
LS2088A Security (SEC) Reference Manual

Also supports LS2048A, LS2084A and LS2044A

Document Number: LS2088ASECRM
Rev. 0, 04/2018





Contents

Section number	Title	Page
Chapter 1		
Overview of SEC (security engine) functionality		
Chapter 2		
Feature summary		
Chapter 3		
SEC implementation		
3.1	SEC submodules.....	91
3.2	Cryptographic engines implemented in SEC.....	91
3.3	SEC Export-Controlled vs. non-Export-Controlled Versions.....	92
Chapter 4		
SEC modes of operation		
4.1	Security Monitor (SecMon) security states.....	95
4.1.1	The effect of security state on volatile keys.....	96
4.1.2	The effect of security state on non-volatile keys.....	97
4.2	Keys available in different security modes.....	97
4.2.1	Keys available in trusted mode.....	97
4.2.2	Keys available in secure mode.....	98
4.2.3	Keys available in non-secure mode.....	98
4.2.4	Keys available in fail mode.....	99
Chapter 5		
SEC hardware functional description		
5.1	System Bus Interfaces.....	102
5.1.1	AXI master (DMA) interface.....	102
5.1.1.1	DMA read-safe transactions.....	102
5.1.1.2	DMA interface write-safe transactions.....	103
5.1.1.3	DMA write-efficient transactions.....	103
5.1.1.4	DMA bursts that may read past the end of data structures.....	104
5.1.2	Register interface (IP bus).....	105

Section number	Title	Page
5.2	SEC service interface concepts.....	106
5.2.1	SEC descriptors.....	106
5.2.2	Job termination status/error codes.....	108
5.2.3	Frames and flows.....	115
5.2.4	Frame descriptors and frames.....	116
5.2.5	Frame descriptor flow and flow context.....	116
5.2.6	Buffer allocation, release, and reuse.....	117
5.2.7	User data access control and isolation.....	118
5.3	Service interfaces.....	118
5.3.1	Job Ring interface.....	119
5.3.1.1	Configuring and managing the input/output rings, overview.....	120
5.3.1.2	Managing the input rings.....	122
5.3.1.3	Managing the output rings.....	123
5.3.1.4	Controlling access to Job Rings.....	124
5.3.1.5	Order of job completion.....	124
5.3.1.6	Initializing Job Rings.....	125
5.3.1.7	Job Ring Registers.....	125
5.3.1.8	Asserting Job Ring interrupts.....	125
5.3.2	Queue Manager Interface (QI).....	126
5.3.2.1	Requesting and receiving frame descriptors from QMan.....	127
5.3.2.2	Building job descriptors for QI jobs.....	129
5.3.2.3	Controlling QI access to frame queues and data.....	129
5.3.2.4	Tracking the completion order of QI jobs.....	130
5.3.2.5	Initializing the Queue Manager Interface.....	130
5.3.2.6	Done/error notification for QI jobs.....	131
5.3.3	Advanced IO Processor Interface (AI).....	132
5.3.3.1	Receiving frame descriptors from AIOP.....	133
5.3.3.2	Building job descriptors for AI jobs.....	134
5.3.3.3	Controlling AI access to data.....	135

Section number	Title	Page
5.3.3.4	Tracking the order of completion of AI jobs.....	135
5.3.3.5	Initializing the AIOP interface.....	136
5.3.3.6	Done/error notification for AI jobs.....	136
5.3.4	Register-based service interface.....	137
5.4	Job scheduling.....	139
5.4.1	Job scheduling - default algorithm.....	139
5.4.2	Job scheduling - adaptive resource reservation algorithm.....	142
5.4.3	Job scheduling - DECO-specific jobs.....	145
5.5	Job execution hardware.....	146
5.5.1	Descriptor controller (DECO) and cryptographic control block (CCB).....	146
5.5.1.1	Alignment blocks.....	147
5.5.2	Cryptographic hardware accelerators (CHAs) (overview).....	148

Chapter 6

Frame queues, frame descriptors, and buffers

6.1	Frame queues.....	151
6.1.1	Dequeue response.....	152
6.2	Multi-partition resource access.....	153
6.2.1	Multi-partition resource access modes.....	154
6.2.2	Flow context selection restrictions.....	157
6.2.3	Inline job descriptor restrictions.....	158
6.2.4	Replacement job descriptor restrictions.....	158
6.2.5	Non-local jump limitations.....	159
6.2.6	Multi-partition resource access restriction summary.....	159
6.3	Frame descriptors.....	160
6.3.1	Flow Context.....	160
6.3.2	Processing single frame jobs.....	168
6.3.3	Processing frame list jobs.....	170
6.3.4	Frame descriptor error handling.....	171
6.3.5	Job descriptor construction from frame descriptor.....	172

Section number	Title	Page
Chapter 7		
Descriptors and descriptor commands		
7.1	Job descriptors.....	175
7.2	Trusted descriptors.....	177
7.3	Shared descriptors.....	179
7.3.1	Executing shared descriptors in proper order.....	180
7.3.2	Specifying different types of shared descriptor sharing.....	181
7.3.2.1	Error sharing.....	183
7.3.3	Changing shared descriptors.....	183
7.4	Using in-line descriptors.....	184
7.5	Using replacement job descriptors.....	185
7.6	Scatter/gather tables (SGTs).....	187
7.7	Using descriptor commands.....	190
7.7.1	Command execution order.....	190
7.7.1.1	Executing commands when SHR = 0.....	192
7.7.1.2	Executing commands when SHR = 1.....	193
7.7.1.3	Executing commands when REO = 0.....	193
7.7.1.4	Executing commands when REO = 1.....	194
7.7.1.5	Executing additional HEADER commands.....	195
7.7.1.6	Jumping to another job descriptor.....	196
7.7.2	Command properties.....	197
7.7.2.1	Blocking commands.....	197
7.7.2.2	Load/store checkpoint.....	197
7.7.2.3	Done checkpoint.....	197
7.7.3	Command types.....	198
7.7.4	SEQ vs non-SEQ commands.....	199
7.7.4.1	Creating a sequence.....	200
7.7.4.2	Using sequences for fixed and variable length data.....	201
7.7.4.3	Transferring meta data.....	202

Section number	Title	Page
7.7.4.4	Rewinding a sequence.....	203
7.7.5	Information FIFO entries.....	204
7.7.6	Output FIFO Operation.....	204
7.7.7	Output Checksum logic.....	207
7.7.8	Cryptographic class.....	208
7.7.9	Address pointers.....	209
7.8	HEADER command.....	210
7.9	KEY commands.....	215
7.10	LOAD commands.....	219
7.11	FIFO LOAD command.....	231
7.11.1	Bit length data.....	234
7.11.2	FIFO LOAD input data type	235
7.12	ECPARAM command.....	237
7.13	STORE command.....	241
7.14	FIFO STORE command.....	249
7.15	MOVE, MOVEB, MOVEDW, and MOVE_LEN commands.....	255
7.16	ALGORITHM OPERATION command.....	264
7.17	PROTOCOL OPERATION commands.....	270
7.18	PKHA OPERATION command.....	290
7.18.1	PKHA OPERATION: clear memory function.....	292
7.18.2	PKHA OPERATION: Arithmetic Functions.....	293
7.18.3	PKHA OPERATION: copy memory functions.....	299
7.18.4	PKHA OPERATION: Elliptic Curve Functions.....	302
7.19	SIGNATURE command.....	305
7.20	JUMP (HALT) command.....	308
7.20.1	Jump type.....	308
7.20.1.1	Local conditional jump.....	309
7.20.1.2	Local conditional increment/decrement jump.....	309
7.20.1.3	Non-local conditional jump.....	310

Section number	Title	Page
7.20.1.4	Conditional halt.....	310
7.20.1.5	Conditional halt with user-specified status.....	311
7.20.1.6	Conditional subroutine call.....	312
7.20.1.7	Conditional subroutine return.....	312
7.20.2	Test type.....	313
7.20.3	JSL and TEST CONDITION fields.....	313
7.20.4	JUMP command format.....	315
7.21	MATH and MATHI Commands.....	318
7.22	SEQ IN PTR command.....	325
7.23	SEQ OUT PTR command.....	329

Chapter 8 Public Key Cryptography Operations

8.1	Conformance considerations.....	333
8.2	Discrete-log key-pair generation.....	334
8.2.1	Inputs to the discrete-log key-pair generation function.....	335
8.2.2	Assumptions of the discrete-log key-pair generation function.....	335
8.2.3	Outputs from the discrete-log key-pair generation function.....	335
8.2.4	Operation of the discrete-log key-pair generation function.....	335
8.2.5	Notes associated with the discrete-log key-pair generation function	336
8.3	Using the Diffie_Hellman function.....	340
8.3.1	Diffie_Hellman requirements.....	340
8.3.2	Inputs to the Diffie-Hellman function.....	340
8.3.3	Assumptions of the Diffie-Hellman function.....	341
8.3.4	Outputs from the Diffie-Hellman function.....	341
8.3.5	Operation of the Diffie-Hellman function.....	341
8.3.6	Notes associated with the Diffie-Hellman function.....	341
8.4	Generating DSA and ECDSA signatures.....	342
8.4.1	Inputs to the DSA and ECDSA signature generation function.....	343
8.4.2	Assumptions of the DSA and ECDSA signature generation function.....	343

Section number	Title	Page
8.4.3	Outputs from the DSA and ECDSA signature generation function.....	343
8.4.4	Operation of the DSA and ECDSA signature generation function	344
8.4.5	Notes associated with the DSA and ECDSA Signature Generation function.....	344
8.5	Verifying DSA and ECDSA signatures.....	347
8.5.1	Inputs to the DSA and ECDSA signature verification function.....	348
8.5.2	Assumptions of the DSA and ECDSA signature verification function.....	348
8.5.3	Outputs from the DSA and ECDSA signature verification function.....	348
8.5.4	Operation of the DSA and ECDSA signature verification function	348
8.5.5	Notes associated with the DSA and ECDSA Signature Verification function	349
8.6	RSA Finalize Key Generation (RFKG).....	352
8.7	Implementation of the RSA encrypt operation.....	353
8.8	Implementation of the RSA decrypt operation.....	355

Chapter 9 Protocol acceleration

9.1	IPsec ESP encapsulation and decapsulation overview.....	363
9.1.1	IPsec ESP encapsulation and decapsulation mode support.....	365
9.1.2	IPsec ESP error codes.....	365
9.1.3	Programming for IPsec	366
9.1.3.1	PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation.....	367
9.1.3.2	Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation.....	370
9.1.3.3	Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register.....	373
9.1.3.4	PDB format for IPsec ESP Tunnel encapsulation	374
9.1.3.5	Common PDB format descriptions for IPsec ESP Tunnel decapsulation.....	377
9.1.3.6	Overriding ESP Tunnel PDB content with the DECO Protocol Override Register.....	380
9.1.3.7	IPsec ESP encapsulation CBC-specific PDB segment format descriptions.....	382
9.1.3.8	IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions.....	383
9.1.3.9	IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions.....	383
9.1.3.10	IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions.....	384

Section number	Title	Page
9.1.3.11	IPsec ESP decapsulation CBC-specific PDB segment format descriptions.....	384
9.1.3.12	IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions.....	385
9.1.3.13	IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions.....	385
9.1.3.14	IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions.....	385
9.1.4	IPsec ESP Transport (and Legacy Tunnel) encapsulation overview.....	386
9.1.4.1	Encapsulating the IP header in tunnel mode.....	387
9.1.4.2	Encapsulating the IP header in transport mode.....	387
9.1.4.3	Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation.....	388
9.1.5	IPsec ESP Cryptographic Encapsulation.....	389
9.1.5.1	Process for IPsec encapsulation when using AES-CBC or DES-CBC.....	389
9.1.5.2	Process for IPsec encapsulation when using AES-CTR.....	391
9.1.5.3	Process for IPsec encapsulation when using AES-CCM.....	392
9.1.5.4	Process for IPsec encapsulation when using AES-GCM.....	394
9.1.6	IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure overview.....	395
9.1.6.1	IPsec ESP Transport Mode outer IP header decapsulation procedure.....	397
9.1.6.2	IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode).....	397
9.1.7	IPsec ESP Cryptographic Decapsulation.....	398
9.1.7.1	IPsec decapsulation procedure when using AES-CBC or DES-CBC.....	398
9.1.7.2	Process for IPsec decapsulation when using AES-CTR.....	400
9.1.7.3	Process for IPsec decapsulation when using AES-CCM.....	401
9.1.7.4	Process for IPsec decapsulation when using AES-GCM.....	402
9.1.7.5	Use of SPI and the sequence number in decapsulation.....	403
9.1.7.6	Optional use of ESN in ESP decapsulation.....	404
9.1.7.7	Anti-replay checking in IPsec ESP decapsulation.....	404
	9.1.7.7.1 When anti-replay checking is enabled.....	405
	9.1.7.7.2 When anti-replay checking is disabled.....	405
9.1.7.8	ICV checking during IPsec ESP decapsulation.....	406
9.1.8	IPsec ESP Tunnel encapsulation overview.....	406

Section number	Title	Page
9.1.8.1	Handling the Outer IP Header during ESP Tunnel encapsulation.....	406
9.1.8.2	Outer IP Header handling with UDP-encapsulated-ESP.....	408
9.1.8.3	ESP Tunnel Outer IP Header manipulation.....	408
9.1.8.4	ESP Tunnel handling of Next Header.....	409
9.1.9	IPsec ESP tunnel decapsulation overview.....	409
9.1.9.1	Input material preceding the outer IP header.....	410
9.1.9.2	Handling the Outer IP Header during ESP Tunnel decapsulation.....	410
9.1.9.3	Manipulation of the Inner IP Header during ESP Tunnel decapsulation.....	411
9.1.9.4	Decapsulation Output Frame Length.....	411
9.2	SSL/TLS/DTLS record encapsulation and decapsulation overview.....	412
9.2.1	Programming and processing details common to all versions of SSL, TLS, and DTLS.....	413
9.2.1.1	PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation.....	413
9.2.1.1.1	PDB for SSL, TLS, and DTLS when a Block Cipher is used.....	414
9.2.1.1.2	PDB for SSL, TLS, and DTLS when AES-Counter mode is used.....	415
9.2.1.1.3	PDB for TLS and DTLS when AES-GCM is used.....	416
9.2.1.1.4	PDB for TLS and DTLS when AES-CCM is used.....	417
9.2.1.1.5	Programming the Options byte with the PDB for SSL, TLS and DTLS.....	417
9.2.1.2	Overriding the PDB for SSL, TLS, and DTLS Encapsulation.....	419
9.2.1.3	Computing the pre-encrypted record length during decapsulation.....	420
9.2.1.4	SSL, TLS, DTLS Decapsulation Output frame options.....	422
9.2.1.5	SSL / TLS / DTLS error codes.....	423
9.2.2	Process for SSL 3.0 and TLS 1.0 record encapsulation.....	424
9.2.2.1	Differences between SSL 3.0 and TLS 1.0 (record encapsulation).....	425
9.2.2.2	Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers.....	425
9.2.3	Process for SSL 3.0 and TLS 1.0 record decapsulation.....	427
9.2.3.1	SSL 3.0 and TLS 1.0 Record Decapsulation for block ciphers.....	427
9.2.3.2	Differences between SSL 3.0 and TLS 1.0 (record decapsulation).....	428
9.2.4	Process for TLS 1.1 and TLS 1.2 record encapsulation.....	429
9.2.4.1	Differences between TLS 1.0, TLS 1.1, and TLS 1.2 Record Encapsulation.....	429

Section number	Title	Page
9.2.4.2	Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation.....	430
9.2.4.3	Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers (AES or DES)..	432
9.2.4.4	Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers.....	433
9.2.4.5	Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers.....	434
9.2.5	Process for TLS 1.1 and TLS 1.2 record decapsulation.....	435
9.2.5.1	Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used.....	436
9.2.5.2	Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used.....	438
9.2.5.3	Decapsulation of TLS 1.2 records when an AEAD is used.....	439
9.2.6	Process for DTLS record encapsulation.....	440
9.2.6.1	Differences between DTLS and TLS.....	441
9.2.6.2	Process of DTLS Record Encapsulation when using a Block Cipher.....	441
9.2.6.3	Process of DTLS Record Encapsulation when using a Stream Cipher.....	443
9.2.6.4	DTLS 1.2 Record Encapsulation when using an AEAD Cipher.....	444
9.2.7	Process for DTLS record decapsulation.....	445
9.2.7.1	Differences between DTLS and TLS.....	446
9.2.7.2	Process of DTLS Record Decapsulation when using a Block Cipher.....	446
9.2.7.3	Process of DTLS Record Decapsulation when using a Stream Cipher.....	448
9.2.7.4	DTLS 1.2 Record Decapsulation when using an AEAD Cipher.....	449
9.3	SRTP packet encapsulation and decapsulation.....	451
9.3.1	Building the initial counter value (Counter IV).....	452
9.3.2	Building the AEAD Nonce.....	452
9.3.3	Constructing the AESA context from the SRTP AEAD Nonce for AES-CCM mode.....	453
9.3.4	SRTP encapsulation.....	454
9.3.4.1	Process for SRTP encapsulation.....	455
9.3.4.2	Handling the optional MKI.....	456
9.3.4.3	SRTP encapsulation PDB format descriptions.....	456
9.3.4.4	SRTP encapsulation error conditions.....	457
9.3.5	SRTP decapsulation overview.....	457
9.3.5.1	Process for SRTP decapsulation.....	458

Section number	Title	Page
	9.3.5.2 SRTP decapsulation PDB format descriptions.....	459
	9.3.5.3 SRTP decapsulation error conditions.....	461
9.4	IEEE 802.1AE MACsec encapsulation and decapsulation overview.....	461
9.4.1	Process for 802.1AE MACsec encapsulation.....	462
9.4.1.1	Using the frame check sequence (FCS).....	464
9.4.1.2	Additional notes for GMAC support.....	465
9.4.2	MACsec encapsulation PDB format descriptions.....	466
9.4.3	Process for 802.1AE MACSec decapsulation.....	467
9.4.3.1	Automatically switching between two keys.....	469
9.4.3.2	Additional notes for GMAC support (decapsulation).....	470
9.4.4	MACsec decapsulation PDB format descriptions.....	470
9.5	IEEE 802.11 -2012 WPA2 MPDU encapsulation and decapsulation.....	471
9.5.1	Processing Common to WPA2 Encapsulation and Decapsulation.....	472
9.5.1.1	Constructing the AAD for WPA2 encapsulation and decapsulation.....	472
9.5.1.2	Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation.....	473
9.5.1.3	Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation.....	473
9.5.2	Process for WPA2 encapsulation.....	474
9.5.2.1	Constructing the CCMP header for WPA2 encapsulation.....	475
9.5.2.2	WPA2 Payload Encapsulation.....	476
9.5.2.3	Computing the FCS for WPA2 encapsulation.....	476
9.5.2.4	WPA2 encapsulation PDB format descriptions.....	476
9.5.2.5	WPA2 encapsulation error conditions.....	478
9.5.3	Process for WPA2 decapsulation.....	478
9.5.3.1	WPA2 Decapsulation Anti-replay checking.....	479
9.5.3.2	Using automatic key-switching.....	480
9.5.3.3	WPA2 decapsulation PDB format descriptions.....	480
9.5.3.4	WPA2 decapsulation error conditions.....	481
9.6	IEEE 802.16 WiMAX encapsulation and decapsulation overview.....	482
9.6.1	Process for IEEE 802.16 WiMAX encapsulation.....	483

Section number	Title	Page
9.6.2	IEEE 802.16 WiMAX encapsulation PDB format descriptions.....	485
9.6.3	WiMax encapsulation error conditions.....	486
9.6.4	Procedure for IEEE 802.16 WiMAX decapsulation.....	486
9.6.4.1	Transforming the GMH (WiMAX decapsulation).....	488
9.6.4.2	Automatic key switching (WiMAX decapsulation).....	489
9.6.5	IEEE 802.16 WiMAX decapsulation PDB format descriptions.....	489
9.6.6	WiMAX decapsulation error conditions.....	490
9.7	Anti-Replay built-in checking.....	490
9.8	Process for 3G double-CRC encapsulation and decapsulation.....	493
9.8.1	3G double-CRC encapsulation process.....	494
9.8.1.1	Calculating the 7-bit CRC of the PDU header for encapsulation.....	494
9.8.1.2	Calculating the 11-bit CRC of the PDU header for encapsulation.....	495
9.8.1.3	Calculating the 16-bit payload CRC for encapsulation.....	496
9.8.2	3G double-CRC encapsulation PDB format descriptions.....	496
9.8.3	3G double-CRC decapsulation process.....	496
9.8.3.1	Calculating the 7-bit CRC of the PDU header for decapsulation.....	497
9.8.3.2	Calculating the 11-bit CRC of the PDU header for decapsulation.....	497
9.8.3.3	Calculating the 16-bit payload CRC for decapsulation.....	498
9.8.4	3G double-CRC decapsulation PDB format descriptions.....	498
9.9	3G RLC PDU Encapsulation and Decapsulation overview.....	499
9.9.1	3G RLC PDU encapsulation overview.....	499
9.9.2	Process for 3G RLC PDU encapsulation.....	500
9.9.3	3G RLC PDU encapsulation PDB format descriptions.....	501
9.9.4	3G RLC PDU decapsulation overview.....	502
9.9.5	Process for 3G RLC PDU decapsulation.....	503
9.9.6	3G RLC PDU decapsulation PDB format descriptions.....	504
9.9.7	Overriding the PDB for 3G RLC PDU encapsulation and decapsulation.....	504
9.10	LTE PDCP PDU encapsulation and decapsulation overview.....	505
9.10.1	LTE PDCP PDU IV generation.....	506

Section number	Title	Page
9.10.2	LTE PDCP PDU encapsulation process for confidentiality only.....	509
9.10.3	LTE PDCP PDU encapsulation for confidentiality and integrity.....	511
9.10.4	LTE PDCP PDU decapsulation process for confidentiality only.....	512
9.10.5	LTE PDCP PDU decapsulation for confidentiality and integrity.....	513
9.10.6	LTE PDCP shared descriptor PDB format descriptions.....	515
9.10.7	Overriding the PDB for LTE PDCP encapsulation and decapsulation.....	515

Chapter 10

Key agreement functions

10.1	IKEv2 PRF overview.....	517
10.1.1	Using IKE PRF to generate SKEYSEED.....	518
10.1.2	Using IKE PRF+ to generate keying material for the IKEv2 SA.....	518
10.1.3	Using IKE PRF+ to generate Child SA key material.....	519
10.1.4	Restrictions on programming control blocks.....	519
10.1.5	IKE PRF PDB format descriptions.....	520
10.1.6	Implementation details for IKE PRF function.....	523
10.1.7	Implementation Details for IKE PRF+ function.....	524
10.2	SSL/TLS/DTLS pseudo-random functions (PRF).....	525
10.2.1	SSL 3.0 PRF overview.....	525
10.2.1.1	SSL 3.0 PRF definitions.....	526
10.2.2	Process for SSL 3.0 PRF.....	527
10.2.3	SSL 3.0 PRF PDB format descriptions.....	527
10.2.4	TLS 1.0/TLS 1.1/DTLS PRF overview.....	530
10.2.4.1	TLS PRF RFC definitions.....	532
10.2.5	Process for TLS 1.0, TLS 1.1, DTLS PRF.....	533
10.2.5.1	How TLS uses PRF material.....	533
10.2.5.2	Concatenating input material into one input string (TLS 1.0/1.1/DTLS).....	534
10.2.6	TLS 1.0, TLS 1.1, DTLS PRF PDB format descriptions.....	535
10.2.7	TLS 1.2 PRF overview.....	538
10.2.8	Process for TLS 1.2 PRF.....	539

Section number	Title	Page
10.2.8.1	Concatenating input material into one input string (TLS 1.2).....	540
10.2.8.2	How TLS uses PRF material (TLS 1.2).....	540
10.2.9	TLS 1.2 PRF PDB format descriptions.....	541
10.3	Implementation of the derived key protocol.....	543
10.3.1	Using DKP with HMAC keys.....	544
10.3.2	Implementation of the Blob Protocol.....	545

Chapter 11

Cryptographic hardware accelerators (CHAs)

11.1	Public-key hardware accelerator (PKHA) functionality.....	548
11.1.1	Modular math.....	549
11.1.2	About Montgomery values.....	549
11.1.3	Non-modular Math.....	551
11.1.4	Elliptic-Curve Math.....	551
11.1.4.1	ECC_MOD: Point math on a standard curve over a prime field (Fp).....	552
11.1.4.2	ECC_F2M: Point math on a standard curve over a binary field (F2m).....	553
11.1.5	PKHA Mode Register.....	553
11.1.6	PKHA functions.....	554
11.1.6.1	Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_S SZ).....	555
11.1.6.2	Clear Memory (CLEAR_MEMORY) function.....	556
11.1.6.3	Arithmetic Functions.....	557
11.1.6.3.1	Integer Modular Addition (MOD_ADD) function.....	557
11.1.6.3.2	Integer Modular Subtraction (MOD_SUB_1) function.....	558
11.1.6.3.3	Integer Modular Subtraction (MOD_SUB_2) function.....	558
11.1.6.3.4	Integer Modular Multiplication (MOD_MUL).....	559
11.1.6.3.5	Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM).....	559
11.1.6.3.6	Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function.....	560
11.1.6.3.7	Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ).....	561
11.1.6.3.8	Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function.....	562

Section number	Title	Page
11.1.6.3.9	Integer Simultaneous Modular Exponentiation (MOD_SML_EXP).....	562
11.1.6.3.10	Integer Modular Square (MOD_SQR and MOD_SQR_TEQ).....	563
11.1.6.3.11	Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ).....	564
11.1.6.3.12	Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ).....	564
11.1.6.3.13	Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ).....	565
11.1.6.3.14	Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ).....	566
11.1.6.3.15	Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ).....	566
11.1.6.3.16	Integer Modulo Reduction (MOD_AMODN).....	567
11.1.6.3.17	Integer Modular Inversion (MOD_INV).....	568
11.1.6.3.18	Integer Montgomery Factor Computation (MOD_R2).....	568
11.1.6.3.19	Integer Greatest Common Divisor (MOD_GCD).....	569
11.1.6.3.20	Miller_Rabin Primality Test (PRIME_TEST).....	569
11.1.6.3.21	Right Shift A (RIGHT_SHIFT_A) function.....	570
11.1.6.3.22	Compare A B (COMPARE) function.....	570
11.1.6.3.23	Evaluate A (EVALUATE) function.....	571
11.1.6.3.24	Binary Polynomial (F2m) Addition (F2M_ADD) function.....	571
11.1.6.3.25	Binary Polynomial (F2m) Modular Multiplication (F2M_MUL).....	572
11.1.6.3.26	Binary Polynomial (F2m) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function.....	573
11.1.6.3.27	Binary Polynomial (F2m) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function.....	573
11.1.6.3.28	Binary Polynomial (F2m) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ).....	574
11.1.6.3.29	Binary Polynomial (F2m) Simultaneous Modular Exponentiation (F2M_SML_EXP).....	575
11.1.6.3.30	Binary Polynomial (F2m) Modular Square (F2M_SQR and F2M_SQR_TEQ).....	576

Section number	Title	Page
11.1.6.3.31	Binary Polynomial (F2m) Modular Square, Montgomery Input (F2M_SQR_IM and F2M_SQR_IM_TEQ).....	576
11.1.6.3.32	Binary Polynomial (F2m) Modular Square, Montgomery Input and Output (F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ).....	577
11.1.6.3.33	Binary Polynomial (F2m) Modular Cube (F2M_CUBE and F2M_CUBE_TEQ).....	578
11.1.6.3.34	Binary Polynomial (F2m) Modular Cube, Montgomery Input (F2M_CUBE_IM and F2M_CUBE_IM_TEQ).....	579
11.1.6.3.35	Binary Polynomial (F2m) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ).....	579
11.1.6.3.36	Binary Polynomial (F2m) Modulo Reduction (F2M_AMODN).....	580
11.1.6.3.37	Binary Polynomial (F2m) Modular Inversion (F2M_INV).....	581
11.1.6.3.38	Binary Polynomial (F2m) R2 Mod N (F2M_R2) Function.....	581
11.1.6.3.39	Binary Polynomial (F2m) Greatest Common Divisor (F2M_GCD) Function.....	582
11.1.6.4	Elliptic Curve Functions.....	582
11.1.6.4.1	ECC Fp Point Add, Affine Coordinates (ECC_MOD_ADD) Function.....	582
11.1.6.4.2	ECC Fp Point Add, Affine Coordinates, R2 Mod N Input (ECC_MOD_ADD_R2) Function.....	583
11.1.6.4.3	ECC Fp Point Double, Affine Coordinates (ECC_MOD_DBL) Function.....	584
11.1.6.4.4	ECC Fp Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function.....	584
11.1.6.4.5	ECC Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function.....	586
11.1.6.4.6	ECC Fp Check Point (ECC_MOD_CHECK_POINT) Function.....	587
11.1.6.4.7	ECC Fp Check Point, R2 Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function.....	588
11.1.6.4.8	ECC F2m Point Add, Affine Coordinates (ECC_F2M_ADD) Function.....	589
11.1.6.4.9	ECC F2m Point Add, Affine Coordinates, R2 Mod N Input (ECC_F2M_ADD_R2) Function.....	589
11.1.6.4.10	ECC F2m Point Double - Affine Coordinates (ECC_F2M_DBL) Function.....	590
11.1.6.4.11	ECC F2m Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function.....	591

Section number	Title	Page
11.1.6.4.12	ECC F2m Point Multiply, R2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function.....	592
11.1.6.4.13	ECC F2m Check Point (ECC_F2M_CHECK_POINT) Function.....	593
11.1.6.4.14	ECC F2m Check Point, R2 (ECC_F2M_CHECK_POINT_R2) Function.....	594
11.1.6.4.15	ECM Modular Multiplication (ECM_MOD_MUL_X and ECM_MOD_MUL_X_TEQ) Function.....	595
11.1.6.4.16	ECM Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECM_MOD_MUL_X_R2 and ECM_MOD_MUL_X_R2_TEQ) Function..	596
11.1.6.4.17	ECT Modular Multiplication (ECT_MOD_MUL and ECT_MOD_MUL_TEQ) Function.....	597
11.1.6.4.18	ECT Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ) Function.....	599
11.1.6.4.19	ECT Fp Point Add, Affine Coordinates (ECT_MOD_ADD) Function.....	600
11.1.6.4.20	ECT Fp Point Add, Affine Coordinates, R2 Mod N Input (ECT_MOD_ADD_R2) Function.....	600
11.1.6.4.21	ECT Fp Check Point (ECT_MOD_CHECK_POINT) Function.....	601
11.1.6.4.22	ECT Fp Check Point, R2 (ECT_MOD_CHECK_POINT_R2) Function.....	602
11.1.6.4.23	Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ).....	603
11.1.6.4.24	Right Shift A (R_SHIFT) function.....	604
11.1.6.4.25	Compare A B (COMPARE) function.....	604
11.1.6.4.26	Evaluate A (EVALUATE) function.....	605
11.1.6.5	Special values for common ECC domains.....	606
11.2	Kasumi f8 and f9 hardware accelerator(KFHA) functionality.....	625
11.2.1	KFHA use of the Mode Register.....	625
11.2.2	KFHA use of the Context Register.....	626
11.2.3	KFHA use of the Key Register.....	627
11.2.4	KFHA use of the Data Size Register.....	627
11.2.5	KFHA error conditions.....	627
11.3	Data encryption standard accelerator (DES) functionality.....	628
11.3.1	DESA use of the Mode Register.....	628

Section number	Title	Page
11.3.2	DESA use of the Key Register.....	629
11.3.3	DESA use of the Key Size Register.....	629
11.3.4	DESA use of the Data Size Register.....	629
11.3.5	DESA Context Register.....	630
11.3.6	Save and store operations in DESA context data.....	630
11.4	Cyclic-redundancy check accelerator (CRCA) functionality.....	630
11.4.1	CRCA modes of operation.....	631
11.4.2	CRCA use of the Mode Register.....	631
11.4.3	CRCA Key Register.....	633
11.4.4	CRCA Key Size Register.....	633
11.4.5	CRCA Data Size Register.....	633
11.4.6	CRCA Context Register.....	633
11.4.7	Save and restore operations in CRCA context data.....	634
11.5	Random-number generator (RNG) functionality.....	634
11.5.1	RNG features summary.....	634
11.5.2	RNG functional description	635
11.5.2.1	RNG state handles.....	635
11.5.2.2	RNG NIST certification.....	635
11.5.3	RNG operations.....	637
11.5.4	RNG use of the Key Registers.....	638
11.5.5	RNG use of the Context Register.....	638
11.5.6	RNG use of the Data Size Register.....	639
11.6	SNOW 3G f8 accelerator functionality.....	639
11.6.1	Differences between SNOW 3G f8 and SNOW 3G f9.....	639
11.6.2	SNOW 3G f8 use of the Mode Register.....	640
11.6.3	SNOW 3G f8 use of the Context Register.....	641
11.6.4	SNOW 3G f8 use of the Data Size Register.....	641
11.6.5	SNOW 3G f8 use of the Key Register.....	642
11.6.6	SNOW 3G f8 use of the Key Size Register.....	642

Section number	Title	Page
11.7	SNOW 3G f9 accelerator functionality.....	642
11.7.1	SNOW 3G f9 use of the Mode Register.....	643
11.7.2	SNOW 3G f9 use of the Context Register.....	644
11.7.3	SNOW 3G f9 use of the Data Size Register.....	645
11.7.4	SNOW 3G f9 use of the Key Register.....	645
11.7.5	SNOW 3G f9 use of the Key Size Register.....	646
11.7.6	SNOW 3G f9 use of ICV check.....	646
11.8	Message digest hardware accelerator (MDHA) functionality.....	646
11.8.1	MDHA use of the Mode Register.....	647
11.8.2	MDHA use of the Key Register.....	648
11.8.2.1	Using the MDHA Key Register with normal keys.....	648
11.8.2.2	Using the MDHA Key Register with IPAD/OPAD "split keys".....	648
11.8.2.2.1	Definition and function of IPAD/OPAD split keys.....	649
11.8.2.2.2	Process flow of using the Key Register with split keys.....	649
11.8.2.2.3	Using padding with the split key type to align with storage.....	649
11.8.2.2.4	Length of a split key.....	649
11.8.2.2.5	Loading/storing a split key with a key command.....	649
11.8.2.2.6	Loading/storing a split key with a FIFO STORE command.....	650
11.8.2.2.7	Sizes of split keys.....	650
11.8.2.2.8	Constructing an HMAC-SHA-1 split key in memory.....	650
11.8.2.3	MDHA use of the Key Size Register.....	651
11.8.3	MDHA use of the Data Size Register.....	651
11.8.4	MDHA use of the Context Register.....	652
11.8.5	Save and restore operations in MDHA context data.....	652
11.9	AES accelerator (AESA) functionality.....	652
11.9.1	Differences between the AES encrypt and decrypt keys.....	652
11.9.2	AESA as both Class 1 and Class 2 CHA.....	653
11.9.3	AESA modes of operation.....	654
11.9.4	AESA use of registers.....	655

Section number	Title	Page
11.9.5	AESA use of the parity bit.....	656
11.9.6	AES ECB mode.....	656
11.9.6.1	AES ECB mode use of the Mode Register.....	656
11.9.6.2	AES ECB mode use of the Context Register.....	657
11.9.6.3	AES ECB Mode use of the Data Size Register	657
11.9.6.4	AES ECB Mode use of the Key Register.....	657
11.9.6.5	AES ECB Mode use of the Key Size Register.....	657
11.9.7	AES CBC, OFB, CFB128 modes.....	658
11.9.7.1	AES CBC, OFB, and CFB128 modes use of the Mode Register.....	658
11.9.7.2	AES CBC, OFB, and CFB128 modes use of the Context Register.....	659
11.9.7.3	AES CBC, OFB, and CFB128 modes use of the Data Size Register.....	659
11.9.7.4	AES CBC, OFB, and CFB128 modes use of the Key Register.....	660
11.9.7.5	AES CBC, OFB, and CFB128 modes use of the Key Size Register.....	660
11.9.8	AES CTR mode.....	660
11.9.8.1	AES CTR mode use of the Mode Register.....	660
11.9.8.2	AES CTR mode use of the Context Register.....	661
11.9.8.3	AES CTR mode use of the Data Size Register.....	661
11.9.8.4	AES CTR mode use of the Key Register.....	661
11.9.8.5	AES CTR mode use of the Key Size Register.....	662
11.9.9	AES XTS mode.....	662
11.9.9.1	AES XTS mode use of the Mode Register.....	662
11.9.9.2	AES XTS mode use of the Context Register.....	662
11.9.9.3	AES XTS mode use of the Data Size Register.....	663
11.9.9.4	AES XTS mode use of the Key Register.....	663
11.9.9.5	AES XTS mode use of the Key Size Register.....	664
11.9.10	AES XCBC-MAC and CMAC modes.....	664
11.9.10.1	AES XCBC-MAC and CMAC modes use of the Mode Register.....	664
11.9.10.2	AES XCBC-MAC and CMAC Modes use of the Context Register.....	666
11.9.10.3	AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register.....	666

Section number	Title	Page
11.9.10.4	AES XCBC-MAC and CMAC modes use of the Data Size Register.....	667
11.9.10.5	AES XCBC-MAC and CMAC modes use of the Key Register.....	667
11.9.10.6	AES XCBC-MAC and CMAC modes use of the Key Size Register.....	667
11.9.10.7	ICV checking in AES XCBC-MAC and CMAC modes.....	667
11.9.11	AESA CCM mode.....	668
11.9.11.1	Generation encryption.....	668
11.9.11.2	Decryption verification.....	668
11.9.11.3	AES CCM mode use of the Mode Register.....	669
11.9.11.4	AES CCM mode use of the Context Register.....	670
11.9.11.5	AES CCM mode use of the Data Size Register.....	671
11.9.11.6	AES CCM mode use of the Key Register.....	671
11.9.11.7	AES CCM mode use of the Key Size Register.....	671
11.9.11.8	AES CCM mode use of the ICV check.....	672
11.9.12	AES GCM mode.....	672
11.9.12.1	GMAC.....	673
11.9.12.2	GCM data types.....	673
11.9.12.3	IV processing.....	673
11.9.12.4	GCM initialization.....	673
11.9.12.5	AES GCM mode use of the Mode Register.....	673
11.9.12.6	AES GCM mode use of the Context Register.....	675
11.9.12.7	AES GCM Mode use of the Data Size Register.....	675
11.9.12.8	AES GCM mode use of the Class 1 IV Size Register.....	676
11.9.12.9	AES GCM mode use of the AAD Size Register.....	676
11.9.12.10	AES GCM mode use of the Class 1 ICV Size Register.....	676
11.9.12.11	AES GCM mode use of the Key Register.....	676
11.9.12.12	AES GCM mode use of the Key Size Register.....	676
11.9.12.13	AES GCM mode use of the ICV check.....	677
11.9.13	AESA optimization modes.....	677
11.9.13.1	CTR-XCBC and CTR-CMAC modes data format.....	677

Section number	Title	Page
11.9.13.2	CTR-XCBC and CTR-CMAC modes message format.....	677
11.9.13.3	CTR-CMAC-LTE for LTE PDCP control-plane processing.....	678
11.9.13.4	Authentication-only data.....	678
11.9.13.5	AES optimization modes use of the Mode Register.....	678
11.9.13.6	AES optimization modes use of the Context Register.....	680
11.9.13.7	AES optimization modes use of the Data Size Register.....	683
11.9.13.8	AES optimization modes use of the AAD Size Register.....	684
11.9.13.9	AES optimization modes use of the Class 1 ICV Size Register.....	684
11.9.13.10	AES optimization modes use of the Class 1 Key Register.....	685
11.9.13.11	AES optimization modes use of the Class 2 Key Register.....	685
11.9.13.12	AES optimization modes use of the Class 1 Key Size Register.....	685
11.9.13.13	AES optimization modes use of the Class 2 Key Size Register.....	686
11.9.13.14	AES optimization modes use of the ICV check.....	686
11.9.13.15	AES optimization modes error conditions.....	686
11.10	ZUC encryption accelerator (ZUCE) functionality.....	688
11.10.1	Differences between ZUCE and ZUCA.....	688
11.10.2	ZUCE use of the Mode Register.....	689
11.10.3	ZUCE use of the Context Register.....	689
11.10.4	ZUCE use of the Data Size Register.....	690
11.10.5	ZUCE use of the Key Register.....	690
11.10.6	ZUCE use of the Key Size Register	691
11.11	ZUC authentication accelerator (ZUCA) functionality.....	691
11.11.1	ZUCA use of the Mode Register.....	691
11.11.2	ZUCA use of the Context Register.....	692
11.11.3	ZUCA use of the Data Size Register.....	694
11.11.4	ZUCA use of the Key Register.....	694
11.11.5	ZUCA use of the Key Size Register.....	694
11.11.6	ZUCA use of ICV checking.....	694

Chapter 12

Section number	Title	Page
Trust Architecture modules		
12.1	Run-time integrity checker (RTIC).....	697
12.1.1	RTIC modes of operation.....	697
12.1.2	RTIC initialization and operation.....	697
12.1.3	RTIC use of the Throttle Register.....	698
12.1.4	RTIC use of command, configuration, and status registers.....	698
12.1.5	Initializing RTIC.....	699
12.1.6	RTIC Memory Block Address/Length Registers.....	699
12.2	SEC virtualization and security domain identifiers (SDIDs).....	700
12.2.1	Virtualization.....	700
12.2.2	Security domain identifiers (SDIDs).....	700
12.2.3	TrustZone SecureWorld.....	701
12.3	Special-purpose cryptographic keys.....	701
12.3.1	Initializing and clearing black and trusted descriptor keys.....	702
12.3.2	Black keys and JDKEK/TDKEK.....	702
12.3.3	Trusted descriptors and TDSK.....	702
12.3.4	Master key and blobs.....	702
12.4	Black keys.....	703
12.4.1	Black key encapsulation schemes.....	703
12.4.2	Differences between black and red keys.....	703
12.4.3	Loading red keys.....	704
12.4.4	Loading black keys.....	704
12.4.5	Avoiding errors when loading red and black keys.....	704
12.4.6	Encapsulating and decapsulating black keys.....	705
12.4.7	Types of black keys and their use.....	707
12.4.8	Types of blobs for key storage.....	707
12.5	Trusted descriptors.....	708
12.5.1	Why trusted descriptors are needed.....	708
12.5.2	Trusted-descriptor key types and uses.....	709

Section number	Title	Page
12.5.3	Trusted descriptors encrypting/decrypting black keys.....	709
12.5.4	Trusted-descriptor blob types and uses.....	709
12.5.5	Configuring the system to create trusted descriptors properly.....	710
12.5.6	Creating trusted descriptors.....	710
12.5.6.1	Trusted descriptors and descriptor-header bits.....	711
12.5.6.2	Trusted-descriptor execution considerations.....	711
12.6	Blobs.....	712
12.6.1	Blob protocol.....	713
12.6.2	Why blobs are needed.....	713
12.6.3	Blob conformance considerations.....	713
12.6.4	Encapsulating and decapsulating blobs.....	715
12.6.5	Blob types.....	716
12.6.5.1	Blob types differentiated by format.....	716
12.6.5.2	Blob types differentiated by content.....	717
12.6.5.2.1	Red blobs (for general data).....	718
12.6.5.2.2	Black blobs (for cryptographic keys).....	718
12.6.5.2.3	Enforcing blob content type.....	718
12.6.5.3	Blob types differentiated by security state.....	719
12.6.6	Blob encapsulation.....	719
12.6.7	Blob decapsulation.....	720
12.7	Critical security parameters.....	721
12.8	Manufacturing-protection chip-authentication process.....	722
12.8.1	Providing data to the manufacturing-protection authentication process.....	723
12.8.1.1	Providing data to the MPPrivK-generation function.....	724
12.8.1.2	Providing data to the MPPubK-generation function.....	724
12.8.1.3	Providing data to the MPSign function.....	724
12.8.1.4	Role of the ROM-resident secure boot firmware.....	724
12.8.2	MPPrivK-generation function.....	725

Section number	Title	Page
12.8.2.1	Differences between the MPPrivK-generation function and the DL KEY PAIR GEN function.....	725
12.8.2.2	MPPrivK-generation function parameters and operation.....	726
12.8.2.3	Protocol data block (PDB) for the MPPrivK-generation function.....	726
12.8.3	MPPubK-generation function.....	727
12.8.3.1	Differences between the MPPubK-generation function and the DL KEY PAIR GEN function.....	727
12.8.3.2	MPPubK-generation function parameters and operation.....	728
12.8.3.3	Protocol data block (PDB) for the MPPubK-generation function.....	728
12.8.3.4	Running the MPPubK generation function at the OEM's facility.....	729
12.8.4	MPSign function.....	730
12.8.4.1	MPSign function parameters and operation.....	730
12.8.4.2	Protocol data block (PDB) MPSign function.....	731

Chapter 13

SEC service error detection, recovery (reset), and reconfiguration

13.1	Software SEC Reset.....	733
13.2	Job ring error detection, recovery, reset and reconfiguration.....	734
13.2.1	Job ring user services.....	734
13.2.1.1	Error recovery.....	734
13.2.1.2	Unrecoverable conditions.....	736
13.2.1.3	User reconfiguration options.....	737
13.2.2	Job ring management services.....	737
13.2.2.1	Recoverable error interrupt/status notifications.....	737
13.2.2.2	Ring user access termination procedure.....	738
13.2.2.3	Ring user (re-)assignment procedure.....	738
13.3	QMan interface error detection, recovery, reset, and reconfiguration.....	738
13.3.1	QI user services.....	739
13.3.1.1	Recoverable error conditions.....	739
13.3.1.2	Unrecoverable error conditions.....	740
13.3.1.3	User reconfiguration options.....	740

Section number	Title	Page
13.3.1.4	Flow and queue flush procedures and options.....	740
13.3.2	QI management services.....	741
13.3.2.1	Recoverable error conditions.....	741
13.3.2.2	Unrecoverable error conditions.....	741
13.3.2.3	Single QI user SEC queue use termination procedure.....	742
13.3.2.4	Single QI user SEC access termination procedure.....	742
13.3.2.5	Single QI user access (re-)enable procedure.....	742
13.3.2.6	Global QI user access controls.....	742
13.3.2.7	Queue and ICID flush procedures and options.....	743
13.4	AIOP interface error detection, recovery, reset, and reconfiguration.....	743
13.4.1	Single AI task services.....	744
13.4.1.1	Recoverable error conditions.....	744
13.4.1.2	Unrecoverable error conditions.....	744
13.4.1.3	Task reconfiguration options.....	744
13.4.2	AI management services.....	745
13.4.2.1	Global AIOP task services.....	745
13.4.2.2	Flow and ICID flush procedures and options.....	745
13.5	RTIC error detection, recovery, reset, and reconfiguration.....	745
13.5.1	RTIC user services.....	746
13.5.2	RTIC management services.....	746
13.5.2.1	Recoverable error conditions.....	747
13.5.2.2	Unrecoverable error conditions.....	747
13.5.2.3	Reconfiguration procedure.....	747
13.6	Global and DECO error detection, recovery, reset, and reconfiguration.....	748
13.6.1	Global and DECO user services.....	748
13.6.2	Global SEC and DECO management services.....	748
13.6.2.1	Error detection.....	748
13.6.2.2	Recover procedure.....	749

Chapter 14

Section number	Title	Page
SEC register descriptions		
14.1	SEC Memory map.....	753
14.2	Master Configuration Register (MCFGR).....	823
14.2.1	Offset.....	823
14.2.2	Function.....	823
14.2.3	Diagram.....	823
14.2.4	Fields.....	824
14.3	Security Configuration Register (SCFGR).....	827
14.3.1	Offset.....	827
14.3.2	Function.....	828
14.3.3	Diagram.....	829
14.3.4	Fields.....	830
14.4	Job Ring a ICID Register - most significant half (JR0ICID_MS - JR3ICID_MS).....	831
14.4.1	Offset.....	831
14.4.2	Function.....	831
14.4.3	Diagram.....	832
14.4.4	Fields.....	832
14.5	Job Ring a ICID Register - least significant half (JR0ICID_LS - JR3ICID_LS).....	833
14.5.1	Offset.....	833
14.5.2	Function.....	833
14.5.3	Diagram.....	834
14.5.4	Fields.....	834
14.6	Queue Manager Interface SDID Register (QISDID).....	835
14.6.1	Offset.....	835
14.6.2	Function.....	835
14.6.3	Diagram.....	835
14.6.4	Fields.....	836
14.7	Debug Control Register (DEBUGCTL).....	836
14.7.1	Offset.....	836

Section number	Title	Page
14.7.2	Function.....	836
14.7.3	Diagram.....	837
14.7.4	Fields.....	837
14.8	Job Ring Start Register (JRSTARTR).....	838
14.8.1	Offset.....	838
14.8.2	Function.....	838
14.8.3	Diagram.....	838
14.8.4	Fields.....	838
14.9	RTIC ICID Register for Block a - most significant half (RTICAICID_MS - RTICDICID_MS).....	840
14.9.1	Offset.....	840
14.9.2	Function.....	840
14.9.3	Diagram.....	840
14.9.4	Fields.....	840
14.10	RTIC ICID Register for Block a - least significant half (RTICAICID_LS - RTICDICID_LS).....	841
14.10.1	Offset.....	841
14.10.2	Function.....	841
14.10.3	Diagram.....	841
14.10.4	Fields.....	842
14.11	Protocol Configuration Register (PROTCFG).....	842
14.11.1	Offset.....	842
14.11.2	Function.....	842
14.11.3	Diagram.....	843
14.11.4	Fields.....	843
14.12	DECO Request Source Register (DECORSR).....	843
14.12.1	Offset.....	843
14.12.2	Function.....	844
14.12.3	Diagram.....	844
14.12.4	Fields.....	844
14.13	DECO Request Register (DECORR).....	845

Section number	Title	Page
14.13.1	Offset.....	845
14.13.2	Function.....	845
14.13.3	Diagram.....	846
14.13.4	Fields.....	846
14.14	DECOa ICID Register - most significant half (DECO0ICID_MS - DECO5ICID_MS).....	847
14.14.1	Offset.....	847
14.14.2	Function.....	847
14.14.3	Diagram.....	848
14.14.4	Fields.....	848
14.15	DECOa ICID Register - least significant half (DECO0ICID_LS - DECO5ICID_LS).....	848
14.15.1	Offset.....	848
14.15.2	Function.....	848
14.15.3	Diagram.....	849
14.15.4	Fields.....	849
14.16	DECO Availability Register (DAR).....	850
14.16.1	Offset.....	850
14.16.2	Function.....	850
14.16.3	Diagram.....	850
14.16.4	Fields.....	851
14.17	DECO Reset Register (DRR).....	851
14.17.1	Offset.....	851
14.17.2	Function.....	852
14.17.3	Diagram.....	852
14.17.4	Fields.....	852
14.18	Adaptive Resource Reservation Register (ARRR).....	853
14.18.1	Offset.....	853
14.18.2	Function.....	853
14.18.3	Diagram.....	854
14.18.4	Fields.....	855

Section number	Title	Page
14.19	DMA Control Register (DMAC - DMA_CTRL).....	855
14.19.1	Offset.....	855
14.19.2	Function.....	855
14.19.3	Diagram.....	856
14.19.4	Fields.....	856
14.20	Peak Bandwidth Smoothing Limit Register (PBSL).....	857
14.20.1	Offset.....	857
14.20.2	Function.....	857
14.20.3	Diagram.....	857
14.20.4	Fields.....	858
14.21	DMAa_AIDL_MAP_MS (DMA0_AIDL_MAP_MS - DMA1_AIDL_MAP_MS).....	858
14.21.1	Offset.....	858
14.21.2	Function.....	858
14.21.3	Diagram.....	859
14.21.4	Fields.....	860
14.22	DMAa_AIDL_MAP_LS (DMA0_AIDL_MAP_LS - DMA1_AIDL_MAP_LS).....	860
14.22.1	Offset.....	860
14.22.2	Function.....	860
14.22.3	Diagram.....	861
14.22.4	Fields.....	861
14.23	DMAa_AIDM_MAP_MS (DMA0_AIDM_MAP_MS - DMA1_AIDM_MAP_MS).....	861
14.23.1	Offset.....	862
14.23.2	Function.....	862
14.23.3	Diagram.....	862
14.23.4	Fields.....	862
14.24	DMAa_AIDM_MAP_LS (DMA0_AIDM_MAP_LS - DMA1_AIDM_MAP_LS).....	863
14.24.1	Offset.....	863
14.24.2	Function.....	863
14.24.3	Diagram.....	863

Section number	Title	Page
14.24.4	Fields.....	864
14.25	DMAf AXI ID Enable Register (DMA0_AID_ENB - DMA1_AID_ENB).....	864
14.25.1	Offset.....	864
14.25.2	Function.....	865
14.25.3	Diagram.....	865
14.25.4	Fields.....	865
14.26	DMAa AXI Read Timing Check Register (DMA0_ARD_TC - DMA1_ARD_TC).....	866
14.26.1	Offset.....	866
14.26.2	Function.....	867
14.26.3	Diagram.....	867
14.26.4	Fields.....	868
14.27	DMAf Read Timing Check Latency Register (DMA0_ARD_LAT - DMA1_ARD_LAT).....	869
14.27.1	Offset.....	869
14.27.2	Function.....	869
14.27.3	Diagram.....	870
14.27.4	Fields.....	870
14.28	DMAa AXI Write Timing Check Register (DMA0_AWR_TC - DMA1_AWR_TC).....	870
14.28.1	Offset.....	870
14.28.2	Function.....	871
14.28.3	Diagram.....	871
14.28.4	Fields.....	872
14.29	DMAf Write Timing Check Latency Register (DMA0_AWR_LAT - DMA1_AWR_LAT).....	873
14.29.1	Offset.....	873
14.29.2	Function.....	873
14.29.3	Diagram.....	874
14.29.4	Fields.....	874
14.30	Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63).....	874
14.30.1	Offset.....	874
14.30.2	Function.....	875

Section number	Title	Page
14.30.3	Diagram.....	875
14.30.4	Fields.....	875
14.31	Manufacturing Protection Message Register (MPMR0 - MPMR31).....	875
14.31.1	Offset.....	875
14.31.2	Function.....	876
14.31.3	Diagram.....	876
14.31.4	Fields.....	876
14.32	Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31).....	876
14.32.1	Offset.....	877
14.32.2	Function.....	877
14.32.3	Diagram.....	877
14.32.4	Fields.....	877
14.33	Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7).....	877
14.33.1	Offset.....	877
14.33.2	Function.....	878
14.33.3	Diagram.....	878
14.33.4	Fields.....	879
14.34	Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7).....	879
14.34.1	Offset.....	879
14.34.2	Function.....	879
14.34.3	Diagram.....	880
14.34.4	Fields.....	880
14.35	Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7).....	880
14.35.1	Offset.....	881
14.35.2	Function.....	881
14.35.3	Diagram.....	881
14.35.4	Fields.....	881
14.36	Secure Key Nonce Register (SKNR).....	882
14.36.1	Offset.....	882

Section number	Title	Page
14.36.2	Function.....	882
14.36.3	Diagram.....	882
14.36.4	Fields.....	883
14.37	DMA Status Register (DMA_STA).....	883
14.37.1	Offset.....	883
14.37.2	Diagram.....	884
14.37.3	Fields.....	884
14.38	DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP).....	885
14.38.1	Offset.....	885
14.38.2	Function.....	885
14.38.3	Diagram.....	886
14.38.4	Fields.....	886
14.39	DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP).....	886
14.39.1	Offset.....	886
14.39.2	Function.....	887
14.39.3	Diagram.....	887
14.39.4	Fields.....	887
14.40	DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP).....	888
14.40.1	Offset.....	888
14.40.2	Function.....	888
14.40.3	Diagram.....	888
14.40.4	Fields.....	889
14.41	DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP).....	889
14.41.1	Offset.....	889
14.41.2	Function.....	889
14.41.3	Diagram.....	889
14.41.4	Fields.....	890
14.42	DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN).....	890
14.42.1	Offset.....	890

Section number	Title	Page
14.42.2	Function.....	891
14.42.3	Diagram.....	891
14.42.4	Fields.....	891
14.43	DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL).....	892
14.43.1	Offset.....	892
14.43.2	Function.....	893
14.43.3	Diagram.....	893
14.43.4	Fields.....	894
14.44	DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC).....	895
14.44.1	Offset.....	895
14.44.2	Function.....	895
14.44.3	Diagram.....	896
14.44.4	Fields.....	896
14.45	DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC).....	896
14.45.1	Offset.....	896
14.45.2	Function.....	897
14.45.3	Diagram.....	897
14.45.4	Fields.....	898
14.46	DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT).....	898
14.46.1	Offset.....	898
14.46.2	Function.....	898
14.46.3	Diagram.....	899
14.46.4	Fields.....	899
14.47	DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL).....	899
14.47.1	Offset.....	899
14.47.2	Function.....	900
14.47.3	Diagram.....	900
14.47.4	Fields.....	901
14.48	DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC).....	902

Section number	Title	Page
14.48.1	Offset.....	902
14.48.2	Function.....	902
14.48.3	Diagram.....	903
14.48.4	Fields.....	903
14.49	DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC).....	903
14.49.1	Offset.....	903
14.49.2	Function.....	904
14.49.3	Diagram.....	904
14.49.4	Fields.....	905
14.50	DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT).....	905
14.50.1	Offset.....	905
14.50.2	Function.....	905
14.50.3	Diagram.....	906
14.50.4	Fields.....	906
14.51	RNG TRNG Miscellaneous Control Register (RTMCTL).....	906
14.51.1	Offset.....	906
14.51.2	Function.....	906
14.51.3	Diagram.....	907
14.51.4	Fields.....	907
14.52	RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC).....	908
14.52.1	Offset.....	909
14.52.2	Function.....	909
14.52.3	Diagram.....	909
14.52.4	Fields.....	909
14.53	RNG TRNG Poker Range Register (RTPKRRNG).....	910
14.53.1	Offset.....	910
14.53.2	Function.....	910
14.53.3	Diagram.....	910
14.53.4	Fields.....	911

Section number	Title	Page
14.54	RNG TRNG Poker Maximum Limit Register (RTPKRMAX).....	911
14.54.1	Offset.....	911
14.54.2	Function.....	911
14.54.3	Diagram.....	911
14.54.4	Fields.....	912
14.55	RNG TRNG Poker Square Calculation Result Register (RTPKRSQ).....	912
14.55.1	Offset.....	912
14.55.2	Function.....	912
14.55.3	Diagram.....	913
14.55.4	Fields.....	913
14.56	RNG TRNG Seed Control Register (RTSDCTL).....	913
14.56.1	Offset.....	913
14.56.2	Function.....	914
14.56.3	Diagram.....	914
14.56.4	Fields.....	914
14.57	RNG TRNG Sparse Bit Limit Register (RTSBLIM).....	914
14.57.1	Offset.....	915
14.57.2	Function.....	915
14.57.3	Diagram.....	915
14.57.4	Fields.....	915
14.58	RNG TRNG Total Samples Register (RTTOTSAM).....	916
14.58.1	Offset.....	916
14.58.2	Function.....	916
14.58.3	Diagram.....	916
14.58.4	Fields.....	916
14.59	RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN).....	917
14.59.1	Offset.....	917
14.59.2	Function.....	917
14.59.3	Diagram.....	917

Section number	Title	Page
14.59.4	Fields.....	918
14.60	RNG TRNG Frequency Count Register (RTFRQCNT).....	918
14.60.1	Offset.....	918
14.60.2	Function.....	918
14.60.3	Diagram.....	919
14.60.4	Fields.....	919
14.61	RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX).....	919
14.61.1	Offset.....	919
14.61.2	Function.....	920
14.61.3	Diagram.....	920
14.61.4	Fields.....	920
14.62	RNG TRNG Statistical Check Monobit Count Register (RTSCMC).....	921
14.62.1	Offset.....	921
14.62.2	Function.....	921
14.62.3	Diagram.....	921
14.62.4	Fields.....	921
14.63	RNG TRNG Statistical Check Monobit Limit Register (RTSCML).....	922
14.63.1	Offset.....	922
14.63.2	Function.....	922
14.63.3	Diagram.....	922
14.63.4	Fields.....	923
14.64	RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C).....	923
14.64.1	Offset.....	923
14.64.2	Function.....	923
14.64.3	Diagram.....	924
14.64.4	Fields.....	924
14.65	RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L).....	925
14.65.1	Offset.....	925
14.65.2	Function.....	925

Section number	Title	Page
14.65.3	Diagram.....	925
14.65.4	Fields.....	926
14.66	RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C).....	926
14.66.1	Offset.....	926
14.66.2	Function.....	927
14.66.3	Diagram.....	927
14.66.4	Fields.....	927
14.67	RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L).....	928
14.67.1	Offset.....	928
14.67.2	Function.....	928
14.67.3	Diagram.....	928
14.67.4	Fields.....	929
14.68	RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C).....	929
14.68.1	Offset.....	929
14.68.2	Function.....	930
14.68.3	Diagram.....	930
14.68.4	Fields.....	930
14.69	RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L).....	931
14.69.1	Offset.....	931
14.69.2	Function.....	931
14.69.3	Diagram.....	931
14.69.4	Fields.....	932
14.70	RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C).....	932
14.70.1	Offset.....	932
14.70.2	Function.....	932
14.70.3	Diagram.....	933
14.70.4	Fields.....	933
14.71	RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L).....	933
14.71.1	Offset.....	934

Section number	Title	Page
14.71.2	Function.....	934
14.71.3	Diagram.....	934
14.71.4	Fields.....	934
14.72	RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C).....	935
14.72.1	Offset.....	935
14.72.2	Function.....	935
14.72.3	Diagram.....	935
14.72.4	Fields.....	936
14.73	RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L).....	936
14.73.1	Offset.....	936
14.73.2	Function.....	936
14.73.3	Diagram.....	937
14.73.4	Fields.....	937
14.74	RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC).....	938
14.74.1	Offset.....	938
14.74.2	Function.....	938
14.74.3	Diagram.....	938
14.74.4	Fields.....	938
14.75	RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL).....	939
14.75.1	Offset.....	939
14.75.2	Function.....	939
14.75.3	Diagram.....	939
14.75.4	Fields.....	940
14.76	RNG TRNG Status Register (RTSTATUS).....	940
14.76.1	Offset.....	940
14.76.2	Function.....	940
14.76.3	Diagram.....	941
14.76.4	Fields.....	941
14.77	RNG TRNG Entropy Read Register (RTENT0 - RTENT15).....	943

Section number	Title	Page
14.77.1	Offset.....	943
14.77.2	Function.....	943
14.77.3	Diagram.....	943
14.77.4	Fields.....	944
14.78	RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10).....	944
14.78.1	Offset.....	944
14.78.2	Function.....	944
14.78.3	Diagram.....	944
14.78.4	Fields.....	945
14.79	RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32).....	945
14.79.1	Offset.....	945
14.79.2	Function.....	945
14.79.3	Diagram.....	946
14.79.4	Fields.....	946
14.80	RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54).....	946
14.80.1	Offset.....	946
14.80.2	Function.....	947
14.80.3	Diagram.....	947
14.80.4	Fields.....	947
14.81	RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76).....	947
14.81.1	Offset.....	948
14.81.2	Function.....	948
14.81.3	Diagram.....	948
14.81.4	Fields.....	948
14.82	RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98).....	949
14.82.1	Offset.....	949
14.82.2	Function.....	949
14.82.3	Diagram.....	949
14.82.4	Fields.....	949

Section number	Title	Page
14.83	RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA).....	950
14.83.1	Offset.....	950
14.83.2	Function.....	950
14.83.3	Diagram.....	950
14.83.4	Fields.....	950
14.84	RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC).....	951
14.84.1	Offset.....	951
14.84.2	Function.....	951
14.84.3	Diagram.....	951
14.84.4	Fields.....	952
14.85	RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE).....	952
14.85.1	Offset.....	952
14.85.2	Function.....	952
14.85.3	Diagram.....	953
14.85.4	Fields.....	953
14.86	RNG DRNG Status Register (RDSTA).....	953
14.86.1	Offset.....	953
14.86.2	Function.....	954
14.86.3	Diagram.....	954
14.86.4	Fields.....	954
14.87	RNG DRNG State Handle 0 Reseed Interval Register (RDINT0).....	955
14.87.1	Offset.....	955
14.87.2	Function.....	955
14.87.3	Diagram.....	956
14.87.4	Fields.....	956
14.88	RNG DRNG State Handle 1 Reseed Interval Register (RDINT1).....	956
14.88.1	Offset.....	956
14.88.2	Function.....	956
14.88.3	Diagram.....	957

Section number	Title	Page
14.88.4	Fields.....	957
14.89	RNG DRNG Hash Control Register (RDHCNTL).....	957
14.89.1	Offset.....	957
14.89.2	Function.....	958
14.89.3	Diagram.....	958
14.89.4	Fields.....	958
14.90	RNG DRNG Hash Digest Register (RDHDIG).....	959
14.90.1	Offset.....	959
14.90.2	Function.....	959
14.90.3	Diagram.....	959
14.90.4	Fields.....	959
14.91	RNG DRNG Hash Buffer Register (RDHBUF).....	960
14.91.1	Offset.....	960
14.91.2	Function.....	960
14.91.3	Diagram.....	960
14.91.4	Fields.....	960
14.92	Recoverable Error Interrupt Status (REIS).....	961
14.92.1	Offset.....	961
14.92.2	Function.....	961
14.92.3	Diagram.....	961
14.92.4	Fields.....	962
14.93	Recoverable Error Interrupt Enable (REIE).....	962
14.93.1	Offset.....	962
14.93.2	Function.....	963
14.93.3	Diagram.....	963
14.93.4	Fields.....	963
14.94	Recoverable Error Interrupt Force (REIF).....	964
14.94.1	Offset.....	964
14.94.2	Function.....	964

Section number	Title	Page
14.94.3	Diagram.....	965
14.94.4	Fields.....	965
14.95	Recoverable Error Interrupt Halt (REIH).....	966
14.95.1	Offset.....	966
14.95.2	Function.....	966
14.95.3	Diagram.....	966
14.95.4	Fields.....	967
14.96	SEC Version ID Register, most-significant half (SECVID_MS).....	967
14.96.1	Offset.....	967
14.96.2	Function.....	968
14.96.3	Diagram.....	968
14.96.4	Fields.....	969
14.97	SEC Version ID Register, least-significant half (SECVID_LS).....	969
14.97.1	Offset.....	969
14.97.2	Function.....	969
14.97.3	Diagram.....	970
14.97.4	Fields.....	970
14.98	Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR).....	970
14.98.1	Offset.....	971
14.98.2	Function.....	971
14.98.3	Diagram.....	971
14.98.4	Fields.....	972
14.99	Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR).....	972
14.99.1	Offset.....	972
14.99.2	Function.....	973
14.99.3	Diagram.....	973
14.99.4	Fields.....	974
14.100	Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS).....	974
14.100.1	Offset.....	974

Section number	Title	Page
14.100.2	Function.....	974
14.100.3	Diagram.....	974
14.100.4	Fields.....	975
14.101	Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS).....	976
14.101.1	Offset.....	976
14.101.2	Function.....	977
14.101.3	Diagram.....	977
14.101.4	Fields.....	977
14.102	Holding Tank Status (HT0_STATUS).....	978
14.102.1	Offset.....	978
14.102.2	Function.....	978
14.102.3	Diagram.....	978
14.102.4	Fields.....	979
14.103	Job Queue Debug Select Register (JQ_DEBUG_SEL).....	980
14.103.1	Offset.....	980
14.103.2	Function.....	980
14.103.3	Diagram.....	980
14.103.4	Fields.....	981
14.104	Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS).....	981
14.104.1	Offset.....	981
14.104.2	Function.....	981
14.104.3	Diagram.....	982
14.104.4	Fields.....	982
14.105	Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC).....	983
14.105.1	Offset.....	983
14.105.2	Function.....	983
14.105.3	Diagram.....	984
14.105.4	Fields.....	984
14.106	Job Ring Job-Done Job ID FIFO (JRJDJIF).....	984

Section number	Title	Page
14.106.1	Offset.....	984
14.106.2	Diagram.....	985
14.106.3	Fields.....	985
14.107	Job Ring Job-Done Source 1 (JRJDS1).....	985
14.107.1	Offset.....	985
14.107.2	Function.....	986
14.107.3	Diagram.....	986
14.107.4	Fields.....	986
14.108	Job Ring Job-Done Descriptor Address 0 Register (JRJDDA).....	986
14.108.1	Offset.....	986
14.108.2	Function.....	987
14.108.3	Diagram.....	987
14.108.4	Fields.....	988
14.109	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ).....	988
14.109.1	Offset.....	988
14.109.2	Function.....	989
14.109.3	Diagram.....	989
14.109.4	Fields.....	990
14.110	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ).....	990
14.110.1	Offset.....	990
14.110.2	Function.....	991
14.110.3	Diagram.....	991
14.110.4	Fields.....	992
14.111	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ).....	992
14.111.1	Offset.....	992
14.111.2	Function.....	993
14.111.3	Diagram.....	993
14.111.4	Fields.....	994
14.112	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT).....	994

Section number	Title	Page
14.112.1	Offset.....	994
14.112.2	Function.....	995
14.112.3	Diagram.....	995
14.112.4	Fields.....	996
14.113	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT).....	996
14.113.1	Offset.....	996
14.113.2	Function.....	997
14.113.3	Diagram.....	998
14.113.4	Fields.....	998
14.114	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT).....	998
14.114.1	Offset.....	999
14.114.2	Function.....	999
14.114.3	Diagram.....	1000
14.114.4	Fields.....	1000
14.115	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED).....	1000
14.115.1	Offset.....	1001
14.115.2	Function.....	1001
14.115.3	Diagram.....	1002
14.115.4	Fields.....	1002
14.116	CHA Revision Number Register, most-significant half (CRNR_MS).....	1003
14.116.1	Offset.....	1003
14.116.2	Function.....	1003
14.116.3	Diagram.....	1003
14.116.4	Fields.....	1004
14.117	CHA Revision Number Register, least-significant half (CRNR_LS).....	1004
14.117.1	Offset.....	1004
14.117.2	Function.....	1005
14.117.3	Diagram.....	1005
14.117.4	Fields.....	1006

Section number	Title	Page
14.118	Compile Time Parameters Register, most-significant half (CTPR_MS).....	1006
14.118.1	Offset.....	1006
14.118.2	Function.....	1007
14.118.3	Diagram.....	1007
14.118.4	Fields.....	1008
14.119	Compile Time Parameters Register, least-significant half (CTPR_LS).....	1010
14.119.1	Offset.....	1010
14.119.2	Function.....	1011
14.119.3	Diagram.....	1011
14.119.4	Fields.....	1011
14.120	Fault Address Register (FAR).....	1012
14.120.1	Offset.....	1012
14.120.2	Function.....	1013
14.120.3	Diagram.....	1013
14.120.4	Fields.....	1014
14.121	Fault Address ICID Register (FAICID).....	1014
14.121.1	Offset.....	1014
14.121.2	Function.....	1015
14.121.3	Diagram.....	1015
14.121.4	Fields.....	1016
14.122	Fault Address Detail Register (FADR).....	1016
14.122.1	Offset.....	1016
14.122.2	Function.....	1017
14.122.3	Diagram.....	1017
14.122.4	Fields.....	1017
14.123	SEC Status Register (SSTA).....	1018
14.123.1	Offset.....	1018
14.123.2	Function.....	1019
14.123.3	Diagram.....	1019

Section number	Title	Page
14.123.4	Fields.....	1020
14.124	RTIC Version ID Register (RVID).....	1020
14.124.1	Offset.....	1020
14.124.2	Function.....	1021
14.124.3	Diagram.....	1021
14.124.4	Fields.....	1022
14.125	CHA Cluster Block Version ID Register (CCBVID).....	1023
14.125.1	Offset.....	1023
14.125.2	Function.....	1023
14.125.3	Diagram.....	1023
14.125.4	Fields.....	1024
14.126	CHA Version ID Register, most-significant half (CHAVID_MS).....	1024
14.126.1	Offset.....	1024
14.126.2	Function.....	1025
14.126.3	Diagram.....	1025
14.126.4	Fields.....	1025
14.127	CHA Version ID Register, least-significant half (CHAVID_LS).....	1026
14.127.1	Offset.....	1026
14.127.2	Function.....	1026
14.127.3	Diagram.....	1027
14.127.4	Fields.....	1027
14.128	CHA Number Register, most-significant half (CHANUM_MS).....	1028
14.128.1	Offset.....	1028
14.128.2	Function.....	1028
14.128.3	Diagram.....	1029
14.128.4	Fields.....	1029
14.129	CHA Number Register, least-significant half (CHANUM_LS).....	1030
14.129.1	Offset.....	1030
14.129.2	Function.....	1030

Section number	Title	Page
14.129.3	Diagram.....	1030
14.129.4	Fields.....	1031
14.130	Input Ring Base Address Register for Job Ring a (IRBAR_JR0 - IRBAR_JR3).....	1031
14.130.1	Offset.....	1031
14.130.2	Function.....	1032
14.130.3	Diagram.....	1032
14.130.4	Fields.....	1033
14.131	Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR3).....	1033
14.131.1	Offset.....	1033
14.131.2	Function.....	1034
14.131.3	Diagram.....	1034
14.131.4	Fields.....	1035
14.132	Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR3).....	1035
14.132.1	Offset.....	1035
14.132.2	Function.....	1035
14.132.3	Diagram.....	1036
14.132.4	Fields.....	1036
14.133	Input Ring Jobs Added Register for Job Ring a (IRJAR_JR0 - IRJAR_JR3).....	1037
14.133.1	Offset.....	1037
14.133.2	Function.....	1037
14.133.3	Diagram.....	1037
14.133.4	Fields.....	1038
14.134	Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3).....	1038
14.134.1	Offset.....	1038
14.134.2	Function.....	1038
14.134.3	Diagram.....	1039
14.134.4	Fields.....	1040
14.135	Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR3).....	1040
14.135.1	Offset.....	1040

Section number	Title	Page
14.135.2	Function.....	1041
14.135.3	Diagram.....	1041
14.135.4	Fields.....	1042
14.136	Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR3).....	1042
14.136.1	Offset.....	1042
14.136.2	Function.....	1042
14.136.3	Diagram.....	1043
14.136.4	Fields.....	1043
14.137	Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR3).....	1044
14.137.1	Offset.....	1044
14.137.2	Function.....	1044
14.137.3	Diagram.....	1044
14.137.4	Fields.....	1045
14.138	Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3).....	1045
14.138.1	Offset.....	1045
14.138.2	Function.....	1045
14.138.3	Diagram.....	1046
14.138.4	Fields.....	1046
14.139	Job Ring Interrupt Status Register for Job Ring a (JRINTR_JR0 - JRINTR_JR3).....	1047
14.139.1	Offset.....	1047
14.139.2	Function.....	1047
14.139.3	Diagram.....	1047
14.139.4	Fields.....	1048
14.140	Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS). ..	1049
14.140.1	Offset.....	1049
14.140.2	Function.....	1050
14.140.3	Diagram.....	1050
14.140.4	Fields.....	1050
14.141	Job Ring Configuration Register for Job Ring a, least-significant half (JRCFGR_JR0_LS - JRCFGR_JR3_LS)....	1051

Section number	Title	Page
14.141.1	Offset.....	1051
14.141.2	Function.....	1051
14.141.3	Diagram.....	1051
14.141.4	Fields.....	1052
14.142	Input Ring Read Index Register for Job Ring a (IRRIR_JR0 - IRRIR_JR3).....	1053
14.142.1	Offset.....	1053
14.142.2	Function.....	1053
14.142.3	Diagram.....	1053
14.142.4	Fields.....	1054
14.143	Output Ring Write Index Register for Job Ring a (ORWIR_JR0 - ORWIR_JR3).....	1054
14.143.1	Offset.....	1054
14.143.2	Function.....	1054
14.143.3	Diagram.....	1055
14.143.4	Fields.....	1055
14.144	Job Ring Command Register for Job Ring a (JRCCR_JR0 - JRCCR_JR3).....	1056
14.144.1	Offset.....	1056
14.144.2	Function.....	1056
14.144.3	Diagram.....	1058
14.144.4	Fields.....	1058
14.145	Job Ring a Address-Array Valid Register (JR0AAV - JR3AAV).....	1058
14.145.1	Offset.....	1058
14.145.2	Function.....	1059
14.145.3	Diagram.....	1059
14.145.4	Fields.....	1059
14.146	Job Ring a Address-Array Address b Register (JR0AAA0 - JR3AAA7).....	1060
14.146.1	Offset.....	1060
14.146.2	Function.....	1060
14.146.3	Diagram.....	1061
14.146.4	Fields.....	1061

Section number	Title	Page
14.147	Recoverable Error Interrupt Record 0 for Job Ring a (REIR0JR0 - REIR0JR3).....	1062
14.147.1	Offset.....	1062
14.147.2	Function.....	1062
14.147.3	Diagram.....	1062
14.147.4	Fields.....	1063
14.148	Recoverable Error Interrupt Record 2 for Job Ring a (REIR2JR0 - REIR2JR3).....	1063
14.148.1	Offset.....	1063
14.148.2	Function.....	1064
14.148.3	Diagram.....	1064
14.148.4	Fields.....	1064
14.149	Recoverable Error Interrupt Record 4 for Job Ring a (REIR4JR0 - REIR4JR3).....	1065
14.149.1	Offset.....	1065
14.149.2	Function.....	1065
14.149.3	Diagram.....	1065
14.149.4	Fields.....	1065
14.150	Recoverable Error Interrupt Record 5 for Job Ring a (REIR5JR0 - REIR5JR3).....	1066
14.150.1	Offset.....	1066
14.150.2	Function.....	1066
14.150.3	Diagram.....	1067
14.150.4	Fields.....	1067
14.151	AIOP Interface Configuration Register (AICFG).....	1067
14.151.1	Offset.....	1067
14.151.2	Function.....	1068
14.151.3	Diagram.....	1068
14.151.4	Fields.....	1068
14.152	AIOP Interface Control Register (AICTL).....	1069
14.152.1	Offset.....	1069
14.152.2	Function.....	1069
14.152.3	Diagram.....	1069

Section number	Title	Page
14.152.4	Fields.....	1069
14.153	AIOP Interface Status Register (AISTA).....	1070
14.153.1	Offset.....	1070
14.153.2	Function.....	1070
14.153.3	Diagram.....	1071
14.153.4	Fields.....	1071
14.154	AIOP Interface Dequeue Event Filter Control Register 0 (ADQEFC0).....	1071
14.154.1	Offset.....	1071
14.154.2	Function.....	1072
14.154.3	Diagram.....	1072
14.154.4	Fields.....	1072
14.155	AIOP Interface Dequeue Event Filter Control Register 1 (ADQEFC1).....	1073
14.155.1	Offset.....	1073
14.155.2	Function.....	1073
14.155.3	Diagram.....	1073
14.155.4	Fields.....	1074
14.156	AIOP Interface Dequeue Event Filter Control Register 2 (ADQEFC2).....	1074
14.156.1	Offset.....	1074
14.156.2	Function.....	1075
14.156.3	Diagram.....	1075
14.156.4	Fields.....	1075
14.157	AIOP Interface Enqueue Event Filter Control Register 0 (AEQEFC0).....	1076
14.157.1	Offset.....	1076
14.157.2	Function.....	1076
14.157.3	Diagram.....	1076
14.157.4	Fields.....	1076
14.158	AIOP Interface Enqueue Event Filter Control Register 1 (AEQEFC1).....	1077
14.158.1	Offset.....	1077
14.158.2	Function.....	1077

Section number	Title	Page
14.158.3	Diagram.....	1077
14.158.4	Fields.....	1078
14.159	AIOP Interface Enqueue Event Filter Control Register 2 (AEQEFC2).....	1078
14.159.1	Offset.....	1078
14.159.2	Function.....	1079
14.159.3	Diagram.....	1079
14.159.4	Fields.....	1079
14.160	Jobs in Use Register for AIOP Interface (JOBS_IN_USE_AI).....	1080
14.160.1	Offset.....	1080
14.160.2	Function.....	1080
14.160.3	Diagram.....	1080
14.160.4	Fields.....	1080
14.161	Jobs Ready Register for AIOP Interface (JOBS_READY_AI).....	1081
14.161.1	Offset.....	1081
14.161.2	Function.....	1081
14.161.3	Diagram.....	1081
14.161.4	Fields.....	1081
14.162	Jobs Transfer Blocking Disabled Register for AIOP Interface (JOBS_XFR_BLK_DIS_AI).....	1082
14.162.1	Offset.....	1082
14.162.2	Function.....	1082
14.162.3	Diagram.....	1082
14.162.4	Fields.....	1083
14.163	Jobs Transferred Register for AIOP Interface (JOBS_XFRD_AI).....	1083
14.163.1	Offset.....	1083
14.163.2	Function.....	1083
14.163.3	Diagram.....	1084
14.163.4	Fields.....	1084
14.164	Jobs Executing Register for AIOP Interface (JOBS_EXEC_AI).....	1084
14.164.1	Offset.....	1084

Section number	Title	Page
14.164.2	Function.....	1085
14.164.3	Diagram.....	1085
14.164.4	Fields.....	1085
14.165	Jobs Done Register for AIOP Interface (JOBS_DONE_AI).....	1086
14.165.1	Offset.....	1086
14.165.2	Function.....	1086
14.165.3	Diagram.....	1086
14.165.4	Fields.....	1086
14.166	Job Select Register for AIOP Interface (JOB_SELECT_AI).....	1087
14.166.1	Offset.....	1087
14.166.2	Function.....	1087
14.166.3	Diagram.....	1087
14.166.4	Fields.....	1087
14.167	AIOP Interface Job Data Register 0 (AIJOB0).....	1088
14.167.1	Offset.....	1088
14.167.2	Diagram.....	1088
14.167.3	Fields.....	1089
14.168	AIOP Interface Job Data Register 1 (AIJOB1).....	1090
14.168.1	Offset.....	1090
14.168.2	Diagram.....	1090
14.168.3	Fields.....	1090
14.169	AIOP Interface Job Data Register 2 (AIJOB2).....	1091
14.169.1	Offset.....	1091
14.169.2	Diagram.....	1091
14.169.3	Fields.....	1091
14.170	AIOP Interface Job Data Register 3 (AIJOB3).....	1092
14.170.1	Offset.....	1092
14.170.2	Diagram.....	1092
14.170.3	Fields.....	1092

Section number	Title	Page
14.171	AIOP Interface Job Data Register 4 (AIJOB4).....	1093
14.171.1	Offset.....	1093
14.171.2	Diagram.....	1093
14.171.3	Fields.....	1093
14.172	AIOP Interface Job Data Register 5 (AIJOB5).....	1093
14.172.1	Offset.....	1093
14.172.2	Diagram.....	1094
14.172.3	Fields.....	1094
14.173	AIOP Interface Job Data Register 6 (AIJOB6).....	1095
14.173.1	Offset.....	1095
14.173.2	Diagram.....	1095
14.173.3	Fields.....	1095
14.174	AIOP Interface Job Data Register 7 (AIJOB7).....	1096
14.174.1	Offset.....	1096
14.174.2	Diagram.....	1096
14.174.3	Fields.....	1096
14.175	AIOP Interface Job Data Register 8 (AIJOB8).....	1097
14.175.1	Offset.....	1097
14.175.2	Diagram.....	1097
14.175.3	Fields.....	1098
14.176	AIOP Interface Job Data Register 9 (AIJOB9).....	1098
14.176.1	Offset.....	1098
14.176.2	Diagram.....	1098
14.176.3	Fields.....	1099
14.177	AIOP Interface Job Data Register 10 (AIJOB10).....	1099
14.177.1	Offset.....	1099
14.177.2	Diagram.....	1099
14.177.3	Fields.....	1100
14.178	AIOP Interface Job Data Register 11 (AIJOB11).....	1100

Section number	Title	Page
14.178.1	Offset.....	1100
14.178.2	Diagram.....	1100
14.178.3	Fields.....	1101
14.179	AIOP Interface Job Data Register 12 (AIJOB12).....	1101
14.179.1	Offset.....	1101
14.179.2	Diagram.....	1101
14.179.3	Fields.....	1102
14.180	AIOP Interface Job Data Register 13 (AIJOB13).....	1102
14.180.1	Offset.....	1102
14.180.2	Diagram.....	1103
14.180.3	Fields.....	1103
14.181	AIOP Interface Job Data Register 14 (AIJOB14).....	1103
14.181.1	Offset.....	1103
14.181.2	Diagram.....	1104
14.181.3	Fields.....	1104
14.182	AIOP Interface Job Data Register 15 (AIJOB15).....	1105
14.182.1	Offset.....	1105
14.182.2	Diagram.....	1105
14.182.3	Fields.....	1105
14.183	AIOP Interface Job Data Register 16 (AIJOB16).....	1105
14.183.1	Offset.....	1106
14.183.2	Diagram.....	1106
14.183.3	Fields.....	1106
14.184	AIOP Interface Job Data Register 17 (AIJOB17).....	1106
14.184.1	Offset.....	1106
14.184.2	Diagram.....	1107
14.184.3	Fields.....	1107
14.185	AIOP Interface Job Data Register 18 (AIJOB18).....	1107
14.185.1	Offset.....	1107

Section number	Title	Page
14.185.2	Diagram.....	1108
14.185.3	Fields.....	1108
14.186	AIOP Interface Job Data Register 19 (AIJOB19).....	1108
14.186.1	Offset.....	1108
14.186.2	Diagram.....	1108
14.186.3	Fields.....	1109
14.187	AIOP Interface Job Data Register 20 (AIJOB20).....	1109
14.187.1	Offset.....	1109
14.187.2	Diagram.....	1109
14.187.3	Fields.....	1110
14.188	AIOP Interface Job Data Register 21 (AIJOB21).....	1110
14.188.1	Offset.....	1110
14.188.2	Diagram.....	1111
14.188.3	Fields.....	1111
14.189	AIOP Interface Job Data Register 22 (AIJOB22).....	1112
14.189.1	Offset.....	1112
14.189.2	Diagram.....	1112
14.189.3	Fields.....	1112
14.190	AIOP Interface Job Data Register 23 (AIJOB23).....	1113
14.190.1	Offset.....	1113
14.190.2	Diagram.....	1113
14.190.3	Fields.....	1114
14.191	AIOP Interface Job Data Register 24 (AIJOB24).....	1114
14.191.1	Offset.....	1114
14.191.2	Diagram.....	1114
14.191.3	Fields.....	1115
14.192	AIOP Interface Job Data Register 25 (AIJOB25).....	1115
14.192.1	Offset.....	1115
14.192.2	Diagram.....	1115

Section number	Title	Page
14.192.3	Fields.....	1116
14.193	AIOP Interface Job Data Register 26 (AIJOB26).....	1116
14.193.1	Offset.....	1116
14.193.2	Diagram.....	1116
14.193.3	Fields.....	1117
14.194	AIOP Interface Job Data Register 27 (AIJOB27).....	1117
14.194.1	Offset.....	1117
14.194.2	Diagram.....	1117
14.194.3	Fields.....	1118
14.195	AIOP Interface Job Data Register 28 (AIJOB28).....	1118
14.195.1	Offset.....	1118
14.195.2	Diagram.....	1118
14.195.3	Fields.....	1119
14.196	AIOP Interface Job Data Register 29 (AIJOB29).....	1119
14.196.1	Offset.....	1119
14.196.2	Diagram.....	1119
14.196.3	Fields.....	1120
14.197	AIOP Interface Job Data Register 30 (AIJOB30).....	1120
14.197.1	Offset.....	1120
14.197.2	Diagram.....	1120
14.197.3	Fields.....	1121
14.198	AIOP Interface Job Data Register 31 (AIJOB31).....	1121
14.198.1	Offset.....	1121
14.198.2	Diagram.....	1121
14.198.3	Fields.....	1122
14.199	AIOP Interface Job Data Register 32 (AIJOB32).....	1122
14.199.1	Offset.....	1122
14.199.2	Diagram.....	1122
14.199.3	Fields.....	1123

Section number	Title	Page
14.200	AIOP Interface Job Data Register 33 (AIJOB33).....	1123
14.200.1	Offset.....	1123
14.200.2	Diagram.....	1123
14.200.3	Fields.....	1124
14.201	AIOP Interface Job Data Register 34 (AIJOB34).....	1124
14.201.1	Offset.....	1124
14.201.2	Diagram.....	1125
14.201.3	Fields.....	1125
14.202	Recoverable Error Interrupt Record 0 for the AIOP Interface (REIR0AI).....	1125
14.202.1	Offset.....	1125
14.202.2	Function.....	1126
14.202.3	Diagram.....	1126
14.202.4	Fields.....	1126
14.203	Recoverable Error Interrupt Record 1 for the AIOP Interface (REIR1AI).....	1127
14.203.1	Offset.....	1127
14.203.2	Function.....	1127
14.203.3	Diagram.....	1127
14.203.4	Fields.....	1128
14.204	Recoverable Error Interrupt Record 2 for the AIOP Interface (REIR2AI).....	1128
14.204.1	Offset.....	1128
14.204.2	Function.....	1128
14.204.3	Diagram.....	1128
14.204.4	Fields.....	1129
14.205	Recoverable Error Interrupt Record 4 for the AIOP Interface (REIR4AI).....	1129
14.205.1	Offset.....	1129
14.205.2	Function.....	1130
14.205.3	Diagram.....	1130
14.205.4	Fields.....	1130
14.206	Recoverable Error Interrupt Record 5 for the AIOP Interface (REIR5AI).....	1131

Section number	Title	Page
14.206.1	Offset.....	1131
14.206.2	Function.....	1131
14.206.3	Diagram.....	1131
14.206.4	Fields.....	1132
14.207	RTIC Status Register (RSTA).....	1132
14.207.1	Offset.....	1132
14.207.2	Function.....	1132
14.207.3	Diagram.....	1132
14.207.4	Fields.....	1133
14.208	RTIC Command Register (RCMD).....	1135
14.208.1	Offset.....	1135
14.208.2	Function.....	1135
14.208.3	Diagram.....	1135
14.208.4	Fields.....	1135
14.209	RTIC Control Register (RCTL).....	1136
14.209.1	Offset.....	1136
14.209.2	Function.....	1137
14.209.3	Diagram.....	1137
14.209.4	Fields.....	1137
14.210	RTIC Throttle Register (RTHR).....	1139
14.210.1	Offset.....	1139
14.210.2	Function.....	1139
14.210.3	Diagram.....	1139
14.210.4	Fields.....	1140
14.211	RTIC Watchdog Timer (RWDOG).....	1140
14.211.1	Offset.....	1140
14.211.2	Function.....	1140
14.211.3	Diagram.....	1141
14.211.4	Fields.....	1141

Section number	Title	Page
14.212	RTIC Memory Block a Address b Register (RMAA0 - RMDA1).....	1142
14.212.1	Offset.....	1142
14.212.2	Function.....	1142
14.212.3	Diagram.....	1142
14.212.4	Fields.....	1143
14.213	RTIC Memory Block a Length b Register (RMAL0 - RMDL1).....	1143
14.213.1	Offset.....	1143
14.213.2	Function.....	1144
14.213.3	Diagram.....	1144
14.213.4	Fields.....	1145
14.214	RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31).....	1145
14.214.1	Offset.....	1145
14.214.2	Function.....	1156
14.214.3	Diagram.....	1156
14.214.4	Fields.....	1157
14.215	Recoverable Error Interrupt Record 0 for RTIC (REIR0RTIC).....	1157
14.215.1	Offset.....	1157
14.215.2	Function.....	1157
14.215.3	Diagram.....	1158
14.215.4	Fields.....	1158
14.216	Recoverable Error Interrupt Record 2 for RTIC (REIR2RTIC).....	1158
14.216.1	Offset.....	1158
14.216.2	Function.....	1159
14.216.3	Diagram.....	1159
14.216.4	Fields.....	1159
14.217	Recoverable Error Interrupt Record 4 for RTIC (REIR4RTIC).....	1160
14.217.1	Offset.....	1160
14.217.2	Function.....	1160
14.217.3	Diagram.....	1160

Section number	Title	Page
14.217.4	Fields.....	1160
14.218	Recoverable Error Interrupt Record 5 for RTIC (REIR5RTIC).....	1161
14.218.1	Offset.....	1161
14.218.2	Function.....	1161
14.218.3	Diagram.....	1162
14.218.4	Fields.....	1162
14.219	Queue Interface Configuration Register (QICFG).....	1162
14.219.1	Offset.....	1162
14.219.2	Function.....	1163
14.219.3	Diagram.....	1163
14.219.4	Fields.....	1163
14.220	Queue Interface Control Register (QICTL).....	1164
14.220.1	Offset.....	1164
14.220.2	Function.....	1164
14.220.3	Diagram.....	1164
14.220.4	Fields.....	1164
14.221	Queue Interface Status Register (QISTA).....	1165
14.221.1	Offset.....	1165
14.221.2	Function.....	1165
14.221.3	Diagram.....	1165
14.221.4	Fields.....	1166
14.222	Queue Interface Dequeue Configuration Register (QIDQC).....	1166
14.222.1	Offset.....	1166
14.222.2	Function.....	1166
14.222.3	Diagram.....	1167
14.222.4	Fields.....	1167
14.223	Queue Interface Dequeue Event Filter Control Register 0 (QDQEFC0).....	1168
14.223.1	Offset.....	1168
14.223.2	Function.....	1168

Section number	Title	Page
14.223.3	Diagram.....	1168
14.223.4	Fields.....	1169
14.224	Queue Interface Dequeue Event Filter Control Register 1 (QDQEFC1).....	1169
14.224.1	Offset.....	1169
14.224.2	Function.....	1170
14.224.3	Diagram.....	1170
14.224.4	Fields.....	1170
14.225	Queue Interface Dequeue Event Filter Control Register 2 (QDQEFC2).....	1171
14.225.1	Offset.....	1171
14.225.2	Function.....	1171
14.225.3	Diagram.....	1171
14.225.4	Fields.....	1171
14.226	Queue Interface Enqueue Event Filter Control Register 0 (QEQEFC0).....	1172
14.226.1	Offset.....	1172
14.226.2	Function.....	1172
14.226.3	Diagram.....	1172
14.226.4	Fields.....	1173
14.227	Queue Interface Enqueue Event Filter Control Register 1 (QEQEFC1).....	1173
14.227.1	Offset.....	1173
14.227.2	Function.....	1174
14.227.3	Diagram.....	1174
14.227.4	Fields.....	1174
14.228	Queue Interface Enqueue Event Filter Control Register 2 (QEQEFC2).....	1175
14.228.1	Offset.....	1175
14.228.2	Function.....	1175
14.228.3	Diagram.....	1175
14.228.4	Fields.....	1175
14.229	Jobs in Use Register for QM Interface (JOBS_IN_USE_QI).....	1176
14.229.1	Offset.....	1176

Section number	Title	Page
14.229.2	Function.....	1176
14.229.3	Diagram.....	1176
14.229.4	Fields.....	1177
14.230	Jobs Ready Register for QM Interface (JOBS_READY_QI).....	1177
14.230.1	Offset.....	1177
14.230.2	Function.....	1177
14.230.3	Diagram.....	1178
14.230.4	Fields.....	1178
14.231	Jobs Transfer Blocking Disabled Register for QM Interface (JOBS_XFR_BLK_DIS_QI).....	1178
14.231.1	Offset.....	1178
14.231.2	Function.....	1179
14.231.3	Diagram.....	1179
14.231.4	Fields.....	1179
14.232	Jobs Transferred Register for QM Interface (JOBS_XFRD_QI).....	1180
14.232.1	Offset.....	1180
14.232.2	Function.....	1180
14.232.3	Diagram.....	1180
14.232.4	Fields.....	1180
14.233	Jobs Executing Register for QM Interface (JOBS_EXEC_QI).....	1181
14.233.1	Offset.....	1181
14.233.2	Function.....	1181
14.233.3	Diagram.....	1181
14.233.4	Fields.....	1182
14.234	Jobs Done Register for QM Interface (JOBS_DONE_QI).....	1182
14.234.1	Offset.....	1182
14.234.2	Function.....	1182
14.234.3	Diagram.....	1182
14.234.4	Fields.....	1183
14.235	Subportal Frame Count Register for Queue Interface (SUBPORT_FC_QI).....	1183

Section number	Title	Page
14.235.1	Offset.....	1183
14.235.2	Function.....	1183
14.235.3	Diagram.....	1184
14.235.4	Fields.....	1184
14.236	Job Select Register for QI Interface (JOB_SELECT_QI).....	1185
14.236.1	Offset.....	1185
14.236.2	Function.....	1185
14.236.3	Diagram.....	1185
14.236.4	Fields.....	1185
14.237	Queue Interface Job Data Register 0 (QIJOB0).....	1186
14.237.1	Offset.....	1186
14.237.2	Diagram.....	1186
14.237.3	Fields.....	1187
14.238	Queue Interface Job Data Register 1 (QIJOB1).....	1188
14.238.1	Offset.....	1188
14.238.2	Diagram.....	1188
14.238.3	Fields.....	1188
14.239	Queue Interface Job Data Register 2 (QIJOB2).....	1189
14.239.1	Offset.....	1189
14.239.2	Diagram.....	1189
14.239.3	Fields.....	1189
14.240	Queue Interface Job Data Register 3 (QIJOB3).....	1190
14.240.1	Offset.....	1190
14.240.2	Diagram.....	1190
14.240.3	Fields.....	1190
14.241	Queue Interface Job Data Register 4 (QIJOB4).....	1191
14.241.1	Offset.....	1191
14.241.2	Diagram.....	1191
14.241.3	Fields.....	1191

Section number	Title	Page
14.242	Queue Interface Job Data Register 5 (QIJOB5).....	1192
14.242.1	Offset.....	1192
14.242.2	Diagram.....	1192
14.242.3	Fields.....	1192
14.243	Queue Interface Job Data Register 6 (QIJOB6).....	1193
14.243.1	Offset.....	1193
14.243.2	Diagram.....	1193
14.243.3	Fields.....	1194
14.244	Queue Interface Job Data Register 7 (QIJOB7).....	1194
14.244.1	Offset.....	1194
14.244.2	Diagram.....	1195
14.244.3	Fields.....	1195
14.245	Queue Interface Job Data Register 8 (QIJOB8).....	1196
14.245.1	Offset.....	1196
14.245.2	Diagram.....	1196
14.245.3	Fields.....	1196
14.246	Queue Interface Job Data Register 9 (QIJOB9).....	1197
14.246.1	Offset.....	1197
14.246.2	Diagram.....	1197
14.246.3	Fields.....	1197
14.247	Queue Interface Job Data Register 10 (QIJOB10).....	1197
14.247.1	Offset.....	1197
14.247.2	Diagram.....	1198
14.247.3	Fields.....	1198
14.248	Queue Interface Job Data Register 11 (QIJOB11).....	1198
14.248.1	Offset.....	1198
14.248.2	Diagram.....	1199
14.248.3	Fields.....	1199
14.249	Queue Interface Job Data Register 12 (QIJOB12).....	1199

Section number	Title	Page
14.249.1	Offset.....	1199
14.249.2	Diagram.....	1199
14.249.3	Fields.....	1200
14.250	Queue Interface Job Data Register 13 (QIJOB13).....	1200
14.250.1	Offset.....	1200
14.250.2	Diagram.....	1201
14.250.3	Fields.....	1201
14.251	Queue Interface Job Data Register 14 (QIJOB14).....	1201
14.251.1	Offset.....	1201
14.251.2	Diagram.....	1202
14.251.3	Fields.....	1202
14.252	Queue Interface Job Data Register 15 (QIJOB15).....	1203
14.252.1	Offset.....	1203
14.252.2	Diagram.....	1203
14.252.3	Fields.....	1203
14.253	Queue Interface Job Data Register 16 (QIJOB16).....	1203
14.253.1	Offset.....	1204
14.253.2	Diagram.....	1204
14.253.3	Fields.....	1204
14.254	Queue Interface Job Data Register 17 (QIJOB17).....	1204
14.254.1	Offset.....	1204
14.254.2	Diagram.....	1205
14.254.3	Fields.....	1205
14.255	Queue Interface Job Data Register 18 (QIJOB18).....	1205
14.255.1	Offset.....	1205
14.255.2	Diagram.....	1206
14.255.3	Fields.....	1206
14.256	Queue Interface Job Data Register 19 (QIJOB19).....	1206
14.256.1	Offset.....	1206

Section number	Title	Page
14.256.2	Diagram.....	1206
14.256.3	Fields.....	1207
14.257	Queue Interface Job Data Register 20 (QIJOB20).....	1207
14.257.1	Offset.....	1207
14.257.2	Diagram.....	1207
14.257.3	Fields.....	1208
14.258	Queue Interface Job Data Register 21 (QIJOB21).....	1208
14.258.1	Offset.....	1208
14.258.2	Diagram.....	1209
14.258.3	Fields.....	1209
14.259	Queue Interface Job Data Register 22 (QIJOB22).....	1210
14.259.1	Offset.....	1210
14.259.2	Diagram.....	1210
14.259.3	Fields.....	1210
14.260	Queue Interface Job Data Register 23 (QIJOB23).....	1211
14.260.1	Offset.....	1211
14.260.2	Diagram.....	1211
14.260.3	Fields.....	1212
14.261	Queue Interface Job Data Register 24 (QIJOB24).....	1212
14.261.1	Offset.....	1212
14.261.2	Diagram.....	1212
14.261.3	Fields.....	1213
14.262	Queue Interface Job Data Register 25 (QIJOB25).....	1213
14.262.1	Offset.....	1213
14.262.2	Diagram.....	1213
14.262.3	Fields.....	1214
14.263	Queue Interface Job Data Register 26 (QIJOB26).....	1214
14.263.1	Offset.....	1214
14.263.2	Diagram.....	1214

Section number	Title	Page
14.263.3	Fields.....	1215
14.264	Queue Interface Job Data Register 27 (QIJOB27).....	1215
14.264.1	Offset.....	1215
14.264.2	Diagram.....	1215
14.264.3	Fields.....	1216
14.265	Queue Interface Job Data Register 28 (QIJOB28).....	1216
14.265.1	Offset.....	1216
14.265.2	Diagram.....	1216
14.265.3	Fields.....	1217
14.266	Queue Interface Job Data Register 29 (QIJOB29).....	1217
14.266.1	Offset.....	1217
14.266.2	Diagram.....	1217
14.266.3	Fields.....	1218
14.267	Queue Interface Job Data Register 30 (QIJOB30).....	1218
14.267.1	Offset.....	1218
14.267.2	Diagram.....	1218
14.267.3	Fields.....	1219
14.268	Queue Interface Job Data Register 31 (QIJOB31).....	1219
14.268.1	Offset.....	1219
14.268.2	Diagram.....	1219
14.268.3	Fields.....	1220
14.269	Queue Interface Job Data Register 32 (QIJOB32).....	1220
14.269.1	Offset.....	1220
14.269.2	Diagram.....	1221
14.269.3	Fields.....	1221
14.270	Queue Interface Job Data Register 33 (QIJOB33).....	1221
14.270.1	Offset.....	1221
14.270.2	Diagram.....	1222
14.270.3	Fields.....	1222

Section number	Title	Page
14.271	Recoverable Error Interrupt Record 0 for the Queue Interface (REIR0QI).....	1223
14.271.1	Offset.....	1223
14.271.2	Function.....	1223
14.271.3	Diagram.....	1223
14.271.4	Fields.....	1223
14.272	Recoverable Error Interrupt Record 1 for the Queue Interface (REIR1QI).....	1224
14.272.1	Offset.....	1224
14.272.2	Function.....	1224
14.272.3	Diagram.....	1224
14.272.4	Fields.....	1225
14.273	Recoverable Error Interrupt Record 2 for the Queue Interface (REIR2QI).....	1225
14.273.1	Offset.....	1225
14.273.2	Function.....	1225
14.273.3	Diagram.....	1226
14.273.4	Fields.....	1226
14.274	Recoverable Error Interrupt Record 4 for the Queue Interface (REIR4QI).....	1226
14.274.1	Offset.....	1227
14.274.2	Function.....	1227
14.274.3	Diagram.....	1227
14.274.4	Fields.....	1227
14.275	Recoverable Error Interrupt Record 5 for the Queue Interface (REIR5QI).....	1228
14.275.1	Offset.....	1228
14.275.2	Function.....	1228
14.275.3	Diagram.....	1228
14.275.4	Fields.....	1229
14.276	CCB a Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK - C5C1MR_NPK).....	1229
14.276.1	Offset.....	1229
14.276.2	Function.....	1230
14.276.3	Diagram.....	1230

Section number	Title	Page
14.276.4	Fields.....	1231
14.277	CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C5C1MR_PK).....	1234
14.277.1	Offset.....	1234
14.277.2	Function.....	1234
14.277.3	Diagram.....	1235
14.277.4	Fields.....	1235
14.278	CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG).....	1235
14.278.1	Offset.....	1236
14.278.2	Function.....	1236
14.278.3	Diagram.....	1236
14.278.4	Fields.....	1237
14.279	CCB a Class 1 Key Size Register (C0C1KSR - C5C1KSR).....	1239
14.279.1	Offset.....	1240
14.279.2	Function.....	1240
14.279.3	Diagram.....	1240
14.279.4	Fields.....	1240
14.280	CCB a Class 1 Data Size Register (C0C1DSR - C5C1DSR).....	1241
14.280.1	Offset.....	1241
14.280.2	Function.....	1241
14.280.3	Diagram.....	1242
14.280.4	Fields.....	1242
14.281	CCB a Class 1 ICV Size Register (C0C1ICVSR - C5C1ICVSR).....	1243
14.281.1	Offset.....	1243
14.281.2	Function.....	1243
14.281.3	Diagram.....	1243
14.281.4	Fields.....	1244
14.282	CCB a CHA Control Register (C0CCTRL - C5CCTRL).....	1244
14.282.1	Offset.....	1244
14.282.2	Function.....	1244

Section number	Title	Page
14.282.3	Diagram.....	1245
14.282.4	Fields.....	1245
14.283	CCB a Interrupt Control Register (C0ICTL - C5ICTL).....	1247
14.283.1	Offset.....	1247
14.283.2	Function.....	1247
14.283.3	Diagram.....	1247
14.283.4	Fields.....	1248
14.284	CCB a Clear Written Register (C0CWR - C5CWR).....	1252
14.284.1	Offset.....	1252
14.284.2	Function.....	1252
14.284.3	Diagram.....	1252
14.284.4	Fields.....	1252
14.285	CCB a Status and Error Register, most-significant half (C0CSTA_MS - C5CSTA_MS).....	1254
14.285.1	Offset.....	1254
14.285.2	Function.....	1254
14.285.3	Diagram.....	1255
14.285.4	Fields.....	1255
14.286	CCB a Status and Error Register, least-significant half (C0CSTA_LS - C5CSTA_LS).....	1256
14.286.1	Offset.....	1256
14.286.2	Function.....	1256
14.286.3	Diagram.....	1256
14.286.4	Fields.....	1257
14.287	CCB a AAD Size Register (C0AADSZR - C5AADSZR).....	1259
14.287.1	Offset.....	1259
14.287.2	Function.....	1259
14.287.3	Diagram.....	1259
14.287.4	Fields.....	1260
14.288	Class 1 IV Size Register (C0C1IVSZR - C5C1IVSZR).....	1260
14.288.1	Offset.....	1260

Section number	Title	Page
14.288.2	Function.....	1260
14.288.3	Diagram.....	1260
14.288.4	Fields.....	1261
14.289	PKHA A Size Register (C0PKASZR - C5PKASZR).....	1261
14.289.1	Offset.....	1261
14.289.2	Function.....	1261
14.289.3	Diagram.....	1262
14.289.4	Fields.....	1262
14.290	PKHA B Size Register (C0PKBSZR - C5PKBSZR).....	1262
14.290.1	Offset.....	1262
14.290.2	Function.....	1263
14.290.3	Diagram.....	1263
14.290.4	Fields.....	1263
14.291	PKHA N Size Register (C0PKNSZR - C5PKNSZR).....	1263
14.291.1	Offset.....	1263
14.291.2	Function.....	1264
14.291.3	Diagram.....	1264
14.291.4	Fields.....	1264
14.292	PKHA E Size Register (C0PKESZR - C5PKESZR).....	1265
14.292.1	Offset.....	1265
14.292.2	Function.....	1265
14.292.3	Diagram.....	1265
14.292.4	Fields.....	1265
14.293	CCB a Class 1 Context Register Word b (C0C1CTXR0 - C5C1CTXR15).....	1266
14.293.1	Offset.....	1266
14.293.2	Function.....	1266
14.293.3	Diagram.....	1267
14.293.4	Fields.....	1268
14.294	CCB a Class 1 Key Registers Word b (C0C1KR0 - C5C1KR7).....	1268

Section number	Title	Page
14.294.1	Offset.....	1268
14.294.2	Function.....	1268
14.294.3	Diagram.....	1269
14.294.4	Fields.....	1270
14.295	CCB a Class 2 Mode Register (C0C2MR - C5C2MR).....	1270
14.295.1	Offset.....	1270
14.295.2	Function.....	1270
14.295.3	Diagram.....	1270
14.295.4	Fields.....	1271
14.296	CCB a Class 2 Key Size Register (C0C2KSR - C5C2KSR).....	1272
14.296.1	Offset.....	1272
14.296.2	Function.....	1272
14.296.3	Diagram.....	1273
14.296.4	Fields.....	1273
14.297	CCB a Class 2 Data Size Register (C0C2DSR - C5C2DSR).....	1273
14.297.1	Offset.....	1273
14.297.2	Function.....	1274
14.297.3	Diagram.....	1274
14.297.4	Fields.....	1275
14.298	CCB a Class 2 ICV Size Register (C0C2ICVSZR - C5C2ICVSZR).....	1275
14.298.1	Offset.....	1276
14.298.2	Function.....	1276
14.298.3	Diagram.....	1276
14.298.4	Fields.....	1276
14.299	CCB a Class 2 Context Register Word b (C0C2CTXR0 - C5C2CTXR17).....	1277
14.299.1	Offset.....	1277
14.299.2	Function.....	1277
14.299.3	Diagram.....	1278
14.299.4	Fields.....	1278

Section number	Title	Page
14.300	CCB a Class 2 Key Register Word b (C0C2KEYR0 - C5C2KEYR31).....	1278
14.300.1	Offset.....	1278
14.300.2	Function.....	1278
14.300.3	Diagram.....	1279
14.300.4	Fields.....	1279
14.301	CCB a FIFO Status (C0FIFOSTA - C5FIFOSTA).....	1279
14.301.1	Offset.....	1279
14.301.2	Function.....	1280
14.301.3	Diagram.....	1280
14.301.4	Fields.....	1280
14.302	CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C5NFIFO).....	1281
14.302.1	Offset.....	1281
14.302.2	Function.....	1281
14.302.3	Diagram.....	1282
14.302.4	Fields.....	1282
14.303	CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2).....	1285
14.303.1	Offset.....	1285
14.303.2	Function.....	1285
14.303.3	Diagram.....	1285
14.303.4	Fields.....	1285
14.304	CCB a Input Data FIFO (C0IFIFO - C5IFIFO).....	1287
14.304.1	Offset.....	1287
14.304.2	Function.....	1288
14.304.3	Diagram.....	1288
14.304.4	Fields.....	1288
14.305	CCB a Output Data FIFO (C0OFIFO - C5OFIFO).....	1289
14.305.1	Offset.....	1289
14.305.2	Function.....	1289
14.305.3	Diagram.....	1290

Section number	Title	Page
14.305.4	Fields.....	1290
14.306	DECOa Job Queue Control Register, most-significant half (D0JQCR_MS - D5JQCR_MS).....	1291
14.306.1	Offset.....	1291
14.306.2	Function.....	1291
14.306.3	Diagram.....	1291
14.306.4	Fields.....	1292
14.307	DECOa Job Queue Control Register, least-significant half (D0JQCR_LS - D5JQCR_LS).....	1293
14.307.1	Offset.....	1293
14.307.2	Function.....	1293
14.307.3	Diagram.....	1294
14.307.4	Fields.....	1294
14.308	DECOa Descriptor Address Register (D0DAR - D5DAR).....	1294
14.308.1	Offset.....	1294
14.308.2	Function.....	1295
14.308.3	Diagram.....	1295
14.308.4	Fields.....	1295
14.309	DECOa Operation Status Register, most-significant half (D0OPSTA_MS - D5OPSTA_MS).....	1296
14.309.1	Offset.....	1296
14.309.2	Function.....	1296
14.309.3	Diagram.....	1296
14.309.4	Fields.....	1297
14.310	DECOa Operation Status Register, least-significant half (D0OPSTA_LS - D5OPSTA_LS).....	1298
14.310.1	Offset.....	1298
14.310.2	Function.....	1298
14.310.3	Diagram.....	1299
14.310.4	Fields.....	1299
14.311	DECOa Checksum Register (D0CKSUMR - D5CKSUMR).....	1299
14.311.1	Offset.....	1299
14.311.2	Function.....	1300

Section number	Title	Page
14.311.3	Diagram.....	1300
14.311.4	Fields.....	1300
14.312	DECOa Control and Output ICID Status Register (D0COICIDSR - D5COICIDSR).....	1301
14.312.1	Offset.....	1301
14.312.2	Function.....	1301
14.312.3	Diagram.....	1301
14.312.4	Fields.....	1301
14.313	DECOa SDID and Input ICID Status Register (D0SIICIDSR - D5SIICIDSR).....	1302
14.313.1	Offset.....	1302
14.313.2	Function.....	1302
14.313.3	Diagram.....	1302
14.313.4	Fields.....	1303
14.314	DECOa Math Register b_MS (D0MTH0_MS - D5MTH7_MS).....	1303
14.314.1	Offset.....	1303
14.314.2	Function.....	1304
14.314.3	Diagram.....	1304
14.314.4	Fields.....	1304
14.315	DECOa Math Register b_LS (D0MTH0_LS - D5MTH7_LS).....	1304
14.315.1	Offset.....	1304
14.315.2	Function.....	1305
14.315.3	Diagram.....	1305
14.315.4	Fields.....	1305
14.316	DECOa Gather Table Register b (D0GTR0 - D5GTR3).....	1305
14.316.1	Offset.....	1305
14.316.2	Function.....	1306
14.316.3	Diagram.....	1306
14.316.4	Fields.....	1307
14.317	DECOa Scatter Table Register b (D0STR0 - D5STR3).....	1308
14.317.1	Offset.....	1308

Section number	Title	Page
14.317.2	Function.....	1308
14.317.3	Diagram.....	1308
14.317.4	Fields.....	1309
14.318	DECOa Descriptor Buffer Word b (D0DESB0 - D5DESB63).....	1310
14.318.1	Offset.....	1310
14.318.2	Function.....	1310
14.318.3	Diagram.....	1311
14.318.4	Fields.....	1311
14.319	DECOa Debug Job (D0DJR - D5DJR).....	1311
14.319.1	Offset.....	1311
14.319.2	Function.....	1312
14.319.3	Diagram.....	1312
14.319.4	Fields.....	1312
14.320	DECOa Debug DECO (D0DDR - D5DDR).....	1314
14.320.1	Offset.....	1314
14.320.2	Function.....	1314
14.320.3	Diagram.....	1314
14.320.4	Fields.....	1315
14.321	DECOa Debug Job Pointer (D0DJP - D5DJP).....	1316
14.321.1	Offset.....	1316
14.321.2	Function.....	1316
14.321.3	Diagram.....	1317
14.321.4	Fields.....	1317
14.322	DECOa Debug Shared Pointer (D0SDP - D5SDP).....	1318
14.322.1	Offset.....	1318
14.322.2	Function.....	1318
14.322.3	Diagram.....	1318
14.322.4	Fields.....	1319
14.323	DECOa Debug_ICID, most-significant half (D0DIR_MS - D5DIR_MS).....	1319

Section number	Title	Page
14.323.1	Offset.....	1319
14.323.2	Function.....	1320
14.323.3	Diagram.....	1320
14.323.4	Fields.....	1320
14.324	DECOa Debug ICID, least-significant half (D0DIR_LS - D5DIR_LS).....	1321
14.324.1	Offset.....	1321
14.324.2	Function.....	1321
14.324.3	Diagram.....	1321
14.324.4	Fields.....	1322
14.325	Sequence Output Length Register (SOL0 - SOL5).....	1322
14.325.1	Offset.....	1322
14.325.2	Function.....	1323
14.325.3	Diagram.....	1323
14.325.4	Fields.....	1323
14.326	Variable Sequence Output Length Register (VSOL0 - VSOL5).....	1323
14.326.1	Offset.....	1324
14.326.2	Function.....	1324
14.326.3	Diagram.....	1324
14.326.4	Fields.....	1324
14.327	Sequence Input Length Register (SIL0 - SIL5).....	1325
14.327.1	Offset.....	1325
14.327.2	Function.....	1325
14.327.3	Diagram.....	1325
14.327.4	Fields.....	1326
14.328	Variable Sequence Input Length Register (VSIL0 - VSIL5).....	1326
14.328.1	Offset.....	1326
14.328.2	Function.....	1326
14.328.3	Diagram.....	1327
14.328.4	Fields.....	1327

Section number	Title	Page
14.329	Protocol Override Register (D0POVRD - D5POVRD).....	1327
14.329.1	Offset.....	1327
14.329.2	Function.....	1328
14.329.3	Diagram.....	1328
14.329.4	Fields.....	1328
14.330	Variable Sequence Output Length Register; Upper 32 bits (UVSOL0 - UVSOL5).....	1329
14.330.1	Offset.....	1329
14.330.2	Function.....	1329
14.330.3	Diagram.....	1329
14.330.4	Fields.....	1330
14.331	Variable Sequence Input Length Register; Upper 32 bits (UVSIL0 - UVSIL5).....	1330
14.331.1	Offset.....	1330
14.331.2	Function.....	1330
14.331.3	Diagram.....	1330
14.331.4	Fields.....	1331



Chapter 1

Overview of SEC (security engine) functionality

SEC is the chip's cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous cryptographic accelerators. This provides significant improvement to system-level performance. SEC includes the following interfaces:

- Register interface for the processor to write configuration and command information, and to read status information
- 2 DMA interfaces that allow SEC to read/write data from external memory
- Queue Manager interface that allows SEC to accept jobs directly from the Queue Manager module
- 1 AIOP interface that allows SEC to accept jobs directly from the Advanced Input/Output Processor module
- Job Queue Controller with 4 Job Rings
- 6 Descriptor Controllers (DECOs):
 - Responsible for executing descriptors and managing sequencing of keys, context, and data through the various CHAs
 - Responsible for performing header and trailer processing as defined by the descriptor
- Run-Time Integrity Checker (RTIC)
- Crypto Hardware Accelerators (CHAs)
 - Public Key Hardware Accelerator (PKHA)
 - A Random Number Generator (RNG)
 - 6 Advanced Encryption Standard Hardware Accelerators (AESAs)
 - 6 Message Digest Hardware Accelerators (MDHAs)
 - SNOW 3G f9 Hardware Accelerator (SNOW f9)
 - 6 SNOW 3G f8 Hardware Accelerators (SNOW f8)
 - 6 ZUC Encryption Hardware Accelerators (ZUCE)

- ZUC Authentication Hardware Accelerator (ZUCA)
- 6 Data Encryption Standard Hardware Accelerators (DESA)
- 6 Cyclic-Redundancy Check Hardware Accelerators (CRCA)
- 6 Kasumi f8 and f9 Hardware Accelerators (KFHA)

This figure shows the block diagram for SEC.

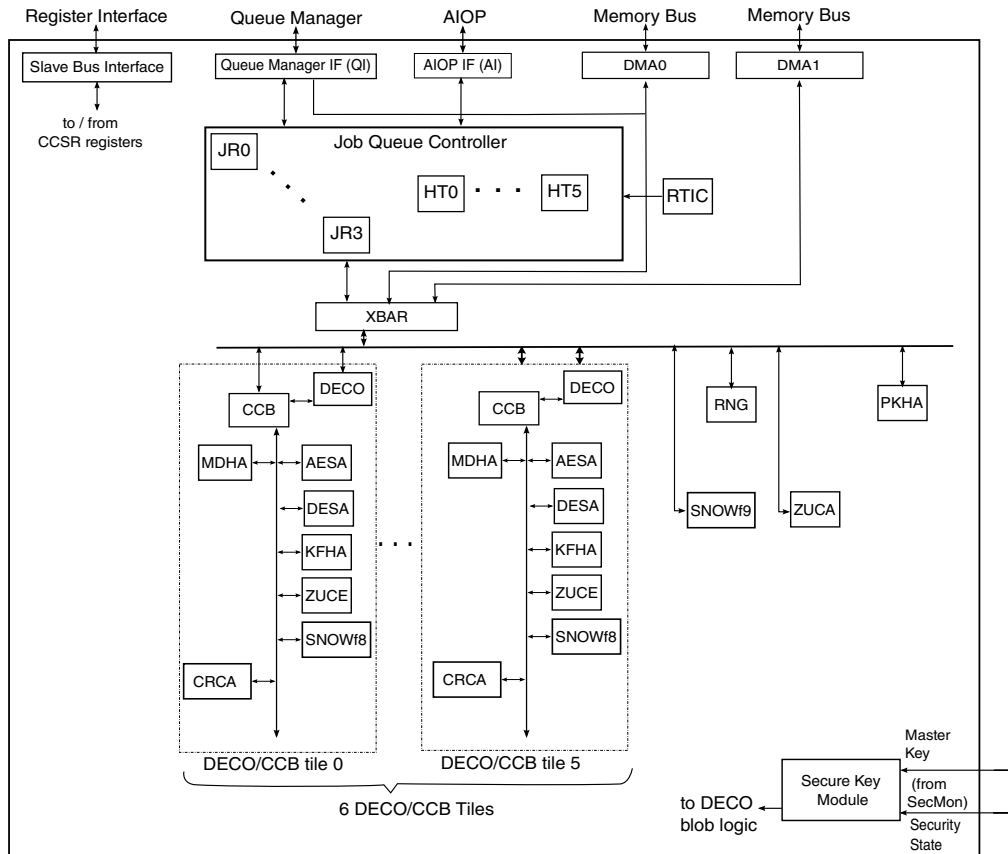


Figure 1-1. SEC block diagram

Chapter 2

Feature summary

SEC includes the following features:

- Offloading of cryptographic functions via a programmable job descriptor language
 - Job descriptors can contain multiple function commands.
 - Job descriptors can be chained to additional job descriptors.
 - Job descriptors can be submitted via 4 separate hardware-implemented Job Rings.
 - Job descriptors can be submitted via Data Path Acceleration Architecture (DPAA) Queue Manager portals.
 - Job descriptors can be submitted via the Advanced IO Processor Accelerator Proxy (AAP) interface.
- 32-bit register bus interface
- 2 128-bit DMA interfaces
 - Automatic byte, half-word, word and double-word ordering of data read/written
 - Scatter/gather support for data
- Special-purpose cryptographic keys
 - Black keys
 - Keys stored in memory in encrypted form and decrypted on-the-fly when used
 - AES_ECB or AES_CCM encryption using a 256-bit key
 - Export and import of cryptographic blobs
 - Data encapsulated in a cryptographic data structure for storage in non-volatile memory
 - AES_CCM encryption using a 256-bit key
 - Each blob encrypted using its own randomly generated blob key.
 - Blob key encrypted using a non-volatile blob key encryption key
 - Blob key encryption key derived from non-volatile master key input
 - Separate blob key encryption keys for trusted mode, secure mode, and non-secure mode
- Public key cryptography
 - Modular Arithmetic

- Addition, subtraction, multiplication, exponentiation, reduction, inversion, greatest common denominator
- Both integer and binary polynomial functions
- Modulus size up to 4096 bits
- Arithmetic operations performed with 128-bit-digit arithmetic unit
- Timing-equalized and normal versions of modular exponentiation
- DSA
 - DSA sign and verify
 - Verify with private key
 - DSA key generation
 - Non-timing-equalized versions of private-key operations
 - Timing-equalized version signing and key generation
 - Non-timing-equalized versions of sign and key generation
- Diffie-Hellman
 - Diffie-Hellman (DH) key agreement
 - Key generation
 - Timing-equalized versions of key agreement and key generation
 - Non-timing-equalized versions of key agreement and key generation
- RSA
 - Modulus size up to 4096 bits
 - Public and Private Key operations
 - Private keys in (n,d) , (p,q,d) , or 5-part (p,q,dp,dq,c) forms
 - Private Key operations (decrypt, sign) timing equalized to thwart side channel attack
 - Non-timing-equalized versions of private-key operations
- Primality testing
 - Maximum size 4096 bits
- Elliptic curve cryptography
 - Point add, point double, point multiply on both prime field and binary polynomial field curves
 - Point validation (is point on curve) both prime field and binary polynomial field curves
 - Timing-equalized and normal versions point multiplication
 - Public Key validation
 - Elliptic curve digital signature algorithm (ECDSA) sign and verify
 - ECDSA verify with private key
 - Elliptic curve Diffie-Hellman key agreement
 - ECDSA and ECDH key generation
 - Modulus size up to 1024 bits
 - Timing-equalized versions of ECDSA sign and key generation
 - Non-timing-equalized versions of sign and key generation

- Cryptographic authentication
 - Hashing algorithms
 - MD5
 - SHA-1
 - SHA-224
 - SHA-256
 - SHA-384
 - SHA-512
 - SHA-512/224
 - SHA-512/256
 - Message authentication codes (MAC)
 - HMAC-all hashing algorithms
 - SSL 3.0 MAC (SMAC-MD5, SHA-1 only)
 - AES-CMAC
 - AES-XCBC-MAC
 - Kasumi f9
 - SNOW 3G f9
 - ZUC authentication
 - Auto padding
 - ICV checking
- Authenticated encryption algorithms
 - AES-CCM (counter with CBC-MAC)
 - AES-GCM (Galois counter mode)
- Symmetric key block ciphers
 - AES (128-bit, 192-bit or 256-bit keys)
 - DES (64-bit keys, including key parity)
 - 3DES (128-bit or 192-bit keys, including key parity)
 - Kasumi f8 with support for 3g, ECSD/EDGE, and GSM
 - SNOW 3G f8
 - ZUC encryption
 - Cipher modes
 - ECB, CBC, CFB, OFB for all block ciphers
 - CTR and XTS for AES
- Random-number generation
 - Entropy is generated via an independent free running ring oscillator
 - For lower-power consumption, oscillator is off when not generating entropy
 - NIST-compliant, pseudo random-number generator seeded using hardware-generated entropy
- Run-time integrity checking
 - SHA-256 message authentication
 - SHA-512 message authentication

- Segmented data-gathering to support non-contiguous data blocks in memory
- Support for up to four independent memory blocks
- Advanced protocol support
 - IPsec
 - SSL/TLS
 - DTLS
 - SRTP
 - IEEE 802.11-2012 WPA2 MPDU for WiFi
 - IEEE 802.16 WiMAX
 - IEEE 802.1AE MacSec/LinkSec
 - Support for protocol-specific padding
 - 3GPP Release 11 (LTE) PDCP layer protocol
- Extensive virtualization features
 - Job rings can be time-shared by multiple security domains
 - Black keys are cryptographically separated per security domain
 - Blobs are cryptographically separated per security domain
 - Trusted descriptors are cryptographically separated per security domain

Chapter 3

SEC implementation

SEC provides platform assurance by working with security monitor (SecMon), which is a companion logic block that tracks the security state of the chip. SEC is programmed using SEC job descriptors (not to be confused with frame descriptors (FDs)) that indicate the operations to be performed and that point to the message and associated data. SEC incorporates two DMA engines to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC can read and write data scattered in memory.

3.1 SEC submodules

The SEC core contains the following submodules:

- Master bus interfaces
- Register bus interface
- Job queue controller (JQC)
- Queue Manager Interface (QI)
- Advanced IO Processor Interface (AI)
- Run-Time Integrity Checker (RTIC)
- Descriptor Controllers (DECOs)
- Cryptographic control blocks (CCBs)
- Multiple cryptographic hardware accelerators (CHAs)

JQC fetches descriptors that tell SEC which cryptographic operations to perform and on what data to operate. DECO decodes descriptors and executes the commands within them. For those descriptor commands that use CHAs, DECO communicates with the CHAs by means of the CCB.

3.2 Cryptographic engines implemented in SEC

The cryptographic engines provided are:

- Public key hardware accelerator (PKHA)
- Data encryption standard (DES) accelerator (DESA)
- Advanced encryption standard (AES) accelerator (AESA)
- Message digest (hashing) hardware accelerator (MDHA)
- Random-number generator (RNG)
- SNOW 3G f8 (SNOW encryption algorithm) Hardware Accelerator (SNOWf8)
- SNOW 3G f9 (SNOW authentication algorithm) Hardware Accelerator (SNOWf9)
- Cyclic redundancy check accelerator (CRCA)
- Kasumi f8 and f9 (Kasumi encryption and authentication) hardware accelerator (KFHA)
- ZUC encryption algorithm hardware accelerator (ZUCE)
- ZUC authentication hardware accelerator (ZUCA)

3.3 SEC Export-Controlled vs. non-Export-Controlled Versions

Some SoCs implement cryptographic algorithms that are subject to United States Government export control regulations requiring export licenses when shipping the SoCs to certain countries, and prohibiting shipping the SoCs to certain other countries. These are referred to as export-controlled SoCs. Versions of these SoCs that have these algorithms permanently disabled avoid these export control restrictions. These are referred to as non-export-controlled SoCs.

In non-export-controlled SoCs all the CHAs that implement these algorithms are permanently disabled. In non-export-controlled SoCs the following bit fields will return 0:

- CHA Number register, AESNUM bit field
- CHA Number register, DESNUM bit field
- CHA Number register, SNW8NUM bit field
- CHA Number register, SNW9NUM bit field
- CHA Number register, KASNUM bit field
- CHA Number register, ZENUM bit field
- CHA Number register, ZANUM bit field
- CHA Number register, PKNUM bit field

When these CHAs are disabled SEC acts as if these CHAs are not implemented. A descriptor that attempts to use one of these CHAs will terminate with error code: Source=2h (CCB), ERRID=Fh (Invalid CHA). Note that the following CHAs will continue to operate even in non-export-controlled SoCs:

- MDHA
- RNG
- CRCA

It is also possible to disable just the ZUC CHAs (ZUCE and ZUCA), when licensing or regulation considerations warrant. If the ZUC CHAs are disabled, the ZENUM and ZANUM bit fields will read as 0.

Chapter 4

SEC modes of operation

SEC can operate in the following security modes:

- Trusted
- Secure
- Non-secure
- Fail

SecMon controls these modes based on SecMon's current security state (that is, init, check, trusted, secure, non-secure, and fail). The primary difference between these modes is that they make different cryptographic keys available. Within each mode there are keys that are volatile (that is, a different key value is used for each power-on session) and keys that are non-volatile (that is, the same key value is available during each power-on session).

4.1 Security Monitor (SecMon) security states

The current security mode can be identified in the SEC status register's MOO field.

This figure shows an overview of the SecMon security state transitions. Note that SEC can detect certain security alarm conditions and can signal an alarm to SecMon. Depending upon the settings and configuration of SecMon, this may cause the SecMon security state machine to transition to fail state, which would then put SEC into its fail mode.

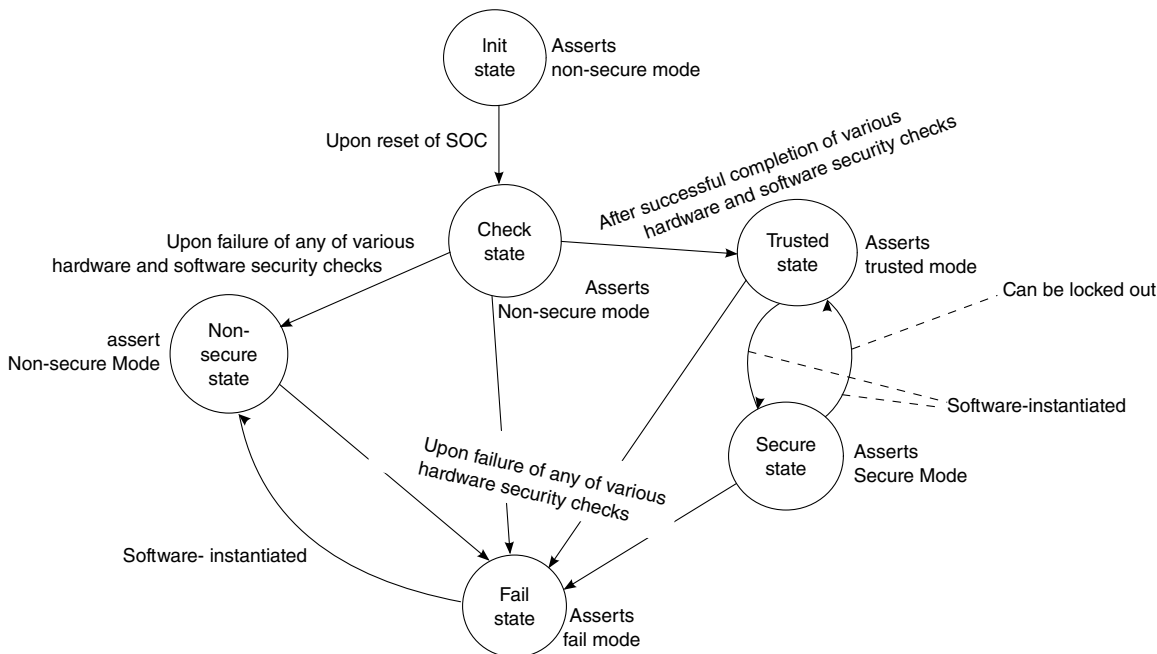


Figure 4-1. SecMon security state machine diagram

4.1.1 The effect of security state on volatile keys

SEC implements three 256-bit volatile cryptographic keys. At each power up, boot code must test and instantiate the RNG. After instantiation, (or as part of RNG instantiation), the three volatile secret keys must be generated. These values are stored within secure key registers in SEC. The values are zeroized when SEC transitions to fail mode (in other words, when the SecMon's security state machine transitions to fail state).

The available volatile keys, (which are located in SEC's secure key module), are as follows:

- Job descriptor key encryption key (JDKEK) - used by job descriptors for encrypting black keys (encrypted keys)
- Trusted descriptor key encryption key (TDKEK) - used by trusted descriptors for encrypting black keys
- Trusted descriptor signing key (TDSK) - used to authenticate trusted descriptors (digitally signed job descriptors)

Note that the JDKEK, the TDKEK, and the TDSK are all available for use by SEC in trusted mode, secure mode, and non-secure mode¹, but this does not cause any security issue. The reason that this is not a security issue is that the trusted mode and secure mode

1. The JDKEK, TDKEK, and TDSK are readable and writable while in non-secure mode to facilitate hardware testing.

are intended to use the same values for these keys, and these key values will be different when in non-secure mode (which is not allowed to obtain the trusted/secure state mode values of these keys). The reason that SEC cannot obtain the trusted/secure state mode values of these keys when in non-secure mode is that new values for these keys are generated by SEC's hardware RNG at each POR, and these keys are zeroized when entering fail mode. As shown in [Figure 4-1](#), the only paths from trusted state or secure state to non-secure state pass through fail state or through a hardware reset, and in each case the keys will be cleared. The only path from non-secure state to either trusted state or secure state is through a hardware reset, which clears the keys. Consequently, when operating in non-secure mode SEC does not have access to previous trusted mode/secure mode values of these keys.

4.1.2 The effect of security state on non-volatile keys

Data that must be retained when the system is powered off must be stored in external non-volatile storage. Some of this data is disclosure-sensitive (such as data rights management keys) and must be protected even when the system is powered off. SEC implements non-volatile cryptographic keys that can be used to encrypt sensitive data during one power-on cycle, and then decrypt it during a subsequent power-on cycle. These non-volatile keys (blob key encryption keys) are derived from the master key input that SEC receives from SecMon.

When SEC is operating in trusted mode or secure mode, SEC derives blob key encryption keys (BKEKs) from its master key input. When SEC is operating in non-secure mode or fail mode, BKEKs are derived from the non-volatile test key, a hardwired constant used for known-answer testing.

4.2 Keys available in different security modes

The primary difference between SEC's security modes is that different cryptographic keys available are available in the different modes. See each mode's section for the description of the mode's special keys.

4.2.1 Keys available in trusted mode

While in trusted mode, SEC can use special keys as listed in this table.

Table 4-1. Special keys used in trusted mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> At POR, a new value (shared with secure mode but not shared with non-secure mode) should be generated from the RNG after instantiation Zeroized when entering fail mode 	Used for automatic key encryption and decryption when executing Job Descriptors, Trusted Descriptors and Shared Descriptors
Trusted descriptor key encryption key		Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors.
Trusted descriptor signing key		Used for signing, verifying and re-signing Trusted Descriptors
Master key derivation key	Non-volatile, shared with secure mode, but uses a different key derivation function input to generate keys not shared with trusted mode, non-secure mode or fail mode	Used for blob encapsulation or decapsulation operations

4.2.2 Keys available in secure mode

While in secure mode, SEC can use special keys as listed in this table.

Table 4-2. Special keys used in secure mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> At POR a new value (shared with trusted mode but not shared with non-secure mode) should be generated from the RNG after instantiation Zeroized when entering fail mode 	Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors and shared descriptors
Trusted descriptor key encryption key		Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors
Trusted descriptor signing key		Used for signing, verifying and re-signing Trusted Descriptors
Master key derivation key	Non-volatile, shared with trusted mode, but uses a different key derivation function input to generate keys not shared with trusted mode, non-secure mode or fail mode	Used for blob encapsulation or decapsulation operations

4.2.3 Keys available in non-secure mode

In non-secure mode a fixed default key with a known value is used in place of the master key derivation key. This allows the cryptographic blob mechanism to be tested using known test results. The volatile key registers are read and write accessible until they are locked, which allows testing using known test results. While in non-secure mode SEC can use special keys as listed below.

Table 4-3. Special keys used in non-secure mode

Key	Characteristic(s)	Function(s)
Job descriptor key encryption key	<ul style="list-style-type: none"> At POR, a new value (not shared with trusted mode or secure mode) should be generated from the RNG after instantiation Zeroized when entering fail mode 	<ul style="list-style-type: none"> Can be read and overwritten for testing Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors, and shared descriptors
Trusted descriptor key encryption key		<ul style="list-style-type: none"> Can be read and overwritten for testing Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by a trusted descriptors
Trusted descriptor signing key		<ul style="list-style-type: none"> Can be read and overwritten for testing Used for testing the signing, verifying and re-signing of trusted descriptors
Master key derivation key	Non-volatile, fixed, and not shared with trusted mode or secure mode	Used for testing blob encapsulation or decapsulation operations

4.2.4 Keys available in fail mode

When SEC transitions to fail mode, SEC clears all registers that could potentially hold sensitive data². Because of this, cryptographic operations that are in progress when the transition occurs will likely not produce the correct result. If this is the case, the operation completes with an error indication.

Although SEC cleans up ongoing operations after a transition to fail mode, SEC is not intended to continue operating in this mode. After removing all causes for entering the fail mode, software can initiate a transition from fail mode to non-secure mode by commanding the SecMon security state machine to transition from fail state to non-secure state (unless this transition has previously been locked out via software). However, since all key registers were cleared when SEC entered fail mode, the only useful actions that can be performed after the transition to non-secure mode would be those required to investigate the cause of the transition to Fail mode.

Note that it is not possible to transition from fail mode back to secure mode or trusted mode.

2. The registers that are cleared include the class 1 and class 2 key registers, the class 1 and class 2 context registers, the math registers, the JDKEK, TDKEK and TDSK registers, the PKHA E memory, the input data FIFO, the output data FIFO, and the descriptor buffer.

Chapter 5

SEC hardware functional description

As shown in [Figure 1-1](#), SEC functionality is aligned with several major subcomponents. This table describes these subcomponents.

Table 5-1. SEC subcomponents

Description	Cross-reference(s)
Interfaces	
Register interface Used for access to configuration, control, status and debugging registers	Register interface (IP bus)
Job execution interfaces	
Job Ring Interface (JR)	Job Ring interface
Queue Manager Interface (QI)	Queue Manager Interface (QI)
AIOP Interface (AI)	Advanced IO Processor Interface (AI)
Job Queue Controller	
Schedules tasks for the descriptor processor	Job scheduling
Descriptor processor	
Descriptor controller (DECO)	Descriptors and descriptor commands
Cryptographic control block (CCB)	Descriptor controller (DECO) and cryptographic control block (CCB)
Cryptographic hardware accelerators (CHAs)	
Public key hardware accelerator (PKHA)	Public-key hardware accelerator (PKHA) functionality
Kasumi f8 and f9 hardware accelerator (KFHA)	Kasumi f8 and f9 hardware accelerator(KFHA) functionality
DES and 3DES hardware accelerator (DESA)	Data encryption standard accelerator (DES) functionality
CRC hardware accelerator (CRCA)	Cyclic-redundancy check accelerator (CRCA) functionality
Random Number Generator (RNG)	Random-number generator (RNG) functionality
SNOW f8 Hardware Accelerator (SNOWf8)	SNOW 3G f8 accelerator functionality
SNOW f9 Hardware Accelerator (SNOWf9)	SNOW 3G f9 accelerator functionality
Message Digest Hardware Accelerator (MDHA)	Message digest hardware accelerator (MDHA) functionality
AES Hardware Accelerator (AES/A)	AES accelerator (AES/A) functionality
ZUC encryption hardware accelerator (ZUCE)	ZUC encryption accelerator (ZUCE) functionality
ZUC authentication hardware accelerator (ZUCA)	ZUC authentication accelerator (ZUCA) functionality
ChaCha20 Hardware Accelerator (CCHA)	ChaCha20 hardware accelerator (CCHA) CHA functionality

Table continues on the next page...

Table 5-1. SEC subcomponents (continued)

Description	Cross-reference(s)
Poly1305 Hardware Accelerator (PTHA)	Poly1305 Hardware Accelerator (PTHA) functionality
Trust Architecture modules	
Run-time integrity checker (RTIC)	Run-time integrity checker (RTIC)
Secure key module	Black keys Blobs Trusted descriptors

5.1 System Bus Interfaces

SEC is connected to a SoC-wide bus for access to SEC registers. See [Register interface \(IP bus\)](#). Two AXI master interfaces connect to the SoC bus fabric for DMA access to system memory.

5.1.1 AXI master (DMA) interface

DMA access to system memory is implemented through 2 independent AXI master interfaces. SEC DMA always asserts normal (AKA user) mode rather than privileged (AKA supervisor) mode, and always asserts data access rather than instruction access (i.e. fetch). SEC DMA can be configured to assert either TrustZone SecureWorld or TrustZone NonSecureWorld for different bus transactions. SEC DMA can also be configured to assert specified ICID values for various bus transactions. For high throughput these interfaces allow simultaneous transfers over 128-bit data buses.

The AXI master interface configuration defaults are chosen to enhance performance where possible, however ideal configuration for performance is not the default and should not be assumed for any application. The DMAs read and write data in data-bus-aligned bursts, whenever possible. The LARGE_BURST field default value is '0' but better performance will be achieved with a value of '1' (See the Master Configuration Register (MCFGR)) Other notable, performance enhancements include the use of read-safe, write-safe, and write-efficient transactions, described in the following sections.

5.1.1.1 DMA read-safe transactions

A read-safe transaction may read data preceding and/or following the target bytes to align the starting and ending byte addresses to data bus or burst address boundaries, even if not all of the data read is needed. This improves performance because the bus system is optimized for address-aligned transactions.

NOTE

When reading hardware registers, it is not always safe to read extra bytes beyond the limits of the register due to potential side effects. For example, a register at an adjacent address might automatically clear itself when its contents are read. Although SEC's DMA would not normally be directed to read hardware registers, read-safe operation can be disabled if necessary (see the RSE field in the DMA Control Register).

5.1.1.2 DMA interface write-safe transactions

A write-safe transaction is similar to a read-safe transaction in that it accesses a set of memory locations even if not all of those locations need to be accessed. In the case of write-safe transactions, SEC writes zeros to addresses past the targeted locations up to the next data bus or burst address boundary, depending upon the offset within the cache line. This improves performance because the bus system is optimized for cacheline-aligned transactions.

SEC's DMA uses write-safe transactions only when the following conditions are all met:

- A SEQ STORE or SEQ FIFO STORE command is being executed.
- A SEQ OUT PTR command defined the sequence.
- The EWS (Enable Write-Safe) bit was set in the SEQ OUT PTR command.
- The total number of bytes written does not exceed the available buffer space.

Note that even when performing a write-safe transaction, SEC's DMA does not write more bytes than was specified by the LENGTH field of the SEQ OUT PTR command (plus any extensions to the length specified by SEQ OUT PTR commands with the PRE bit set). If a SEQ OUT PTR command with the REW (Rewind) field set to 10b or 11b has been executed, write-safe transactions are not performed unless output length counting has been re-enabled via the DECO Control Register. Note that write-safe transactions can be disabled if necessary (see WSE field in the DMA Control Register).

5.1.1.3 DMA write-efficient transactions

In addition to “write-safe” transactions, the DMA interface also allows “write-efficient” transactions to be used for updating Descriptors in memory. Most of the built-in protocols utilize a Protocol Data Block (PDB) that is located within the Job Descriptor or Shared Descriptor that contains the PROTOCOL OPERATION command. The PDB specifies various options, but may also contain protocol state information (such as sequence numbers) that must be updated for each protocol data unit that is processed. Any updates that were made to portions of the PDB while the Descriptor was executing must be written back to the Descriptor in memory once the Descriptor terminates so that the new state information is available for subsequent executions of the same Descriptor. These write-back operations are more efficient if the write transaction is aligned to data bus and burst address boundaries. Write-efficient bus transactions (specified by STORE command SRC values 45h and 46h) will write out more of the Descriptor Buffer than just the updated portions of the Descriptor if this is both possible and more efficient. Extending (either toward the beginning or toward the end, or both) the portion of the Descriptor Buffer to be written out is considered possible if the extended portion contains only the Shared Descriptor or only the Job Descriptor that must be updated. Extending the portion of the Descriptor Buffer to be written out is considered efficient if this causes the write transaction to be aligned to address boundaries that minimizes the number of bus transactions. If extending the portion to be written is not possible or not efficient, then the bus transactions associated with STORE commands using SRC values 45h and 46h will be identical to bus transactions associated with SRC values 41h and 42h.

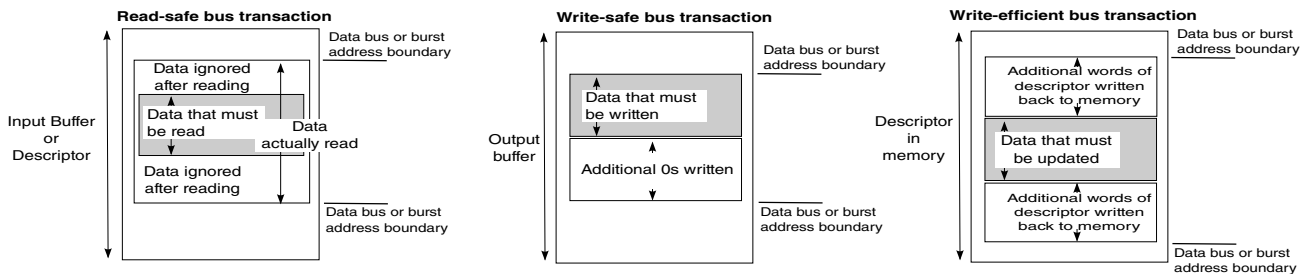


Figure 5-1. Read-Safe, Write-Safe and Write-Efficient Bus Transactions

5.1.1.4 DMA bursts that may read past the end of data structures

SEC DMA accesses do not read a full burst if the read would need to cross a 4 Kbyte address boundary. SEC also does not read a full burst from a Job Ring input or output ring if it would need to read past the end of the ring. However, as illustrated in the figure below, SEC may read past the end of a descriptor or scatter/gather table (SGT) when fetching them because it does not know the length of the descriptor or SGT before issuing the read transaction.

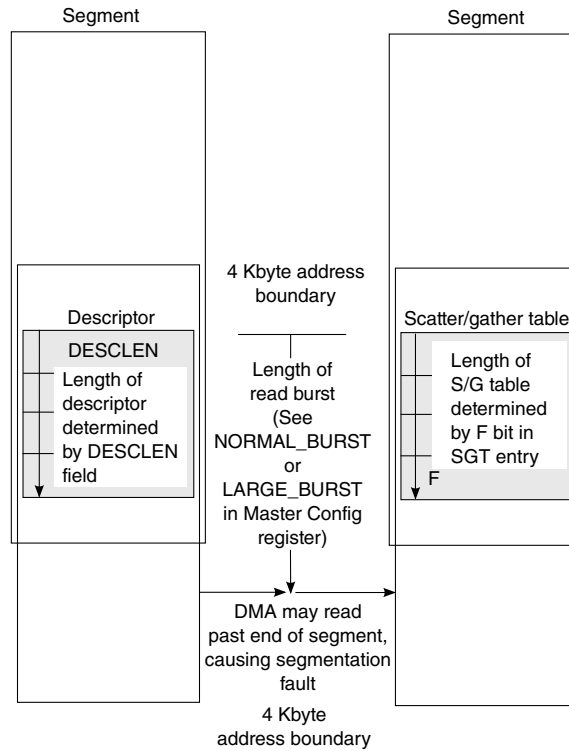


Figure 5-2. DMA may read past end of descriptor or SGT

5.1.2 Register interface (IP bus)

SEC's register interface (32-bit IP bus) is used to read and write registers within SEC for the following purposes:

Table 5-2. Summary of register interface uses

Purpose	For more information, see
During chip initialization time	
To configure SEC, including initialization of the Job Rings, Queue Manager Interface and AIOP Interface.	<ul style="list-style-type: none"> • Initializing Job Rings • Initializing the Queue Manager Interface • Initializing the AIOP interface
Change the default settings for SEC's AXI DMA interfaces	Master Configuration Register (MCFGR) in the memory map
Configure RTIC	Initializing RTIC
During normal steady-state operations	
Manage SEC's Job Ring interface	Job Ring interface
During hardware and software debugging	
Read status registers and single-step descriptor commands	<ul style="list-style-type: none"> • Register-based service interface • SEC Status Register • RNG TRNG Status Register • RNG DRNG Status Register • Holding Tank Status Register

Table 5-2. Summary of register interface uses

Purpose	For more information, see
	<ul style="list-style-type: none"> • Secure Memory Status Register • Job Ring Output Status Register • Job Ring Interrupt Status Register • CCB Interrupt Status Register • CCB Status and Error Register • CCB FIFO Status • DECO Operation Status Register

NOTE

Accesses to registers other than the DECO and CCB registers must use full-word (32-bit) reads or writes. Reads and writes to the DECO and CCB registers permit byte access.

5.2 SEC service interface concepts

SEC delivers cryptographic services through a set of interfaces optimized for different use models (see [Service interfaces](#)). All service interfaces share a number of common objects and concepts, which are explained in more detail in the subsections below.

5.2.1 SEC descriptors

SEC provides cryptographic services by executing a series of commands specified in SEC descriptors. Each SEC descriptor is formed from SEC commands and embedded data. The set of available commands includes conditional branches, loops, subroutine calls, or jumps to other descriptors, as well as mathematical, cryptographic and data move operations. Except as specified by the branch and call commands, the commands within SEC descriptors normally execute in sequence until the descriptor has completed (or has been aborted due to external management action). Descriptors cannot change their own execution priority or directly affect the scheduling of other descriptors, but SEC descriptors do have mechanisms to ensure coherency of data shared between descriptors.

SEC descriptors access input, output, and control data by means of memory addresses and job-specific memory access control attributes, or, for jobs submitted via Job Rings, job-ring-specific memory access control attributes. SEC descriptors also cannot allocate or release buffers to/from buffer pools.

SEC implements different types of descriptors to address specific processing needs:

- Job Descriptor (JD) (see [Job descriptors](#))

Every SEC job is defined by at least one JD. The JD may be provided by the SEC service user directly via the register-based service interface or via the Job Ring-based service interface, or the JD may be created internally within SEC in response to a service request from the Queue Manager Interface (QI) or the Advanced IO Processor Interface (AI) or the Run-Time Integrity Checker (RTIC). It is also possible for a JD to invoke a Shared Descriptor (SD) or to jump to another JD, which allows a job to consist of an arbitrarily large number of commands and data objects.

- Shared Descriptor (SD) (see [Shared descriptors](#))

SDs provide a mechanism to group and reuse instructions and data that are common in the processing of more than one related job, e.g., processing protocol data units of a network connection. A key feature of SDs is to select and coordinate sharing of the descriptor information between multiple DECOs. Using SDs may also increase performance by improving the probability of finding an SD within SEC that has already been read for a preceding job requiring the same processing. While primarily not intended for single, one-off jobs, it may be more efficient for DPAA2 software using the QI or AI service interface to utilize SDs instead of Inline Job Descriptors (IJDs) even for one-off jobs because the SD information does not need to be inserted in the SEC input data stream (as would be required for IJDs).

- Trusted Descriptor (TD) (see [Trusted descriptors](#))

TDs are essentially the same as JDs, but they are cryptographically signed. When a TD is presented for execution, SEC first checks the signature and executes the TD only if the signature is correct. TDs are intended to ensure that special access privileges are usable only by descriptors that are known to employ those privileges properly. TDs would be created by trusted software (such as secure boot software or a hypervisor), and then cryptographically signed to ensure that they were not altered by untrusted software.

- Inline Job Descriptor (IJD) (see [Using in-line descriptors](#))

IJDs are simply JDs that are made available to SEC through the input data stream. JDs submitted via Job Rings may direct SEC to execute commands from an IJD at any time using the SEQ IN PTR command with the INL option selected. QI-generated and AI-generated JDs imply the service user's intent to execute an IJD from the absence of an SD specification (SD length is set to 0). In this case SEC is directed to execute commands from an IJD starting at the first byte of the input data stream and immediately after output and input data stream addresses are defined in the internally generated JD. IJDs were originally intended for the processing of one-off jobs or job variations in earlier first generation DPAA products where SEC users could not select an SD on a per service request basis without also requiring the setup

of dedicated queues. This limitation does not exist in DPAA2, but IJDs remain available to provide backward compatibility and reduce the need for software updates.

- Replacement Job Descriptor (RJD and CRJD) (see [Using replacement job descriptors](#))

RJDs and Control RJDs (CRJDs) are intended to support job processing variations or updates of immediate or state data defined in SDs. Both kind of RJDs replace the JD that invoked them and can be executed either before or after the execution of the SD. Thus RJDs and CRJDs provide the capability to permanently update or temporarily change the processing defined by SDs. RJDs may be supplied with 2 methods: The normal RJD is supplied inline (like the IJD) embedded in the input data stream. Alternatively, a CRJD associated with a specific SD may be utilized. A CRJD must be located in memory immediately following the SD. The execution of an RJD is initiated with the SEQ IN PTR command by setting the RJD control bit. A CRJD requires to additionally set the CTRL bit.

5.2.2 Job termination status/error codes

SEC reports the termination status of all¹ jobs, allowing software to determine whether the descriptor completed normally, with warnings, or with an error. The reporting mechanism always involves writing a job termination status word to memory. Depending on the selected service interface, SEC may also update service interface status registers.

An all-zero status indicates that the job completed without warnings or errors. If a warning or an error was encountered, the code in the source field of the status word indicates which component within SEC detected the condition. The remaining status word coding provides additional component-specific detail.

For jobs submitted through the Job Ring interface the job termination status is written to the Job Ring Output Status register and to the output ring in the word following the pointer to the completed job descriptor. The job termination status can be read from the Job Ring Output Status register, but because the termination status of each newly completed job will overwrite the previous job's termination status this mechanism is primarily intended to support debug and ring management (for executing single jobs or after the ring is halted). A selection of severe error conditions (potentially indicating malicious users or software instability) is stored together with additional error and/or access violation attributes in job-ring specific recoverable error record registers.

1. Receiving a frame descriptor with a non-0 ERR field on the Queue Manager Interface (QI) is not reported as an error to the service user but to the Management Complex (MC) to prevent infinite error reporting loops.

For jobs submitted through the Queue Manager Interface (QI) or the Advanced IO Processor Interface (AI) the job completion status is written to the FRC field in the response/output frame descriptor returned to the user. If QI or AI detects an error, the status word's source field is set to 5 or 1, respectively. A selection of severe error conditions (potentially indicating malicious users or software instability) is stored together with additional error and/or access violation attributes in the QI and AI status and recoverable error record registers.

Besides the standardized job termination status format described above (and listed in more detail in the table below) SEC also supports an optional, compressed, and 'self-identifying' status word format for QI and AI users. In the compressed format the most significant byte of the standard status word is shifted right by 8 bit (consuming reserved space of the standard format status word). The vacated space in the status word is left clear to indicate unconditional successful completion and coded as 83h for all other status indications (errors or warnings). A self-identifying status code enables simplified implementation of software that need to process errors or warnings from multiple sources that are not guaranteed to report unique status codes. With the availability of the compressed status format the most significant bit can be interpreted as a sign bit and software can quickly identify whether any special handling is necessary. The less significant 7 bits of the most significant byte are used to identify the source of the status (SEC is assigned the unique code 3) and the remaining coding of the status word follows the same principles as documented for the standard status word format. AI utilizes the compressed status format by default. For backward compatibility with existing QI software the compressed status format is by default disabled.

Table 5-3. Job termination status word

bits 31-28		bits 27-0				
Source		Source-specific error or warning codes				
0h (None)		0000000h - No errors or warnings				
bits 31-28		bits 27-8			bits 7-0	
Source		Reserved			Error code	
1h (AI)		00000h			See QI (Source 5h) below. AI and QI use the same error and warning codes. In addition, AI reports an AIOp Accelerator Proxy detected error condition: DFh - AAP read error	
bits 31-28	bit 27	bit 26	bits 25-16	bits 15-8	bits 7-4	bits 3-0
Source	(JMP)	(MLK)	Reserved	(DESC INDEX)	(CHAID)	(ERRID)
2h (CCB)	See footnote ¹	See footnote ²	0000h	The number of words from the start of the descriptor where the error was	0h - CCB 1h - AESA (all modes of AES)	0h - No error 1h - Mode error 2h - Data size error

Table continues on the next page...

Table 5-3. Job termination status word (continued)

				detected. In some cases this value may be off by one or more words due to timing issues.	2h - DESA (DES and 3DES) 4h - MDHA (MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256) 5h - RNG 6h - SNOWf8 (SNOW encrypt/decrypt) 7h - KFHA f8/9 Kasumi encrypt/decrypt and authentication) 8h - PKHA (all public key operations) 9h - CRCA (CRC processing) Ah - SNOWf9 (SNOW authentication) Bh - ZUCE (ZUC encrypt/decrypt) Ch - ZUCA (ZUC authentication)	3h - Key size error 3h - (RNG) Instantiate error 4h - (RNG) Not instantiated error 4h - (PKHA) A size error 5h - (RNG) Test instantiate error 5h - (PKHA) B size error 6h - (RNG) Prediction resistance error 6h - Data out-of-sequence error 6h - (PKHA) "c" is zero for ECC F2M 7h - (RNG) Prediction resistance & test request error 7h - (PKHA) Divide by 0 error 8h - (PKHA) Modulus even error 9h - (DES) key parity error 9h - (RNG) Secure Key Generation error Ah - ICV check failed Bh - Hardware error Ch - (AESA) CCM AAD size error Ch - (RNG) Continuous check error Ch - Invalid key write Dh - (CCB) Class 1 or class 2 CHA is not reset, or, a second CHA of the same class is selected prior to resetting the first selection Eh - (CCB) Invalid CHA combination selected Fh - (CCB) Invalid CHA
bits 31-28	bit 27	bit 26	bits 25-16	bits 15-8	bits 7-0	
Source	(JMP)	(MLK)	Reserved	(DESC INDEX)	User-defined value	
3h (Jump Halt User Status)	See footnote ¹	See footnote ²	0000h	The number of words from the start of the descriptor where the JUMP	The value in the LOCAL OFFSET field of the JUMP command is written into these bits of the termination status word. The user is free to assign any interpretation	

Table continues on the next page...

Table 5-3. Job termination status word (continued)

bits 31-28	bit 27	bit 26	bits 25-16	bits 15-8	bits 7-0
Source	(JMP)	(MLK)	Reserved	(DESC INDEX)	Error Code
4h (DECO)	See footnote ¹	See footnote ²	0000h	HALT Command was encountered. The number of words from the start of the descriptor where the error was detected. In some cases this value may be off by one or more words due to timing issues.	to these bits, such as using them to distinguish among different instances of the JUMP command. 00h - No error 01h - SGT length error (The descriptor is trying to read more data than is contained in the SGT table.) 02h - Unused SGT entry error (Extension bit set in unused SGT entry.) 03h - Job Ring Control Error (There is a bad value in the Job Ring Control register.) 04h - Invalid Descriptor Command 06h - Invalid KEY Command 07h - Invalid LOAD Command 08h - Invalid STORE Command 09h - Invalid OPERATION Command 0Ah - Invalid FIFO LOAD Command 0Bh - Invalid FIFO STORE Command 0Ch - Invalid MOVE/MOVE_LEN Command 0Dh - Invalid JUMP Command (a non-local JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a TD to a JD, or because the target descriptor contains an SD) 0Eh - Invalid MATH Command 0Fh - Invalid SIGNATURE Command 10h - Invalid Sequence Command (A SEQ IN PTR or SEQ OUT PTR Command is invalid or a SEQ KEY, SEQ LOAD, SEQ FIFO LOAD, or SEQ FIFO STORE decremented the input or Output Sequence length below 0. This error may result if a built-in PROTOCOL Command has encountered a malformed PDU.) 11h - Skip data type invalid (The type must be Eh or Fh.) 12h - Shared Descriptor Header Error 13h - Header Error (Invalid length or parity, or certain other problems.) 14h - Burster Error (Burster has gotten into an illegal state.) 15h: Context Register Length Error. The descriptor is trying to read or write past the end of the Context Register. A SEQ LOAD or SEQ STORE with the VLF bit set was executed with too large a length in the variable length register (VSOL for SEQ STORE or VSIL for SEQ LOAD). 16h - DMA Error 1Ah - Job failed due to Job Ring reset or RTIC error

Table continues on the next page...

Table 5-3. Job termination status word

					<p>1Bh - Job failed due to transition to Fail Mode</p> <p>1Ch - DECO Watchdog timer timeout error</p> <p>1Dh - Error when copying key from another DECO (other DECO's key registers were locked)</p> <p>1Eh - Error when copying data from another DECO (other DECO had unmasked descriptor error)</p> <p>1Fh - ICID mismatch error (DECO was trying to share from itself or from another DECO but the two Non-SEQ ICID values didn't match or the "shared from" DECO's Descriptor required that the SEQ ICID and TZ/SDID values be the same but they aren't.)</p> <p>20h - DECO has completed a reset initiated via the DRR register</p> <p>21h - Nonce error (When using EKT (CCM) key encryption option in the FIFO STORE Command, the Nonce counter reached its maximum value and this encryption mode can no longer be used.)</p> <p>22h - Leading meta data is too large for rewind operation</p> <p>23h - Read Input Frame error (A read input frame was attempted, but the protocol executed does not support it or a SEQ IN PTR command has not been executed.)</p> <p>24h - JDKEK, TDKEK or TDSK was needed, but value has not yet been initialized.</p> <p>25h - Error while prefetching</p> <p>26h - A job has DECO select value for a different DECO</p> <p>27h - Rewind input frame attempted but input buffers already released</p> <p>30h - DWORD load error</p> <p>31h - DWORD store error</p> <p>32h - Invalid PKCURVE Command</p> <p>33h - Burster buffer reuse error (address or length went negative)</p> <p>80h - DNR (do not run) error (A Job Descriptor or Shared Descriptor had the DNR bit set.)</p> <p>81h - undefined protocol command</p> <p>82h - invalid setting in PDB</p> <p>83h - Anti-replay LATE error</p> <p>84h - Anti-replay REPLAY error</p> <p>85h - Sequence number overflow</p> <p>86h - Invalid signature</p> <p>87h - DSA Sign Illegal test descriptor</p>
--	--	--	--	--	--

Table continues on the next page...

Table 5-3. Job termination status word (continued)

				<p>88h - Protocol Format Error (A protocol has seen an error in the format of data received. When running RSA, this means that formatting with random padding was used, and did not follow the form: 00h, 02h, 8-to-N bytes of non-zero pad, 00h, F data.)</p> <p>89h - Protocol size error</p> <p>8Ah - Key not written before start of protocol</p> <p>8Bh - IPsec decap CE DROP (ECN issue) error</p> <p>8Ch - RFBG P & Q upper 100 bits the same</p> <p>8Dh - RFBG computed D too small</p> <p>8Eh - RFBG PDB and computed N sizes differ</p> <p>C1h - Undefined Blob mode</p> <p>C4h - Black Blob key or input size error</p> <p>C5h - Invalid key destination in blob command</p> <p>C8h - Trusted/Secure mode error in blob command</p> <p>CAh - LTE C-plane ICV error</p> <p>F0h - Warning: Descriptor completed normally, but IPsec TTL or hop limit field either came in as 0, or was decremented to 0</p> <p>F1h - Warning: Descriptor completed normally, but HFN matches or exceeds the Threshold</p> <p>F2h - Warning: IPsec padding check error found</p> <p>FFh - Warning: Output frame length rollover</p>
bits 31-28		bits 27-8		bits 7-0
Source		Reserved		Error code
5h (QI)		0000h		<p>0Fh - no error</p> <p>1Fh - job terminated by FQ or ICID flush</p> <p>20h - FD format error</p> <p>21h - FD command format error</p> <p>23h - FL format error</p> <p>25h - Control RJD specified in FD but not enabled in FLC</p> <p>30h - max buffer size too small (allocate mode, single buffer format)</p> <p>31h - DHR exceeds max buffer size (allocate mode, scatter/gather format)</p> <p>32h - SGT exceeds max buffer size (allocate mode, scatter/gather format)</p> <p>33h - size over/underflow (allocate mode)</p> <p>34h - size over/underflow (reuse mode)</p> <p>35h - length exceeds max short length (allocate mode, scatter/gather format)</p>

Table continues on the next page...

Table 5-3. Job termination status word (continued)

			<p>36h - memory footprint exceeds max value (allocate mode, scatter/gather format)</p> <p>41h - SBC frame format not supported (allocate mode)</p> <p>42h - pool 0 invalid or pool 1 size less than pool 0 size (allocate mode)</p> <p>43h - annotation output enabled but ASAR=0 (allocate mode)</p> <p>44h - unsupported or reserved frame format or SGHR=1 (reuse mode)</p> <p>45h - DHR correction underflow (reuse mode, single buffer format)</p> <p>46h - annotation length exceeds offset (reuse mode)</p> <p>48h - annotation output enabled but ASA is limited by ASAR (reuse mode)</p> <p>49h - data offset correction exceeds input frame data length (reuse mode)</p> <p>4Bh - annotation output enabled but ASA cannot be expanded (frame list)</p> <p>51h - unsupported IF reuse use mode</p> <p>52h - unsupported FL use mode</p> <p>53h - unsupported RJD use mode</p> <p>54h - unsupported inline descriptor use mode</p> <p>C0h - table buffer pool 0 depletion</p> <p>C1h - table buffer pool 1 depletion</p> <p>C2h - data buffer pool 0 depletion, no OF allocated</p> <p>C3h - data buffer pool 1 depletion, no OF allocated</p> <p>C4h - data buffer pool 0 depletion, partial OF allocated</p> <p>C5h - data buffer pool 1 depletion, partial OF allocated</p> <p>D0h - FLC read error</p> <p>D1h - FL read error</p> <p>D2h - FL write error</p> <p>D3h - OF SGT write error</p> <p>D4h - PTA read error</p> <p>D5h - PTA write error</p> <p>D6h - OF SGT F-bit write error</p> <p>D7h - ASA write error</p> <p>E1h - FLC[ICR]=0 ICID error</p> <p>E2h - FLC[ICR]=1 ICID error</p> <p>E4h - source of ICID flush not trusted (BDI = 0)</p> <p>F0h - misdirected frame error (AI only)</p>
bits 31-28	bits 27-12	bits 11-8	bits 7-0

Table continues on the next page...

Table 5-3. Job termination status word (continued)

Source	Reserved			NADDR	Error code
6h (Job Ring)	0000h			Number of descriptor addresses requested for error code 1Eh, otherwise 0h	00h - No error 1Eh - Error reading the Descriptor address 1Fh - Error reading the Descriptor
bits 31-28	bit 27	bit 26	bits 25-16	bits 15-8	bits 7-0
Source	(JMP)	(MLK)	Reserved	(DESC INDEX)	(COND)
7h (Jump Halt with Condition Codes)	See footnote ¹	See footnote ²	0000h	The number of words from the start of the descriptor where the JUMP HALT Command was encountered.	PKHA/Math condition codes field from JUMP HALT Command.

1. If JMP = 1, the descriptor made a jump to another descriptor. When this bit is 1, the DESC INDEX field will contain the index into the descriptor that was jumped to rather than into the original descriptor.
2. Memory Leak. If MLK=1 a memory leak has occurred due to an error that prevents user requested buffer releases, i.e., this condition can only be detected for jobs generated by QI or AI.

5.2.3 Frames and flows

SEC borrows the term ‘frame’ from network protocols. A frame simply refers to some number of sequential bytes that are usually, but not necessarily, part of a segmented byte stream and delimited by implicit or explicit start and end markers. Explicit markers are formed by so-called protocol headers and/or trailers of protocol-specific length including a possible length of 0. Implicit markers are out-of-band information defining where frame data starts and ends. For SEC the meaning of a frame is generalized to also include designated space into which SEC-generated frame data can be stored, as well as data that is completely unrelated to networking protocols, e.g., a piece of program code that needs to be cryptographically signed or a sequence of SEC-generated random data.

While frames define a logical sequence of bytes, the frame data itself does not need to be necessarily stored in a single, contiguous region of memory (also referred to as a buffer). Segmented, multi-buffer frames can be formed by utilizing scatter/gather tables (stored in additional buffers) where table entries are used to keep track of frame segment address, offset, length, and other segment attributes. For details see [Scatter/gather tables \(SGTs\)](#).

A SEC flow simply refers to a sequence of two or more frames requiring the same kind of processing. Whether the frame data is stored in single buffer frames, multi-buffer frames, or a mix of both is irrelevant. The key criteria of a flow is that all frames are processed in the same fashion. Some flows require that the frames be processed in a specific order. By using shared descriptors, SEC can control the order in which frames are processed and delivered to the service user by selecting an appropriate sharing type. For details see [Shared descriptors](#).

5.2.4 Frame descriptors and frames

A Frame Descriptor (FD) is a standardized Data Path Acceleration Architecture (DPAA) data structure used to pass information between services users and service providers. The FD may convey input and/or output data, output data buffers, as well as context data, how input data should be or has been processed, service user-controlled data access attributes, and a number of controls to support FD queuing options, congestion and buffer management, user data stashing and tracing of FDs for debug and performance analysis. For a detailed generic description of FDs see the DPAA overview chapter of your SoC reference manual.

As described in [Frames and flows](#), a frame refers to a defined sequence of bytes and a SEC FD may convey 0, 1, or 2 frame definitions. A frame in an FD is either defining input data to be consumed, output buffer(s) to be filled, output data that has been produced, or combinations thereof. The frame specific information of the FD may refer to a single or a multi-buffer frame structure and identifies the context data a recipient may need to process the frame data or status information on how the data was processed.

The Queue Manager Interface and the Advanced IO Processor Interface of SEC support 4 different types of FDs:

- FDs with no input or output frame specification
- Single frame input and/or output FDs
- Two-entry input/output list FDs
- Management command FDs

and 2 types of frames:

- Single buffer frames
- Multi-buffer frames

For detailed information on SEC FDs see [Frame descriptors](#)

5.2.5 Frame descriptor flow and flow context

For SEC services provided through QI or AI, a flow refers to two or more Frame Descriptors (FDs) utilizing the same context to process frames. Whether FDs are providing one or two frames (or no frame at all) or whether the frame data is stored in one or multiple buffers is irrelevant. The key criteria of such flows is that all FDs of the flow are processed with the same SEC descriptor(s) and flow context data or controls.

A SEC flow context consists of one or more data structures used to define flow specific frame processing parameters. In all use cases the flow context precedes an optional, but usually present, Shared Descriptor (SD). A SEC flow context defines information in the following categories:

- Data and job processing attributes to optimize frame data access and job scheduling
- Input buffer reuse, release and output buffer allocation controls
- Annotation data pass-through and error handling controls
- User context, user context access, and stashing controls for response/output FDs
- Additional queue and resource access controls for multi-partition use cases (optional)
- An optional, but usually present SD and the SD length (0 if SD is not present)
- An optional Control Replacement Job Descriptor (CRJD)

For detailed information on SEC flow contexts see [Flow Context](#).

5.2.6 Buffer allocation, release, and reuse

One of the key functions of the SEC flow context is to define how frame input and output buffers are managed. In combination with frame descriptor attributes, SEC provides the following options:

- SEC service users may use two-entry input/output frame list descriptors to provide both input and output frames. This enables the user to fully control both the input and the output buffer structures (and bypass any SEC buffer management limitations). Note, this approach may also be utilized to reuse part or all of the input buffers as output buffers. In this case it is the user's responsibility to provide enough output buffer space to hold the output data produced by the requested SEC service and ensure that SEC output does not destroy any input data needed to produce the output data before it can be consumed by SEC.
- SEC service users may use a flow context configuring SEC buffer management for reusing the user's input buffer structure for output data. The same user responsibilities apply as listed in the two-entry input/output frame list descriptor option described above.
- SEC service users may use flow context parameter settings and configure SEC buffer management to optionally release input buffers to their associated buffer pool, allocate output buffers from up to two user-defined buffer pools, and form a variety of output frame structures. In this use case it is the user's responsibility to maintain a sufficient amount of output buffers in the utilized buffer pool(s).

For detailed information on SEC buffer management options see [Flow Context](#).

5.2.7 User data access control and isolation

SEC supports user data access control and SEC-service user isolation through its ability to use service interface-specific Isolation Context Identifiers (ICIDs) and Access Management Qualifiers (AMQs) to tag all related memory and DPAA resource transactions. The ICID/AMQ tag is utilized by the SMMU as a Stream ID to select table entries that define system software enforced address validity checking, optional address and attribute translation as well as other SMMU functions. For details see the SMMU chapter of your SoC reference manual.

SEC can be configured to assign different tags for input, for output, and for control and context data when performing associated memory accesses or utilizing shared DPAA resources like buffer pools and queues. Control data refers to any SEC instructions or data utilized to process an input data stream in order to produce an output data stream or to just output status (e.g., an indication that the signature of some input data is correct). Depending on the selected service interface and user application, access to SEC-utilized memory and other DPAA resources is managed by up to three ICIDs and AMQs.

When accessing memory the Job Ring, RTIC and register service interfaces are limited to one set of AMQ and ICID values defined by a trusted management entity in interface-specific SEC registers.

The SEC Queue Manager service Interface (QI) and Advanced IO Processor service Interface (AI) support three sets of AMQs and ICIDs. The values of these access control attributes as well as an AMQ Usage Control (AUC) are passed to SEC over the service interface to specify how AMQs and ICIDs are applied to input, output, and/or control data.

The SEC QI receives the set of AUC, AMQs, and ICIDs for up to 3 Frame Descriptors from a single queue in the so-called dequeue summary information. SEC's AI receives the same type of information for each Frame Descriptor in the Frame Descriptor transfer attributes. The ICID set of AUC, AMQs and ICIDs always originate from the Queue Manager's or AIOP's access control attributes, which are configured by a trusted management entity.

For a more detailed description of single- and multi-partition resource access applications and associated use limitations see [Multi Partition Resource Access Restriction Summary](#).

5.3 Service interfaces

SEC services may be invoked via the following types of service interfaces:

- A [Register-based service interface](#)
- A [Job Ring interface](#)
- A [Queue Manager Interface \(QI\)](#)
- An [Advanced IO Processor Interface \(AI\)](#)
- A [Run-time integrity checker \(RTIC\)](#)

The register-based interface is primarily intended for management entities to use for simple one-off jobs during startup, run-time testing of SEC functionality, or debugging SEC descriptors. It is not intended for repetitive or high throughput activities.

The Job Ring interface provides single user/driver job queuing, job completion interrupt services, and support for dynamic service interface virtualization via a software management entity. SEC implements 4 Job Ring interfaces that can be independently assigned (and re-assigned) to different users. This service interface type is primarily intended to be (at least temporarily) assigned to either the Arm TrustZone (TZ), system management entities or application entities. Note that creating trusted descriptors can only be accomplished via the Job Ring interface.

QI provides efficient sharing of SEC service for all users having access to a Queue Manager portal and an assigned frame queue pair linking the portal to SEC. Once these conditions are met a bi-directional data path is enabled for service requests and responses between a user and SEC. In this use mode the Queue Manager provides access attributes for input, output, and control (context) data associated with the service request and response. The user's service request identifies the data including details about data buffer locations and how the data is to be processed. Optionally, the SEC service user may utilize queues that convey access attributes for input, output and control data stored in different user domains.

AI provides low-latency SEC service to all Advanced IO Processor (AIOP) tasks through an adaptive resource reservation mechanism and direct access to a dedicated set of SEC internal job staging slots. AI supports the same input, output, and control data access capabilities as QI, including the option to access the 3 types of differentiated SEC data from up to 3 different partition/user domains.

Jobs can also be internally initiated by SEC's RTIC submodule. RTIC is typically configured at startup (and optionally reconfigured thereafter) by a trusted or privileged management entity, and then operates autonomously, periodically submitting specialized SEC descriptors to the job queue controller.

5.3.1 Job Ring interface

The Job Ring interface is a software job programming interface. For each job submitted to SEC, software must create a job descriptor that explicitly describes the data to be processed and the keys and context to be used for the processing (see [Job descriptors](#)). In most high-speed networking products the AI and the QI submit the majority of job descriptors executed by SEC (see [Queue Manager Interface \(QI\)](#) and [Advanced IO Processor Interface \(AI\)](#)), but the Job Ring interface may be used to create security associations and perform other one-time cryptographic operations.

SEC implements 4 Job Ring interfaces. Each Job Ring interface provides an input ring for job descriptors and an output ring for results. This queuing mechanism allows software to schedule multiple SEC jobs for execution and then retrieve the results as convenient. The input rings and output rings are implemented as circular buffers (also called rings) that are located in system memory.

The entries in the input rings are pointers to job descriptors that are located elsewhere in memory (see [Address pointers](#)). Each entry in an output ring consists of a pointer to a job descriptor followed by a job termination status word (see [Job termination status/error codes](#)). They may also be followed by a word containing the number of bytes written by SEQ STORE and SEQ FIFO STORE commands during the descriptor's execution (see INCL_SEQ_OUT field in the Job Ring Configuration Register). The job descriptor pointers in the output rings allow software to correlate result status with the particular job descriptor that SEC executed to produce that result.

5.3.1.1 Configuring and managing the input/output rings, overview

Software configures the input and output rings and then manages them jointly with SEC. The following table describes the uses of the input and output ring registers:

Table 5-4. Input/output ring registers

Register	Description
Input/Output Ring Base Address Register	Describes the base address of the ring buffer, which must be a multiple of four bytes
Input/Output Ring Size Register	Describes the size of the ring buffer measured in the number of entries
Input Ring Jobs Added/Output Ring Jobs Removed Register	Tells SEC how many jobs software placed in the input ring or removed from the output ring
Input Ring Slots Available/Output Ring Slots Full Register	Tells software how many spaces are available to add jobs to the input ring or how many jobs are in the output ring ready for software processing
Input Ring Read Index	Points to the head of the queue within the input ring, that is, where SEC finds the next job descriptor to read from the Job Ring
Output Ring Write Index	Points to the tail of the queue within the output ring, that is, where SEC places the results of the next completed job descriptor for that Job Ring

Table continues on the next page...

Table 5-4. Input/output ring registers (continued)

Register	Description
Job Ring Configuration Register	Used to configure Job Ring interrupts and select whether the optional sequence out length word appears in Output Ring entries

[Figure 5-3](#) shows an example input ring and output ring. The physical ring buffers are shown in the boxes on the right. The logical queues located within these ring buffers are shown in shaded boxes to the left. Each input ring entry consists of a pointer to a job descriptor. Each output ring entry consists of a pointer to a job descriptor followed by a 32-bit word indicating the job completion status.

In this example, jobs 10 through 15 are in the input ring waiting for SEC to process them. The results for jobs 4 through 8 are in the output ring, waiting for software to retrieve them. SEC has removed job 9 from the input ring, but has not yet written the results to the output ring. Old entries that have not yet been overwritten are shown in italics.

Note that job 7 completed ahead of job 6. In versions of SEC that implement more than one DECO, it is possible for jobs submitted through the same Job Ring to complete out of order. See [Order of job completion](#).

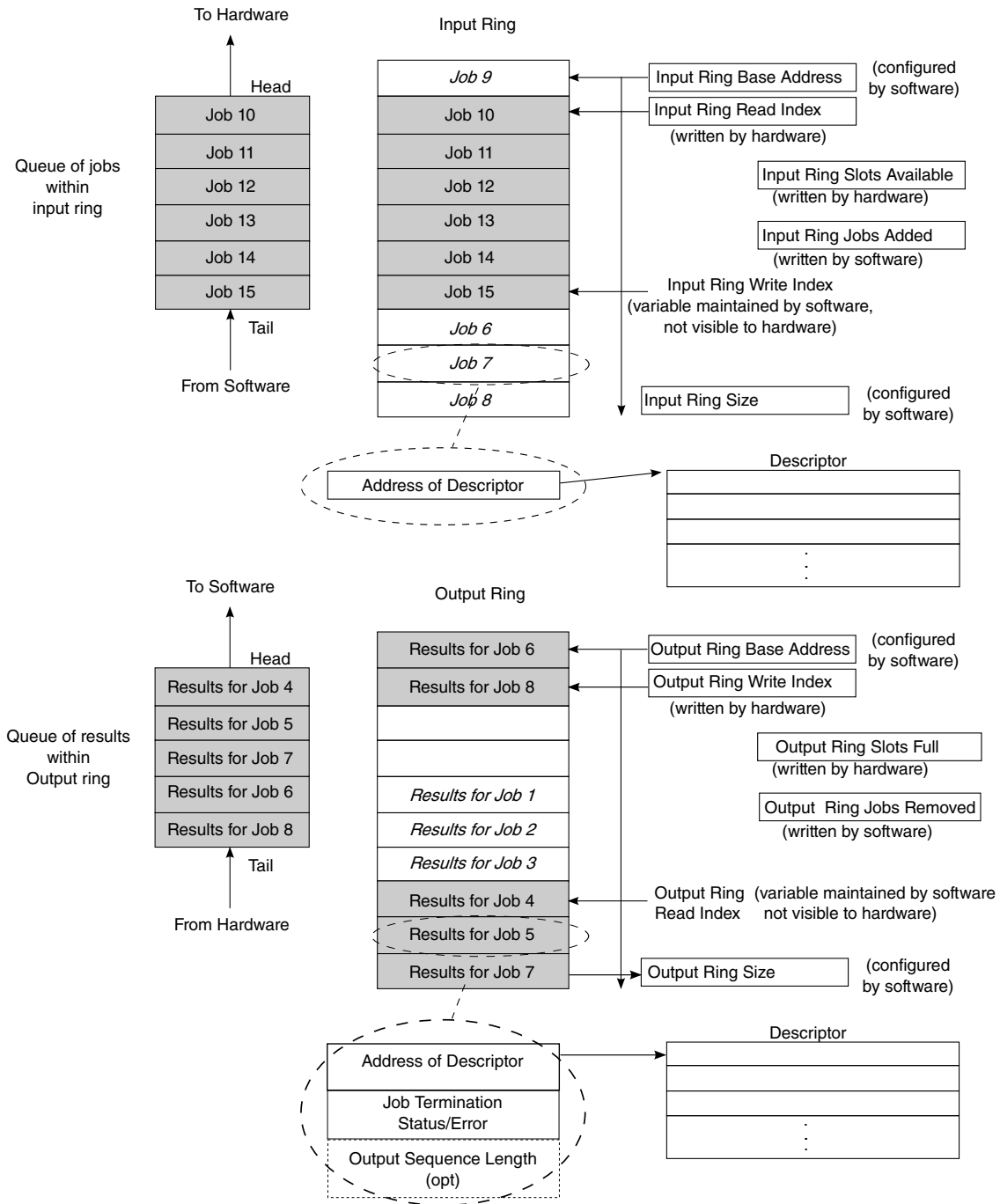


Figure 5-3. Input and output ring example

5.3.1.2 Managing the input rings

For the input ring, software is the producer, meaning that software

- Writes descriptor addresses into the job descriptor queue within the input ring.
- Writes the number of new jobs to the Input Ring Jobs Added Register

The address added to the input ring must point directly to the start of a job descriptor, not to a scatter/gather table. A job descriptor queue entry is determined by the size of a pointer, as specified in the PS field of the MCFGR Register (one word for 32-bit pointers, two words for 49-bit pointers). (See [Address pointers](#).) Software maintains its own write pointer for the input ring, and SEC does not have direct access to that pointer.

For the input ring, SEC is the consumer, meaning that it increments the Input Ring Slots Available Register upon pulling descriptor addresses out of the queue. When software writes a new value to the Input Ring Jobs Added Register, SEC decrements the Input Ring Slots Available Register by the value that was written by software. SEC maintains a read index that it increments as it reads jobs from the input ring.

5.3.1.3 Managing the output rings

For the output ring, the roles are reversed from the input ring. SEC is the producer and software is the consumer.

- When SEC adds completed jobs to the output ring within the Job Ring, it increments the Output Ring Slots Full Register, which tells software how many results are available for software to retrieve. An interrupt may or may not be generated, depending upon the Job Ring configuration at the time (for more details, see [Asserting Job Ring interrupts](#)).
- When software removes jobs from the output ring for processing, it writes the number of jobs removed to the Output Ring Jobs Removed Register. SEC decrements the output ring slots full value by the new value that software wrote to the Output Ring Jobs Removed Register.

Note that each entry in the output ring consists of a job descriptor address and a job termination status word. See [Job termination status/error codes](#) for the format of this status word. Therefore, the size of an entry in the output ring is the size of a pointer plus one word for status, plus an optional word containing the output sequence length. SEC maintains an output ring write index that it increments as it places completed jobs and status into the output ring. Software can read this register to determine the current tail of the output ring.

Note that it is possible for a bus error to occur when the job queue controller is writing the completion status to the output ring. This results in an error code type 1 indication for that particular Job Ring. The correct response to any Job Ring error code 1 indication is to perform a Job Ring reset (See Job Ring Command Register RESET field), a software SEC reset or a power on reset. If a Job Ring reset is performed, it will clear all registers

for that particular Job Ring except the REIR, IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers. The REIR registers should be manually reset after a Job Ring reset. The IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers can be reprogrammed or not, as appropriate, after a Job Ring reset.

5.3.1.4 Controlling access to Job Rings

Access to a Job Ring can be restricted to a particular software entity because each Job Ring's registers are in a separate register address page. An OS or a hypervisor can enforce the restrictions by means of a memory management unit.

A process with permission to access a particular Job Ring's registers can:

- Schedule jobs for SEC by writing the address of a job descriptor or trusted descriptor at the tail of the queue within the input ring.
- Retrieve job completion status by reading the entry at the head of the queue within the corresponding output ring.

Each Job Ring can be configured so that SEC's DMA asserts different ICID values when executing jobs on behalf of that Job Ring. This allows slave devices or chip-level memory management units to make memory access control decisions based upon the Job Ring from which the job was initiated.

5.3.1.5 Order of job completion

Job descriptors submitted through different Job Rings are not guaranteed to complete in the order they were submitted by software, even if they reference the same shared descriptor and use SERIAL or WAIT sharing. (See [Shared descriptors](#) for details about shared descriptors and sharing concepts).

As shown in the example in [Figure 5-3](#), it is possible for results to be written into the output ring in a different order than the order in which the corresponding jobs appear in the input ring (see jobs 6 and 7, where 7 is a short job submitted after 6, which requires more processing time). Because jobs are assigned to DECOs as the DECOs become available, successive job descriptors generally execute in parallel in different DECOs. Therefore, the order of job completion is affected by the time required to process the data, the presence of a shared descriptor, and the sharing mode.

The only way to guarantee that jobs input on a single Job Ring complete in the order they were added to the input ring is to both:

- Have those jobs reference a shared descriptor
- Set the SERIAL sharing bits in the descriptor headers.

Note that the majority of the processing information can be included in the job descriptor with the shared descriptor enforcing serialized processing.

WAIT sharing can be used if all that is required is to guarantee that certain commands in one job are complete before another job is started. Types of sharing and impact on job completion ordering are further described in [Specifying different types of shared descriptor sharing](#).

5.3.1.6 Initializing Job Rings

Minimal configuration for SEC operation using the Job Ring interface requires initializing at least one Job Ring by specifying the base addresses for the input ring and output ring and the size of these rings (see the Input Ring Base Address Register (IRBAR), the Output Ring Base Address Register (ORBAR), the Input Ring Size Register (IRSR), and the Output Ring Size Register (ORSR)). Most cases (with the possible exception of debugging with test data in use) also require specifying the ICID, PL and BMT values associated with the Job Ring (see the JRaICID register). These values should be configured by a trusted SoC / ring management entity. The job Rings can also be configured for whether to include the optional sequence out length word in the Output Ring entries (see Job Ring Configuration Register (JRCONFGR)).

5.3.1.7 Job Ring Registers

If the Job Ring is allocated to TrustZone SecureWorld, the Job Ring registers associated with this ring can be written only via a TrustZone secure bus transaction. Nonsecure writes to Job Ring registers owned by Trustzone SecureWorld will be ignored.

When virtualization is enabled (VIRT_EN=1 in the Security Configuration register), the Job Ring registers in pages 1..4 can be written only if the corresponding Job Ring has been "started", that is, the JRSTARTR[Start_JR] bit for that Job Ring is 1. Conversely, the Job Ring configuration registers in page 0 (for example, the JRaICID register) can be written only if the Start_JR bit for that Job Ring is 0. The Job Ring registers are reset when the Job Ring is stopped and virtualization is enabled, in order to prepare the Job Ring for a new owner. The input ring slots available, input ring read index, output ring slots full and output ring write index registers are read-only when virtualization is disabled. These registers are writable when virtualization is enabled.

For the Job Ring register descriptions, see IRBAR and the following register descriptions.

5.3.1.8 Asserting Job Ring interrupts

To notify the driver software that job results are available from the Job Ring, each Job Ring interface asserts an interrupt request on a shared interrupt request line. Note that the software context switch overhead could have a severe performance impact if interrupts were asserted for every job completion. Therefore, SEC supports a configurable interface that allows the driver to specify how full the output ring can be before SEC generates a job completion interrupt request. To prevent any job from waiting too long for software completion processing, the driver can also specify a time out value. This value allows SEC to generate an interrupt if job results are available and too much time has elapsed since software last removed any completed jobs from the output ring. These values are programmed via fields described in the Job Ring Configuration Register (JRCFGR).

The Job Ring interrupt does not clear automatically when jobs are removed from the output ring. Software must clear the interrupts by writing to the Job Ring Interrupt Status Register. Note that one or more additional jobs can complete while software is clearing the interrupt. Depending on the interrupt coalescing settings, an additional interrupt may immediately be generated for these newly completed jobs.

5.3.2 Queue Manager Interface (QI)

QI is a high performance, flow-oriented job execution interface used in devices implementing the Data Path Acceleration Architecture (DPAA). Once initialized and enabled, QI performs the following functions:

- Request and receive (dequeue) frame descriptors from QMan for processing
- Acquire buffers from the Buffer Manager (BMan) for output data (optional)
- Prepare data for processing by assembling a job descriptor
- Pass the job descriptor to the job queue controller for job scheduling
- Receive job completion status from a descriptor controller (DECO)
- Release input frame buffers (optional)
- Send (enqueue) frame descriptors with processing results to QMan

In contrast to the Job Ring interface, which requires the dedication of a Job Ring to a relatively small group of select users, all users can be given access to a shared SEC by providing services via DPAA frame descriptors exchanged with SEC in user-dedicated queues. Frame queues are initialized with a user identity and resource access attributes authorizing access to the user's resources. The frame descriptor provides a pointer to a SEC Flow Context (FLC) typically followed by an optional but usually present shared descriptor that may supply session keys, context, and other processing instructions. Every

frame descriptor enqueued to SEC is processed according to the information in the FLC and the shared descriptor. For more information, see [Flow Context](#) and [Shared descriptors](#).

From an application software perspective, mapping a security processing flow to a frame queue is an infrequent occurrence, corresponding to establishment or refresh of a security session, e.g. an Internet Key Exchange (IKE) establishing an IPsec tunnel. The enqueueing of frame descriptors to the right frame queue, SEC's dequeue and processing of the data associated with those frame descriptors, and SEC's return of the frame descriptors to software corresponds to a data path.

Refer to the DPAA overview chapter of your SoC reference manual for a comprehensive explanation of frame queues and frame descriptors. Refer to the DPAA reference manual for your SoC to see a detailed description of Queue Manager and Buffer Manager. For an abbreviated discussion of these topics relevant to SEC, see [Frame queues, frame descriptors, and buffers](#).

QI is usually configured and enabled during system initialization, but can be stopped, resumed, and (re-)configured later. The specific mandatory configuration steps are described in [Initializing the Queue Manager Interface](#).

5.3.2.1 Requesting and receiving frame descriptors from QMan

In response to each dequeue command, QMan sends QI one to three frame descriptors from a selected frame queue. The number of requested frame descriptors is determined by QI's configuration (see the Queue Interface Dequeue Configuration Register (QIDQC) for details). However, it is the QMan programming that controls the choice of frame queues and frame descriptors in response to the dequeue request, not the configuration of the QI.

The QI uses a QMan mechanism called subportals to ensure that QMan selects frame descriptors from different frame queues. Each dequeue command specifies a subportal ID. When issuing these commands QI uses as many subportal IDs as the number of job queue holding tanks. This allows SEC to process frames from more than one queue simultaneously, increasing opportunities for parallel processing. If jobs from a few frame queues were to fill up all available job slots QI might not be able to fill all the holding tanks in the job queue controller, possibly resulting in idle DECO resources and decreased throughput. To avoid this condition QI limits the number of outstanding dequeue requests for any subportal to one, and it does not issue a request for a subportal if the number of jobs from the subportal equals or exceeds the threshold defined by the SPFCNT field in the QIDQC register.

In response to each dequeue request, QMan sends QI one to three frame descriptors and summary information applying to all dequeued frames:

- Number of Frames dequeued
- Dequeue response (input) frame queue ID (IFQID)
- Dequeue response Bypass DPAA Resource Isolation (BDI)
- Dequeue response Isolation Context ID (ICID)
- Dequeue response Privileged Level (PL)
- Dequeue response Frame Queue Descriptor Context Format (FF)
- Dequeue response Frame Queue Descriptor Context (FQD_CTX)
 - Enqueue response (output) frame queue ID (RFQID) - conditionally used
 - Access Management Qualifier (AMQ) Usage Control (AUC)
 - Isolation Context ID 2 (ICID_2) - optional
 - Isolation Context ID 3 (ICID_3) - optional
 - Privileged Level 2 (PL_2) - optional
 - Privileged Level 3 (PL_3) - optional
 - Security Domain ID (SDID) - optional

The optional and conditional information elements are interpreted and utilized dependent on the settings of the FF and AUC.

1. If AUC = 0:

- Input ICID = ICID, input PL = PL
- Control ICID = ICID, control PL = PL
- Output ICID = ICID, output PL = PL
- If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
- Response (error) / output FQID = RFQID
- All unreferenced optional parameters are not utilized

2. If AUC = 1:

- Input ICID = ICID, input PL = PL
- Control ICID = ICID, control PL = PL
- Output ICID = ICID_2, output PL = PL_2
- If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
- Response (error) FQID = RFQID, Output FQID = FLC.OFQID
- All unreferenced optional parameters are not utilized

3. If AUC = 2:

- Input ICID = ICID, input PL = PL
- Control ICID = ICID_2, control PL = PL_2
- Output ICID = ICID_2, output PL = PL_2
- If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
- Response (error) FQID = RFQID, Output FQID = FLC.OFQID
- All unreferenced optional parameters are not utilized

4. If AUC = 3:

- Input ICID = ICID, input PL = PL
- Control ICID = ICID_2, control PL = PL_2
- Output ICID = ICID_3, output PL = PL_3
- If (FF == 0) SDID = ICID, else (FF == 1) SDID = FLC.SDID
- Response (error) FQID = RFQID, Output FQID = FLC.OFQID

5.3.2.2 Building job descriptors for QI jobs

QI uses the QMan dequeue summary information and the frame descriptors to build an internal job descriptor for each job. The summary information is primarily used to direct responses and provide isolation and bus access attributes to input and output frame buffers as well as frame processing control and context data structures. For most purposes, SEC hardware does not distinguish between job descriptors created by software and submitted to SEC through the Job Ring interface and job descriptors created by QI using dequeue information. The term Job Descriptor (JD) is used generically when describing processing instructions or embedded data utilized by SEC to process jobs.

Information for each job, including the input and output frame addresses, is stored in job buffers internal to QI. Job buffers are used to save all data needed to build a job descriptor, maintain processing status, track job order, and eventually return (enqueue) output to QMan.

On every cycle QI performs the following:

1. Searches its list of jobs waiting for transfer to the job queue controller
2. Determines each job's eligibility for transfer
3. Assigns selection priorities based on several criteria
4. Selects the next job, if any, for transfer

Priorities are assigned to eligible jobs to maximize sharing and throughput. QI builds the job descriptor for the selected job and notifies the job queue controller that the job is ready for transfer into an available holding tank.

5.3.2.3 Controlling QI access to frame queues and data

The use of frame queues in DPAA allows a large number of user software processes to share SEC. Users are prevented from directly accessing each other's private memory space through proper configuration of the CPU MMUs. When processing a job on behalf of a user process, SEC is prevented from accessing the private memory of other processes

by means of the System Memory Management Unit (SMMU) and one or more ICID values assigned to each job. A general description of ICID usage in DPAA can be found in the DPAA overview chapter of your SoC reference manual.

The dequeue response summary information associated with a frame descriptor dequeued from QMan specifies up to 3 sets of ICIDs and associated access attributes (PL and BMT) as well as a usage control code (AUC). Depending on the AUC value in the summary information, one to three ICID and attributes value sets may be used to access either input, output, or control data (or multiple data types). Control data includes the FLC, an optional, but usually present SD, and an optional CRJD. When QI reads the FLC, it uses the control ICID and its associated access attributes. If the frame format is a frame list, QI reads and writes the frame list table using the input ICID and its associated access attributes. For accesses to the output frame, QI uses the output ICID and its associated access attributes. The three ICIDs and access attribute sets are passed with the job from QI to the job queue controller and DECO so that the appropriate ICIDs and attributes can be used by those processors for their memory accesses.

For information on how to configure ICIDs, access attributes, and AUC values passed to QI in the dequeue summary information and frame descriptor, see the QMan chapter in the DPAA2 reference manual.

5.3.2.4 Tracking the completion order of QI jobs

QI uses ordered lists to track the dequeue order of jobs with the same FLC address and associated memory access attributes. When job processing is done, QI uses the lists to enqueue jobs with the same FLC address in the same order they were dequeued.

When a DECO notifies the QI that a job has finished, QI checks the appropriate list to determine whether the job is the oldest job with that FLC address. If so, QI enqueues the job's frame descriptor. Otherwise the enqueue is delayed to maintain order. Under normal operating conditions (except system error or invalid queue IDs) QMan will not reject SEC enqueues.

See the Queue Manager section of the reference manual for a discussion of the hardwired settings of the SEC Direct Connect portal (DCP). QMan must be configured by software to select an error queue to store frames of SEC's rejected enqueues.

5.3.2.5 Initializing the Queue Manager Interface

Configuring and enabling QI requires initializing:

- Queue Interface Dequeue Configuration Register (optional)

- Queue Interface Configuration Register (optional)
- Queue Interface Control Register (at a minimum enable dequeues)

Additional QI registers are accessible through the SEC register interface to assist with debugging.

Note that in addition to the SEC QI configuration, at least two frame queues must be initialized in QMan so that software can enqueue frame descriptors to SEC, and SEC can enqueue results back to software.

5.3.2.6 Done/error notification for QI jobs

There are no user-observable SEC interrupts associated with QI. QI is utilized through QMan portals and queues. For all jobs handled by QI, completion and error status is indicated in the 32-bit FRC field of the response frame descriptor SEC sends to QMan. The most-significant four bits of this field identify where the error was detected within SEC. For example, the value '5' indicates that QI detected the error and '4' indicates that DECO detected the error. The remaining bits provide more specific status details. See [Table 5-3](#) for a complete list and formatting of error codes. If the RSIDE bit in the QI Configuration Register (QICFG) is set and the job completion status is non-zero, the 32-bit code is reformatted to include a SEC identifier. A description of this alternate format and an explanation for its use is provided in [Job termination status/error codes](#).

While there are no user-observable QI interrupts, QI reports a small set of potentially severe, but recoverable errors via interrupts to the device Management Complex (MC). The reporting of such errors is intended to support the detection of system instability or malicious users. The following error conditions are reported and detailed error information is latched for debug and management consumption in the QI Recoverable Error Interrupt Registers (REIRxQI):

- Memory access errors with specific transaction details, including address, attributes, and SEC internal block details
- SEC-detected, ICID-related, FLC access violations for multi-ICID use cases
- Misdirected frame errors where the ERR field of the dequeued frame descriptor is non-0.

For more information on QI error handling associated with frame queue and frame descriptor operations, see [Frame descriptor error handling](#).

During job processing, SEC produces job-specific annotation that QI will write to the ASA section of the output frame if enabled by EAO=1 in the Flow Context. This annotation is comprised of a 2-byte checksum and a 4-byte encrypted/decrypted byte count. The byte count is written to the least significant 4 bytes of a 64-byte ASA block and the checksum to the next 2 bytes. See [Flow Context](#) for an explanation of annotation

output controls. Note, for testing purposes the Watch Dog Fast (WDF) bit in the Master Configuration Register (MCFGR) may be set, which will force the encrypted/decrypted byte count to increment by 64KB instead of 1.

5.3.3 Advanced IO Processor Interface (AI)

AI is a specialized low latency, pseudo-synchronous job execution interface found in devices implementing the Data Path Acceleration Architecture 2 (DPAA2). AI is a dedicated interface utilized by AIOP tasks via the AIOP Accelerator Proxy (AAP) interface. AAP provides QMan DCP-like frame descriptor transfers between AIOP and SEC while suspending/resuming the AIOP tasks requesting SEC service (in a pseudo-synchronous fashion when viewed from an AIOP task perspective). Once initialized and enabled, AI performs the following functions:

- Receiving frame descriptors from AIOP for processing
- Acquiring buffers from the Buffer Manager (BMan) for output (optional)
- Preparing data for processing by assembling a job descriptor
- Passing the job descriptor to the job queue controller for job scheduling
- Receiving job completion status from a descriptor controller (DECO)
- Releasing input frame buffers (optional)
- Sending frame descriptors with processing results to AIOP

In contrast to SEC's QMan Interface (QI), AI-based SEC service is made available to only a relatively small number of AIOP tasks, but can be indirectly used (via AIOP services) by all users given access to SEC-based AIOP services. Similar to SEC use with QI, AIOP tasks pass a frame descriptor and data representing the equivalent information passed as dequeue summary information by QMan. But contrary to QI where this information is delivered by QMan, AIOP software is in complete control to either use information delivered to it by QMan or generate such information on demand to provide AIOP services relying on SEC services.

The AI SEC interface diverges from the QI SEC interface in only three aspects:

- Instead of receiving a response frame queue in the summary dequeue information, the AIOP AAP interface transfers an AIOP task number and frame storage pointer to SEC. This will be returned to AAP together with the result frame descriptor.
- AI does not return a response or output FLC in the result frame descriptor. Instead, SEC optionally returns additional SEC results in the frame descriptor FLC field and also returns an AIOP task number and frame storage pointer, both of which were transferred with the original service request from AIOP's AAP interface.
- In addition to passing a frame descriptor and summary dequeue information, SEC may optionally receive FD-specific optional storage profile data. This description

overrides the storage profile description present in the SEC FLC and referred to by the passed frame descriptor.

Refer to the DPAA overview chapter of your SoC reference manual for a comprehensive explanation of frame queues and frame descriptors. Refer to the DPAA2 reference manual for a detailed description of the Advanced IO Processor, Queue Manager, and Buffer Manager. For an abbreviated discussion of these topics relevant to SEC, see [Frame queues, frame descriptors, and buffers](#).

AI is usually configured and enabled during system initialization, but can be stopped, resumed, and (re-)configured later. The specific mandatory configuration steps are described in [Initializing the AIOP interface](#).

5.3.3.1 Receiving frame descriptors from AIOP

If enabled, AI will accept a job from AIOP's AAP SEC interface whenever it has room for processing another AIOP job. If AAP receives or already has a pending SEC service request from an AIOP task, AAP will transfer a frame descriptor and dequeue summary information from the AIOP task internal work space. AAP may also transfer (if requested by the AIOP task) a storage profile overriding the storage profile contained in the Flow Context specified by the FLC field in the transferred frame descriptor.

With each job transfer, AAP sends AI a frame descriptor and the following additional information:

- Transfer request Bypass DPAA Resource Isolation (BDI)
- Transfer request Isolation Context ID (ICID)
- Transfer request Privileged Level (PL)
- Transfer request Frame Queue Descriptor Context Format (FF)
- Transfer request Frame Queue Descriptor Context (FQD_CTX)
 - Transfer request AIOP task number
 - Transfer request AIOP response frame storage pointer
 - Access Management Qualifier (AMQ) Usage Control (AUC)
 - Isolation Context ID 2 (ICID_2) - optional
 - Isolation Context ID 3 (ICID_3) - optional
 - Privileged Level 2 (PL_2) - optional
 - Privileged Level 3 (PL_3) - optional
 - Security Domain ID (SDID) - optional
- Frame-specific storage profile (FSSP) - optional

The optional and conditional information elements are interpreted and utilized dependent on the settings of the FF and AUC.

1. If AUC = 0:

- Input ICID = ICID, input PL = PL
 - Control ICID = ICID, control PL = PL
 - Output ICID = ICID, output PL = PL
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - If (FSSP passed) FLC.SP = FSSP
 - All unreferenced optional parameters are not utilized
2. If AUC = 1:
- Input ICID = ICID, input PL = PL
 - Control ICID = ICID, control PL = PL
 - Output ICID = ICID_2, output PL = PL_2
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - If (FSSP passed) FLC.SP = FSSP
 - All unreferenced optional parameters are not utilized
3. If AUC = 2:
- Input ICID = ICID, input PL = PL
 - Control ICID = ICID_2, control PL = PL_2
 - Output ICID = ICID_2, output PL = PL_2
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - If (FSSP passed) FLC.SP = FSSP
 - All unreferenced optional parameters are not utilized
4. If AUC = 3:
- Input ICID = ICID, input PL = PL
 - Control ICID = ICID_2, control PL = PL_2
 - Output ICID = ICID_3, output PL = PL_3
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = FLC.SDID
 - If (FSSP passed) FLC.SP = FSSP

5.3.3.2 Building job descriptors for AI jobs

AI uses the AIOP-dequeued or AIOP-software-generated summary information and the frame descriptors to build an internal job descriptor for each job. The summary information is primarily used to provide isolation and bus access attributes for input and output frame buffers as well as frame processing control and context data structures. For most purposes, SEC hardware does not distinguish between job descriptors created by software and submitted to SEC through the Job Ring interface and job descriptors created by AI using dequeue information. The term Job Descriptor (JD) is used generically when describing processing instructions or embedded data utilized by SEC to process jobs.

Information for each job, including the input and output frame addresses, is stored in job buffers internal to AI. Job buffers are used to save all data needed to build a job descriptor, maintain processing status, track job order, and eventually return (enqueue) output to AIOP.

On every cycle AI performs the following:

1. Searches its list of jobs waiting for transfer to the job queue controller
2. Determines each job's eligibility for transfer
3. Assigns selection priorities based on several criteria
4. Selects the next job, if any, for transfer

Priorities are assigned to eligible jobs to maximize sharing and throughput. AI builds the job descriptor for the selected job and notifies the job queue controller that the job is ready for transfer into an available holding tank.

5.3.3.3 Controlling AI access to data

When SEC services are utilized by AIOP the DPAA access controls made available to AIOP tasks via QMan de-queues can be either changed or passed in unmodified form to SEC to address specific AIOP and SEC service requirements. The dequeue response summary information received by AI specifies the ICIDs and related access attributes (PL and BMT) to use for input, output, and control data access. Depending on the value of AUC in the summary information, the ICIDs and attributes may be the same or different. Control data includes the FLC, the optional, but usually present SD, and the optional CRJD. When AI reads the FLC, it uses the control ICID and its associated access attributes. If the frame format is frame list, AI reads and writes the frame list table using the input ICID and its associated access attributes. For accesses to the output frame, AI uses the output ICID and its associated access attributes. The three ICIDs and attributes follow the job from AI to the job queue controller and DECO so that the appropriate ICIDs and attributes can be used by those processors for memory accesses.

5.3.3.4 Tracking the order of completion of AI jobs

AI uses ordered lists to track the order in which jobs with the same FLC address and associated memory access attributes are received from AIOP. AI can be configured to return completed jobs immediately, regardless of the order in which they were received, or to return completed jobs with the same FLC address and associated attributes in the same order they were received. Order preservation is enabled by setting the EOP bit in the AI Configuration Register (AICFG). If enabled, when DECO notifies AI that a job has finished, AI checks the appropriate FLC list to determine whether the job is the oldest

in that list. If it is, AI transfers the job to AIOP using a frame descriptor. Otherwise, the transfer is delayed until older jobs in that list have been transferred. The ordered lists are also used to support job management functions such as flush requests.

5.3.3.5 Initializing the AIOP interface

Use of AI requires, at a minimum, writing the AI Control Register to enable dequeues. Additional AI registers, accessible through the SEC register interface, are available for configuration, control and debug.

To meet the low-latency requirements of AIOP, software may reserve job processing resources, holding tanks and DECOs, for AIOP jobs. The SEC Adaptive Resource Reservation Register is used to enable this feature and to specify how many holding tanks and DECOs should be reserved.

5.3.3.6 Done/error notification for AI jobs

There are no user-observable SEC interrupts associated with AI. The SEC AI is accessed and utilized through AIOP's AAP interface. For all jobs handled by AI, completion and error status is indicated in the 32-bit FRC field of the response frame descriptor AI sends to AIOP. The most-significant four bits of this field identify where the error was detected within SEC. For example, the value '1' indicates that AI detected the error and '4' indicates that DECO detected the error. The remaining bits provide more specific status details. See [Table 5-3](#) for a complete list and formatting of error codes. If the RSIDE bit in the AI Configuration Register (AICFG) is set and the job completion status is non-zero, the 32-bit code is reformatted to include a SEC identifier. A description of this alternate format and an explanation for its use is provided in [Job termination status/error codes](#).

While there are no user-observable AI interrupts, AI reports a few potentially severe, but recoverable errors via interrupts to the device Management Complex (MC). The reporting of such errors is intended to support the detection of system instability or malicious users. The following error conditions are reported and detailed error information is latched for debug and management consumption in the AI Recoverable Error Interrupt Registers (REIRxAI):

- Memory access errors with specific transaction details, including address, attributes, and SEC internal block details
- SEC-detected, ICID-related, FLC access violations for multi-ICID use cases

For more information on AI error handling associated with frame descriptor operations, see [Frame descriptor error handling](#).

During job processing, SEC produces job-specific annotation that AI returns in the FLC field of the response frame descriptor it sends to AIOP. AI will also write the annotation to the ASA section of the output frame if enabled by EAO=1 in the Flow Context. This annotation is comprised of a 2-byte checksum and a 4-byte encrypted/decrypted byte count. The byte count is returned in the least significant 4 bytes of the frame descriptor FLC field and the checksum in the next 2 bytes. When writing the annotation to the output frame, the byte count is written to the least significant 4 bytes of a 64-byte ASA block and the checksum to the next 2 bytes. See [Flow Context](#) for an explanation of annotation output controls. Note, for testing purposes the Watch Dog Fast (WDF) bit in the Master Configuration Register (MCFGR) may be set, which will force the encrypted/decrypted byte count to increment by 64KB instead of 1.

5.3.4 Register-based service interface

It is possible to use the register interface to perform entire cryptographic operations. For the purposes of debugging descriptors, it is possible to execute descriptors one descriptor at a time, or even one descriptor command at a time.² This method bypasses all job scheduling performed by the job queue controller. Software can perform CHA operations by writing and reading registers in the CCB directly, without using a Job Ring or the QI or the AI to run descriptors. When descriptors or commands are executed in this mode, software can examine the content of most DECO and CCB registers after each descriptor or descriptor command completes. This can assist with debugging hardware and descriptor programs.

To execute descriptors or commands in this fashion, software must request direct use of a DECO by writing into the DECO REQ register.³ But before requesting a DECO, software must specify the SDID and ICID, PL and BMT values that will be used when executing descriptors under direct software control. When virtualization is disabled these values are specified via the DECO ICID Register for the selected DECO. When virtualization is enabled, the DECO Request Source register is used to select a particular Job Ring to act as the source for the ICID, PL and BMT and SDID values. The DECO Request Source register must be written prior to writing the RQDn bit in the DECO REQ register.

To use the register-based job service interface, the DECO must be programmed in proper order so that a descriptor runs correctly. The steps are:

1. Specify the ICID, PL and BMT and SDID values using the DECO Request Source register or appropriate DECO ICID Register

2. Note that trusted descriptors cannot be executed via the register-based service interface.

3. Note that the ability to control DECOs through the register-based service interface is disabled when adaptive resource reservation is enabled.

2. Set the RQDn bit in the DECO Request Register. This RQDn bit must remain asserted during the entire time that software wants to access that DECO/CCB block directly. This indicates to the job queue controller that it should not assign any jobs to the requested DECO block. After the job queue controller sees the RQDn bit set to 1, it waits for the corresponding DECO block to complete any pending tasks.
3. Wait for the DENn bit in the DECO Request Register to be set to 1. The job queue controller sets the DENn bit to 1 when the DECO block becomes available. When this bit is set, software can use the DECO/CCB block by submitting descriptors by means of SEC's register interface.
4. Write at least the first burst of a descriptor into the descriptor buffer. If there is a shared descriptor, offset the descriptor into the descriptor buffer by the length of the shared descriptor.
5. Write the address of the descriptor into the DECO Descriptor Address Register so DECO knows where to find the descriptor. This is only required if the WHL bit (see the next step) is not set or if the descriptor attempts to do a STORE of type 41h or 45h to write back part, or all, of the descriptor to memory.
6. Write the Job Queue Control Register. If fewer than 4 words are in the first burst, the FOUR bit must be 0. If the entire descriptor has already been loaded, set the WHL bit. If the WHL bit in the DECO Job Queue Control Register is not set, DECO attempts to fetch the rest of the descriptor from memory regardless of whether portions of the descriptor beyond the first burst were already written to the descriptor buffer. SHR_FROM is not used in this format and will not be checked.
7. Wait until the DECO is done. To determine whether DECO is done, read the VALID and DECO_STATE fields in the DECOa DBG_DBG_REG register. While the job is running, VALID will be 1 and DECO state will change values as the descriptor is processed. If DECO_STATE is Dh, then an error occurred. Read other fields and registers to determine the cause of the error. Note that VALID will likely remain asserted in the event of an error. If DECO_STATE is 0h and VALID is 0, then the job finished normally.
8. Read registers of interest.
9. Done or start over.
10. When software is finished using the DECO/CCB block, it must clear the RQDn bit so that the DECO is available to the job queue controller for normal processing. The job queue controller then de-asserts the DENn bit, which resets the DECO and CCB.

Note that there are restrictions imposed when executing a descriptor under software control:

- The special cryptographic keys used to encrypt or decrypt Black Keys are not available, so Black Keys cannot be used.
- The master cryptographic key used to encrypt or decrypt Blobs is not available, so Blobs cannot be used.

- Sharing of Shared Descriptors is not permitted.
- Trusted Descriptors are not allowed.
- When virtualization is enabled, a Job Ring source must be selected in the DECO Request Source Register before executing any job under direct software control. All jobs running under direct software control will then utilize the ICID and SDID values for the Job Ring selected in the DECO Request Source Register. When virtualization is disabled, any job under direct software control will utilize the ICID and SDID values specified in the DECO ICID register, and the SRC field in the Job Queue Control Register must be programmed to indicate the job is running on behalf of one of the Job Rings. Jobs cannot be executed under direct software control if those jobs appear to be from other possible SEC sources, such as:
 - AI
 - QI
 - RTIC

The normal use case for the register based service interface is to debug descriptors. When such a descriptor is run through the interface and the descriptor encounters an error, once analysis of the error is done, the only way to recover is to release the DECO. The user can recover by releasing the DECO or by writing a 1 to the STEP bit in the DECO Job Queue Control Register. The second method allows another descriptor to be loaded and run as described above.

5.4 Job scheduling

The job queue controller is the job scheduler within SEC. The default job scheduling algorithm is used when AI adaptive resource reservation is disabled (AR_EN=0 in the Adaptive Resource Reservation Register). The default job scheduling algorithm operates as follows. The job queue controller pulls jobs to be sent to the holding tanks in round-robin fashion from the Job Rings, then from QI, then from AI, and then from RTIC. When AI adaptive resource reservation is enabled, a specified number of holding tanks and DECOs are reserved for AI jobs. This number is programmed in the Adaptive Resource Reservation Register. The job queue controller will prioritize AI jobs over Job Ring jobs, QI jobs and RTIC jobs to keep those reserved holding tanks filled with AI jobs. When those reserved holding tanks are full or when there are no new AI jobs, then the Job Ring, QI and RTIC jobs will be processed in round-robin fashion to fill the unreserved holding tanks.

5.4.1 Job scheduling - default algorithm

When AI adaptive resource reservation is disabled the job queue controller uses the default job scheduling algorithm. Each time that the Job Ring's turn comes up in rotation and there is a job available in that Job Ring's input queue, a job is selected from that Job Ring. But because SEC buffers input ring entries for efficiency, several jobs may be scheduled from one Job Ring before a job is scheduled from the next Job Ring. Eventually all Job Rings will be serviced.

Each time that the QI's turn comes up in rotation and there is a QI job that can be processed, one QI job will be selected for transfer to JQ. QI selects that job from its list of jobs waiting for transfer using several criteria to determine eligibility and priority. QI builds its list of jobs eligible for transfer by dequeuing a number of jobs from one subportal and then switching to a different subportal for the next dequeue. This prevents starving any subportal.

This selection method favors jobs with the potential for sharing. The priority of a QI job will be reduced if the job uses CRID to specify that it requires a critical resource that cannot be used by all DECOs in parallel and all of those resources are already in use, unless there is already a job from this flow in a DECO (see [Flow Context](#)). A selection priority is calculated for each job waiting for transfer to the job queue controller, in decreasing order of priority:

1. Job is not the first job of a dequeue and another job from the same flow is currently in a DECO.
2. Job is first of a dequeue and another job from the same flow is currently in a DECO.
3. No job from the same flow is currently in a DECO and either CRID is not set or the critical resource is available.
4. No job from the same flow is currently in a DECO, CRID is set, and the critical resource is not available.

Each time that the AI's turn comes up in rotation and there is an AI job that can be processed, one AI job will be selected. Like QI, selection favors jobs with the potential for sharing. CRID can be used to modify the priority for AI jobs in the same way as described for QI jobs. A selection priority is calculated for each job waiting for transfer to a job queue as follows, in decreasing order of priority:

1. Another job from the same flow is currently in a DECO.
2. No job from the same flow is currently in a DECO and either CRID is not set or the critical resource is available.
3. No job from the same flow is currently in a DECO, CRID is set, and the critical resource is not available.

Note that RTIC requests at most one job at a time.

The following figure illustrates the algorithm for selecting a job for an available holding tank.

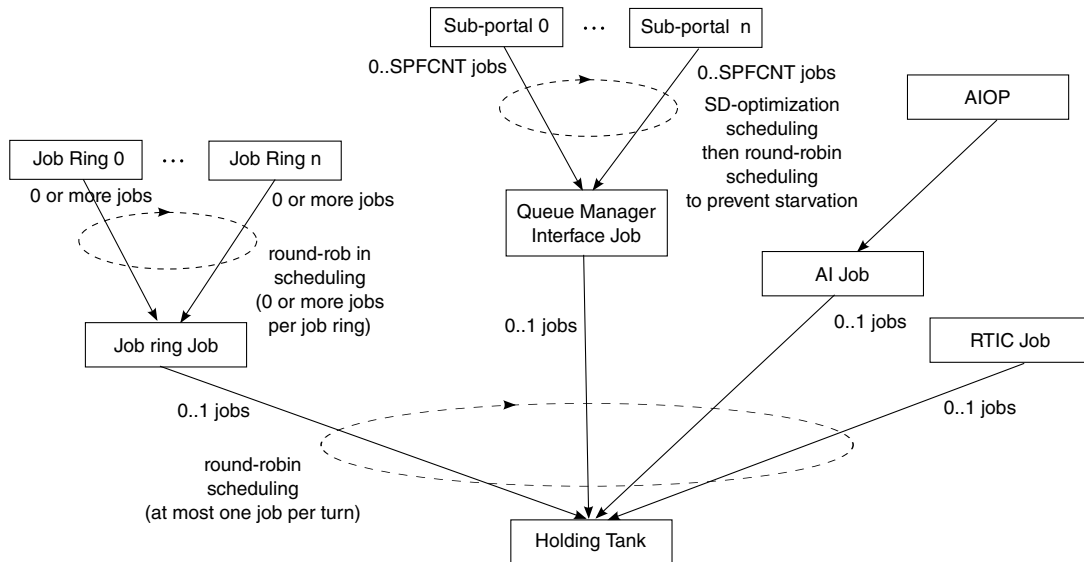


Figure 5-4. Selecting job for available holding tank when adaptive resource reservation is disabled

The job queue controller prefetches some or all of the selected job descriptor and possibly the shared descriptor (if any, and it is not already in a DECO) and places them in a buffer referred to as a holding tank. After a job has been put in a holding tank, it is then eligible for dispatching to a DECO. If the job source is the QI or AI the holding tank also fetches up to two bursts' worth of input frame data (up to the second burst boundary). SEC generally implements the same number of holding tanks as DECOs.

This prefetching of the job descriptor allows the job queue controller to take shared descriptors into consideration when allocating jobs to DECOs. SEC attempts to dispatch jobs to available DECOs as efficiently as possible. If (1) a job descriptor with a shared descriptor is currently executing in DECO n, (2) this descriptor can be shared, and (3) there is another job descriptor in a holding tank that references the same shared descriptor, the job descriptor in the holding tank is marked as pending for DECO n.

The following figure illustrates the SEC's dispatching algorithm, which favors reuse of already fetched descriptors but avoids starving Job Rings or QI queues or AI. Note that when using the default scheduling method, the job source (QI, AI, RTIC, Job Ring) is not considered when deciding which holding tank job should be assigned to the available DECO.

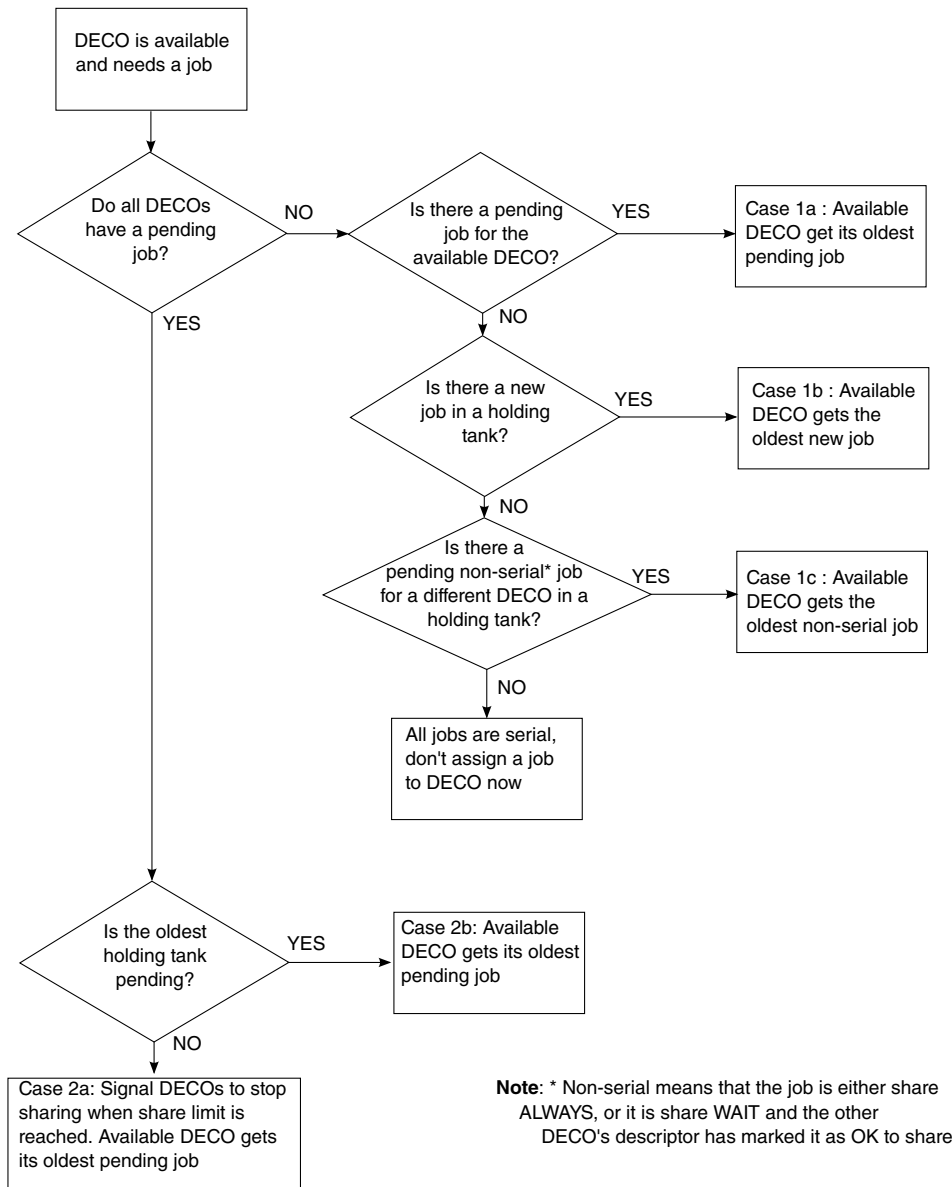


Figure 5-5. Job queue controller's job scheduling algorithm

This approach gives preference to serial sharing whenever doing so does not cause a job to remain indefinitely in a holding tank. Serial sharing occurs in cases 1a, 2a, and 2b in the algorithm described above. Serial sharing is the most efficient since the information that the new job shares with the previous job is already in the DECO. Jobs are shared between DECOs (case 1c) only when the available DECO has no pending job and there are no non-pending jobs in the holding tanks. This is done because sharing between DECOs is less efficient than sharing serially, since the shared information must be copied to the available DECO.

5.4.2 Job scheduling - adaptive resource reservation algorithm

The adaptive resource reservation scheduling algorithm is used for job scheduling when AR_EN=1 in the Adaptive Resource Reservation Register. In this case the specified number of DECOs are reserved for use by the AIOP Interface. The number of DECOs that are reserved is determined by the setting of the AR_MIN, AR_MAX and AR_DELTA fields in the Adaptive Resource Reservation Register. Note that the number of DECOs reserved for processing jobs from the AIOP Interface reduces the number of DECOs available for processing jobs submitted via the Job Rings or QI or RTIC. When AR_EN=1 and there are no uncompleted AI jobs in SEC, AR_MIN DECOs are reserved for AI jobs. Whenever SEC dequeues an AI job for processing, the job counter (JBCOUNT) is incremented. Whenever a DECO writes back status for a completed AI job, JBCOUNT is decremented. If $(JBCOUNT + AR_DELTA) < AR_MIN$, AR_MIN DECOs are reserved for AI jobs. If $AR_MIN < (JBCOUNT + AR_DELTA) \leq AR_MAX$, $JBCOUNT + AR_DELTA$ DECOs are reserved for AI jobs. If $(JBCOUNT + AR_DELTA) > AR_MAX$, AR_MAX DECOs are reserved for AI jobs. It is possible to configure the system with AR_MIN = AR_MAX and AR_DELTA = 0. This would reserve AR_MAX DECOs at all times for AI jobs. It is also possible to configure the system with AR_MIN = 0 and AR_DELTA = AR_MAX. This would reserve no DECOs for AI when there are no AI jobs running, but always reserve AR_MAX DECOs when one or more AI jobs are running.

The same algorithm is used to reserve holding tanks.

When adaptive resource reservation is enabled, scheduling jobs to holding tanks operates as follows. Assume HCOUNT is the number of HTs with AI jobs, DCOUNT is the number of DECOs with AI jobs, and RCOUNT is the number of HTs and DECOs to reserve for AI jobs. To allow resource sharing between AI and the other job sources, the assignment of jobs to holding tanks follows this scheme:

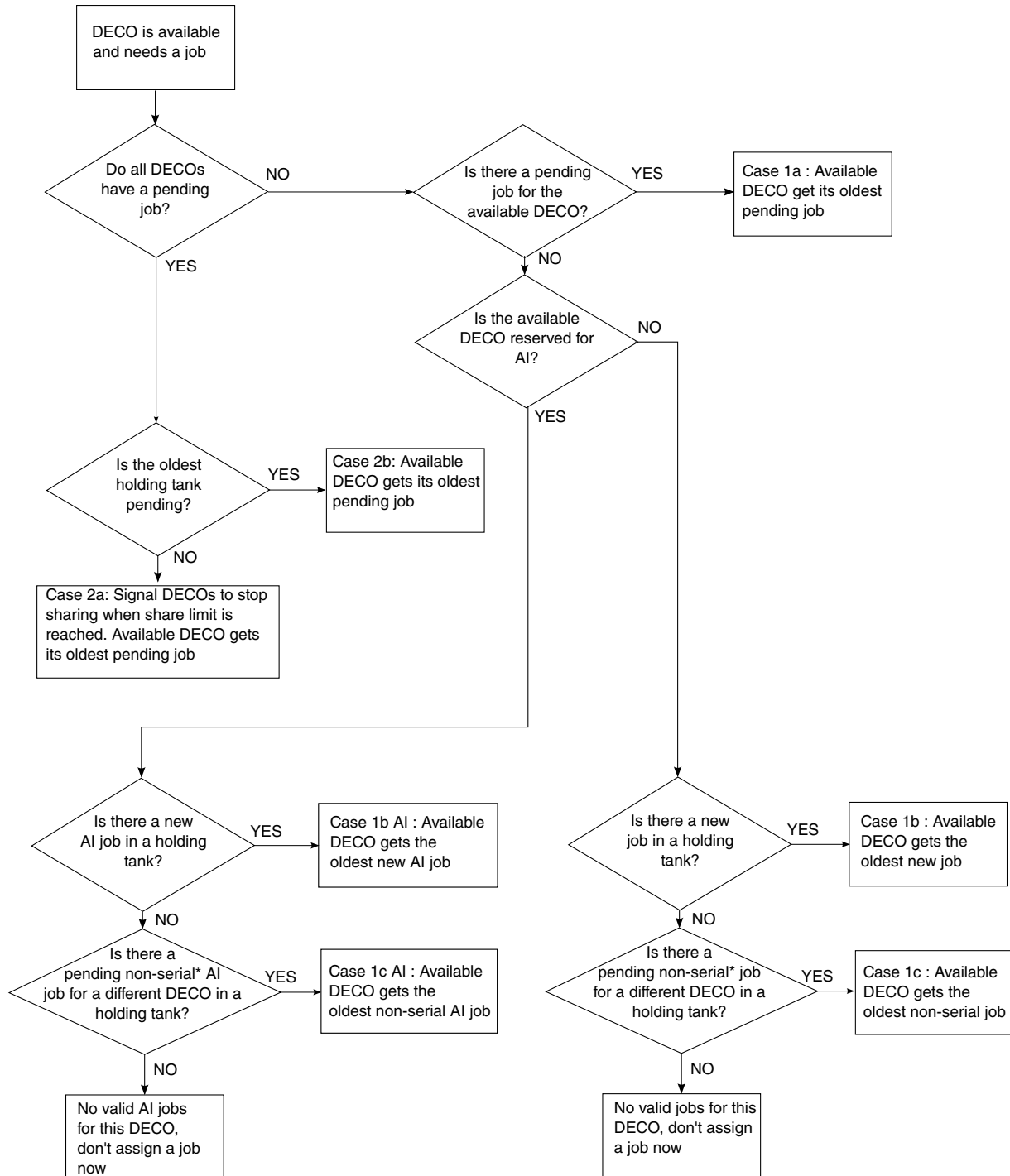
- If not enough HTs or DECOs are running AI jobs ($HCOUNT < RCOUNT$ or $DCOUNT < RCOUNT$) and there is an AI job that can be processed, the next HT gets that AI job.
- Else if HCOUNT plus the number of empty HTs exceeds RCOUNT, and if DCOUNT plus the number of empty DECOs exceeds RCOUNT, and if there is a JQ or QI or RTIC job ready, the next HT gets a job from the JQ/QI/RTIC round-robin.
- Else if there is an AI job, the next HT gets that AI job.

When holding tanks are assigned to DECOs:

- If the available DECO has a pending job, it gets its oldest pending job from any job source.

- Else if DCOUNT plus the number of empty DECOs is less than or equal to RCOUNT (available DECO is reserved for AI) and there is a non-pending (new) AI job in a holding tank, the oldest non-pending AI job is assigned to the available DECO (or the DECO is left empty if there is no AI job).
- Else if DCOUNT plus the number of empty DECOs is less than or equal to RCOUNT and a holding tank has a non-serial AI job pending on a different DECO, the oldest non-serial pending AI job is assigned to the available DECO.
- Else if DCOUNT plus the number of empty DECOs is less than or equal to RCOUNT, there is no valid AI job for this DECO. Do not assign a job at this time.
- Else if DCOUNT plus the number of empty DECOs is greater than RCOUNT and there is a non-pending job in a holding tank, the oldest non-pending job from any source is assigned to the available DECO.
- Else if DCOUNT plus the number of empty DECOs is greater than RCOUNT and a holding tank has a non-serial job pending on a different DECO, the oldest non-serial pending job is assigned to the available DECO.
- Else if DCOUNT plus the number of empty DECOs is greater than RCOUNT, there is no valid job for this DECO. Do not assign a job at this time.

This job assignment flow when AI adaptive resource reservation is enabled is detailed in the following figure.



Note: * Non-serial means that the job is either share ALWAYS, or it is share WAIT and the other DECO's descriptor has marked it as OK to share.

Figure 5-6. Job scheduling algorithm with adaptive resource reservation

5.4.3 Job scheduling - DECO-specific jobs

If there is more than one DECO it is possible to specify that a job should be run in a specific DECO by using the job header extension word. This can be used to support hardware testing. Once a DECO-specific job enters a holding tank, it remains there until the specified DECO becomes available, with the following exception. If the DECO-specific job contains a shared descriptor, specifies serial sharing, and the shared descriptor currently resides in a DECO other than the specified DECO, the DECO-specific job runs serially in the DECO that already contains the shared descriptor, resulting in a DECO-select error job termination code.

NOTE

DECO-specific jobs can create a deadlock in SEC when they are used as part of a flow. Therefore, it is strongly recommended that DECO-specific jobs should not be used in a flow.

5.5 Job execution hardware

The following modules in SEC execute cryptographic acceleration jobs:

- Descriptor controller/cryptographic control block
- Cryptographic hardware accelerators

5.5.1 Descriptor controller (DECO) and cryptographic control block (CCB)

The descriptor controller (DECO) is responsible for executing SEC job descriptors. After the job descriptor and any shared descriptor referenced by that job descriptor are loaded, DECO begins processing. Each DECO has a dedicated CCB (cryptographic control block) that it uses to access any cryptographic hardware accelerators (CHAs) needed to perform cryptographic functions.

When executing a descriptor, DECO activates the DMA controller to read the required inputs, and uses the CCB to dispatch the job to the appropriate CHAs. As data is produced by the CHAs, DECO activates the DMAs to write the results and job completion status information out to the locations specified in the descriptor. When a descriptor finishes, either successfully or with errors, DECO informs the job source (Job Ring interface, QI, AI or RTIC), which then takes appropriate action.

The CCB contains all the hardware necessary to control the various CHAs included in SEC. Every CCB has access to every type of CHA so that every DECO/CCB pair can perform all functions that can be performed by SEC. Note that there may be fewer instances of a CHA than there are CCBs. In such cases, a CCB may have to wait until the CHA it wants is available before proceeding. Arbitration for shared CHAs is automatically handled by SEC in all operating modes.

The hardware inside the DECO/CCB includes the input FIFO, output FIFO, information FIFO (NFIFO), mode registers, context registers, key registers, descriptor buffer, math registers, scatter/gather tables, alignment blocks and interconnects. DECO/CCB uses all of this hardware to process descriptors.

5.5.1.1 Alignment blocks

SEC 's internal data pathways and cryptographic engines generally operate on 64-bit data, but the information that SEC obtains from memory need not be aligned to 64-bit boundaries. To concatenate and left-align information passed to certain destinations within SEC, SEC architecture includes three alignment blocks:

- Class 1 alignment block
- Class 2 alignment block
- DECO alignment block

Note that even if the data is aligned in memory, the alignment blocks may still need to align some portions of the data because a subset of the data may be passed to more than one destination, and the subset may need to be aligned separately for each destination.

The following figure illustrates the interconnections of one of the alignment blocks. All three alignment blocks have the inputs shown in the figure. The Class 1 alignment block contains a nibble shift register, which allows the Class 1 alignment block to handle data that needs to be shifted by a nibble. The only other difference between the alignment blocks is the consumer (Class 1 CHA, Class 2 CHA, or DECO).

The entry pulled from the NFIFO tells the NFIFO controller the data source that will be used with the alignment block, and whether the alignment block will be flushed when the data transfer is complete. The alignment blocks normally transfer eight bytes of data at a time to the consumer. When the amount of data needed by the consumer is not a multiple of eight bytes, a "flush" flag or "last" flag is required to transfer the last one to seven bytes from the alignment block to the consumer.

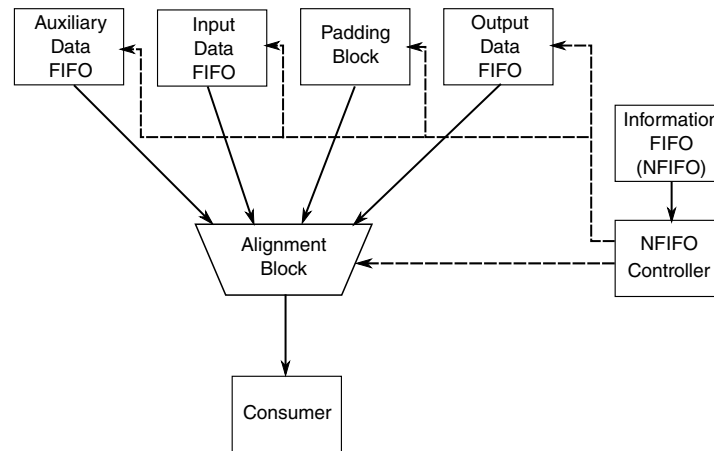


Figure 5-7. Alignment block interconnections

All data entering Class 1 CHAs first passes through the Class 1 alignment block, which ensures that the data presented to the Class 1 CHA is properly concatenated and left-aligned. Note that the Class 1 alignment block can also serve as the source for a MOVE command (see [Table 7-32](#), SRC value 9h and AUXLS = 1).

Similarly, all data entering Class 2 CHAs first passes through the Class 2 alignment block, which ensures that the data presented to the Class 2 CHA is properly concatenated and left-aligned. The Class 2 alignment block can also serve as a source for the MOVE command (see [Table 7-32](#), SRC value 9h and AUXLS = 0).

The third alignment block is the DECO alignment block, which can be used as a data source for a MATH command (see [MATH and MATHI Commands](#), SRC0 and SRC1 fields) and as a data source for a MOVE command (see [Table 7-32](#), SRC field).

Note that the only way to put data into an alignment block is with an info FIFO entry. Therefore, when using an alignment block as the data source for a MOVE command, the data source for the alignment block must have been specified with an info FIFO entry. This info FIFO entry may be automatically or manually generated. In order to use data stored in the input FIFO, that data must be passed through one of the alignment blocks. The only other way to take data out of the input FIFO is by resetting the input FIFO which also resets the alignment blocks.

5.5.2 Cryptographic hardware accelerators (CHAs) (overview)

SEC contains multiple cryptographic hardware accelerators (CHAs), each of which accelerates an encryption (Class 1) algorithm or message integrity (Class 2) algorithm.

- PKHA (public key hardware accelerator), see [Public-key hardware accelerator \(PKHA\) functionality](#).
- DESA (DES accelerator), see [Data encryption standard accelerator \(DES\) functionality](#).
- AESA (AES accelerator), see [AES accelerator \(AESA\) functionality](#).
- MDHA (message digest hardware accelerator), see [Message digest hardware accelerator \(MDHA\) functionality](#).
- RNG (random number generator), see [Random-number generator \(RNG\) functionality](#).
- STHA f8 (SNOW 3G f8 hardware accelerator), see [SNOW 3G f8 accelerator functionality](#).
- STHA f9 (SNOW 3G f9 hardware accelerator), see [SNOW 3G f9 accelerator functionality](#).
- CRCA (cyclic redundancy check accelerator), see [Cyclic-redundancy check accelerator \(CRCA\) functionality](#).
- KFHA f8/f9 (Kasumi hardware accelerator), see [Kasumi f8 and f9 hardware accelerator \(KFHA\) functionality](#).
- ZUCE (ZUC encryption hardware accelerator), see [ZUC encryption accelerator \(ZUCE\) functionality](#).
- ZUCA (ZUC authentication hardware accelerator), see [ZUC authentication accelerator \(ZUCA\) functionality](#).

Chapter 6

Frame queues, frame descriptors, and buffers

Frame queues (FQs), frame descriptors (FDs), and (managed) frame buffers are concepts introduced in the DPAA overview chapter of your SoC reference manual and described in detail in the Queue Manager (QMan) and Buffer Manager (BMan) chapters of the DPAA2 reference manual.

As a component of DPAA, SEC interacts with QMan to dequeue and enqueue FDs, AIOP to exchange FDs, and with BMan to obtain and release managed buffers. These interactions and the processing required to turn FDs into SEC job descriptors are primarily handled by SEC's QMan Interface (QI) and the AIOP Interface (AI), allowing the inner portions of SEC (the job queue and descriptor controllers) to be mostly ignorant of DPAA data structures and related processing requirements.

This section provides a functional overview of SEC's usage of FQs and FDs, including related interactions with QMan, AIOP, and BMan.

6.1 Frame queues

A pair of frame queues is used to submit jobs to SEC via QI and return results back to the service user. From SEC's perspective, a frame queue and a Job Ring are similar: each is a source of input and a destination for output. The major difference (besides the hardware/software interface) is the ability to efficiently share SEC services for multiple users and to support the optional use of BMan managed buffers.

A SEC Job Ring is a highly constrained hardware resource that can only be efficiently operated by a single owner, i.e., job requests from multiple users must either be coordinated by the Job Ring owner (e.g., an OS driver) while users (e.g., OS kernel threads) share common memory access rights (tied to the Job Ring configuration), or management software must grant full or temporary ownership of a Job Ring to individual users (e.g., a user process or thread) and memory access rights are (re-)configured by the management software on a per user basis.

The SEC QI on the other hand can receive frame descriptors and queue-specific user resource access rights (the basis for QI generated job descriptors) from millions of queues thus eliminating the need for control software to manage SEC service access for many users dynamically. Instead, control software configures queues (and associated user resource access rights) and assigns those queues to specific users thus enabling SEC to receive user-specific access rights as well as other processing parameters when one or more frame descriptors are de-queued from the user's queue for processing.

Besides access rights, software initializing SEC frame queues is required to define the source and response frame queue IDs as well as other FQD_CTX parameters in the Frame Queue Descriptor (FQD) which is utilized by QMan to enqueue, store, and dequeue frame descriptors destined to DPAA accelerators like SEC and return results back to a user. The source queue is needed for queue flush operations while the response queue is needed to report results or simply status back to the service user.

6.1.1 Dequeue response

Each time SEC's QI dequeues work (in the form of 1-3 frame descriptors) from a frame queue, it also receives dequeue summary information from the QMan's Frame Queue Descriptor (FQD) in the response to a dequeue command issued by SEC. FQDs are initialized by management software during the creation of frame queues. The first word of the dequeue response of QMan, called the summary information, includes the following fields of interest to SEC:

- Number of Frames dequeued
- Dequeue response (input) frame queue ID (IFQID)
- Dequeue response Bypass DPAA resource Isolation (BDI)
- Dequeue response Isolation Context ID (ICID)
- Dequeue response Privileged Level (PL)
- Dequeue response Frame Queue Descriptor Context Format (FF)
- Dequeue response Frame Queue Descriptor Context (FQD_CTX)
 - Enqueue response (output) frame queue ID (RFQID) - conditionally used
 - Access Management Qualifier (AMQ) Usage Control (AUC)
 - Isolation Context ID 2 (ICID_2) - optional
 - Isolation Context ID 3 (ICID_3) - optional
 - Privileged Level 2 (PL_2) - optional
 - Privileged Level 3 (PL_3) - optional
 - Security Domain ID (SDID) - optional

For a generic description of the details on all dequeue summary information fields see the QMan chapter in the reference manual of your SoC. The optional and conditional information elements are interpreted and utilized dependent on the settings of the FF and AUC.

1. If AUC = 0:
 - Input ICID = ICID, input PL = PL
 - Control ICID = ICID, control PL = PL
 - Output ICID = ICID, output PL = PL
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - Response (error) / output FQID = RFQID
 - All unreferenced optional parameters are not utilized
2. If AUC = 1:
 - Input ICID = ICID, input PL = PL
 - Control ICID = ICID, control PL = PL
 - Output ICID = ICID_2, output PL = PL_2
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - Response (error) FQID = RFQID, Output FQID = FLC.OFQID
 - All unreferenced optional parameters are not utilized
3. If AUC = 2:
 - Input ICID = ICID, input PL = PL
 - Control ICID = ICID_2, control PL = PL_2
 - Output ICID = ICID_2, output PL = PL_2
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = ICID_3
 - Response (error) FQID = RFQID, Output FQID = FLC.OFQID
 - All unreferenced optional parameters are not utilized
4. If AUC = 3:
 - Input ICID = ICID, input PL = PL
 - Control ICID = ICID_2, control PL = PL_2
 - Output ICID = ICID_3, output PL = PL_3
 - If (FF == 0) SDID = ICID, else (FF == 1) SDID = FLC.SDID
 - Response (error) FQID = RFQID, Output FQID = FLC.OFQID

6.2 Multi-partition resource access

DPAA2-supporting SoCs provide user partitioning via isolation and access management controls. User isolation and resource access control is implemented using Isolation Context Identifiers (ICIDs) and Access Management Qualifiers (AMQs). A minimum of one set of ICID and AMQs is needed to limit and grant access to user partition-specific resources like user memory, queues, buffer pools or other DPAA resources. The ICID

and most AMQs are managed and assigned by trusted management entities. Different user partitions are usually unaware of their ICID and most AMQs. In any case, users cannot change ICIDs and can change only a limited and qualified set of AMQs, i.e., the ICID enforces a non-spoof-able identity and the combination of ICID and AMQs supports mandatory access control to user partition-specific resources.

When only a single ICID and one set of AMQs are utilized, all resources needed to perform a SEC service request are provided by the user requesting the service. This implies that SEC's job descriptors are fetched from, input data is read from, and the output data is written to memory controlled by a user from the partition that is requesting service. If SEC's service is requested via AI, any required buffer pools need to be "owned" by the service-requesting partition. If the service is requested via QI, the source and response queues need to be "owned" by the service-requesting partition as well. In this use case SEC essentially acts as a server and the user is completely in charge of what to do, what input data to consume, and where to place results. SEC and all supporting DPAA HW blocks ensure that the user cannot use any resource not associated with the user partition's ICID or that violates the AMQ settings.

A partition's ICID and AMQ settings are conveyed to SEC either in a Job Ring register configuration that can be set only by a trusted SoC management entity (because the configuration registers are protected by CPU and System MMUs) or via QMan queue descriptors and software portal configurations (also protected by CPU and System MMUs). Internally, SEC always differentiates between input, control, and output partition transactions and utilizes their (potentially different) associated ICIDs and AMQ settings. In the single-ICID/AMQ use case, SEC uses the same ICID and AMQ settings for all three types of transactions.

While most SEC user service requests can be satisfied using only a single user's resources there is sometimes a need to combine and access resources from multiple users or user partitions to control or provide global, generic, or shared services. For this purpose the DPAA provides means to convey up to three sets of ICIDs and AMQs to trusted SW or HW components, which guarantee by design that their services cannot be abused (either by error or intent) to bypass partition isolation or access control. The following subsections describe how SEC provides services involving resources from multiple users or user partitions and what use restrictions exist to assure that SEC services do not provide means to bypass isolation and/or access controls.

6.2.1 Multi-partition resource access modes

Multi-partition resource access support from SEC is available only through QI from QMan or indirectly through AI from QMan via AIOP and not available for jobs launched through the Job Ring service interface. Independent of whether SEC jobs are received via

QI or AI, control over how many sets and values of ICID and AMQ attributes are utilized is primarily managed by trusted SW through QMan software portals or AIOP software configurations. The relevant multi-partition resource access information is conveyed to SEC in a Frame Queue Descriptor (FQD) and user-provided (but qualified) AMQ modifiers (BMT and CBMT) passed in each Frame Descriptor (FD). The following multi-partition resource use modes are standardized and supported through different FQD configurations and SEC services. (Note, while all ICID/AMQ use modes are initially sourced from QMan software portal configurations, AIOP has unrestricted capabilities to change or create multi-partition resource use modes because AIOP's software can either modify or generate new FQD and FD information before passing this data to SEC):

- Single-partition resource access mode (FQD.FQD_CTX.AUC=0)

In single-partition resource access mode the FQD.ICID and FQD.BDI AMQ attribute are used to access any buffer pools and frame queues. The FQD.PL AMQ attribute is used to default and qualify user-provided AMQ attributes from the FD, and the combination of FQD.ICID and the qualified AMQ attributes are used to access input, control, and output data.

The service user is fully in control of all selected resources and the data SEC consumes and generates, but cannot use any resources from other partitions.

- Input-controlled dual-partition resource access mode (FQD.FQD_CTX.AUC=1)

In input-controlled dual-partition resource access mode the ICID and AMQ attributes used in the single-partition resource access mode are limited to input and control information. Access to output resources and output data is controlled by FQD_CTX.ICID_2 and FQD_CTX.PL_2. The FQD.BDI attribute applies to accesses of both input and output pools and frame queues.

Because the input partition has the ability to determine what SEC does and thus what gets stored in the output partition's memory this use case should be viewed as a master/slave type of relationship where the SoC input/control partition is assumed to be more trustworthy than the output partition.

- Output-controlled dual-partition resource access mode (FQD.FQD_CTX.AUC=2)

In output-controlled dual-partition resource access mode the ICID and AMQ attributes used in the single-partition resource access mode are limited to the input information only. Access to control and output resources is controlled by FQD_CTX.ICID_2 and FQD_CTX.PL_2. The FQD.BDI attribute applies to accesses of both input and output pools and frame queues.

Because the output partition has the ability to determine what SEC does and thus what gets read from the input partition's memory, this use case should be viewed as a master/slave type of relationship where the SoC control/output partition is assumed to be more trustworthy than the input partition.

- Three-partition resource access mode (FQD.FQD_CTX.AUC=3)

In three-partition resource access mode the ICID and AMQ attributes used in the single-partition resource access mode are limited to the input information only. Access to control and output resources is controlled by two sets of additional ICID and AMQ attributes, FQD_CTX.ICID_2 and FQD_CTX.PL_2 for accesses to control partition resources, and FQD_CTX.ICID_3 and FQD_CTX.PL_3 for accesses to output partition resources. The FQD.BDI attribute applies to accesses of both input and output pools and frame queues.

Because neither the input nor the output partition controls what SEC does and neither input nor output side has the ability to access the other partition's data, the three-partition resource access mode provides red/black information isolation assuming the control partition is a trusted entity.

Three-partition resource access mode also supports a traditional but safely isolated server/client relationship model when both input and output are associated with the same SoC partition, i.e., ICID == ICID_3, and where globally controlled services (by SEC) are maintained and provided by a trusted entity.

To preserve isolation guarantees and enforce obligatory DPAA resource access controls, SEC must restrict and/or automatically change the scope of any features that would allow any partition to get unintended access to other partition's resources in all multi-partition use modes. Assuming the assignment and configuration of multi-partition usage controls (ICIDs and AMQs) is trusted, the SEC implementation must provide the following to guarantee isolation and obligatory access controls while providing multi-partition resource access services:

- Restrict use of Flow Contexts (FLCs), and thus Shared Descriptors (SDs), to partitions that meet restrictions specified in the FLC ICR, IICID, and OICID fields.
- Restrict use of Inline Job Descriptors (IJD) conveyed in the input frame to multi-partition use cases where the input and control partitions are the same.
- Restrict use of Replacement Job Descriptors (RJDs) conveyed in the input frame to multi-partition use cases where the input and control partitions are the same.
- Restrict use of Control RJDs (CRJDs) that sequentially follow the SD in memory (and thus reside in the SD partition) to use cases where the FLC CRJD bit is set.

The following sections describe in more detail how specific DPAA and SEC features are intended to be used, why unrestricted use of those features in combination with multi-partition resource access capability could be used to bypass DPAA isolation guarantees, and what SEC provides to enforce adequate multi-partition resource access restrictions.

6.2.2 Flow context selection restrictions

DPAA2 enables SEC service users to select a Flow Context (FLC) (and for SEC an associated Shared Descriptor (SD)) on a per-service request basis. Similarly, the user selects the service request queue (which is associated with one or more ICID and AMQ settings) on a per request basis. While the user cannot change the ICID and AMQ settings, a user may mix and match service requests selecting specific SEC flow contexts and use any of its assigned queues to generate service requests, i.e., a user can in principle select a flow context that is intended to produce output for a different partition using a queue that is configured to generate output in the user's partition. This would in many cases fail because the user would not have the anticipated buffer pool IDs (his buffer pool IDs would have to match the buffer pools allocated by the output partition), but a malicious user could attempt to discover through trial and error matching buffer pool IDs and thus the user may manage to redirect output and get access to output data that was supposed to be not accessible to the user unless SEC prevents such use.

SEC can restrict a user's ability to select any FLC (and SD) in a multi-partition use case to prevent un-trusted users from bypassing user isolation capabilities as follows:

- An FLC can be programmed to prevent its use with other than the intended input and output ICIDs when the input and control ICIDs may be different (as indicated by FQD.FQD_CTX.AUC=2 or 3). With this restriction, a service-requesting user can select only FLCs that are intended to be used by the user and that produce output for the intended target output partition. This restriction is enabled by setting the ICR field in FLC to 0 and the IICID and OICID fields to the intended input and output ICIDs, respectively.
- An FLC can be programmed to require that the input and output ICIDs be the same when FQD programming allows them to be different (as indicated by FQD.FQD_CTX.AUC=2 or 3). With this restriction, a service-requesting user can use an FLC with a particular service queue only if the input and output ICIDs for the queue are the same. (The control ICID may be different.) This restriction is enabled by setting the ICR field in FLC to 1.
- FLC selection checking on the basis of input and/or output ICIDs will introduce use restrictions for trusted entities requiring SEC services utilizing multi-partition resource access (e.g., SEC use by AIOP combining AIOP-internal resources with user-partition resources to access input, control, and output data or buffer pools).

When FQD programming allows input and output ICIDs to be different (as indicated by FQD.FQD_CTX.AUC=2 or 3), FLC-based restrictions on input and output ICIDs can be disabled for trusted users by setting the ICR field in FLC to 3.

If a service request selects an FLC and the FLC defined ICID matching rules are not met SEC generates an isolation violation error.

6.2.3 Inline job descriptor restrictions

Inline Job Descriptors (IJDs) are intended to support user-controlled SEC processing on a per-FD basis. While IJDs provided a useful capability to process 'one-off' jobs in earlier DPPA1-based SoCs, DPAA2 provides for more efficient per-FD processing via Flow Contexts (FLCs). Unlike IJDs, which must be linked or physically copied to FD buffers, FLCs and sequentially following Shared Descriptors (SDs) are simply referred to by a pointer in the FD. Thus the overhead of copying IJDs or the use of scatter/gather tables may be avoided.

Unrestricted support of IJDs in all multi-partition resource access use cases would enable SEC users to break DPAA isolation guarantees and resource access controls. The user's ability to both select SEC instructions and select assigned frame queues (which are associated with up to three sets of ICID and AMQ settings) would allow users to access any of the resources associated with ICIDs and AMQ settings defined in the associated FQD. For this reason SEC allows the use of IJDs in multi-partition resource use cases when the FQD.FQD_CNCTX.AUC code indicates the input and control ICIDs and associated AMQs are the same (AUC codes 0 and 1). Since the input and control partitions are the same, the IJD can only access what the user partition is able to access with an SD following the flow context, i.e., isolation guarantees are maintained.

6.2.4 Replacement job descriptor restrictions

A Replacement Job Descriptor (RJD) is a special form of an Inline Job Descriptor (IJD) intended to support user controlled per-FD processing before or after a Shared Descriptor (SD) is executed (e.g., RJDs can be used to update key material or other SD state (before, after, or without any normal frame processing defined by the SD)). Just like IJDs, RJD instructions have been historically conveyed to SEC in the input frame and thus the input partition gains control over what SEC does. Therefore, in multi-partition resource access use cases, RJD users could bypass the intended SD function while being able to access other partition's resources. To prevent isolation violations in multi-partition use cases,

SEC only allows this type of RJD to run when the FQD.FQD_CNTRX.AUC code indicates the input and control ICIDs and associated AMQs are the same (the AUC code is 0 or 1).

An alternative to conveying the RJD in the input frame is to place the RJD instructions in the control partition immediately following the SD. This type of RJD (referred to as Control RJD (CRJD)) can ensure that intended isolation restrictions are observed. Since the Flow Context, SD, and CRJD are only accessible using the control ICID and associated AMQs, the owner of the control partition can design a suitable and safe RJD. Access to the space following the SD is only allowed when the CRJD bit in the Flow Context is set. The user of the Flow Context and SD can only choose to run or not run the CRJD on a per-FD basis. The user cannot decide what CRJD to run or what the CRJD does.

6.2.5 Non-local jump limitations

A non-local JUMP is a SEC descriptor command that causes SEC to terminate execution of the current SEC descriptor and begin execution of a different SEC descriptor. A non-local JUMP command defines the target address of the new descriptor. While the non-local JUMP instruction enables the descriptor writer to specify a memory address, the command does not allow the ICID and AMQ settings governing descriptor fetch memory accesses to be changed. Therefore the use of non-local JUMP commands does not provide a means to bypass isolation guarantees.

6.2.6 Multi-partition resource access restriction summary

The following table summarizes access restrictions for select SEC descriptors and command types in multi-partition use cases:

Descriptor/Command Type	AUC = 0 (input, control, and output ICID/AMQs are identical)	AUC = 1 (input and control ICID/AMQs are identical, output ICID/AMQ is treated as different)	AUC = 2 (control and output ICID/AMQs are identical, input ICID/AMQ is treated as different)	AUC = 3 (input, control and output ICID/AMQs are treated as different)
Inline Job Descriptor (IJD)	no restrictions		unconditional error	
Replacement Job Descriptor (RJD)	no restrictions		unconditional error	
Control RJD (CRJD)	no restrictions if FLC.CRJD = 1; otherwise, error			

Table continues on the next page...

6.3 Frame descriptors

A frame is some number of bytes of space or information stored in memory and represents room for or the actual data of a single message, packet, or protocol data unit. This space or data can be located in one or more buffers in memory. Multi-buffer frames use one or more tables, called scatter/gather tables, to keep track of buffer order, addresses, and associated information.

A Frame Descriptor (FD) is a standardized DPAA data structure that defines a frame's starting address in memory, data length, and other frame attributes. The FD is a vehicle to transfer frames between the QMan and SEC's QI through use of a QMan hardware portal that provides frame dequeue and enqueue services. The FD is also utilized to transfer frames between the AIOP and SEC's AI through use of AIOP's AAP interface, which suspends an AIOP task after receiving a frame to be transferred to SEC and resumes the AIOP task after AAP has received results in a frame from SEC.

Note that FDs should not be confused with the job and shared descriptors used by SEC to describe a sequence of operations to be performed by SEC.

The FD includes a format field that describes the format of the corresponding frame. There are two major frame formats: single frame and frame list.

- A single frame FD describes a single frame that may define the input to or the output of SEC data processing. Note, the single frame format is also used for FDs that do not convey a frame.
- A frame list FD generically describes a list of frames. SEC supports only a special form of frame list, a two-entry input/output frame list, using a fixed two-entry scatter/gather-like table where each entry is an FD structure.

When QI dequeues or AI receives a job with an input/output frame list FD, it returns the processed job with the same frame table list and the same input and output frame buffers. When QI or AI obtains a single frame FD, it may return the processed job with the same or new single frame buffers for the output data. The flow context defined in the received FD's FLC field specifies how to use frame data buffers for result return.

6.3.1 Flow Context

The Flow Context (FLC) field in the Frame Descriptor (FD) is used to associate data, buffers, commands, and other attributes conveyed through the FD with specific SEC processing requirements typically related to a 'flow' of jobs (versus one-off jobs) sharing the same kind of processing. Except for FDs that request services for more than one flow, SEC interprets the FLC field as a pointer to a 64-byte memory structure preceding an optional SEC Shared Descriptor (SD). Software may use the most-significant 15 bits of the SEC FLC to carry a token to assist in address translation or serve other needs. The SEC FLC address should be 64-byte aligned for optimal performance (and must be 64-byte aligned if the FD[SC] bit is set to indicate that the least significant 6 bits of the FLC are used as stash controls). Access control to the FLC data is provisioned through the 'control' ICID and PL as defined by AUC code and associated configuration fields in the QMan dequeue summary and AIOP frame transfer attributes as well as the CBMT bit defined in the FD.

The SEC FLC data structure (see the table below) contains the following types of information:

- Job scheduling and processing options including the shared descriptor length
- Response flow context and attributes
- User authentication and security domain ID association (optional)
- Output flow context and attributes (optional)
- Storage profile (SP) for output buffer management (AI may provide per-frame SP)

The SDL, CRID, CRJD, EWS, DAC, and EAO fields form the group of flow-specific SEC job scheduling and processing options. SDL serves a dual purpose: It is used to define both the length of an optional Shared Descriptor (SD) and the entry point of an optional Control Replacement Job Descriptor (CRJD). The CRID field defines the internal ID of a critical SEC CHA resource that may not be available to be used by all DECOs in parallel, but is required to process the service request. The CRJD bit indicates the presence of a CRJD following the SD and the EWS bit is used to enable write-safe SEQ stores to output buffers. The DAC configuration may be used to disable the copying of Pass-Through Annotation (PTA) from input to output frames and the EAO configuration is used to enable SEC-generated Application-Specific Annotation (ASA) in the output frame.

The RSC, RBMT and RFLC fields form the group of flow-specific, response flow context and attribute configurations. The content of these fields is used to generate the SC, CBMT, and FLC fields of any response FDs transferred to the originator of service requests (input partition) using this SEC FLC. These fields are not used and reserved when the FLC is referenced by an FD processed by the AI service interface.

The OFQID, OSC, OBMT, and OFLC fields form the group of flow-specific, output queue and flow context and attribute configurations. The content of these fields is used to define an output queue and generate the SC, CBMT, and FLC fields of any response FDs transferred to an output partition for service requests using this SEC FLC. These fields are only utilized for multi-ICID use cases where the AUC code indicates that the output partition is different from the input partition. These fields are not used and reserved when the FLC is referenced by an FD processed by the AI service interface.

The SDID, ICR, IICID, and OICID fields form the user authentication and security domain ID association group of configurations. The content of these fields is used only in multi-ICID use cases where the AUC code indicates that the control, input, and output partitions may be different. SEC will use the SDID specified in the FLC only if the requested service requires an SDID and if AUC=3 (indicating that the ICID_3 field in the QMan dequeue summary is utilized to convey the output partition ICID). For certain use cases, depending on the ICR setting, SEC will apply different matching requirements between the ICIDs provided in the QMan dequeue summary and the AIOP frame transfer attributes and the ICIDs defined in the IICID and OICID fields in the FLC. These ICIDs are used to determine whether the originator of the service request is allowed to utilize the FLC (and implicitly any associated SD and/or CRJD) and to direct the results of the processing to the provided output frame queue. For a detailed description of the different ICR settings and the associated use of IICID and OICID, see the respective entries in the FLC field description table below.

The BS, FF, DLC, PTAR, SGHR, DHR, DL, BMTn, BPIDn, PBSn, SRn, and BPVn fields form the group of flow-specific Storage Profile (SP) configurations. For a SEC-specific description of the SP configuration fields see the entries in the FLC field description table below. A generic description of the fields is available in the DPAA2 reference manual. The BDP and BP settings described in the generic SP pool control records do not apply and are silently ignored by SEC.

Table 6-1. SEC FLC Format

Word 0	31-12										11-0		
	reserved										SDID		
Word 1	31-26	25	24	23-19	18	17	16	15	14-12	11-8	7-6	5-0	
	reserved	RB MT	R S C	reserved	E A O	D A C	E W S	CR JD	reserved	CRID	reser ved	SDL	
Word 2	31-0												
	RFLC[31-0]												

Table continues on the next page...

**Table 6-1. SEC FLC Format
(continued)**

Word 3	31-0													
	RFLC[63-32]													
Word 4	31-16						15-0							
	OICID						IICID							
Word 5	31-30	29-26	25	24	23-0									
	ICR	reserved	OB MT	O S C	OFQID									
Word 6	31-0													
	OFLC[31-0]													
Word 7	31-0													
	OFLC[63-32]													
Word 8	31-16						15-0							
	reserved						DL							
Word 9	31	3	29-28	27-25	24	23	22-21	20	19-16	15-12	11-0			
	BS	rs v	FF	reserved	DL C	PT AR	reser ved	SG HR	ASAR	reserved	DHR			
Word 10	31	3	29-16						15-6			5	4-1	0
	BM T1	rs v	BPID1						PBS1			S R 1	reserved	BP V1
Word 11	31	3	29-16						15-6			5	4-1	0
	BM T2	rs v	BPID2						PBS2			S R 2	reserved	BP V2
Word 12	31-0													
	reserved													
Word 13	31-0													
	reserved													
Word 14	31-0													
	reserved													

Table continues on the next page...

Table 6-1. SEC FLC Format (continued)

Word 15	31-0
	reserved

Table 6-2. Flow Context Field Descriptions

Field Names	Descriptions
ASAR	<p>The Accelerator Specific Annotation Room field is part of the SP group of parameters. ASAR is encoded in terms of 64-byte increments and its value is used to generate (preserve in the reuse mode) HW annotation space in the output frame.</p> <p>If output buffer source is set to allocate output buffers (BS=0) the ASAR field is used to generate room for ASA. In the single buffer output case ASAR is combined with DHR and any necessary PTAR to form FD[OFFSET] and thus create room for both annotations and data head room. In the SG buffer output frame case ASAR is combined with SGHR and any necessary PTAR to form FD[OFFSET] and thus create room for both annotations and any SG head room. DHR is used to set the OFFSET in the first SGT entry.</p> <p>If output buffer source is set to reuse input buffers (BS=1) the ASAR field is used to leave (or preserve) room for a defined amount of (preexisting) ASA. In the single buffer output case ASAR is combined with any necessary PTAR to determine whether the requested amount of DHR correction is possible and to report an error if not. In the SG buffer output case SGE[OFFSET] is used to perform the check and to report an error if the correction is not possible.</p>
BMT1 / BMT2	<p>The Bypass Memory address Translation bits are part of the buffer pool control record 1 and 2 in the SP parameter group. The BMTn setting indicates whether the pool buffers reside in memory that must be accessed without System MMU memory address translation, i.e., the buffer memory must be accessed with physical addresses.</p> <p>Note: If the output buffer source is set to allocate buffers (BS=0), buffer pool record 1 may need to be marked valid (BPV1=1) so that BMT1 can be set as a default output BMT available to be used within the SD or CRJD, even if the remaining SP parameters request that QI or AI never allocate output buffers (DLC=1 and DL=0). In this rare use case no other parameters other than BPV1 and BMT1 are utilized in the buffer pool records and the user should set unused and reserved fields to 0.</p>
BPID1 / BPID2	<p>The Buffer Pool ID fields are part of the buffer pool control record 1 and 2 in the SP parameter group. BPIDn defines the buffer pool ID utilized by SEC to allocate buffers on behalf of the service user. Note, in some error cases SEC may use this buffer pool ID to release allocated buffers.</p>
BPV1 / BPV2	<p>The Buffer Pool Valid bits are part of the buffer pool control record 1 and 2 in the SP parameter group. BPVn simply indicates whether the associated buffer pool record is valid or must be ignored. SEC supports the use of up to 2 buffer pool records. The record for buffer pool 1 must be utilized if there is only one available buffer pool.</p> <p>Note: If the output buffer source is set to allocate buffers (BS=0), buffer pool record 1 may need to be marked valid (BPV1=1) so that BMT1 can be set as a default output BMT available to be used within the SD or CRJD, even if the remaining SP parameters request that QI or AI never allocate output buffers (DLC=1 and DL=0). In this rare use case no other parameters other than BPV1 and BMT1 are utilized in the buffer pool records and the user should set unused and reserved fields to 0.</p>
BS	<p>The Buffer Source bit is part of the SP parameter group. If BS is set to</p> <ul style="list-style-type: none"> • 0 - allocate output buffers and release input buffers (some errors may cause different behavior) • 1 - reuse input buffers as output buffers (invalidates DL field) <p>Note: The user may override the BS setting on a per FD basis by providing output buffers via input/output frame lists.</p>

Table continues on the next page...

Table 6-2. Flow Context Field Descriptions (continued)

Field Names	Descriptions
CRID	The Critical Resource ID field is part of the job scheduling and processing parameter group. CRID defines the internal ID of a critical SEC CHA resource that may not be available for use by all DECOs in parallel, but is required to process the service request. If a non-zero CRID is defined, QI and AI check the availability of the resource when selecting the next job for transfer to the job queue controller. If all instances of the critical resource are in use by jobs from other flows, the selection priority of the jobs from this flow is lowered to minimize the possibility that multiple DECOs are stalled waiting for one or a small number of critical resources to become available. FLC-generating SW must be aware of what CHA types the SEC descriptor(s) is/are utilizing. If the available number of the CHA type required to perform the acceleration is less than the number of available DECOs, CRID should be set to the ID of the CHA to support fair and performance-optimized job scheduling. Reading the CHANUM register enables software to query how many DECOs and how many CHAs of each type exist in a given SEC and set CRID if necessary.
CRJD	The Control Replacement Job Descriptor present bit is part of the job scheduling and processing parameter group. The setting of CRJD simply indicates whether there is a CRJD immediately following the SD. This is primarily useful for multi-partition use cases where the input partition must not be able to select and request the execution of a CRJD in cases where the control partition did not provide a CRJD.
DAC	The Disable pass-through Annotation Copy control bit is part of the job scheduling and processing parameter group. DAC is set to disable the copying of Pass-Through Annotation (PTA) from the input to output frame. For most applications DAC should left clear to comply with the general DPAA2 requirement that PTA is always 'Passed'. DAC may need to be set if a control partition needs tighter isolation for applications where input and output data are controlled by different partitions. For more information on multi-partition use cases, see Multi-partition resource access modes
DHR	The Data Head Room field is part of the SP parameter group. DHR is encoded in terms of bytes. If the output buffer source is set to allocate output buffers (BS=0) the DHR field is used to generate data headroom, i.e., the offset in the first data containing entry of SG buffer output frames and in combination with the annotation room specification (see ASAR above) in single buffer output frames. If the output buffer source is set to reuse input buffers as output buffers (BS=1) the DHR field is treated as a signed value (2's-complement) of a data headroom correction and defines by how many bytes an existing offset should be adjusted to make room for additional output data or any need to move the output 'forward' in relation to the input data so that output generation does not destroy input data before it is consumed. Note: In most envisioned use scenarios DHR would be negative but the use of positive values is not excluded.
DL	The Data Length field is part of the SP parameter group. DL is only used in output buffer allocation mode (BS=0) and an estimate for an absolute output data length (DLC=1) or a length correction / adjustment of the output frame length based on input frame length information (DLC=0). The resulting length may need to be treated as an estimate because a precise length may not be possible to determine due to variable padding requirements. Examples: <ul style="list-style-type: none"> • a physical device may use a positive DL value to create additional frame tail room. • a service producing output on the basis of some input data may use DL to calculate an anticipated shorter or longer output frame. In the length correction / adjustment case the number is treated as a signed value (2's-complement), i.e., the output length can be either expanded or reduced relative to a received input frame. Depending on the output length control bit (see DLC below) the output length can be set to an fixed value (including 0, in this case the output FD would only report status or an error if there is unexpected output data to be generated). If using input buffer reuse mode (BS=1), then this field is unused.
DLC	The Data Length Control bit is part of the SP parameter group. DLC is only used in output buffer allocate mode (BS=0) and if set to <ul style="list-style-type: none"> • 0 - use DL to calculate the anticipated output frame buffer length by adding the (signed) DL correction value to the input frame length • 1 - use DL to set a fixed anticipated output frame buffer length QI and AI do not allocate any buffer if the calculated estimate for the output buffer length is zero (0) and no annotation need to be generated or passed to the service user.

Table continues on the next page...

Table 6-2. Flow Context Field Descriptions (continued)

Field Names	Descriptions
EAO	<p>The Enable application-specific Annotation Output control bit is part of the job scheduling and processing parameter group. The EAO bit is used to enable SEC-generated Application-Specific Annotation (ASA) to the output frame. Note, EAO strictly controls writing ASA to the output frame. When SEC is utilized via the AI service interface, SEC ASA is always returned in the response FD FLC field.</p>
EWS	<p>The Enable Write-Safe control bit is part of the job scheduling and processing parameter group. The EWS bit is used to enable write-safe SEQ stores to output buffers. Write-safe is always enabled for output frames allocated by QI and AI since there is no danger of overwriting existing data in allocated output frame buffers. EWS is a means to enable write-safe when an input frame is reused for output or when an output frame is provided through an input/output frame list. See DMA interface write-safe transactions for more details on write-safe operation, controls, and benefits.</p>
FF	<p>The Frame Format field is part of the SP group of parameters. The FF field defines the type of output frame structure SEC should generate. The output frame structure is formed utilizing a sub-set of the elements available from the FD format options and the 'room' specifications in the SP parameter group. The following frame structures are supported by SEC to address different user needs:</p> <ul style="list-style-type: none"> • Single buffer frame: The single buffer frame is formed from a single data buffer. Input FD PTA and PTVn, PTAR, ASAR, and DHR are used to calculate OFFSET. OFFSET, DLC, and DL are used to calculate minimum buffer length. Any non-0 SGHR value is ignored. • Scatter/gather (SG) buffer frame: The SG buffer frame is formed from one or more SGT buffers and one or more data buffers. Input FD PTA and PTVn, PTAR, ASAR, and SGHR are used to calculate OFFSET for the SGT buffer. DHR is used to set OFFSET for the first data buffer. OFFSET for subsequent data buffers is set to 0. The SGE format types 'frame data section' and 'unused' are not utilized. <p>If the output buffer source is set to allocate output buffers (BS=0), the following applies depending on the selected FF value:</p> <ul style="list-style-type: none"> • 0 - Allocate single buffer frame or an SG buffer frame with as many data buffer(s) as needed to generate the required data offset and hold the frame data. • 1 - Reserved • 2 - Force single buffer frame use (generate error if (anticipated) output data with offset does not fit in single buffer). • 3 - Force SG buffer frame use. This may produce an FD with a single entry SG table buffer pointing to a single buffer. <p>If output buffer source is set to reuse input buffers as output buffers (BS=1), the following applies:</p> <ul style="list-style-type: none"> • 0 - Reserved • 1 - Reserved • 2 - Reuse input buffers if they provide sufficient space (generate error if (anticipated) output data with recalculated offset does not fit in the available input buffer(s)). Note: The check for available space relies on the input frame data offset and length information instead of what may be really available in the input buffers because any unused input buffer head or tail room is in many use cases unknown or would require excessive processing overhead to determine). • 3 - Reserved <p>SEC reports an 'SP frame format unsupported or reserved' status if a reserved FF code is encountered.</p>
ICR	<p>The Isolation Context Restrictions field is part of the user authentication parameter group. The ICR code is only utilized in combination with the AUC codes 2 and 3 and provides the ability to restrict access to the FLC if the service request's input and output ICIDs do not match FLC-specific input and output ICID values (IICID and OICID) or ICID relationships. The following is supported:</p> <ul style="list-style-type: none"> • 0 - If AUC = 2 or 3 (input and control ICIDs for the dequeued frame may be different), input and output ICIDs must match the values defined in the IICID and OICID fields, respectively. • 1 - If AUC = 2 or 3 (input and control ICIDs for the dequeued frame may be different), input and output ICIDs must be the same. • 2 - Reserved (defaults to 'No restrictions' but may be assigned a different restriction in the future) • 3 - No restrictions

Table continues on the next page...

Table 6-2. Flow Context Field Descriptions (continued)

Field Names	Descriptions
IICID	The Input Isolation ID field is part of the user authentication parameter group. The IICID field is only utilized in combination with AUC codes 2 or 3 and ICR code 0. If ICR=0 and AUC=2 or AUC=3, IICID must match the input ICID and OICID must match the output ICID in the QMan dequeue summary in order for SEC to accept and execute the service request.
OBMT	The Output Bypass Memory Translation bit is part of the user output parameter group. OBMT is used to configure a user selected CBMT value associated with the user selected Output FLC (see OFLC described below). The CBMT attribute in combination with interpreting OFLC as a pointer can be used to stash up to 3 64-byte blocks of user context data (if enabled via the SC and CS fields in the FD) when QMan de-queues SEC's service output FD from the output queue defined by OFQID. Setting OBMT/CBMT (to bypass memory translation) has no effect unless the user is implicitly authorized to do so by having access to and using a frame queue with a set privileged level (PL=1). Note, OBMT is only utilized for SEC generated FDs directed to the output partition and if the service request was not a management command.
OFLC	The Output FLC field is part of the user output parameter group. OFLC is used to configure a user selected FLC field returned to the user in SEC's service output FD fetched from the output queue defined by OFQID. Note, the OFLC field is fully determined by the service user unless OSC is set. If OSC is set, the LS 6 bits of the OFLC field define QMan stash controls and need to comply with the DPAA FD format specification. Note, the OFLC field is only utilized for SEC generated FDs directed to the output partition and if the service request was not a management command.
OFQID	The Output Frame Queue ID is part of the user output parameter group. OFQID is only used to enqueue SEC's service output FD when input and output partitions (ICIDs) are treated as different (AUC codes 2 or 3) and if the service request was not a management command.
OICID	The Output Isolation ID field is part of the user authentication parameter group. The OICID field is only utilized in combination with AUC codes 2 or 3 and ICR code 0. If ICR=0 and AUC=2 or AUC=3, OICID must match the output ICID and IICID must match the input ICID in the QMan dequeue summary in order for SEC to accept and execute the service request.
OSC	The Output FLC Stash Control bit is part of the user output parameter group. OSC is used to configure a user selected QMan Stash Control (SC) associated with the user selected LS 6 bits of the Output FLC (OFLC) field. Per DPAA FD format specification, a set SC bit enables QMan to interpret the LS 6 bits of the FLC field to control up to 3 64-byte blocks of frame data, context, and/or annotation data to be stashed (loaded in a cache for CPU consumption) when a user de-queues such an FD from a QMan software portal. For more information on SC and related FD fields see the DPAA overview chapter in your SoC's reference manual. Note, OSC is only utilized for SEC generated FDs directed to the output partition and if the service request was not a management command.
PBS1 / PBS2	The Pool Buffer Size fields are part of the buffer pool control record 1 and 2 in the SP parameter group. PBSn is encoded in terms of 64 bytes and reflects the amount of the data that can be stored in buffers allocated from the associated buffer pool. A value of 0 indicates the largest representable buffer size of 64KB. SEC supports the use of up to 2 buffer pool records. The size of buffers in pool 1 must be same or smaller than the size of buffers in pool 2 (if the pool 2 record is valid (BPV=1)).
PTAR	The Pass Through Annotation Room field is part of the SP parameter group. PTAR is encoded in terms of 64-byte and its value is used to optionally generate an additional space of 64 bytes for an anticipated pass through annotation in the front of the frame. The generation of PTAR may be requested even if the entity allocating the buffer does not generate any PTA content. In the latter case FD[PTA] must be set and FD[PTV1/2] must be clear to indicate that the PTA field content is invalid. A description on how PTAR is used can be found in the ASAR field description above. Note, in the input buffer reuse case (BS=1) PTAR is ignored (because the PTAR room must be preserved according to the frame format specification and a change of FD[PTA] would require to move any following ASA in the spirit of reuse) and should be set to 0 to enable alternative use in future SP format extensions.
RBMT	The Response Bypass Memory Translation bit is part of the user response parameter group. RBMT is used to configure a user selected CBMT value associated with the user selected Response FLC (see RFLC described below). The CBMT attribute in combination with interpreting RFLC as a pointer can be used to stash up to 3 64-byte blocks of user context data (if enabled via the SC and CS fields in the FD) when QMan de-queues SEC's service response FD. Setting RBMT/CBMT (to bypass memory translation) has no effect unless the

Table continues on the next page...

Table 6-2. Flow Context Field Descriptions (continued)

Field Names	Descriptions
	user is implicitly authorized to do so by having access to and using a frame queue with a set privileged level (PL=1). Note, RBMT is only utilized for SEC generated FDs directed to the input partition and if the service request was not a management command.
RFLC	The Response FLC field is part of the user response parameter group. RFLC is used to configure a user selected FLC field returned to the user in SEC's service response FD. Note, the RFLC field is fully determined by the service user unless RSC is set. If RSC is set, the LS 6 bits of the RFLC field define QMan stash controls and need to comply with the DPAA FD format specification. Note, the RFLC field is only utilized for SEC generated FDs directed to the input partition and if the service request was not a management command.
RSC	The Response FLC Stash Control bit is part of the user response parameter group. RSC is used to configure a user selected QMan Stash Control (SC) associated with the user selected LS 6 bits of the Response FLC (RFLC) field. Per DPAA FD format specification, a set SC bit enables QMan to interpret the LS 6 bits of the FLC field to control up to 3 64-byte blocks of frame data, context, and/or annotation data to be stashed (loaded in a cache for CPU consumption) when a user de-queues such an FD from a QMan software portal. For more information on SC and related FD fields see the DPAA overview chapter in your SoC's reference manual. Note, RSC is only utilized for SEC generated FDs directed to the input partition and if the service request was not a management command.
SDID	The Security Domain ID field is part of the user authentication parameter group. The SDID field is only used when AUC=3 and the Frame Queue Descriptor Context Format (FQD[FF]) passed via the QMan dequeue summary is set to 1. For all other AUC codes the SDID is assumed to be equal to the input partition ICID if FQD[FF] = 0 and the value stored in the ICID_3 if FQD[FF] = 1. For more information on the use of SDIDs see Trusted descriptors
SDL	The Shared Descriptor Length is part of the job scheduling and processing parameter group. SDL is encoded in terms of 32-bit job descriptor command words and serves a dual purpose: SDL is used to define both the length of an optional (but usually present) SD as well as the entry point of an optional CRJD. For example, SDL=10 defines both a shared descriptor with 10 words and a CRJD entry point at offset 40 relative to the start address of the SD (which is assumed to start at offset 64 relative to the address defined by FLC.) SDL=0 indicates there is no SD and an in-line descriptor is expected at the start of the input frame. See Using in-line descriptors
SGHR	The Scatter/Gather Head Room bit is part of the SP parameter group. SGHR is encoded in terms of 64-byte and its value is used to optionally generate an additional space of 64 bytes for anticipated SG table growth. If the SG buffer frame format is selected SGHR is applied only to the first SGT buffer. A description on how SGHR is used can be found in the ASAR field description above. Note, in the input buffer reuse case (BS=1) SGHR is ignored and should be set to 0 to enable alternative use in future SP format extensions.
SR1 / SR2	The Scarce Resource attribute bits are part of the buffer pool control record 1 and 2 in the SP parameter group. The SRn setting indicates that the amount of memory consumed by the pool buffer need to be reflected in the FD[MEM] encoding and FDs conveying such buffers need to set the SL bit in FD[FMT], reduce the FD[LENGTH] encoding to 18 bit and make room for a 12 bit FD[MEM] encoding. If the buffer is utilized in an SG FD the SGE entry's SL bit in SGE[FMT] and the SGE[SR] bit will be set. The FD[MEM] encoding is utilized by QMan to support resource based congestion management features. For more information see the QMan chapter of the DPAA2 reference manual.

6.3.2 Processing single frame jobs

Two single-frame formats are defined, single-buffer and multi-buffer (or scatter-gather) frames. Single buffer frames contain an address that points directly to a buffer that contains data or space for data. Multi-buffer frames contain an address that points to a

scatter/gather table (SGT). The SGT contains one or more entries that can be marked as unused, point to single buffers, point to a data section within the SGT, or point to an SGT extension. Both formats support short and long length coding options:

- Short length frames support an 18-bit data length and a 12-bit memory footprint encoding.
- Long length frames support a 32-bit data length and no memory footprint encoding.

Preparation for processing a single-frame usually involves preparing space for output. The Storage Profile (SP) prescribes whether QI and AI should allocate a new single-frame for output, prepare to reuse the input frame for output, or prepare for no output. When a new frame is allocated for output, SEC will release all input frame buffers.

Compared to the generic SP features described in the DPAA2 reference manual, the following SEC limitations apply to SP parameter programming:

- 1 or 2 buffer pool records may be utilized
- If only 1 pool is provided, pool record 1 must be used to define the pool characteristics
- BDPn=1 and BPn=1 in buffer pool records are ignored and not supported

The following highlights the steps QI and AI perform to create the optional, but usually present, output frame buffer structure(s):

1. In input buffer reuse mode (BS=1) or if the dequeued frame is a frame list, no buffer pool is used (and no pool records are checked for validity)
2. In output buffer allocate mode (BS=0) buffer pool record 1 must be valid (BPV1=1) even if other SP parameters or the estimated output length requirements indicate that no output buffer is needed (buffer pool record 2 may or may not be valid and may be utilized as described below).
3. If pool record 2 is valid (BPV2=1), the buffers in the pool 2 must be same size or larger than the buffers in pool 1 (PBS2 >= PBS1).
4. If single-buffer or scatter/gather-buffer selection is set to automatic (FF=0) QI and AI will first calculate the space needed for single buffer annotation(s), and/or data headroom, and data
 - a. If the space requirement is met by the size of buffers in pool 1, allocate 1 buffer from pool 1.
 - b. else, if buffer pool record 2 is valid and the space requirement is met by buffers in pool 2, allocate 1 buffer from pool 2.
 - c. else, attempt to allocate a scatter/gather frame by
 1. first calculating the space needed for data headroom and data
 - a. If the space requirement is met by buffers in pool 1, allocate 1 buffer from pool 1 for data headroom and data.
 - b. else, if pool 2 is valid and the space requirement is met by buffers in pool 2, allocate 1 buffer from pool 2 for data headroom and data.

- c. else, allocate 2 or more buffers from the pool with the largest buffers for data head room and data.
 2. and second, calculating the space needed for scatter/gather annotation, headroom, and table entries
 - a. If the space requirement is met by buffers in pool 1, allocate 1 buffer from pool 1 for the scatter/gather table
 - b. else, if pool 2 is valid and the space requirement is met by buffers in pool 2, allocate 1 buffer from pool 2 for the scatter/gather table.
 - c. else, allocate 2 or more buffers from the pool with the largest buffers and use SGT EXT entries to link the tables together.
5. If force single-buffer is selected (FF=2) QI and AI will calculate the space needed for annotation, and/or data headroom and data
 - a. Same as 4.a above
 - b. Same as 4.b above
 - c. If a single buffer does not provide the required space, report an error that a single buffer frame cannot be generated.
6. If force scatter/gather-buffer is selected (FF=3) QI and AI will follow the steps outlined in 4.c above.

6.3.3 Processing frame list jobs

The input/output variant of generic frame lists is used to associate and convey an input and an output frame. Although the software overhead of using frame lists is somewhat higher than using single frames, the input/output frame list enables the use of any user-defined output frame structure and/or to preserve the input frame instead of reusing or releasing it. The output frame structure generated by QI and AI is either a single- or a multi-buffer frame (the inclusion of pass-through annotation, application-specific annotation, and data headroom is supported). But some applications may require a more complex structure with multiple levels of extension and unused entries to facilitate expansion. Input/output frame lists also provide a way to preserve the original data, rather than over-writing it or releasing input frame buffers. When SEC is done processing, it returns the input/output frame list without releasing any of the input frame buffers. This may be useful for multi-cast and retransmission scenarios and for system debug. Software becomes responsible for management of the input and output buffers.

The FD for an input/output frame list points to a two-entry Frame List Table (FLT), where the first entry specifies the output frame and the second entry specifies the input frame. Each FLT entry is similar to the generic DPAA FD format which may point to a single-buffer or a multi-buffer frame. See the DPAA overview chapter in your SoC reference manual for a generic description of the frame list format.

The following use modes of the input/output frame list are possible:

- Output frame address = X, Input frame address = Y. The output is written to the frame at address X. Software must ensure that the total size of the supplied output frame is large enough to hold SEC's output.
- Output frame address = Y, Input frame address = Y. The output overwrites the data in input buffers. Software must ensure that the data to be read in has, in fact, been read in before it is overwritten. For example, the descriptor could output a protocol header before reading the input frame. As a result, the header could overwrite the input frame before the input frame is ready. One easy way to avoid this is to use a non-zero offset for the input FD but have the output FD use a smaller offset. Note that software must also ensure that the total size of the supplied input buffer(s) is large enough to hold SEC's output.

6.3.4 Frame descriptor error handling

SEC can experience errors while accessing FLCs and frame lists, interpreting input FDs and FLC parameters, allocating new frames for output or preparing input frames for reuse, processing input and output frame data, and generating response FDs, as well as updating frame list output frame entries. Allocation errors may be due to buffer pool depletion in BMan. As explained in earlier chapters, SEC reports errors using the FD FRC field. For a full list of detected error conditions see [Job termination status/error codes](#).

Some errors are detected during QI's or AI's preparation of a job for processing, including FD format errors, FLC configuration errors, memory access and buffer pool depletion errors. When these errors occur, QI and AI set the Do Not Run (DNR) bit in the job descriptor HEADER command to instruct DECO not to execute shared descriptor commands. When DECO gets the job, it may only release input buffers (if RBS is set in the SEQ IN PTR command), and, if present, DECO will read the shared descriptor and check whether to propagate and update the DNR setting in the shared descriptor HEADER command if the PD bit is set in the shared descriptor. In either case, no additional job processing occurs and QI or AI report the detected error.

If SEC processing requires only a single output buffer and the buffer could not be acquired (BMan reports pool depletion), QI or AI will respond by returning the input frame to the service user with a buffer pool depletion error status in the FRC field of the FD.

When multiple output buffers are required (or if the service user requested the unconditional generation of an SGT), QI or AI checks whether the buffers in the buffer pool assigned to provide SGT buffers are large enough to accommodate the scatter/gather

annotation, headroom, and an SGT EXT. If not, no buffers will be acquired and QI or AI will return the input frame to the service user with an SGT exceeds max buffer size error status in the FRC field of the FD.

Further multi-buffer frame error handling depends on whether part of the additional frame data buffers were acquired and associated SGT entries were already written to memory when the error condition is detected. If no part of the SGT has been written to memory, all buffers acquired for output are released and the input frame will be returned to the service user with the detected error condition reported in the FRC field of the FD. If part of the SGT entries have already been written, that partially generated SGT buffer and the associated data buffers will not be released, the last utilized SGT entry will be rewritten to set the F (final) bit, and the partially generated (and empty) frame will be passed to the service user with LENGTH set to 0 and a non-0 status reporting the detected error condition. Any buffers acquired from BMan that have not yet been added as entries to the partially generated SGT buffer are released back to BMan.

Note that all QI or AI detected frame write errors indicate that some or all of the content of the service user received multi-buffer SGT or frame list table is at least partially invalid, associated buffer attributes (including the buffer address) need to be treated as suspect, and thus the information conveyed by such FDs is merely useful for debug and analysis.

6.3.5 Job descriptor construction from frame descriptor

QI and AI build an internal SEC Job Descriptor (JD) to process service requests conveyed via DPAA Frame Descriptors (FDs). The precise length of the generated JD depends on the utilized memory address size and the inclusion of an optional load command. If SEC is configured to use 32-bit addresses, the generated JD is 8 words long without the load command or 10 words long with it. If addresses are greater than 32 bits, the JD is 11 words long without the load command or 13 words long with it. Note that the sum total of words for the internally generated JD and an optional (but usually present) Shared Descriptor (SD) must not exceed 64 words, i.e., the maximum size of an SD utilized by QI or AI depends on the size of the internally generated JD, but it is always safe to limit the SD length utilized with QI or AI to 51 words or less.

The internally generated JD is constructed from the following DECO commands:

HEADER
Shared Descriptor Pointer (1 word for 32-bit addressing, 2 words otherwise) (See Address pointers.)
SEQ OUT PTR
Output Pointer (1 word for 32-bit addressing, 2 words otherwise)

Table continues on the next page...

Output Length
SEQ IN PTR
Input Pointer (1 word for 32-bit addressing, 2 words otherwise)
Input Length
LOAD immediate (optional)
Immediate data value (FD FRC field) to be written to DPOVRD Register (optional)

If there is no SD (the Flow Context defines SDLEN as 0), the START WORD field in the HEADER command is set to 2 or 3 to skip over the 1 or 2 words reserved for the SD pointer.

The Frame Context (FRC) field in the Frame Descriptor (FD) enables software to request management services or to modify the processing of individual frames while retaining the advantages of sharing SEC services between many users. The 8 most significant bits of the FD's FRC field are interpreted as follows:

Table 6-3. SEC Frame Descriptor Frame Context (FRC) Field Options

MS byte of FRC Field	Description
0000_0000b	Generate internal default JD as described above
0000_0001b	Reserved (unsupported command error)
0000_0010b	Generate internal default JD and set RJD bit in SEQ IN PTR command (Execute a Replacement Job Descriptor (RJD) stored inline in input buffer)
0000_0011b	Generate internal default JD and set RJD and CTRL bits in SEQ IN PTR command (Execution a Control RJD (CRJD) stored immediately following the SD)
0000_0100b : 0000_0111b	Reserved (unsupported command error)
0000_1000b	Input ICID flush command (wait for unfinished jobs to complete)
0000_1001b	Input ICID flush command (abort any unfinished jobs)
0000_1010b	QI: Input queue flush command (wait for unfinished jobs to complete) AI: Reserved (unsupported command error)
0000_1011b	QI: Input queue flush command (abort any unfinished jobs) AI: Reserved (unsupported command error)
0000_1100b	Input flow context flush command (wait for unfinished jobs to complete)
0000_1101b : 0000_1111b	Reserved (unsupported command error)
1000_xxxx b	Generate internal default JD and add LOAD Immediate command with immediate data equal to FRC field value at end of the internally generated JD (optional in internal JD illustration above)

If the most significant byte (MSB) of the FRC field is zero SEC generates the internal default JD as described above (without the optional LOAD Immediate command).

If the MSB is 0x02 or 0x03 SEC generates the internal default JD and sets the RJD bit in the SEQ IN PTR command, which tells DECO to replace the internally constructed default JD with a Replacement Job Descriptor (RJD). If the MSB in FRC is set to 0x02 the RJD is fetched from the beginning of the frame input buffer(s). If the MSB is set to 0x03 a Control RJD (CRJD) is fetched from the address following the SD (derived from the FLC address and the length of the SD encoded in FLC.SDL). Note, the CRJD bit must be set in the FLC to enable selection and execution of a CRJD following the SD. Also, if a DECO Protocol Override is required, the user must place the LOAD Immediate command in the (C)RJD. The sum total of words for the SD and the (C)RJD may not exceed 64 words.

If the MSB of the FRC field has one of the values 0x08 through 0x0C (excluding 0x0A and 0x0B for AI), SEC interprets the code as a flush command with the scope indicated in the table (ICID, frame queue, or flow context (excluding frame queue flush for AI)). The option to drain (complete pending requests) or abort (immediately terminate or cancel pending requests) is encoded by the least significant bit of the flush code (except for the flow context flush). Flush commands are intended to assist resource management and can be utilized by all users, with the exception of the ICID flush command, which is only available for privileged (trusted) users that can bypass isolation protection (FQD.BDI=1). For more information on flush commands see the DPAA overview chapter of your SoC reference manual.

If the MS nibble of the FRC field has the value 0x8, SEC adds a LOAD Immediate command to the end of the internally generated JD, with immediate data equal to the full 32-bit FRC field. The target of the LOAD is the DECO Protocol Override Register (DPOVRD). The effect of loading specific values into DPOVRD depends on the selected protocol and is described in detail in the protocol section of this document.

If there is no SD (FLC.SDL=0), QI and AI set the INL bit in the SEQ IN PTR command, which tells DECO to execute an Inline Job Descriptor (IJD) fetched from the beginning of input data stream immediately after the SEQ IN PTR command is executed, i.e., setting the DECO Protocol Override option will have no effect because the associated LOAD Immediate following the SEQ IN PTR command will not be executed. If needed, the LOAD Immediate command must be placed in the IJD.

Chapter 7

Descriptors and descriptor commands

Software's primary interaction with SEC is through the submission of descriptors. To make SEC flexible, the descriptor is a program that controls SEC's operation. It is therefore up to the user to provide meaningful descriptors for execution. Descriptors are submitted to SEC in order to process a job, where a job can specify a variety of functions supported by SEC, from initialization of a security parameter, to generation of a random number, to encryption or signing of data, or full security protocol encapsulation of a packet.

Descriptors consist of commands that are executed in sequence, although conditional and unconditional jumps are available to alter the sequence. The size of a single descriptor is limited to 64 32-bit words, but it is possible to jump from one descriptor to another so that, in effect, much larger descriptors can be created. Only the first of these descriptors has to be submitted by means of the job ring, the queue interface, or the AIOP interface; the rest are automatically fetched and executed by SEC.

Job descriptors, trusted descriptors, and shared descriptors can be modified and written back to memory. This is usually done when the processing of a data block is dependent on the result of processing of the prior data block. Such dependencies exist for information such as sequence numbers, counter values, and cryptographic state. Write backs are performed using descriptor commands. Hardware does not make independent decisions regarding the fields that should be written back.

Note that to correctly use sharing flows (wait or serial) in SEC, if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

When a job is submitted via the AIOP interface or the queue interface, the interface automatically creates a job descriptor. This job descriptor is built based on information provided via the submission. See [Job descriptor construction from frame descriptor](#).

7.1 Job descriptors

NOTE

The term 'job descriptor' refers to both control structures created by software and submitted to a Job Ring and to equivalent control structures created by QI or AI based on frame descriptor information.

A job descriptor (JD) is a control structure that causes SEC to execute a single job. Given a pointer to a job descriptor, the job queue controller will fetch from that address to the next burst boundary. If this is less than the number of bytes required to load an entire, maximum size, Job HEADER command, then the job queue controller will fetch the remaining required bytes. If the address size is 32 bits, then the maximum Job HEADER size is 12 bytes. If the address size is greater than 32 bits, then the maximum Job HEADER size is 16 bytes.

NOTE

As stated above, the first read is at least to the next burst boundary even though the descriptor may not be that long. It is up to the user to ensure that reading beyond the end of the job descriptor to the burst boundary will not result in any memory access errors.

If a second read is required to read the entire Job HEADER command, the second read is done from the first burst boundary to the second burst boundary. Note that these reads are all done with a single request and that request may be split under some conditions. Once these first words of the job descriptor are received, the job queue controller makes a decision. If there is a DECO available into which this job may be placed, the job is placed into the DECO for execution. If no such DECO is available, or if the job can't be placed into the DECO due to sharing constraints, the job queue controller will fetch the rest of the job descriptor if the previous reads did not already accomplish this. In addition, if there is a shared descriptor, the job queue controller will also fetch the shared descriptor unless it can be shared and is already present for another job. Once these reads have completed, the job will be eligible for placement into a DECO for execution. By prefetching all of this material, the job queue controller saves the DECO from taking the time to do so, thereby significantly improving performance. If the job queue controller is fetching a shared descriptor, it will also attempt to prefetch the input frame for AI and QI jobs. In the case where the input frame is a single buffer, the job queue controller will prefetch up to 128 bytes from the input frame, but no more than the length of the input frame. In the case where the input frame is a scatter/gather table, the job queue controller will prefetch the first 4 entries of the table. If the input frame is scattered and the output frame is also scattered, the job queue controller will prefetch the first 4 entries of the output table as well. Once these reads have completed, the job will be eligible for

placement into a DECO. Software-generated job descriptors contain the lengths and pointers (see [Address pointers](#)) of the data to be operated on, and either directly embed security keys and context or explicitly point to these keys and context. Keys and context can also be referenced indirectly by pointing to a shared descriptor (SD) that either contains keys and context, or includes pointers to keys and context. A job descriptor can include a shared descriptor by reference, but a shared descriptor cannot include a job descriptor.

Job descriptors use the descriptor commands defined in [Using descriptor commands](#). A job descriptor always begins with a HEADER command. A job descriptor without a shared descriptor typically includes:

- Commands that specify the inputs (such as keys, IV, or data) to a cryptographic operation and where to place them
- Commands that specify where to place the output(s) of the operation
- One or more OPERATION commands that specify the cryptographic work to be done

The job descriptor may also contain MATH commands that perform various calculations and conditional JUMP commands that branch based upon the results of those calculations.

If the job descriptor references a shared descriptor, the memory address pointer to the shared descriptor immediately follows the job descriptor HEADER. In this case the OPERATION command and certain inputs (such as the key) are normally specified in the shared descriptor. The job descriptor typically specifies the location of the memory buffers for the input and output data. In this case, the job HEADER command has the REO (Reverse Execution Order) bit set so that the job descriptor commands execute first (to specify the input and output data buffers), followed by the shared descriptor commands (to specify the operations to be performed on these data buffers). (see [Command execution order](#))

Because the length of the job descriptor is contained in the job HEADER command, no special termination command is required. When execution reaches the command which extends to the end of the job descriptor, DECO knows that the execution of the job descriptor has completed. Note that this endpoint is marked and does not change unless a new descriptor is loaded. Therefore, even if new descriptor material is loaded over the original material via MOVE or LOAD commands, the endpoint will not change and DECO will end execution of the job descriptor there. An error will be generated if DECO detects that the endpoint is inside a command. (For example, an error will be generated if the endpoint is between the words of a 2-word command.)

7.2 Trusted descriptors

A trusted descriptor is a job descriptor (possibly including a shared descriptor) that is integrity-checked at run time and is executed only if the check passes. This provides a mechanism to ensure that particularly sensitive operations are performed only by descriptors that were created by trusted software. Trusted descriptors have the following privileges not available to ordinary job descriptors:

- Access to trusted descriptor-only black keys (See [Black keys](#))
- Access to trusted descriptor-only blobs (See [Blobs](#))

Trusted descriptors allow trusted software to extend these privileges to untrusted software in a carefully controlled fashion. The trusted software can generate trusted descriptors that access specific privileged data objects in specific ways on behalf of specific requestors and deny access to other data objects, access modes, or requestors. Note that each Trusted Descriptor is associated with a particular SDID, and will run only if it is executed with the same SDID as the Job Ring in which the Trusted Descriptor was created. (The signature over the Trusted Descriptor will not validate if the SDID is different.) The Trusted Descriptor can be run in the Job Ring in which it was created, or another Job Ring, or can be run from the QI or AI, as long as the SDID is correct. The only exception to this is Trusted Descriptors created in a Job Ring owned by TrustZone SecureWorld. These "TrustZone Trusted Descriptors" can be run in any Job Ring or can be run from the QI or AI, and will assert the SecureWorld signal when accessing memory.

Trusted descriptors must be created, and are usually run, via jobs submitted via a Job Ring. To run a trusted job via AI or QI, one of three indirect methods must be used. The first method is to have the specified shared descriptor JUMP to the trusted descriptor. The second method is to submit an inline descriptor that is a trusted descriptor. Note that neither of the first two methods allows the trusted descriptor to have a shared descriptor. The third method is to use a replacement job descriptor or a control replacement job descriptor, which can have a shared descriptor.

Any descriptor can jump to a trusted descriptor via any of the various means: CRJD, RJD, nonlocal JUMP, or inline descriptor. However, while a trusted descriptor may use any of those means to jump, the target of such a jump must be another trusted descriptor. Otherwise, an error will be generated.

NOTE

In order to use the derived key protocol (DKP) in a trusted descriptor, the input and output for the protocol must both be via the sequence pointers. That is, the option selected must be from SEQ IN PTR to SEQ OUT PTR. There are no restrictions for other protocols.

7.3 Shared descriptors

Because descriptors can hold a lot of information required to process a job for a particular flow, they can be large, particularly if efficiency is maximized by placing the keys and other information within the descriptor rather than referencing them with pointers. To save overhead, SEC supports a shared descriptor mechanism. A shared descriptor is fetched once and held internally while it is used by several different related jobs. The keys and context information can also be shared among multiple descriptors. This saves bandwidth and latency, particularly when black keys are in use.

A shared descriptor (SD) is constructed with the expectation that it will be used for multiple jobs. The general usage model is to have a shared descriptor for each security session (for example, unidirectional IPsec tunnel). Every time a job related to that security session is required, SEC obtains job-specific information about the data (length, pointer) from the job descriptor and obtains its session context from the shared descriptor. Shared descriptors can store session state and can include commands to update this session state as needed. Shared descriptors are well suited for complex operations, as the software overhead of creating the shared descriptor is amortized over many individual jobs.

In order to optimize performance when a job descriptor references a shared descriptor, use the following guidelines. The job descriptor should contain only commands specific to one job in the sequence of jobs for which the shared descriptor will be used. Such commands include where to find the input data and where to place the output data. In addition, occasional tasks such as executing an RJD or overriding the normal operation of the shared descriptor would also be found here. The shared descriptor should contain all of the generic, flow-specific, commands. That is, references to keys, context, state, operations, etc. This is exactly the type of job descriptor automatically created by the QI and AI.

Job descriptors indicate the presence of an associated shared descriptor by setting the SHR bit in the job descriptor HEADER command. Software creates shared descriptors using the same command set as all other types of descriptors. A shared descriptor always starts with a shared descriptor HEADER.

The following restrictions are specific to shared descriptors:

- A shared descriptor cannot have its own shared descriptor.
- A shared descriptor can be, at most, 62 32-bit words if pointers are 32 bits and, at most, 61 32-bit words if pointers are larger than 32 bits. This limit is imposed because the job descriptor and the shared descriptor must both fit into the 64-word descriptor buffer (see [Figure 7-3](#)), and the minimum job descriptor consists of a one-

word job descriptor HEADER and a pointer to the shared descriptor (See [Address pointers](#)). Note that larger jobs can be created by JUMPing to another job descriptor.

- Some bits in the shared descriptor HEADER and the job descriptor HEADER commands differ.
- The creation of a trusted descriptor involves signing the entire job descriptor, including a referenced shared descriptor, if any. As a result, shared descriptors are signed as part of the job descriptor when creating trusted descriptors. Therefore the final signature is never part of a shared descriptor. Note that the REO bit cannot be set in a trusted descriptor.

The following figure illustrates two descriptors that reference the same shared descriptor.

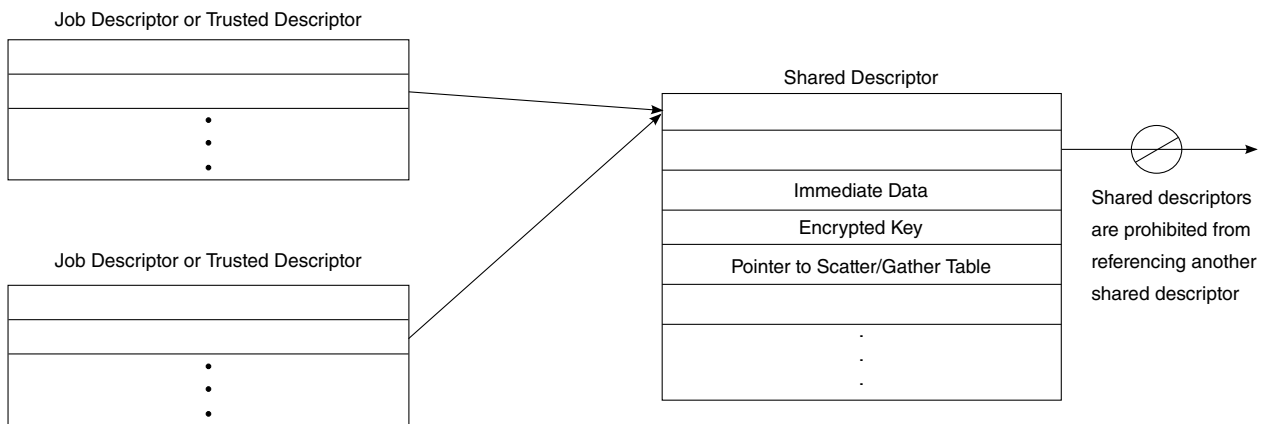


Figure 7-1. Two descriptors referencing the same shared descriptor

7.3.1 Executing shared descriptors in proper order

SEC provides mechanisms that can be used to ensure that jobs referencing the same shared descriptor execute in proper order. A shared descriptor may need to modify keys embedded within the descriptor, or particular fields of a protocol data block within the descriptor before a subsequent job uses the shared descriptor. Use the STORE command to update the shared descriptor. Note that to correctly use sharing flows (wait or serial), if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

NOTE

If a NEVER share shared descriptor is modified during execution, and that modification is not written back to memory,

the modification will NOT be seen by any other job which uses that shared descriptor. If a NEVER share shared descriptor is modified during execution, and if memory is updated with that change, subsequent jobs which reference that shared descriptor might have already fetched the original version or, if fetched during the update, might have a corrupted version of the shared descriptor. Therefore, it is up to the user to ensure that no jobs which use a NEVER share shared descriptor are in flight when the shared descriptor is updated. Clearly, NEVER share shared descriptors are not meant to be updated.

When a shared descriptor uses sequences, the sequence definitions should be in the job descriptor because the definitions can change from job descriptor to job descriptor. In such cases, set the REO bit in the job descriptor header. Note that setting the REO bit in the job descriptor header tells SEC to execute the job descriptor before the shared descriptor.

The sharing type may be changed to NEVER via a write to the DECO Control Register. Doing so prevents the descriptor from being shared from the DECO. The descriptor could be shared following a subsequent read from memory or from another DECO if that DECO has already gotten a copy of the descriptor. If the descriptor is being shared at the time the DECO Control Register is written to set the sharing type to NEVER, the descriptor will be shared.

SERIAL sharing can only take place by sharing the shared descriptor executing in a DECO with the next job to execute in that same DECO. For the same SERIAL shared descriptor to execute in a different DECO, it must be refetched from memory because it could not be shared with its prior execution.

A WAIT shared descriptor may be shared into the same, or a different, DECO. However, any instance of a WAIT shared descriptor may only be shared once. For example, if there are 3 jobs, X, Y, and Z, to be executed using the WAIT shared descriptor, once X has shared to Y, X can't share to Z. So, the sharing would be from X to Y, and then Y to Z. In other words, the WAIT shared descriptor may be shared many times, but each job descriptor can share it only once.

An ALWAYS shared descriptor may be shared as many times as necessary from the same job descriptor. This is because ALWAYS shared descriptors are stateless so that the order in which they are shared is not important. In the example above, X could share to both Y and Z.

7.3.2 Specifying different types of shared descriptor sharing

If two jobs are to be processed for the same data flow, they can share flow-specific data by referencing the same shared descriptor, which would be written to either reference or embed the flow-specific data. Sharing can be in parallel, i.e. two or more DECOs processing jobs using the same shared descriptor at the same time, or sharing can be sequential, i.e. a single DECO uses the same shared descriptor to process several jobs in a row without refetching the shared descriptor. "Self sharing" occurs when a descriptor is shared back into the same DECO. This can happen with "WAIT", "SERIAL", and "ALWAYS" sharing.

SEC distinguishes shared descriptors from each other by the address and ICID used to fetch the shared descriptor.

To share shared descriptors, the SHARE bits in the job descriptor header, and sometimes in the shared descriptor header itself, must be set. This lets SEC know under which circumstances the shared descriptors can be shared.

The following table shows the sharing possibilities supported by the HEADER command. The full details of the Shared Descriptor HEADER command can be found in [HEADER command](#).

Table 7-1. Interpretation of the SHARE fields

SHARE Name	Job Descriptor SHARE (binary)	Shared Descriptor SHARE (binary)	Description
NEVER	000	00	Never share the shared descriptor. Descriptors can execute in parallel, so no dependencies are allowed between them. Fetching the shared descriptor is repeated.
WAIT	001	00	Share the shared descriptor once set up has completed and processing has begun. Sharing can begin after a LOAD Command (or a PROTOCOL OPERATION) has set the OK to Share bit. Class 1 and Class 2 Key Registers are shared if valid.
SERIAL	010	00	Share once the descriptor has completed. The descriptor with which this should be shared will execute in the same DECO/CCB. Class 1 and Class 2 Key Registers are shared, if valid. Context may optionally be shared.
ALWAYS	011	00	Always share the shared descriptor, but keys are not shared. No dependencies can exist between the descriptors.
DEFER	100	00: NEVER 01: WAIT 10: SERIAL 11: ALWAYS	Use the value of the SHARE bits in the shared descriptor to determine the type of sharing.
All other combinations are reserved			

7.3.2.1 Error sharing

Shared descriptors can be marked as:

- NEVER
- WAIT
- SERIAL
- ALWAYS

When the job descriptor or shared descriptor is marked SERIAL or NEVER, no sharing can take place between DECOs. In the SERIAL case, only one DECO at a time is allowed to have a copy of the shared descriptor. In the NEVER case, each DECO receives a new version of the shared descriptor read from memory each time it runs a job referencing that shared descriptor. Note that the shared descriptor being marked as NEVER indicates that it is stateless (contains no information requiring update upon completion of a job). If a DECO reports an error while using the shared descriptor, there is no need to report that error to any other DECOs using an independent copy of the shared descriptor.

In cases where shared descriptor sharing occurs between DECOs, the first DECO to fetch the shared descriptor is the supplier DECO, and other DECOs using shared descriptor information from the supplier DECO's descriptor buffer are the consumer DECOs.

When the descriptor is shared between two jobs which run sequentially in the same DECO, no errors can be propagated.

In the ALWAYS case, errors do not propagate from supplier to consumers. If a supplier DECO has already started sharing the shared descriptor when an error occurs, the consumer DECO's job can complete normally regardless of the presence of an error in the supplier DECO.

In the WAIT case, an error in the supplier DECO can propagate to the consumer DECO while the shared descriptor and keys, if any, are being shared, causing both jobs to terminate with errors. The DECO Control Register can be written with value 0200h (that is, OFFSET = 02h and LENGTH = 00h) to enable error propagation, or 0300h (that is, OFFSET = 03h and LENGTH = 00h) to block error propagation (see value 06h, class 11 in [Table 7-17](#)). Using either of these values sets "OK to share" and tells the supplier DECO to propagate its shared descriptor and keys to the consumer DECO. Once the shared descriptor and keys (if any) have been copied to the consumer DECO, errors in the supplier DECO no longer affect the consumer DECO.

7.3.3 Changing shared descriptors

The best shared descriptors are independent, meaning that they do not need to be modified by software for each job descriptor with which they are used. (Note that this is a different topic than shared descriptors which update themselves.) Shared descriptors are more easily used the more generic they are. However, shared descriptors may have to be changed on occasion; for example, when there is a key change. Replacement job descriptors (see [Using replacement job descriptors](#)) can be used for such changes to avoid requiring software to make the change.

7.4 Using in-line descriptors

In the typical use case, the shared descriptor contains the main processing sequence. However, by setting the INL bit in a SEQ IN PTR command and providing appropriate address and length information, SEC is directed to an in-line descriptor, which is a job descriptor that software prepends to the data defined by an input sequence. (For more information about the SEQ IN PTR command, see [SEQ vs non-SEQ commands](#) and [SEQ IN PTR command](#).)

Note that shared descriptors can point to in-line descriptors, but in-line descriptors cannot point to shared descriptors. This means that the in-line descriptor is loaded at the start of the descriptor buffer, overwriting as much shared descriptor, if one is present, and job descriptor, as needed. This means the shared descriptor will no longer be executable by this job and will no longer be shareable. Note that an in-line descriptor may be scattered by means of an SGT.

If SEC services are accessed via the Queue Manager or AIOP, in-line descriptors are a method for instructing SEC to perform special processing on frames in a given flow. This is done by setting the shared descriptor length field to 0 in the flow context for the flow. In that case, QI or AI will build the job descriptor for those frames with INL set in the SEQ IN PTR command.

Prepending the in-line descriptor to the input data can be accomplished in two ways:

- Exploiting empty space in the buffer provided by the upstream frame producer
- Creating the in-line descriptor in a new buffer, which is then placed at the head of the scatter/gather list for a multi-buffer frame descriptor.

Once the inline descriptor has been loaded, the detection of an error will result in a bit in the completion status being set which indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For Job Ring jobs, the original job descriptor address is placed in the appropriate output ring.

Once an inline descriptor has been loaded, use of the STORE convenience sources for updating the job descriptor (41h and 45h) will result in an error.

In some cases, an inline descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

Note, in multi-ICID use cases, inline job descriptors may only be used when the input and the control ICID are the same.

7.5 Using replacement job descriptors

A replacement job descriptor (RJD) is an in-line descriptor that:

- Replaces the job descriptor that invoked the replacement descriptor. If SEC services are accessed via the Queue Manager, the internal job descriptor generated by the Queue Manager Interface (QI) is replaced. If SEC services are accessed via the AIOP, the internal job descriptor generated by the AIOP Interface (AI) is replaced.
- Does not replace the existing shared descriptor

To invoke the replacement job descriptor, execute a SEQ IN PTR with RJD = 1. This immediately executes the replacement job descriptor. Note that the replacement job descriptor must be at the start of the input sequence data at the time that this SEQ IN PTR command is executed.

The replacement job descriptor can modify the shared descriptor before allowing it to execute. This allows operations such as changing the keys and resetting the sequence number within a shared descriptor, such as for an IPSEC PROTOCOL OPERATION), without having to interrupt the flow of packets. However, because the shared descriptor has already been loaded, the length and address of the shared descriptor must not be modified. Note that when there is no shared descriptor, there is no difference between an in-line descriptor and a replacement job descriptor.

When using the replacement job descriptor capability, the current job descriptor can be replaced with any job descriptor. Other data, including an input frame, can follow the replacement job descriptor in the input sequence data. For example, an IPsec flow can modify the keys or sequence number and then immediately process the packet which follows the replacement job descriptor.

If there is a JUMP HALT command in the replacement job descriptor, the job terminates without executing the shared descriptor. Otherwise, if the job descriptor has the REO bit set (jobs from QI and AI always do), once the replacement job descriptor has finished,

execution continues with the shared descriptor so that data can be processed. If the shared descriptor will process data during this job, before beginning that processing make sure that all the updates made to the shared descriptor have completed both internally and externally (that is, the update to the descriptor buffer has completed and the update to the shared descriptor in memory has completed). This is discussed in the following two paragraphs.

The replacement job descriptor can insert new values in the shared descriptor with either the MOVE command or the LOAD command. The MOVE command's default behavior is to schedule the MOVE operation as soon as possible and then allow the next command to execute. As a result, the MOVE happens in parallel with subsequent commands. Be aware that the MOVE command can take multiple cycles to complete, and it is possible that shared descriptor commands may be executed before the MOVE completes. This could result in the intended updates not being used. If there is a chance that this may occur, use the WC bit in the MOVE command to ensure correct operation. See [MOVE](#), [MOVEB](#), [MOVEDW](#), and [MOVE_LEN](#) commands for additional details about the MOVE command.

If using the LOAD command to modify the shared descriptor, the replacement job descriptor should use the JUMP command, waiting for the NIP (No Input Pending) bit to evaluate true before proceeding. Note that the replacement job descriptor can also be used to transfer data to other destinations, such as memory, context registers, or Math registers. It is the replacement job descriptor's responsibility to ensure that any and all of these transfers have completed before the shared descriptor uses the new data.

Replacement job descriptors can be trusted descriptors, and they must be trusted if the current descriptor is a trusted descriptor. Although trusted descriptors cannot be run directly through QI or AI, trusted descriptors can be indirectly executed using a replacement job descriptor.

SEC also implements a different type of RJD known as a Control RJD (CRJD). In the case of a CRJD, SEC fetches the replacement job descriptor from memory immediately following the shared descriptor. Note that it is an error to use a CRJD for a Job Ring job if there is no shared descriptor.

Note, in multi-ICID use cases, an (in-line) RJD may only be used when the input and control ICID are the same. A CRJD may be used in all multi-ICID use cases.

Once an RJD has been loaded, use of the STORE convenience sources for updating the job descriptor (41h and 45h) will result in an error.

The detection of an error will result in a bit in the completion status being set which indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For Job Ring jobs, the original job descriptor address is placed in the appropriate output ring.

In some cases, an RJD descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

Due to features of the AES encryption algorithm, special handling may be required when using a replacement job descriptor to update a key in a shared descriptor. AES encryption requires each block of data to be processed in a series of cryptographic rounds, and the AES key is successively modified at each round. When decrypting, the AES CHA must start with the fully modified form of the key (also called a decryption key or decap key) and reverse the modifications at each round, eventually ending up with the original encryption key. If the descriptor was shared, AES will have left the decryption key in the key register. However, when using a replacement job descriptor to update a key in a shared descriptor, the updated key is usually an encryption key (also called an encap key). To resolve this problem, use the LOAD command to clear the fact that the shared descriptor was, in fact, shared. That way, AES will expect the encryption key and will automatically generate the decryption key. This avoids having to generate the decryption key as part of the RJD.

7.6 Scatter/gather tables (SGTs)

When submitting jobs to SEC, software can create job descriptors or frame descriptors with address/length entries that point directly to data or indirectly to data by means of scatter/gather tables. An SGT consists of one or more SGT entries. The final entry in the SGT is marked by setting the F (Final) bit in an SGT entry.

NOTE

SGT entries are read from memory four entries at a time. If the number of entries in the table is not a multiple of four, then the read of the entries will go beyond the end of the table. The user is responsible for ensuring that such reads beyond the end of the table do not result in memory access errors.

The following figure illustrates how an SGT might be used in a frame descriptor.

Scatter/gather tables (SGTs)

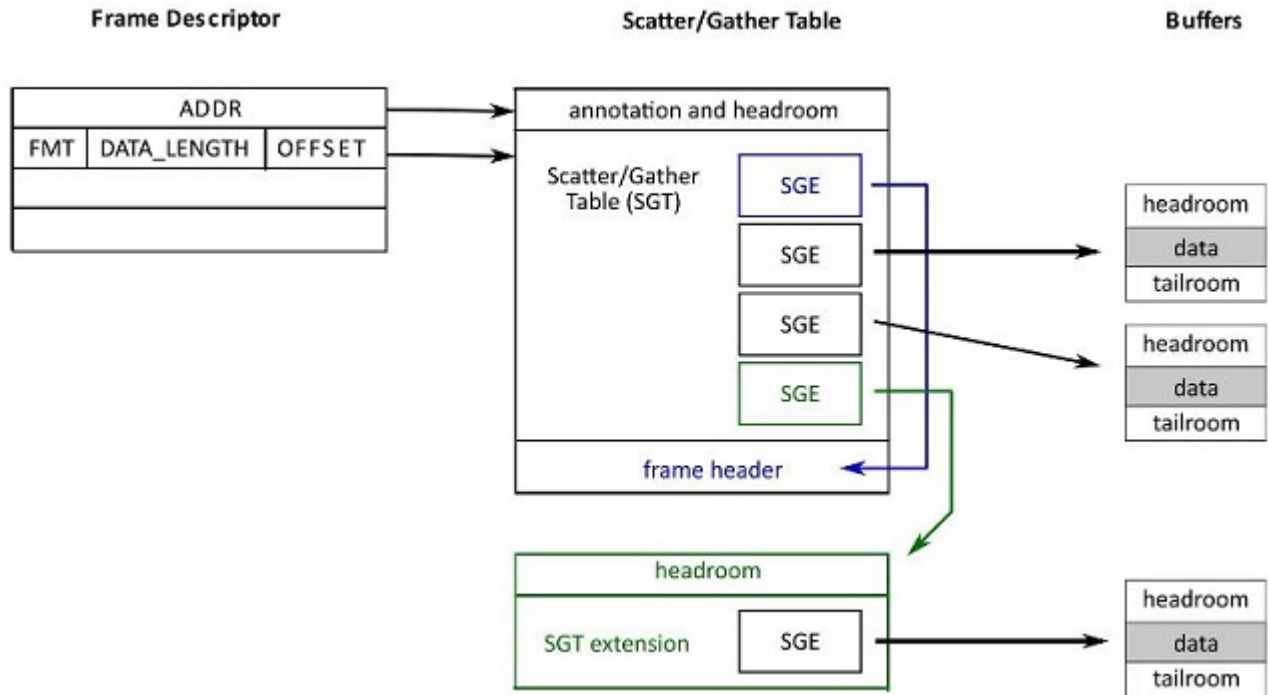


Figure 7-2. Layerscape Scatter-Gather Table Example

The following figure illustrates the format of the Scatter-Gather Table Entries.

First Dword		63-49				48-0				
		SOFTWARE_TOKEN (15 bits)				(physical or virtual) ADDR (49 bits)				
Second Dword	format when SL=0	6 3	62 SL=0	61-60 FMT	59-48 OFFSET (12 bits)	4 4 7 6 BI MV TP	45-32 BPID (14 bits)	31-0 AVAIL_LENGTH (32-bits)		
	format when SL=1	F	SL=1	FMT	OFFSET (12 bits)	BI MV TP	BPID (14 bits)	31-22 BUFF_SIZE (10 bits)	22 10 SS RF	19-17 reserved

Table 7-2. Scatter/gather table field descriptions

Field	Description
SW_TOKEN	Software may use these 15 bits as a token associated with the buffer, to assist in physical to virtual address translation or to serve other needs.
ADDR	49-bit address pointer. The type of data pointed to is indicated by the FMT field.
F	Final Bit. If set to 1, this is the last scatter/gather entry in the scatter/gather table.
SL	Short Length.

Table continues on the next page...

Table 7-2. Scatter/gather table field descriptions (continued)

Field	Description										
	<p>If SL=0 the AVAIL_LENGTH field is 32 bits and the BUFF_SIZE, SR and SF fields are not present in the SGE.</p> <p>If SL=1 the AVAIL_LENGTH field is 17 bits and the BUFF_SIZE, SR and SF fields are present in the SGE.</p> <p>NOTE: The SL bit must be 0 if FMT = 11b. (The SL=1, FMT=11b combination is reserved for future FMT extension).</p>										
FMT	<p>Target Format. FMT specifies the type of data pointed to by ADDR.</p> <table border="1"> <thead> <tr> <th>Value of FMT field</th> <th>Type of data pointed to by ADDR field</th> </tr> </thead> <tbody> <tr> <td>00b</td> <td> <p>Single Data Buffer.</p> <p>ADDR points to a buffer containing frame data. Available length and offset are encoded in the AVAIL_LENGTH and OFFSET fields.</p> </td> </tr> <tr> <td>01b</td> <td> <p>Frame Data Section</p> <p>ADDR points to a section reserved for frame data. The section is located in the same buffer as the SGE. The available length and offset of the frame data section are encoded in the AVAIL_LENGTH and OFFSET fields.</p> </td> </tr> <tr> <td>10b</td> <td> <p>SGT Extension.</p> <p>ADDR points to an extension of the SGT. Processing of SGEs continues with the SGE located at ADDR + OFFSET. Note that the AVAIL_LENGTH, F, and SF fields are ignored. The SL bit must be set to 1. The combination FMT = 10b and SL = 0 is reserved.</p> </td> </tr> <tr> <td>11b</td> <td> <p>Unused.</p> <p>All entries of this type are ignored while processing input frames. This space may be used to add new entries for output. The SL bit must be set to 0. The combination FMT = 11b and SL = 1 is reserved.</p> </td> </tr> </tbody> </table>	Value of FMT field	Type of data pointed to by ADDR field	00b	<p>Single Data Buffer.</p> <p>ADDR points to a buffer containing frame data. Available length and offset are encoded in the AVAIL_LENGTH and OFFSET fields.</p>	01b	<p>Frame Data Section</p> <p>ADDR points to a section reserved for frame data. The section is located in the same buffer as the SGE. The available length and offset of the frame data section are encoded in the AVAIL_LENGTH and OFFSET fields.</p>	10b	<p>SGT Extension.</p> <p>ADDR points to an extension of the SGT. Processing of SGEs continues with the SGE located at ADDR + OFFSET. Note that the AVAIL_LENGTH, F, and SF fields are ignored. The SL bit must be set to 1. The combination FMT = 10b and SL = 0 is reserved.</p>	11b	<p>Unused.</p> <p>All entries of this type are ignored while processing input frames. This space may be used to add new entries for output. The SL bit must be set to 0. The combination FMT = 11b and SL = 1 is reserved.</p>
Value of FMT field	Type of data pointed to by ADDR field										
00b	<p>Single Data Buffer.</p> <p>ADDR points to a buffer containing frame data. Available length and offset are encoded in the AVAIL_LENGTH and OFFSET fields.</p>										
01b	<p>Frame Data Section</p> <p>ADDR points to a section reserved for frame data. The section is located in the same buffer as the SGE. The available length and offset of the frame data section are encoded in the AVAIL_LENGTH and OFFSET fields.</p>										
10b	<p>SGT Extension.</p> <p>ADDR points to an extension of the SGT. Processing of SGEs continues with the SGE located at ADDR + OFFSET. Note that the AVAIL_LENGTH, F, and SF fields are ignored. The SL bit must be set to 1. The combination FMT = 10b and SL = 0 is reserved.</p>										
11b	<p>Unused.</p> <p>All entries of this type are ignored while processing input frames. This space may be used to add new entries for output. The SL bit must be set to 0. The combination FMT = 11b and SL = 1 is reserved.</p>										
OFFSET	The number of bytes from the start of the buffer (ADDR) to the start of significant data in the buffer. The use of an offset permits reuse of a memory buffer without recalculating the address.										
BMT	Bypass Memory Translation. Setting BMT=1 is an indication to the System MMU to bypass memory translation (i.e. treating ADDR as a physical address) whenever an accelerator is performing memory transactions in the buffer referenced by ADDR. As BMT may be set to 1 by an untrusted entity, when BMT=1 the System MMU checks the privilege level of the isolation context.										
IVP	Invalid Pool ID. If IVP is set to 1 it indicates that BPID does not encode a valid BMAN pool ID value and ADDR must not be released to BMAN using BPID.										
BPID	Buffer Pool ID. If IVP=0 BPID indicates the identifier of the buffer pool that owns the buffer referred to by ADDR. If IVP=1 BPID may be used by GPP software to carry a software-defined token associated with the buffer.										
BUFF_SIZE	BUFF_SIZE indicates the size of the target buffer, in 64-byte units. Value 0 is interpreted as 64KB, since buffers of size 0 are not supported. The field is only available in the short format and is valid only for single buffer (FMT = 00b) and SGT extension (FMT = 10b) entries.										
SR	Scarce Resource. SR=1 indicates that ADDR is pointing to a scarce resource buffer, i.e. the size of the buffer may be subject to scarce resource monitoring and is included in the FD or FLE MEM field. If SR=1 BUFF_SIZE will be subtracted from the FD or FLE MEM field if the buffer is removed from the S/G frame structure. If a buffer is added to the frame structure the SR bit will be set based on the storage profile/buffer pool attributes and the FD or FLE MEM field value will be updated accordingly. The bit is only available in the short format (refer to SL bit description). For long format SGEs it is considered reset. SR is valid only for single buffer (FMT = 00b) and SGT extension (FMT = 10b) entries. The SR bit in an extension entry refers only to the SG extension table buffer itself. Frame data sections located in current buffer and unused SGEs do not contribute to scarce memory footprint.										

Table continues on the next page...

Table 7-2. Scatter/gather table field descriptions (continued)

Field	Description
SF	Start of Fragment. The SF bit indicates that the data placed in the target buffer is the start of a new frame fragment. The field is ignored for SGT Extension entries and is only available in the short format (refer to SL bit description). Note that certain types of processing may change the structure of the frame, breaking the original fragment boundaries. In this case out of band information must be passed to the DMA engine to indicate that the SF bits in the S/G are no longer valid and must be ignored.
AVAIL_L ENGTH	Available Buffer Length. AVAIL_LENGTH indicates the number of bytes of data that may be placed in the target buffer (or section), starting from OFFSET. The size of the field is determined by the SL bit. This field is valid only for Single Buffer and Frame Data Section entries. It is ignored for the other SGE FMT types.

7.7 Using descriptor commands

Descriptors contain one or more commands that tell SEC what operations to perform, as well as what data on which to operate. Commands can also be used to enforce data type separation. For example, specifying that input data be treated as a cryptographic key forces SEC to treat it exclusively as a key and prevents the key from being written back out into memory in unencrypted form.

SEC permits a great deal of flexibility in composing descriptors, but it is highly recommended that descriptors be modeled after the examples in drivers or other reference software. Some sequences of commands or combinations of command options may produce unexpected results.

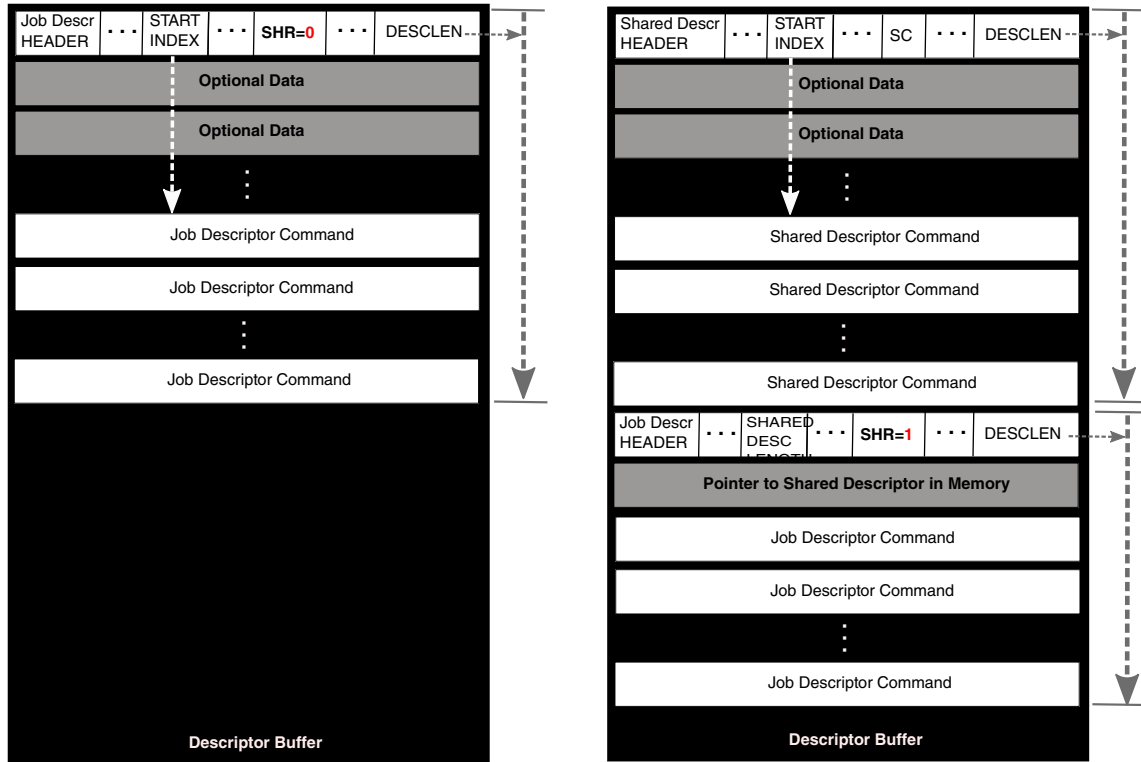
7.7.1 Command execution order

NOTE

In the following discussion, the term "Job Descriptor" should be taken to include both job descriptors and trusted descriptors.

Before a job descriptor begins execution, the portion of the job descriptor contained in the holding tank is loaded into the descriptor buffer. This includes the job descriptor's HEADER command (see [HEADER command](#)), which is the first command executed. Once the remainder of the job descriptor has finished loading, the next command to execute depends upon three fields in the HEADER: SHR, REO, and START INDEX.

The following figure shows the layouts for job descriptors depending on whether SHR = 0 or 1.



7.7.1.1 Executing commands when SHR = 0

When SHR = 0, the job descriptor does not reference a shared descriptor. Therefore, the HEADER's START INDEX field specifies the position of the next command that will execute within the job descriptor. If the START INDEX value is 0, the next command to execute is the command immediately following the HEADER command. Any other value causes a jump to the position indicated by the START INDEX field. Note that within a protocol job descriptor, the START INDEX value is used to skip over the PDB, if any. Before the job descriptor continues execution, the remainder of the job descriptor is fetched from memory and loaded into the DECO's descriptor buffer. The left half of [Figure 7-3](#) shows the layout of a job descriptor that does not reference a shared descriptor.

Commands execute in the order in which they appear in the descriptor buffer until one of the following is executed:

- The last command in the job descriptor
- A JUMP command when the JUMP is taken (see [JUMP \(HALT\) command](#))
- An in-line descriptor (see [Using in-line descriptors](#))
- A replacement job descriptor (see [Using replacement job descriptors](#))

When JUMP commands are executed, the behavior is as follows:

- If DECO executes an unconditional halt type of JUMP or a conditional halt type whose tested condition evaluates to true, execution of the job descriptor terminates.
- If DECO executes a JUMP whose type is conditional halt, local conditional jump, non-local conditional jump, conditional subroutine call, or conditional subroutine return and the tested condition evaluates to false, execution continues with the command following the JUMP.
- If the JUMP type is local or non-local jump or conditional subroutine call and the tested condition evaluates as true, the command indicated by the LOCAL OFFSET field (for local jumps) or by the Pointer Field (for non-local jumps) is the next command to execute.
 - If the jump is local, the target of the JUMP should be within the current job descriptor. If the target is beyond the end of the current job descriptor, it is up to the programmer to ensure there is executable code at the target and that the descriptor will be able to terminate properly. One common method for proper termination is to use a JUMP HALT command.

- If the jump is non-local, the target of the JUMP must be the start of a job descriptor.
- If the JUMP type is conditional subroutine return and the tested condition evaluates as true, the next command to execute is the command following the most recently executed conditional subroutine call.

7.7.1.2 Executing commands when SHR = 1

As described in [Executing commands when SHR = 0](#), the portion of the job descriptor (including the HEADER command) contained in the holding tank is loaded into the descriptor buffer. When SHR = 1, the job descriptor references a shared descriptor (see [Shared descriptors](#)).

In this case, instead of a START INDEX field, the job descriptor HEADER contains a SHR DESCR LENGTH field. This field specifies the length of the shared descriptor, which allows DECO to leave enough space for the shared descriptor when the job descriptor is loaded into the descriptor buffer. The right side of [Figure 7-3](#) shows the layout of a job descriptor that references a shared descriptor.

A pointer to the shared descriptor's location in memory appears in the word (or two words) immediately following the job descriptor HEADER. Note that the job descriptor HEADER may occupy two words in addition to the shared descriptor address (see EXT field in [HEADER command](#)). The pointer, together with the ICID that was used when fetching the descriptor, is used to determine if the shared descriptor is already resident in the same or another DECO and is therefore a candidate for sharing. If the shared descriptor is not resident or cannot be shared, the shared descriptor is fetched from memory using the pointer as the starting address. If the shared descriptor is available from another DECO, processing cannot continue until the shared descriptor (and the keys, if the keys are to be copied) has been copied from the other DECO. Processing cannot continue until the entire shared descriptor is present. The START INDEX field within the shared descriptor's HEADER specifies the position of the next command that will execute within the shared descriptor once the shared descriptor begins execution. If either the shared or job descriptor contain a PROTOCOL OPERATION command, the START INDEX value in the shared descriptor is used to skip over the PDB, if any. Note that when a shared descriptor is present, the PDB is always in the shared descriptor even if the PROTOCOL OPERATION is in the job descriptor.

7.7.1.3 Executing commands when REO = 0

If the job descriptor references a shared descriptor, the REO bit in the HEADER command determines the next command to be executed. ¹ When REO = 0, DECO executes the shared descriptor before the remainder of the job descriptor, as illustrated on the left of [Figure 7-4](#). After the job descriptor HEADER executes, the HEADER command within the shared descriptor (2.0 in the diagram) is the next command to execute. The commands within the shared descriptor then execute. Once the shared descriptor starts executing, any job descriptor HEADER command will be treated as a no-op until a new job descriptor is loaded.

The shared descriptor ceases executing when any of the following occurs:

- an in-line job descriptor is executed
- a replacement job descriptor is executed
- a JUMP HALT command is executed
- a non-local JUMP is executed
- or the shared descriptor "falls through" to the job descriptor

Since the shared descriptor immediately precedes the job descriptor in the descriptor buffer (see right side of [Figure 7-3](#)), unless the last command of the shared descriptor causes a jump, the shared descriptor may complete by simply "falling through" to the job descriptor. Once the shared descriptor completes, DECO executes the job descriptor, starting with the job descriptor HEADER, which will be treated as a no-op. Execution will continue with the next command of the job descriptor. Execution will end following execution of the last command in the job descriptor unless the last command is a taken JUMP.

Once the shared descriptor HEADER command has been executed, any further shared descriptor HEADER commands will be used as absolute, unconditional, jump commands if the START INDEX field is nonzero. The START INDEX field will be used to determine the target. Note that, unlike the JUMP command, the START INDEX is the value of the target index, not a relative index. No other fields in the shared descriptor HEADER command will cause an action to take place although error conditions may be triggered. If a subsequent execution of a shared descriptor HEADER command is done where the START INDEX is zero, then the shared descriptor HEADER command will be treated as a no-op.

It is important to note the difference in how subsequent job descriptor HEADER commands are handled when REO=0 and REO=1.

1. Note that the REO bit cannot be set in a trusted descriptor.

7.7.1.4 Executing commands when REO = 1

When REO is 1, DECO executes the job descriptor before the shared descriptor, as illustrated in the diagram on the right of [Figure 7-4](#). In this case, the job descriptor command (if any) that immediately follows the shared descriptor pointer (1.1 in the diagram), or the extended HEADER word if it is present, executes immediately after the job descriptor HEADER. After the job descriptor completes, DECO then executes the shared descriptor commands, starting with the shared descriptor HEADER (2.0 in the diagram).

Execution of a subsequent job descriptor HEADER, other than one reached via a non-local JUMP, an RJD or inline descriptor, will terminate execution normally. Upon execution of the command which ended the job descriptor, no matter how many times this occurs, with the exception of taken JUMPs, execution will continue with the command at the start of the descriptor buffer. After the first execution of the shared descriptor HEADER, subsequent executions can be used as absolute, unconditional, jumps in the same manner as subsequent shared descriptor HEADER commands are used when REO=0.

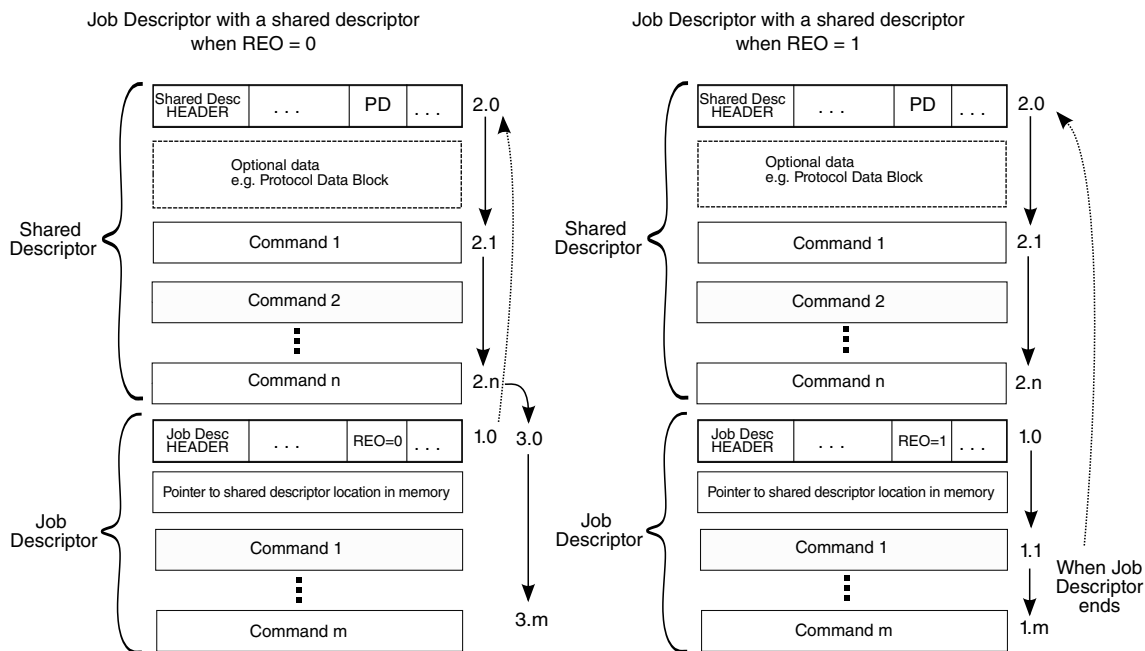


Figure 7-4. Order of command execution if a shared descriptor is referenced

7.7.1.5 Executing additional HEADER commands

A job descriptor must start with a job descriptor header, and a shared descriptor must start with a shared descriptor header. These are typically the only HEADER commands within a descriptor, but it is possible for the descriptor to have additional HEADER commands.

No error is generated if a job descriptor or shared descriptor executes additional shared descriptor HEADER commands. These are essentially no-ops, with one exception. If START INDEX is non-zero, the Shared Descriptor HEADER command causes a jump to that position within the descriptor buffer. That is, the Shared Descriptor HEADER command executes as if it is an unconditional JUMP to an absolute index. Note that this is different from the JUMP command, which uses relative addressing. The first shared descriptor HEADER command is the one that is treated as real. All subsequent shared descriptor HEADER commands executed, including the first one if executed again, are no-ops (other than an absolute jump if the START INDEX is nonzero).

NOTE

It is an error to execute a shared descriptor HEADER command in a job descriptor when there is no shared descriptor (when SHR = 0).

If a job descriptor does not reference a shared descriptor, any additional job descriptor HEADER commands that it executes (for example, by jumping back to the beginning of the job descriptor) are treated as jumps to an absolute address within the descriptor buffer. If a job descriptor does reference a shared descriptor, any additional job descriptor HEADER commands that it executes are treated as no-ops. Executing a job descriptor HEADER command within a shared descriptor terminates the shared descriptor if the shared descriptor runs after the job descriptor runs (that is, REO = 1), but the job descriptor header acts as a no-op if the shared descriptor runs before the job descriptor runs (that is, REO = 0).

7.7.1.6 Jumping to another job descriptor

Note that either a job descriptor or a shared descriptor can execute a non-local JUMP to a job descriptor. In these cases, the current job descriptor or shared descriptor terminates, and the new job descriptor is fetched into the descriptor buffer and executes. Note that this new job descriptor is not permitted to reference a shared descriptor, but can also execute a non-local JUMP to another job descriptor. This mechanism allows the construction of jobs that are larger than the descriptor buffer. Once the entire chain of job descriptors terminates, a single job termination status word (see [Job termination status/error codes](#)) is returned. The return status is as if the original job descriptor had completed. That is, for Job Ring jobs, the original job descriptor address is placed in the

appropriate output ring. If an error is detected following a jump to another descriptor, a bit in the status indicates that a non-local jump was taken. Note that there is no indication of how many non-local jumps were made.

7.7.2 Command properties

Three properties determine how SEC handles each command:

- Blocking
- Load/store checkpoint
- Done checkpoint

7.7.2.1 Blocking commands

A blocking command must complete before the next command can begin. Note that the completion is from the standpoint of the DECO. If the command requires a read and the DECO has scheduled the read, the next command can begin even if the read has not completed.

Many commands are blocking commands. The notable exceptions are commands that perform LOADs, STOREs, MOVEs and OPERATION algorithm commands. (That is, not PROTOCOL or PKHA OPERATIONS, which are blocking.) Note that setting the WC bit in any of the MOVE commands cause the command to become blocking.

7.7.2.2 Load/store checkpoint

If a command is a load/store checkpoint, it must wait for certain prior LOADs and/or STOREs to complete before it can start. This property ensures that LOADs, STOREs, and other commands occur in proper order.

7.7.2.3 Done checkpoint

If a command is a done checkpoint, it must wait until all current cryptographic activity associated with the descriptor is done. The CHAs signal done once their computation has completed. Note that this is different from the descriptor being done, since not all loads and stores may have completed. It merely indicates that the CHAs in use have completed their current tasks. Note that done checkpoints can be for only Class 1, or only Class 2, or both Class 1 and Class2.

7.7.3 Command types

The following is a list of commands that are supported in SEC along with their blocking and checkpoint properties.

Table 7-3. List of command types

Command name	CTYPE	Blocking	Load/store checkpoint	Done checkpoint	See section/page
KEY (& SEQ KEY)	00000 (00001)	Yes, if not immediate or if encrypted	Load/Store, if not immediate or if encrypted	Yes	KEY commands
LOAD (& SEQ LOAD)	00010 (00011)	No	Load, for some destinations	No	LOAD commands
FIFO LOAD (& SEQ FIFO LOAD)	00100 (00101)	No	Load, if immediate and another FIFO LOAD is pending or if not immediate and an immediate FIFO LOAD or MOVE to the input FIFO is pending	No	FIFO LOAD command
STORE (& SEQ STORE)	01010 (01011)	No	if storing the checksum or if storing a scatter/gather table and that table is still being loaded	If from a Context Register the corresponding CHA must be done	STORE command
FIFO STORE (& SEQ FIFO STORE)	01100 (01101)	No	Load checkpoint if encrypting	Yes if encrypting	FIFO STORE command
MOVE (& MOVE_LEN) (& MOVEB) (& MOVEW)	01111 (01110)	Yes, if WC set	Load or Store depending on type of MOVE. It is a checkpoint if the CCB DMA is being used for a prior MOVE or for moving IMM data for KEY, LOAD, or FIFO LOAD commands.	If from a Context Register the corresponding CHA must be done	MOVE, MOVEB, MOVEDW, and MOVE_LEN commands
OPERATION (ALGORITHM OPERATION) (PROTOCOL OPERATION) (PKHA OPERATION)	10000	Yes, if PKHA or protocol	For PKHA	No	ALGORITHM OPERATION command PROTOCOL OPERATION commands PKHA OPERATION command
SIGNATURE	10010	Yes, when verifying or re-signing	Yes if recomputing signature following execution; no pending reads or writes.	No	SIGNATURE command

Table continues on the next page...

Table 7-3. List of command types (continued)

Command name	CTYPE	Blocking	Load/store checkpoint	Done checkpoint	See section/page
JUMP	10100	Yes	Checkpoint based upon condition bits	If Class bit or bits are set	JUMP (HALT) command
MATH and MATHI	10101 11101	Yes	Will wait for data if SRC1 is Input or Output Data FIFO and data is not yet available	No	MATH and MATHI Commands
Job descriptor HEADER	10110	Yes	N/A	N/A	HEADER command
Shared Descriptor HEADER	10111				
ECPARAM	11100	No	Load if an immediate FIFO LOAD or MOVE to the input FIFO is pending	No	see ECPARAM command
SEQ IN PTR	11110	Yes	Yes if there is a pending gather table read. For QI and AI jobs, then also yes if buffer releasing remains to be done for a prior SEQ IN PTR command. Also if an input frame reuse S/G table entry modification write is in flight.	No	SEQ IN PTR command
SEQ OUT PTR	11111	Yes	Yes if there is a pending scatter table read.	No	SEQ OUT PTR command

7.7.4 SEQ vs non-SEQ commands

SEC can process networking protocol packets that consist of separate fields, such as headers, sequence numbers, AADs, payloads, and ICVs. (A complete discussion of network security protocol packet formats is beyond the scope of this document. However, examples using the protocols that SEC supports can be found in [Protocol acceleration](#).) To help process such packets efficiently, SEC provides sequence (SEQ) versions of the following descriptor commands:

- KEY
- LOAD
- STORE
- FIFO LOAD
- FIFO STORE

SEQ and non-SEQ versions of descriptor commands have nearly identical functions, with the major distinction being that the SEQ versions do not require pointers because SEC uses sequence addresses that were defined by previously executed SEQ IN PTR or SEQ OUT PTR commands. Another difference is that SEQ commands (with the exception of SEQ STORE) do not have immediate data modes.

For jobs submitted by means of the Job Ring interface, each Job Ring can be configured via the INCL_SEQ_OUT field in each Job Ring Configuration Register to output an additional word in each entry of the Output Ring that indicates the length of the output sequence (that is, the number of bytes output via SEQ commands).

7.7.4.1 Creating a sequence

Sequences are generally associated with shared descriptors (see [Shared descriptors](#)) which support a one-time definition of a set of commands to be performed on each packet in a flow. The address and length of the input and output packets are usually specified in a job descriptor (see [Job descriptors](#)) that references a shared descriptor containing SEQ-version commands to indicate how to process the data. The shared descriptor is analogous to a subroutine, and the job descriptor is analogous to a software program supplying arguments and then calling that subroutine.

The job descriptor uses the following commands to provide information about the data to be processed by the sequence:

- A SEQ IN PTR command to specify the length and address of the data to be processed (see [SEQ IN PTR command](#)).
- A SEQ OUT PTR command to specify the length and address of the buffer for the output data (see [SEQ OUT PTR command](#)).

The SEQ IN PTR and SEQ OUT PTR commands each have an SGF field which, when set to 1, allows sequence input and/or output areas to be defined by means of scatter/gather tables.

The SEQ IN PTR command is used to create an input sequence. The SEQ OUT PTR is used to create an output sequence. Once the input and/or output sequence pointers have been set, subsequent SEQ commands indicate how to process the packet. The length of the sequence may be extended by issuing additional SEQ IN PTR and SEQ OUT PTR commands with the PRE bit set (see [Table 7-96](#) and [Table 7-98](#)) or by using MATH or MATHI commands to add length directly. DECO tracks how far into the output sequence DECO has progressed, and this information is used if a rewind is needed so that a second pass can be made over the sequence (see RTO field and SOP field in the SEQ IN PTR command and REW field in the SEQ OUT PTR command).

An input sequence ends when any of the following occurs:

- All specified input data is consumed (unless a rewind is then done).
- A new input sequence is started.
- An error occurs.

An output sequence ends when any of the following occurs:

- All specified output space is consumed (unless a rewind is then done).
- A new output sequence is started.
- An error occurs.

There can be at most one scatter/gather table active for input and at most one scatter/gather table active for output in the DECO at any time. Note that non-sequential commands can be executed within the same descriptor while a sequence is running. However, an input gather table can be in use by either an input sequence or by non-SEQ KEY, LOAD, or FIFO LOAD commands, but not both. Likewise, an output scatter table can be in use by either an output sequence or by non-SEQ STORE or FIFO STORE commands, but not both.

To accelerate performance, SEC caches gather table and scatter table entries in registers (see the Gather Table Register (DxGTR) and Scatter Table Register (DxSTR)).

NOTE

If a scatter/gather table is being used for an input or output sequence, and a non-SEQ command references a second scatter/gather table for input or output data, entries from the second scatter/gather table overwrite the entries from the initial scatter/gather table. This can result in the input/output sequence referencing the wrong data. (The opposite case is not a problem. For example, a non-SEQ LOAD command which references a scatter/gather table followed by a SEQ IN PTR which references a scatter/gather table won't be an issue since the LOAD has to complete the use of its scatter/gather table before the SEQ IN PTR command can execute.)

NOTE

Hardware does not flag overwriting the scatter/gather table as an error. The descriptor programmer must ensure this does not happen.

7.7.4.2 Using sequences for fixed and variable length data

Some SEQ commands act on fixed length data (for example, keys, IVs, or packet header fields) whereas other SEQ commands act on data that changes length from packet to packet, such as packet payload. The VLF bit found in all SEQ commands indicates whether the data associated with the SEQ command is a constant length or whether the length is to be found in the corresponding variable length registers (VSIL and VSOL).

Note that the VSIL and VSOL are not accessible through the register bus, but rather they are read from or written to by means of the MATH or MATHI Command (see [Table 7-92](#)). This allows commands within the descriptor to calculate variable lengths. Given a total packet length (from an internally or externally generated job descriptor), the descriptor can calculate the variable length portion of a job and load it into the Variable Length Registers to be referenced by subsequent SEQ commands (by setting the VLF bit).

7.7.4.3 Transferring meta data

When processing data, SEC typically uses the DMA to read input data and write output data. Because SEC is primarily intended to accelerate cryptographic operations, the output data is normally different from the input data. However, it is possible to use SEC's external DMA to transfer data from an input buffer to an output buffer without modifying the data (that is, the identity transformation, also called null encryption), typically to either:

- Benefit from SEC's scatter/gather capabilities
- Transfer meta data in conjunction with cryptographic processing

The latter case is often useful because the meta data may describe the type, the source, the destination, the classification, the priority and/or the amount of the cryptographic data. If this meta data appears ahead of the cryptographically processed data, it is called 'leading meta data'. If it appears after, it is called 'trailing meta data'.

Three different tasks must be scheduled in order to transfer meta data from input to output without modification:

- Data must be read. Most often, the data is read into the input data FIFO. FIFO LOAD and SEQ FIFO LOAD are the most common methods for getting data into the input FIFO.

- Data must be stored. Most often, the data is stored from the output FIFO. FIFO STORE and SEQ FIFO STORE are the most common methods for storing data from the output FIFO.
- If the data is brought into the input data FIFO, it must be moved, via one of the MOVE commands, to the output FIFO.

While it is possible to transfer meta data without going through the input FIFO and output FIFO, such transfer methods are discouraged as timing can be complex. Furthermore, meta data is most often used with sequences so that multiple pointers don't have to be specified.

It is possible to accomplish two or three of the above tasks using a single command. This command is the SEQ FIFO STORE command with the meta data Output Data Type, 3Eh. Depending on how the auxiliary bits are set, this type of SEQ FIFO STORE will adjust the various lengths and obtain the data either from the input frame or from the input data FIFO.

The above procedures work for leading meta data. To handle trailing meta data for a sequence, start by subtracting the length of the meta data from the Sequence Input Length Register. Then, process the input frame. Once the processing is complete, add the length of the meta data back to the Sequence Input Length Register and handle the meta data. (Note that if using SEQ FIFO STORE with meta data Output Data Type, you don't need to add the length back into the register.)

7.7.4.4 Rewinding a sequence

Note that it is possible to rewind a sequence to make an additional pass over the input and output data (see RTO field and SOP field in SEQ IN PTR command and REW field in SEQ OUT PTR command). A rewind can fill in data that was skipped over in a previous pass. For example, a rewind may be necessary if a field contains a hash value that is computed over data that appears later in the output data.

The following built-in protocol operations perform rewinds:

- TLS Decapsulation rewinds the input frame.
- IPSec Decapsulation rewinds the output frame.
- DCRC Encapsulation and Decapsulation rewind the output frame.

Note that owing to hardware limitations, the amount of leading meta data is limited to 2^{15} bytes when using any of these built-in protocols.

An error will be generated if the SEQ IN PTR command is used to rewind the input frame after buffers have been released. Care must be taken when a rewind is done if a nonzero offset modifier is present in the case of input frame reuse and the input is

referenced via a scatter/gather table. Initially, the first 4 entries of the scatter/gather table are fetched. The offset modifier is then applied to the output scatter table but is not applied to the input gather table. Furthermore, the modified output table is written back to memory, overwriting the original. Therefore, the only copy of the original scatter/gather table is stored in the Gather Table inside DECO. There are two scenarios which can occur. The first is when an input frame rewind is requested while the original scatter/gather table is still in the Gather Table. In this scenario, DECO will start reading from the beginning of the original input frame. The second scenario occurs when the input frame rewind is requested after subsequent entries have been loaded into the Gather Table. This will occur if reads have read past the content of the first four entries or if an SGT Extension was encountered. In such a scenario, the input frame rewind will result in reads starting from the beginning of the modified output frame. It is up to the descriptor writer and the system software to ensure that rewinds will result in expected behavior.

7.7.5 Information FIFO entries

Each CCB has an iNformation FIFO (NFIFO). The NFIFO holds entries that describe the corresponding data to obtain from the input data FIFO, the output data FIFO, the auxiliary data FIFO, or the padding module. (The padding module provides a means for generating different types of padding and random numbers.) The data is obtained from each source in the order in which the NFIFO entries are loaded. Note that a single entry is able to describe the same (in-snooping) or different (out-snooping) sources for the class 1 and class 2 alignment blocks.

Typically, a command which loads data to the input data FIFO or pulls data through one of the alignment blocks will result in SEC automatically generating the proper NFIFO entries to handle that data. However, that functionality can be overridden to allow the descriptor to directly specify the NFIFO entries. Entries can be placed into the NFIFO via a LOAD Immediate command with a DST value of 7Ah or 70h through 75h.

7.7.6 Output FIFO Operation

There are three ways to get data into the output FIFO: a LOAD IMM to the output FIFO, a move command where the destination is the output FIFO, and CHAs pushing their results into the output FIFO. It is up to the descriptor writer to ensure that there are no collisions of data from these sources. If such a collision does occur, an error will be generated.

The output FIFO does not track valid bytes. Therefore, it is up to the descriptor writer to know which bytes in the output FIFO are valid. For example, if you push 3 bytes into the output FIFO followed by 5 more bytes, these 8 bytes are not contiguous. The first three bytes are in one dword and the other 5 bytes are in a second dword. However, the output FIFO always stores 8 bytes per push. When a LOAD IMM to the output FIFO is done, the specified number of bytes are left aligned and any other bytes are written as provided. That is, if the immediate data is to be one byte, 55h, but the 4-byte value provided is 55443322h, then all four bytes are written to the output FIFO along with 4 more bytes of 0. Moves to the output FIFO will have similar results. However, if a CHA is pushing 3 bytes into the output FIFO, those bytes will be left aligned and the other 5 bytes will be 0.

The output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the output FIFO so each can track separately allowing consumption of data at different rates. The following list illustrates how these indices work.

- If the current NFIFO entry is not pulling data from the output FIFO, then whenever the external DMA pops an entry off the output FIFO, the two indices increment.
- If the CCB DMA pops an entry off the output FIFO, both indices will increment.
- If the DECO pops an entry off the output FIFO via the MATH command, both indices will increment.
- If the current NFIFO entry is pulling data from the output FIFO, then the two indices will track separately if the NFIFO entry is not STYPE=01 and AST=1. This is a critical point to understand: since the indices are tracking separately, if one of the consumers, either the NFIFO or the external DMA, falls far enough behind the other, the output FIFO can fill and operations will stall until the lagging consumer catches up. If the NFIFO is consuming data but there are no FIFO STOREs to advance the external DMA pointer, then the NFIFO can only consume as much data as the output FIFO can hold before a hang will result.
- If the current NFIFO entry has STYPE=01 and AST=1, the indices will both increment when the NFIFO pops entries from the output FIFO. This scenario is useful when all of the data being pushed into the output FIFO is to be consumed via one of the alignment blocks.

There are two ways to alter the behavior of the output FIFO via descriptor control. The first is the means to set the index shared by the CCB DMA, DECO, and the NFIFO to have the same value as the index used for the external DMA. This is done via a LOAD IMM to the DECO control register. The second method is to reset the output FIFO, which clears the data in the FIFO and resets both indices. The reset can also be done via a LOAD IMM to the DECO control register. Another means of resetting the output FIFO is via a LOAD IMM to the Clear Written register.

Another way to alter access to the output FIFO is via the ofifo offset. This value is tracked by DECO as a means of remembering where the last access left off. For example, if a SEQ FIFO STORE of 3 bytes is done, what happens to the other 5 bytes in the output FIFO entry? Both the NFIFO entry and the FIFO STORE commands allow the descriptor to have these remaining bytes retained or discarded. If the OC bit in the NFIFO entry is set when the NFIFO is pulling data from the output FIFO, the remaining bytes are retained. A subsequent access via DECO, the CCB DMA, or the NFIFO will be able to obtain this data. However, the descriptor writer will be responsible for shifting the data as needed to get to the remaining bytes if the access is done via the DECO or the CCB DMA since only the NFIFO will be tracking where it left off.

If the CONT bit in the FIFO STORE command is set, the remaining bytes are also retained when the the external DMA reads the specified number of bytes. In this case, it is DECO which tracks how many remaining bytes there are so that the subsequent FIFO STORE command will start where the prior one left off. The move commands will also use the ofifo offset, so that it can also start with the remaining data. However, the CCB DMA will always pop entries from which it takes data so that it is not possible for the CCB DMA to leave any trailing bytes in the output FIFO. Please see the section for the move commands for important information on how this works.

In order to provide greater control of access to the output FIFO, the value of the ofifo offset can be changed via a LOAD IMM to the DECO control register. This feature can be useful in several scenarios:

- If there are 7, or fewer, bytes of interest in the current output FIFO entry, and this data needs to be stored to memory and used within DECO but snooping is not convenient, the descriptor could do a FIFO STORE with the CONT bit set, and then do a move from the output FIFO to another destination in DECO or the CCB. (If the original ofifo offset was nonzero, the sum of the original ofifo offset and the number of bytes of interest must be less than 8.
- In order to do a single FIFO STORE of data that was sent to the output FIFO via separate methods, e.g. a LOAD IMM to the output FIFO followed by data from a CHA, the data must be contiguous in the output FIFO. For example, a LOAD IMM to the output FIFO of 3 bytes followed by data from the CHA would have a 5-byte gap between the loaded data and the CHA data. This could be solved by shifting the load data 5 bytes to the right and then doing a LOAD IMM of 8 bytes with those 3 bytes of interest right aligned. But now there are 5 "garbage" bytes at the start of the output FIFO data. These can be skipped over by setting the ofifo offset to 5. Now, a single FIFO STORE can be done with a length of the number of bytes pushed by the CHA plus 3.
- The ofifo offset can also be set to a smaller value than it currently contains. This can allow the same data to be stored twice, although the limit is back to the start of the current entry.

7.7.7 Output Checksum logic

Normally, all data written out through a SEQ STORE or SEQ FIFO STORE command has a checksum computed upon it.

The checksum computed is a sixteen-bit ones complement modular sum checksum, compatible with UDP and TCP. RFC 793 uses this description: "The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes."

By default, all bytes written through a SEQ STORE or SEQ FIFO STORE command are included in the computation. The SEQ FIFO STORE command has special output data types (not available to FIFO STORE) to enable (output data type 31h) or disable (output data type 30h) the checksum logic. The first time the enable is invoked, the contents of the DECO checksum register are cleared so that the checksum appears to have started with the enable. An enabling or disabling SEQ FIFO STORE may be used with a zero length to simply "turn on" or "turn off" the checksum computation without actually writing any contents of the output FIFO to memory.

If the data is stored via multiple SEQ STORE and/or SEQ FIFO STORE commands, each store command will result in a data segment of a specified number of bytes being added to the checksum. If, for one such segment, the last byte ends at an odd boundary, the next data to be checked will start on the odd boundary. In other words, the checksum is computed as if all of the segments to be checked are concatenated. This is true even if there are checksum-disabled bytes between two checksum-enabled segments. If the final length of the data is odd, then the checksum is computed as if a zero byte was appended.

When a job is to be returned to AIOP, the checksum is returned in the FLC field of the response frame descriptor, as described in sec [Done/error notification for AI jobs](#).

When a job is to be returned to Qman, the checksum is returned in the FLC field of the ASA section of the output frame, if enabled by EAO=1 in the flow context. This is described further in sec [Done/error notification for QI jobs](#).

The contents of the checksum computation register can be written at any time by performing a STORE or SEQ STORE of the DCHKSM register.

The checksum computation logic is present in part to enable UDP-encapsulated-ESP as part of IPsec ESP Tunnel encapsulation. See section [Outer IP Header handling with UDP-encapsulated-ESP](#) for details on ESP Tunnel support for UDP-encapsulated-ESP. When ESP Tunnel performs UDP-encapsulated-ESP, the checksum logic controls are

overridden in order to perform the proper UDP checksum computation. The checksum result can be written out using any of the methods above, but that value will duplicate the UDP checksum included in the UDP header. The IPsec ESP tunnel protocol, if NAT and NUC are both set, enable and disable the checksum logic as needed. Otherwise, the IPsec ESP tunnel protocol, and all other protocols including IPsec ESP Transport (and legacy tunnel) neither enable nor disable the checksum logic.

NOTE

In some instances, a protocol may write out more data past the end of the output frame. In the case of IPsec ESP Transport Decapsulation, this is because the protocol doesn't determine the end of the decapsulated payload until decrypting the Pad Length byte found in the ESP trailer. For all instances when this occurs, the computation of the checksum will include all the bytes actually written out to memory, even if the output frame length is subsequently adjusted to hide them.

7.7.8 Cryptographic class

SEC divides cryptographic algorithms into two different classes for the purpose of selecting CHAs. Some key and data movement commands must have a CLASS value associated with them so they are delivered to the proper CHA.

The following table shows the class for each cryptographic algorithm.

Table 7-4. Classes of the cryptographic algorithms

Class	Algorithm
Class 1	AES (all modes) ¹ , DES, 3DES, Kasumi, SNOW f8, ZUCE, Public Key, RNG
Class 2	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SNOW f9, CRC, ZUCA AES authentication modes, MD5

- Note that AES optimization modes (such as, modes that combine encryption and authentication) require that the authentication key be present in the Class 2 Key Register, and the encryption/decryption key be present in the Class 1 Key Register.

NOTE

A descriptor that requests both a Class 1 CHA and a Class 2 CHA must request the Class 2 CHA first. Otherwise, in versions of SEC that implement more than one DECO a deadlock situation could occur as follows:

- Descriptor *x* executing in DECO *x* acquires CHA 1 and then requests CHA 2.

- Descriptor y in DECO y acquires CHA 2 and then requests CHA 1.
- Descriptor x and descriptor y wait until both CHAs are available, but neither will release the CHA that it currently has.

If descriptors always acquire CHAs in the same order, this deadlock situation is avoided. The required order is Class 2 first, then Class 1. An error is generated if a Class 2 CHA is selected after a Class 1 CHA.

Note that software written for versions of SEC that implement only one DECO must still follow this practice to ensure that the software is portable to versions of SEC that implement two or more DECOs.

When specifying classes in commands, a two-bit field is used to specify class as follows:

Table 7-5. Class field

Class Value	Meaning for LOAD and STORE Commands	Meaning for Other Commands
00	CCB class independent	None, sequence data skipped for SEQ FIFO LOAD command.
01	CCB Class 1	Class 1
10	CCB Class 2	Class 2
11	DECO	Both Class 1 and Class 2

7.7.9 Address pointers

Many of the descriptor commands and several data structures used by SEC include Pointer fields. If PS = 0 in the Master Configuration Register, all address pointers used by SEC will be 32-bits in length. If PS = 1 in the Master Configuration Register, all pointers used by SEC will be 49 bits in length, laid out as shown in [Table 7-6](#).

When operating in little-endian mode, it is important to remember that the least significant word of two-word addresses appears prior to the most significant word in all locations. If the descriptor needs to manipulate such addresses, the MOVEW command can be used to move the address to a Math Register such that it arrives as an 8-byte integer ready for manipulation. The MOVEW command can then be used to move the address back

Table 7-6. Format of 49-bit pointers

Dword at address X	least-significant 32 bits of 49-bit address	
Dword at address X +4	15 bits ignored	most-significant 17 bits of 49-bit address

7.8 HEADER command

Every descriptor begins with a HEADER command, which provides basic information about the descriptor itself, such as length, ability of DECOs to share the descriptor, and whether errors are propagated when the descriptor is shared.

Job descriptor and Shared Descriptor HEADER commands share a base format, but some fields are specific to descriptor type. The formats of the Job Descriptor Header Command and Shared Descriptor Header Command are shown in the diagrams below, and the fields of both are described in detail below each format diagram.

Table 7-7. Job descriptor header command format

31-27				26	25	24	23	22	21-16		
CTYPE = 10110b				EXT	Reserv ed	DNR	ONE	Reser ved	START INDEX / SHR DESCR LENGTH		
15	14-13	12	11	10-8		7	6-0				
ZRO	TDES	SHR	REO	SHARE		Reser ved	DESCLEN				
<i>Optional additional words of HEADER command:</i>											
If SHR = 1, a one or two word pointer to a Shared Descriptor is located immediately after the HEADER.											
If EXT = 1, a Job Descriptor HEADER extension is located immediately after the HEADER or, if SHR = 1, immediately after the Shared Descriptor pointer. (see Job Descriptor HEADER extension format, below)											

Table 7-8. Job descriptor header field descriptions

Job descriptor header fields	Description
31-27 CTYPE	Command type 10110b Job Descriptor HEADER
26 EXT	Extended Job Descriptor HEADER If EXT=0 : There is no extended HEADER word. If EXT=1 : The HEADER command contains an extended HEADER word, as illustrated in the format diagram below. Note that if there is no Shared Descriptor a HEADER error (13h) is generated if EXT=1 and START INDEX=1.

Table continues on the next page...

Table 7-8. Job descriptor header field descriptions (continued)

Job descriptor header fields	Description
25 Reserved	Reserved
24 DNR	<p>Do Not Run</p> <p>If DNR=0 : Normal execution of the Job Descriptor.</p> <p>If DNR=1 : Do Not Run the Job Descriptor. There was a problem upstream so this descriptor should not be executed. This allows the job to be passed through the hardware and software pipeline to a point where the problem might be corrected by software and the job resubmitted.</p> <p>NOTE: If this bit is found in a Job Descriptor HEADER, SEC still fetches any associated Shared Descriptor. If the Shared Descriptor HEADER 's PD bit is set and the DNR bit is not set, SEC updates the Shared Descriptor header's DNR bit. As a result, future Job Descriptors that use this Shared Descriptor do not run. Once software clears the DNR bit in the Shared Descriptor, any new Job Descriptors that use this Shared Descriptor run normally. In addition, when the DNR bit is set and the input frame buffers were to be released, DECO will release them.</p>
23 ONE	<p>One</p> <p>The ONE bit is always 1. This bit is used in combination with the ZRO bit to verify that the endianness of the header is correct. This is necessary because SEC is used in chips with both big-endian and little-endian processors.</p>
22	Reserved
21-16 START INDEX/ SHR DESCR LENGTH	<p>Start Index or Shared Descriptor Length</p> <p>If SHR = 0, this is the START INDEX field</p> <p>Start Index specifies the position of the word in the descriptor buffer where execution of the Job Descriptor should continue following execution of the Job Descriptor HEADER. That is, DECO should jump to the specified word to continue processing. Note that if there is a HEADER extension word (EXT=1) the START INDEX must not be 1, else a 13h (header) error will result.</p> <p>If SHR = 1, this is SHR DESCR LENGTH field</p> <p>The Shared Descriptor Length specifies the length of the Shared Descriptor (in 32-bit words).</p>
15 ZRO	<p>Zero</p> <p>The ZRO bit is always 0. This bit is used in combination with the ONE bit to verify that the endianness of the header is correct. This is necessary as SEC is used in NXP product lines with both big-endian and little-endian processors.</p>
14-13 TDES	<p>Trusted Descriptor</p> <p>If TDES=00b : This is a normal Job Descriptor, that is, not a Trusted Descriptor. However, if the AMTD bit is set in the JRaCID register this descriptor can be run as a trusted descriptor by setting the FTD bit in the extended header word. In that case, no SIGNATURE command is required and no signature will be generated or verified. Note that if FTD=1 in the extended header word then TDES must be 00b. An error will be generated if this is not the case.</p> <p>If TDES=01b : This is a TrustZone SecureWorld Trusted Descriptor - a special Trusted Descriptor created by TrustZone SecureWorld.</p> <p>If TDES=10b : This is a TrustZone non-SecureWorld Trusted Descriptor, that is, a Trusted Descriptor created by TrustZone non-SecureWorld.</p> <p>If TDES=11b : This is a candidate Trusted Descriptor, that is, a descriptor that will be made into a Trusted Descriptor by appending a signature to it. If a candidate Trusted Descriptor is submitted via a Job Ring owned by TrustZone SecureWorld the descriptor will be converted to a TrustZone SecureWorld Trusted Descriptor (TDES=01b) when the descriptor is signed. If a candidate Trusted Descriptor is submitted via a</p>

Table continues on the next page...

Table 7-8. Job descriptor header field descriptions (continued)

Job descriptor header fields	Description
	<p>Job Ring owned by TrustZone non-SecureWorld the descriptor will be converted to a TrustZone non-SecureWorld Trusted Descriptor (TDES=10b) when the descriptor is signed. Note that an error will be generated if AMTD=0 in the Job Ring's ICID register. The DESCLEN field must account for the eight 32-bit words of signature which will be added.</p> <p>See the discussion Trusted descriptors for an explanation of Trusted Descriptors.</p>
12 SHR	<p>Shared Descriptor (SHR) flag</p> <p>If SHR=0 : This Job Descriptor does not have a Shared Descriptor and so does not include a Shared Descriptor pointer.</p> <p>If SHR=1 : This descriptor has a Shared Descriptor that is pointed to by the next word or words.</p> <p>SHR controls how START INDEX / SHR DESCR LENGTH is used.</p>
11 REO	<p>Reverse Execution Order (REO). Note that this bit is ignored if SHR = 0 (that is, no Shared Descriptor).</p> <p>If REO=0 : The Shared Descriptor is executed prior to the remainder of the Job Descriptor.</p> <p>If REO=1 : The Job Descriptor will be executed prior to the Shared Descriptor. Setting REO=1 in a Trusted Descriptor results in an error.</p>
10-8 SHARE	<p>Share State (SHARE)</p> <p>This defines if, and when, the Shared Descriptor of this descriptor can be shared with another Job Descriptor. (See Table 7-1.) Also see Specifying different types of shared descriptor sharing for further information.</p>
7	Reserved
6-0 DESCLEN	<p>Descriptor Length</p> <p>This field represents the total length in 4-byte words of the descriptor. A descriptor length of 0 is undefined. The header word is included in the length. Note that the size of the descriptor buffer is 64 words, so that is the maximum size of a single Job Descriptor with no Shared Descriptor.</p>

Table 7-9. Job descriptor header extension format

31-16									
Reserved									
15-9	8	7	6	5	4	3	2	1	0
Reserved	FTD	DSELECTV ALID	Reserved		Reser ved	DECO_SELECT			

Table 7-10. Job descriptor header extension field descriptions

Field	Description
31-9	Reserved
8 FTD	<p>Fake Trusted Descriptor. Treat the current descriptor as a Trusted Descriptor but do not check the signature. If the descriptor is run in a Job Ring owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone SecureWorld Trusted Descriptor, otherwise the descriptor will be treated as a TrustZone non-SecureWorld Trusted Descriptor. Note that an error will be generated if FTD=1 and the source Job Ring's JRaCID register AMTD=0 (that is, the extended header says to run the descriptor as a Trusted Descriptor, but the Job Ring is not allowed to make Trusted Descriptors). In order to use FTD, the TDES field in the first word of the header command must be 00b. An error will be generated if this is not the case.</p>

Table continues on the next page...

Table 7-10. Job descriptor header extension field descriptions (continued)

Field	Description
	If FTD=1, no SIGNATURE commands are required. If any SIGNATURE skip commands are present they will be treated as no-ops. If a final SIGNATURE command is present, it will be treated as the end of the descriptor.
7 DSELECT VALID	DECO_SELECT field is valid. If DSELECTVALID=0 : Any DECO can run the job. The DECO_SELECT field is ignored. If DSELECTVALID=1 : The job must be run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, DECO error 026h will be generated.
6-4	Reserved
3	Reserved
2-0 DECO_SEL ECT	DECO Select If DSELECTVALID = 1, the job is run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, an error will be generated. Note that for programming consistency, a one-DECO version of SEC has a one-bit DECO_SELECT field. NOTE: DECO-specific jobs have the possibility to create a deadlock in SEC when they are used as part of a flow. Therefore, it is strongly recommended that DECO-specific jobs either not be part of a flow or all the jobs in the flow be assigned to the same DECO.

Table 7-11. Shared descriptor header format

31-27					26	25	24	23	22	21-16		
CTYPE = 10111					Reser ved	RIF	DNR	ONE	Reser ved	START INDEX		
15	14	13	12	11	10	9-8		7-6		5-0		
ZRO	Reser ved	CIF	SC	PD	Reser ved	SHARE		Reserved		DESCLEN		

Table 7-12. Shared descriptor header field descriptions

Shared descriptor header fields	Description
31-27 CTYPE	Command type 10111b Shared descriptor
26	Reserved
25 RIF	Read Input Frame As soon as possible, DECO should read the entire input frame as defined in a SEQ IN PTR command in the Job Descriptor. The length of the input frame is placed in the VSIL Register. The data is read into the input data FIFO. This is the equivalent of a SEQ FIFO LOAD of the entire input frame without an NFIFO entry or any data size registers being written. This bit is intended to allow DECO to issue reads from the SEQ IN PTR as soon as possible, thereby reducing processing latency. However, there are contraindications to its use: <ul style="list-style-type: none"> • If the descriptor contains any LOAD or KEY command that is not immediate. • If the descriptor contains a KEY command that loads an encrypted key. The Derived Key PROTOCOL OPERATION command is not included in this restriction. • If the descriptor contains a SEQ IN PTR with RTO (Restore Input Sequence) set • If the descriptor contains a PROTOCOL OPERATION command specifying SSL / TLS Decapsulation

Table continues on the next page...

Table 7-12. Shared descriptor header field descriptions (continued)

Shared descriptor header fields	Description
	<ul style="list-style-type: none"> • If the descriptor contains a PROTOCOL OPERATION command specifying IPsec ESP Tunnel mode encapsulation <i>and</i> PDB Options field OIHI=10 (specifying inclusion of an Outer IP Header referenced by the PDB). • If the descriptor contains a PROTOCOL OPERATION command specifying LTE Control Plane encapsulation or LTE Data Plane encapsulation for RN with either SNOW or ZUC for the confidentiality algorithm <i>and</i> AES for the integrity algorithm. • If the descriptor contains a PROTOCOL Operation command specifying either Blob encapsulation or Blob decapsulation. <p>There are restrictions with the use of RIF with Public Key Cryptography operations and PRFs: RIF may be used, but only if all the input FIFO is drained by other descriptor commands before the PROTOCOL COMMAND is encountered.</p>
<p>24 DNR</p>	<p>Do Not Run</p> <p>0 Normal execution</p> <p>1 Do Not Run. There was a problem upstream so this descriptor should not be executed.</p> <p>NOTE: If this bit is found in a Job Descriptor HEADER, SEC still fetches any associated Shared Descriptor. If the Shared Descriptor HEADER 's PD bit is set and the DNR bit is not set, SEC updates the Shared Descriptor header's DNR bit. As a result, future Job Descriptors that use this Shared Descriptor do not run. Once software clears the DNR bit in the Shared Descriptor, any new Job Descriptors that use this Shared Descriptor run normally. In addition, when the DNR bit is set and the input frame buffers were to be released, DECO will release them.</p>
<p>23 ONE</p>	<p>One</p> <p>The ONE bit is always 1. This bit is used in combination with the ZRO bit to verify that the endianness of the header is correct. This is necessary because SEC is used in chips with both big-endian and little-endian processors.</p>
<p>22</p>	<p>Reserved</p>
<p>21-16 START INDEX</p>	<p>This is the START INDEX field, which specifies the index of the word in the descriptor buffer where execution of the Shared Descriptor should start. This allows protocol or other information to be jumped over.</p>
<p>15 ZRO</p>	<p>Zero</p> <p>The ZRO bit is always 0. This bit is used in combination with the ONE bit to verify that the endianness of the header is correct. This is necessary as SEC is used in NXP product lines with both big-endian and little-endian processors.</p>
<p>14</p>	<p>Reserved</p>
<p>13 CIF</p>	<p>Clear Input FIFO (CIF)</p> <p>If set, the input FIFO and the NFIFO entries are reset between self-Shared Descriptors. That is, these are reset if the next job to be run within the same DECO has the same Shared Descriptor as the previous job run in that same DECO. (The Input FIFO and NFIFO are always reset between descriptors that don't share the same Shared Descriptor.)</p>
<p>12 SC</p>	<p>Save Context (SC)</p> <p>After this descriptor completes, if Serial-Sharing is selected, and if sharing of the Shared Descriptor occurs within the same DECO (self-sharing):</p> <p>0 The context registers are cleared.</p> <p>1 The context registers are maintained and used by the subsequent descriptor.</p> <p>Save Context is intended to allow multiple subsequent Shared Descriptors to maintain cryptographic context when a cryptographic operation is split between multiple jobs.</p>

Table continues on the next page...

Table 7-12. Shared descriptor header field descriptions (continued)

Shared descriptor header fields	Description
11 PD	Propagate DNR (PD) If the Job Descriptor's DNR bit is set and this bit is set, set the DNR bit of the Shared Descriptor HEADER if it is not already set.
10	Reserved
9-8 SHARE	Share State (SHARE) This defines if, and when, the Shared Descriptor of this descriptor can be shared with another Job Descriptor. (See Table 7-1 .) Also see Specifying different types of shared descriptor sharing for further information.
7-6	Reserved
5-0 DESCLEN	Descriptor Length This field represents the total length in 4-byte words of the Shared Descriptor. A Shared Descriptor length of 0 is undefined. The header word is included in the length. Note that the size of the descriptor buffer is 64 words, so the maximum size of a Shared Descriptor is 62 words (assuming that the pointer size is 32 bits and the Job Descriptor consists of only the Job Descriptor HEADER command and the pointer to the Shared Descriptor).

If the SHR bit in a Job Descriptor HEADER command is set, a pointer to the Shared Descriptor immediately follows the header.

7.9 KEY commands

NOTE

In the following discussion, the term 'KEY command' refers to both the SEQ and non-SEQ forms of the command.

KEY commands are used to load keys into one of SEC's key registers: Class 1 or Class 2 Key Register or PKHA E-Memory. The SEQ KEY command is identical to the KEY command except that no address is specified and the VLF bit replaces the SGT bit and the AIDF bit replaces the IMM bit.

If the key to be loaded into a key register is encrypted, SEC can be told to automatically decrypt it as it is loaded into the key register. The MOVE command, LOAD command and KEY command can all be used to load a Red Key into a key register, but only the KEY command can be used to load a Black Key. Note that Black Keys can be loaded only into PKHA E-Memory or key registers because only KEY commands decrypt Black Keys, and these registers are the only possible destinations for KEY commands.

If the KEY command is loading a Black Key, the Class 2 key must be loaded prior to the Class 1 key as the loading of encrypted keys has side effects on the Class 1 Key Register.

If ENC is set (that is, a Black Key is being loaded), the KEY command has significant side effects, including clearing the following:

- Input Data FIFO
- Output Data FIFO
- Class 1 Key Register
- Class 1 Data Size Registers
- Class 1 Mode Register
- Class 1 Context (if EKT is also set)

As a result, the only commands that should precede loading a Black Key are:

- JUMP
- SEQ IN PTR
- SEQ OUT PTR
- LOADs to registers not mentioned above.
- MOVEs to or from registers not mentioned above.

NOTE

The KEY command is blocking under the following circumstances:

1. Decrypting a black key.
2. Loading a red key that is NOT immediate.
3. CHAs are not done.
4. The data must pass through the input FIFO and there are info FIFO entries in the way.
5. The data must be read into the data FIFO and there is other data in the input data FIFO that is in the way. (This does not apply to SEQ KEY AIDF.)
6. The CCB DMA is required but is busy.
7. The hardware which schedules external reads is required but is busy.

Table 7-13. KEY command format

31–27		26–25		24	23	22	21	20	19–18	17–16
CTYPE = 00000 or 00001		CLASS		SGF or VLF	IMM or AIDF	ENC	NWB	EKT	Reserved	KDEST
15	14	13–10			9–0					
TK	PTS	Reserved			LENGTH					
<i>Additional words of KEY command:</i>										
Pointer (one or two words) or Value (if immediate, one or more words)										

Table 7-14. KEY command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=00000b : KEY command If CTYPE=00001b : SEQ KEY command
26-25 CLASS	Class. This defines whether this key is for a Class 1 or Class 2 algorithm. If CLASS=00b : Reserved If CLASS=01b : Class 1 Key If CLASS=10b : Class 2 Key If CLASS=11b : Reserved NOTE: Class must be set to Class 1 if the key DEST field is set to 01b. Class must be set to Class 2 if the key DEST field is set to 11b.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF) If CTYPE = 00000b (KEY), this bit is the Scatter/Gather table Flag (SGF). If SGF=0: Pointer points to actual data. If SGF=1: Pointer points to a scatter/gather table. NOTE: It is an error for this bit to be set if the IMM bit is set. If CTYPE = 00001b (SEQ KEY), this bit is the Variable Length Flag (VLF). If VLF=0: The number of bytes of data to be loaded into the key register is specified by the LENGTH field. If VLF=1: The number of bytes of data to be loaded into the key register is specified by the value in the VSIL register.
23 IMM or AIDF	Immediate Flag (IMM) or Already in Input Data FIFO (AIDF) If CTYPE = 00000 (KEY), this bit is the IMM flag. If IMM=0 : The key value is found at the location pointed to by the pointer in the next word (32-bit pointers) or next dword (> 32-bit pointers). If IMM=1 : The key value follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. NOTE: PKHA E-Register values can be very large and may not fit within the descriptor buffer. If the SGF bit is set, It is an error for the IMM bit to be set. If CTYPE = 00001b (SEQ KEY), this bit is the AIDF flag. If AIDF=0 : Read the Input Data Sequence data and load it into the specified destination. If AIDF=1 : Do not read the Input Data Sequence data since it is already in the Input Data FIFO, but load the data in the Input Data FIFO into the specified destination. It is an error for ENC and AIDF to both be 1.
22 ENC	Key is encrypted If ENC = 0 : The key is assumed to be in plaintext and is loaded into the destination register without decryption. If ENC = 1 : SEC automatically decrypts the key (using the JDKEK, or if this is a trusted descriptor, using the TDKEK if TK = 1) before putting it in the key register. Decrypting a key requires using the AESA, the input and output data FIFOs, Class 1 Mode register, Key Size, context, and key registers. Therefore, Class 2 Black Keys (if any) must be loaded prior to loading the Class 1 register, and Class 1 Black Keys must be loaded prior to loading any of the resources noted above.
21 NWB	No Write Back If NWB=0 : Note that it is not usually possible to write a key back out to memory as plaintext, but if NWB=0 the key can be written out as a Black Key by using the FIFO STORE command.

Table continues on the next page...

Table 7-14. KEY command field descriptions (continued)

Field	Description
	<p>If NWB=1 : Prevents the key that is loaded into the key register from being written back out to memory.</p> <p>NWB applies to all key locations: Class 1 Key Register, Class 2 Key Register, and PKHA E Memory. Setting this bit sets the key register's NWB flag. The No Write Back setting lasts until the end of the descriptor (or sequence of shared descriptors) or until the corresponding key register or CHA is cleared/reset.</p>
20 EKT	<p>Encrypted Key Type</p> <p>The EKT bit determines which decryption mode is used when a Black Key (ENC = 1) is loaded.</p> <p>If EKT=0 : The Black Key is decrypted using AES-ECB.</p> <p>If EKT=1 : The Black Key is decrypted using AES-CCM.</p> <p>A Black Key encrypted with AES-ECB must be decrypted with AES_ECB, and a Black Key encrypted with AES-CCM must be decrypted with AES_CCM. If the wrong mode is selected it is possible that no error will be issued but the value loaded into the key register will be incorrect.</p> <p>Note that an error status is generated if EKT=1 and ENC=0.</p>
19-18	Reserved
17-16 KDEST	<p>Key Destination</p> <p>If KDEST=00b : The key is loaded into the Key Register (either Class 1 or Class 2 as specified in the CLASS field).</p> <p>If KDEST=01b : The key is loaded into the PKHA E-Memory. This key destination requires CLASS = 01b (Class 1 key).</p> <p>If KDEST=11b : The key is regarded as an MDHA Split Key, and is loaded into the Class 2 Key Register. An MDHA split key is the concatenation of the IPAD material followed by the OPAD material. Split keys offer higher performance for HMACs. Note that MDHA IPAD/OPAD values are considered keys and are decrypted to the Class 2 Key Register. This key destination requires CLASS = 10b (Class 2 key).</p> <p>All other values are reserved.</p>
15 TK	<p>Trusted Key</p> <p>This bit is used only by trusted descriptors. If not a trusted descriptor, setting TK = 1 and ENC = 1 is an error. If the ENC bit is not set, this bit is ignored.</p> <p>If TK=0 : Use the Job Descriptor Key Encryption Key (JDKEK) to decrypt the key that is to be loaded into a key register.</p> <p>If TK=1 : A trusted descriptor wants to use the Trusted Descriptor Key Encryption Key (TDKEK) to decrypt the key that is to be loaded into a key register.</p>
14 PTS	<p>Plaintext Store</p> <p>If PTS=0: The key loaded cannot later be stored in plaintext form.</p> <p>If PTS=1: The key loaded can later be stored in plaintext form using a FIFO STORE or SEQ FIFO STORE command. Note the following restrictions:</p> <p>The Class 2 Key register can be stored in plaintext form if a split key was loaded into it with a KEY command with PTS=1 or if a key is loaded into the Class 2 Key register with a KEY command with PTS=1 and the MDHA is run in INIT mode to create a split key.</p> <p>An error is generated</p> <ul style="list-style-type: none"> • if PTS=1 & ENC=1. • if PTS=1 & NWB=1. • if PTS=1 & KDEST=01b (PKHA E-Memory). • if a non-split key is stored from the Class 2 Key register after the Class 2 Key register was loaded with a KEY command with PTS=1. • if the Class 1 Key register is stored after the Class 1 Key register was loaded using a KEY command with PTS=1.

Table continues on the next page...

Table 7-14. KEY command field descriptions (continued)

Field	Description
13-10	Reserved
9-0	Key Length
LENGTH	This field defines the length of the key in bytes. If the key is encrypted, this is the decrypted length of the key material only. The built-in key decryption operation produces output whose length is as specified in the LENGTH field. ECB encrypted keys are padded to 16-byte boundaries, so the KEY command reads enough input to read the entire encrypted key. CCM-encrypted keys have a 6-byte nonce, a 6-byte MAC, and padding of up to 7 bytes. The length is checked to ensure it is not too large for the specified destination.
<i>Additional words of KEY command:</i>	
POINTE R	If IMM = 0, this field is a pointer to the key to be loaded. If IMM = 1, this field is not present. NOTE: This field is not present for SEQ KEY Commands.

7.10 LOAD commands

NOTE

In the following discussion, the term 'LOAD command' refers to both the SEQ and non-SEQ forms of the command.

LOAD commands are used to load values into registers, either directly from the descriptor (a LOAD IMMEDIATE command contains constant data within the command) or from a memory location addressed by a pointer within the command. The SEQ LOAD command is identical to the LOAD command except that no address is specified, the VLF bit replaces the SGF bit, and the immediate bit cannot be set. See [SEQ vs non-SEQ commands](#). (Note that while SEQ KEY and SEQ FIFO LOAD have an AIDF bit, SEQ LOAD does not.)

When reading from an external address, the LOAD command, whether SEQ or non-SEQ, uses hardware within DECO to schedule DMA transactions. This command will block until that hardware is available. For LOAD IMM, if the DMA hardware is required but is in use, the command will block until the DMA hardware becomes available. (The command may block for other reasons as well, as documented in a following table.) Once the command is handed off to the responsible hardware, descriptor execution will continue with the next command. Therefore, the requested data may not be present for some time. It is up to the descriptor writer to ensure that the data arrives prior to attempting to use it. Paying attention to the blocking nature just discussed is critical in order to avoid hanging descriptors.

The definitions of the OFFSET and LENGTH fields in the LOAD command can depend on the CLASS and destination (DST) fields. The first table below shows the LOAD command fields, the second table defines the fields, and the third table defines the legal destinations and how each destination affects the other fields.

Table 7-15. LOAD command format

31–27	26–25	24	23	22–16
CTYPE = 00010 or 00011	CLASS	SGF or VLF	IMM	DST
15–8			7–0	
OFFSET			LENGTH	
<i>Additional words of LOAD command:</i>				
Pointer (one or two words, see Address pointers) or Value (if immediate, one or more words)				

Table 7-16. LOAD command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=00010b : LOAD command If CTYPE=00011b : SEQ LOAD command
26-25 CLASS	Class. The algorithm class of the data to be loaded. If CLASS=00b : Load class-independent objects in CCB. If CLASS=01b : Load Class 1 objects in CCB. If CLASS=10b : Load Class 2 objects in CCB. If CLASS=11b : Load objects in DECO.
24 SGF or VLF	Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF) flag. Meaning depends on CTYPE. If CTYPE = 00010 (LOAD), this bit is the Scatter/Gather table Flag (SGF). If SGF=0 : The pointer points to actual data. If SGF=1 : The pointer points to a scatter/gather table. NOTE: If the IMM bit is set, it is an error for this bit to be set. If CTYPE = 00011 (SEQ LOAD), this bit is the Variable Length Flag (VLF). If VLF=0 : The LENGTH field indicates the length of the data. If VLF=1 : The length of the data is variable. SEC uses the length in the Variable Sequence In Length register rather than the value in the LENGTH field. However, an error will be generated if the values in the VSIL register and OFFSET field are not a valid combination as indicated in table Table 7-17 .
23 IMM	Immediate Flag If CTYPE = 00010 (LOAD) If IMM=0, the data to be loaded is found at the location pointed to by the address pointer. If IMM=1, the data to be loaded follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. NOTE: If the SGF bit is set, it is an error for this bit to be set.

Table continues on the next page...

Table 7-16. LOAD command field descriptions (continued)

Field	Description
	If CTYPE = 00011 (SEQ LOAD) IMM must be set to 0. Setting IMM to 1 generates an automatic error.
22-16 DST	The DST value defines the destination register, such as CONTEXT, ICV, or IV. See Table 7-17 for a list of supported destinations.
15-8 OFFSET	OFFSET defines the start point for writing within the destination.
7-0 LENGTH	Length of the data. The value in the DST field determines whether the length is specified in bytes or words. See Table 7-17 for details.
<i>Additional words of LOAD command:</i>	
31-0 POINTER/ VALUE	Address pointer if IMM = 0 or the immediate value if IMM = 1. Note that the immediate value occupies as many words as required to fit the number of bytes specified in the LENGTH field. Data is left aligned. NOTE: This field is present only for LOAD Commands (that is, not for SEQ LOAD Commands).

SEC can accomplish the data transfer associated with a LOAD immediate command in two different ways:

- Using a direct (non-DMA) path to the register, referred to as a direct immediate load
- Using SEC's internal-transfer DMA

SEC automatically selects the appropriate transfer mechanism as follows:

- SEC selects the direct immediate load data path (the first bullet above) if the restrictions are met because this is the fastest of the transfer mechanisms (see following paragraph).
- If the data length or offset restrictions are not met, SEC automatically selects the internal-transfer DMA data path (the second bullet above).

The direct immediate load is the most efficient of the transfer mechanisms, but it has the following restrictions:

- It can transfer only 4 or 8 bytes, unless the destination is the input Data FIFO, the Auxiliary Data FIFO, or the output Data FIFO, in which case the length must be no more than 8 bytes.
- The sum of the data length and the offset cannot be larger than 8, meaning the legal combinations of length and offset are either
 - 4 bytes with an offset of 0 or 4
 - 8 bytes with an offset of 0
 - 4 bytes with any multiple of a 4-byte offset if the destination is a context register
 - 8 bytes with any multiple of an 8-byte offset if the destination is a context register

As shown in [Table 7-17](#), some registers can be loaded only with a LOAD IMM command. These registers always use the direct immediate load data path. Other registers can be loaded using either the LOAD or LOAD IMM form of the command.

As shown in [Table 7-17](#), some LOAD destinations are control data registers and other destinations are message data registers. Data loaded into or stored from control data registers is regarded as word-oriented data, whereas data loaded into or stored from message data registers is regarded as byte strings.

The same swapping operations can be configured independently for:

- each Job Ring (see the Job Ring Configuration Register (JRCONFGR))
- the Queue Manager Interface (see the Queue Interface Control Register (QICTL))
- the AIOP Interface (see AIOP Interface Configuration Register (AICR))

NOTE

For those destinations that allow immediate loads with a nonzero offset, the combination of offset=0 length=4 is equivalent to the combination of offset=4 length=4. This has been done to maintain backward compatibility. Both of these combinations will load the right-most word of the destination. Therefore, in order to load the left-most word of the destination, the combination must be offset=0 length=8. This is ONLY the case when IMM=1 and, therefore, does not affect the SEQ LOAD command. When IMM=0, offset=0 length=4 will load the left-most word of the destination while offset=4 length=4 will load the right-most word. This behavior does not affect any other commands.

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
01	01	Control	4/0 bytes	Yes	C1KSR	Class 1 Key Size Register	The key size registers are normally written with the KEY command. Once KEY SIZE is written, the user cannot modify the key or key size until the key is cleared.
	10		8/0 bytes 4/4 bytes		C2KSR	Class 2 Key Size Register	
02	01	Control	4/0 bytes	Yes	C1DSR	Class 1 Data Size Register	Writes to the data size registers block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready.
	10		8/0 bytes 4/4 bytes		C2DSR	Class 2 Data Size Register	

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
03	01	Control	4/0 bytes	Yes	C1ICVS	Class 1 ICV Size Register	
	10		8/0 bytes 4/4 bytes		C2ICVS	Class 2 ICV Size Register	
05	11	Control	See below	Yes	DCTRL2	DECO Control Register 2	See notes below.
			<p>The DECO Control Register 2 is used to control the operation of DECO by means of a 1-word command that uses the LOAD command fields that normally represent OFFSET and LENGTH. This LOAD must be IMMEDIATE, which means that this DEST cannot be used with SEQ LOAD. The OFFSET and LENGTH fields are redefined as follows:</p> <p>LENGTH[5]: If data for this job was prefetched by the JQ, invalidate the prefetched data so that subsequent reads of that data come from memory rather than the prefetch buffers. This is useful when the descriptor needs to modify the data and then rewinds the input frame.</p> <p>LENGTH[4]: Transfer the value in the output frame tracking length register to the Variable Sequence Output Length Register.</p> <p>LENGTH[0]: Set bit 5 of the PM EVENT Bus entry for this DECO for one clock.</p> <p>All other bits of OFFSET and LENGTH are reserved and must be 0.</p>				
06	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	CCTRL	CHA Control Register	
	11	Control	See below	Yes	DCTRL	DECO Control Register	See notes below.
<p>The DECO Control Register is used to control the operation of DECO by means of a 1-word command that uses the LOAD command fields that normally represent OFFSET and LENGTH. This LOAD must be IMMEDIATE, which means that this DEST cannot be used with SEQ LOAD. The OFFSET and LENGTH fields are redefined as follows:</p> <p>OFFSET[7:6]: Reserved</p> <p>OFFSET[5:4]: Reserved</p> <p>OFFSET[3]: Disable Automatic NFIFO Entries (If disable and enable are both set, disable dominates)</p> <p>OFFSET[2]: Enable Automatic NFIFO Entries</p> <p>OFFSET[1:0]: Change Share Type</p> <p>00 no change</p> <p>01 NEVER share</p> <p>10 OK to share, do propagate errors</p> <p>11 OK to share, don't propagate errors</p> <p>LENGTH[7]: Turn On Output Sequence Length Counting (turned off by doing sequence output pointer rewind)</p> <p>LENGTH[6]: Reset CHA pointer in Output Data FIFO</p>							

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
			LENGTH[5]: Reset Output Data FIFO LENGTH[4]: Process the Output Data FIFO Offset Field (automatically stalls if write burster is busy) LENGTH[3]: Reserved LENGTH[2:0]: Output Data FIFO Offset				
07	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ICTRL	IRQ Control Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	DPOVRD	DECO Protocol Override Register	If bit 31 = 1 the value loaded into DPOVRD overrides the default values in some protocol PDB fields. See individual protocol sections for usage details. If bit 31 = 0, DPOVRD is not used as an override by the built-in protocols. The other bits are defined on a protocol by protocol basis. This register may be used as a source or destination by MATH and MATHI commands.
08	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	CLRW	Clear Written Register	LOAD to the Clear Written Register will block if there are outstanding loads to a Context Register and the Class 1 Key Register is to be cleared.
	11	Control	0-64/ 0-7 bytes	No	MATH0 W	DECO Math Register 0 (Words)	1, 2
09	11	Control	0-56/ 0-7 bytes	No	MATH1 W	DECO Math Register 1 (Words)	1, 2
0A	11	Control	0-48/ 0-7 bytes	No	MATH2 W	DECO Math Register 2 (Words)	1, 2
	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	CISEL	CHA Instance Select Register	-
0B	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	AADSZ	AAD Size Register	-
	11	Control	0-40/ 0-7 bytes	No	MATH3 W	DECO Math Register 3 (Words)	1, 2

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
			0-7 bytes				
0C	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	C1IVSZ	Class 1 IV Size Register	-
0C	11	Control	0-32/ 0-7 bytes	No	MATH4 W	DECO Math Register 4 (Words)	1, 2
0D	11	Control	0-24/ 0-7 bytes	No	MATH5 W	DECO Math Register 5 (Words)	1, 2
0E	11	Control	0-16/ 0-7 bytes	No	MATH6 W	DECO Math Register 6 (Words)	1, 2
0F	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ALTDS1	Alternate Data Size Class 1 Register (aliased to the Class 1 Data Size Register)	The ALTDS1 destination can be used only with a LOAD Immediate command. Writes to the ALTDS1 block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready. The alternate address for the Class 1 Data Size register limits handling of the data type "Special Authentication Data" as authentication/protection to the performance counters.
0F	11	Control	0-8/ 0-7 bytes	No	MATH7 W	DECO Math Register 7 (Words)	1, 2
10	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKASZ	PKHA A Size Register	This holds the size of the data in the PKHA A RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
11	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKBSZ	PKHA B Size Register	This holds the size of the data in the PKHA B RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
12	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKNSZ	PKHA N Size Register	This holds the size of the data in the PKHA N RAM.

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
							The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
13	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	Yes	PKESZ	PKHA E Size Register	This holds the size of the data in the PKHA E RAM. The descriptor writer must ensure that any bits written to this register above the width of the register are 0.
20	01	Message	0-128/ 0-128 bytes	No	CTX1	Class 1 Context Register	A LOAD IMM to a context register blocks if there are any outstanding external loads to either context register.
	10	Message	0-128/ 0-128 bytes	No	CTX2	Class 2 Context Register	A non_IMM LOAD to a context register blocks if the CCB DMA is writing to either context register.
30	11	Control	0-64/ 0 bytes	No	MATH0D W	DECO Math Register 0 (Double Word)	1, 3
31	11	Control	0-56/ 0 bytes	No	MATH1D W	DECO Math Register 1 (Double Word)	1, 3
32	11	Control	0-48/ 0 bytes	No	MATH2D W	DECO Math Register 2 (Double Word)	1, 3
33	11	Control	0-40/ 0 bytes	No	MATH3D W	DECO Math Register 3 (Double Word)	1, 3
34	11	Control	0-32/ 0 bytes	No	MATH4D W	DECO Math Register 4 (Double Word)	1, 3
35	11	Control	0-24/ 0 bytes	No	MATH5D W	DECO Math Register 5 (Double Word)	1, 3
36	11	Control	0-16/ 0 bytes	No	MATH6D W	DECO Math Register 6 (Double Word)	1, 3
37	11	Control	0-8/ 0 bytes	No	MATH7D W	DECO Math Register 7 (Double Word)	1, 3
38	11	Control	0-64/ 0-7 bytes	No	MATH0B	DECO Math Register 0 (Bytes)	1, 4
39	11	Control	0-56/ 0-7 bytes	No	MATH1B	DECO Math Register 1 (Bytes)	1, 4
3A	11	Control	0-48/ 0 bytes	No	MATH2B	DECO Math Register 2 (Bytes)	1, 4

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
			0-7 bytes				
3B	11	Control	0-40/ 0-7 bytes	No	MATH3B	DECO Math Register 3 (Bytes)	1, 4
3C	11	Control	0-32/ 0-7 bytes	No	MATH4B	DECO Math Register 4 (Bytes)	1, 4
3D	11	Control	0-24/ 0-7 bytes	No	MATH5B	DECO Math Register 5 (Bytes)	1, 4
3E	11	Control	0-16/ 0-7 bytes	No	MATH6B	DECO Math Register 6 (Bytes)	1, 4
3F	11	Control	0-8/ 0-7 bytes	No	MATH7B	DECO Math Register 7 (Bytes)	1, 4
40	01	Message	0-96/0-95 bytes	No	KEY1	Class 1 Key Register	The key registers are normally written by the KEY Command, but can be written by a LOAD Command using this DST value. In this case the KEY SIZE register must be written by a separate command after the KEY register has been loaded. A LOAD IMM to a key register blocks if there are any outstanding external loads to either key register. A non_IMM LOAD to a key register blocks if the CCB DMA is writing to either key register.
	10	Message	0-128/ 0-127 bytes	No	KEY2	Class 2 Key Register	
	11	Control	1-64/ 1-63	No	DESC BUF	DECO descriptor buffer	See comments below. For LOADs into the Descriptor Buffer the values in the LENGTH and OFFSET field are specified in 4-byte words. An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the destination within the descriptor buffer. Note that the OFFSET is relative to the start of the descriptor buffer. For SEQ LOAD, the data written into the descriptor buffer is read from the current location pointed to by the input sequence pointer (there is no offset with respect to the source address).
70	00	Control	4 or 8/ 0 bytes	Yes	NFSL	NFIFO and size register(s)	Using the Immediate data, write an NFIFO entry and load the Size register from the DL or PL field in that NFIFO entry. Also see note below.

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
-	-	-	This creates an NFIFO entry from 4 or 8 bytes of IMM data and also writes to one or more size registers. The entry's DL or PL field is filled in with the same value loaded into the size register(s). The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.				
71	00	Control	0-7 0 bytes	Yes	NFSM	NFIFO and size register(s)	Using the Immediate data write an NFIFO entry and load the size register using for PL or DL the least-significant 32-bits of the MATH register selected by means of the three ls bits of the LENGTH field. Also see note above.
72	00	Control	4 or 8/ 0 bytes	Yes	NFL	NFIFO	Using the Immediate data write an NFIFO entry. This is equivalent to DST value 7Ah. Also see note below.
-	-	-	This creates an NFIFO entry from 4 or 8 bytes of IMM data. No size registers are written.				
73	00	Control	0-7 0 bytes	Yes	NFM	NFIFO	Using the Immediate data write an NFIFO entry filling in the DL or PL field from the least-significant 32-bits of the MATH register selected via the three ls bits of the LENGTH field. Also see note above.
74	00	Control	4 or 8/ 0 bytes	Yes	SL	Size register(s)	With the Immediate data in NFIFO entry format, load the size register(s) from the DL or PL field in the Immediate data but do not write an entry into the NFIFO. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.
75	00	Control	0-7 0 bytes	Yes	SM	Size register(s)	Load the size register(s) from the value in the MATH register that is selected by means of the three ls bits of the LENGTH field. No NFIFO entry is loaded. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.
NOTE: For DST values 70h, 71h, 74h, and 75h, the particular size registers that are loaded depend on the CHAs that are selected and the DTYPE field of the entry that is written into the NFIFO. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.							

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
<p>NOTE: For DST values 70h, 72h, 74h, and 7Ah, the direct destination depends on the value in the LENGTH field. If the LENGTH is 4, then the direct destination is the NFIFO. However, if the LENGTH field is 8, then the direct destination is special control hardware which breaks up large lengths so that the maximum length permitted in an NFIFO entry is not exceeded. This hardware pushes as many entries into the NFIFO as necessary. The special control hardware will stall if the NFIFO is full, resuming when space becomes available.</p> <p>NOTE: For DST values 70h-75h and 7Ah, the LOAD will block if the NFIFO is full. In addition, further access to the NFIFO will block if the hardware which breaks up large entries is in use.</p>							
76	00	Message	4/0 bytes 8/0 bytes 4/4 bytes	Yes	IDFNS	Input Data FIFO Nibble Shift Register	See notes below.
<p>Inserts the rightmost 4 bits of the immediate value into the input to the Class 1 Alignment Block, which causes the remainder of the input data to be shifted by one nibble. This nibble alignment continues until the L1 bit or F1 bit in an NFIFO entry is encountered. Thereafter, input to the Class 1 Alignment Block will not be nibble shifted unless the IDFNSR is written again. Any nibble remaining in the shift register will remain there once the last or flush is seen. This means that if there are N bytes of data, to get the last nibble out requires NFIFO entries totaling N+1 bytes.</p>							
77	00	Message	4/0 bytes 8/0 bytes 4/4 bytes	Yes	ODFNS	Output Data FIFO Nibble Shift Register	See notes below.
<p>Inserts the rightmost 4 bits of the immediate value into the output from a Class 1 CHA, so subsequent data from that Class 1 CHA is now shifted one nibble. Data from other sources (such as MOVE Command or LOAD IMM to the Output Data FIFO) will not be concatenated correctly. This nibble alignment continues until the CHA Done signal is asserted. Thereafter, output from a second operation, even from the same CHA, is not nibble shifted unless the ODFNSR is written again. Any valid nibble remaining is always pushed into the output FIFO following the assertion of CHA Done.</p>							
78	00	Message	1-8/0 bytes	Yes	AUXDAT A	Auxiliary Data FIFO	See notes below.
<p>This DST value can be used to provide data to the Auxiliary Data FIFO. Each LOAD IMM command can load 1-8 left-aligned bytes. Byte swapping is done automatically. The LOAD IMM to AUXDATA will stall if there is no room left in the AUXDATA buffer. The AUXDATA path should be treated in the same way as the input DATA FIFO. If NFIFO entries are not used properly, execution will hang if more LOADs are done to the AUXDATA buffer but room can't be created by draining that buffer via NFIFO entries and the corresponding alignment block being drained by CHAs or the MOVE command. Note that data can also be supplied to the Auxiliary Data FIFO by using a MOVE command.</p>							
7A	00	Control	4/0 bytes 8/0 bytes **	Yes	NFIFO	NFIFO	See notes below.
<p>The NFIFO is normally written by means of the FIFO LOAD command, but the NFIFO can be written using this DST value with a LOAD Immediate command.</p> <p>** If LENGTH = 8, the LOAD command is interpreted as follows: word 1: LOAD IMM, LENGTH = 8, DST = 7Ah</p>							

Table continues on the next page...

Table 7-17. LOAD command DST, LENGTH, and OFFSET field values (continued)

DST value (hex)	Class (binary)	Control data/message data	Legal values in LENGTH/OFFSET fields	Must use IMM?	Tag	Internal register	Comment
			word 2: bits [31:12] contain the NFIFO entry if not padding type, else bits [31:10] contain the NFIFO entry word 3: Extended Length (DECO creates as many NFIFO entries as needed to satisfy the Extended Length - see the notes above)				
7C	00	Message	1-8/0 bytes	Yes	IFIFO	Input Data FIFO	See notes below The input data FIFO is normally written by means of the FIFO LOAD command, but the Input Data FIFO can be written using this DST value with a LOAD Immediate Command. The data must be left-aligned and byte swapping will be done. This LOAD will block if there is no more room in the input data FIFO or if there is other data heading to the input data FIFO. Care should be taken since this block could turn into a hang if the LOAD is unable to proceed.
7E	00	Message	1-8/0 bytes	Yes	OFIFO	Output Data FIFO	Must use a LOAD Immediate command. The data must be left-aligned. This LOAD will block if there is no more room in the output data FIFO. It is up to the descriptor writer to ensure that this LOAD will not collide with a move to the output FIFO or a push to the output FIFO by one of the C1 CHAs. Care should be taken since this block could turn into a hang if the LOAD is unable to proceed.
All combinations of value and class that do not appear in this table are reserved							

1. May be affected by protocols. Note that using the LOAD command to place values in the math registers does not update the MATH status bits (see MNV, MN, MC and MZ). Because the math registers are in contiguous addresses, it is possible to load more than one math register simultaneously. A LOAD IMM to a math register blocks if there are any outstanding external loads to any math register. A non_IMM LOAD to a math register blocks if the CCB DMA is writing to any math register.
2. When this destination is used, the data loaded into the Math register will be treated as words.
3. When this destination is used, the data loaded into the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers. It is recommended that only full double words be loaded.
4. When this destination is used, the data loaded into the Math register will be treated as bytes.

The following table details which size registers are written when loading NFIFO entries when one, or more, size registers are also written. (DST values 70h, 71h, 74h, and 75h.) An entry of "None/Reserved" means that no size register will be written and NXP reserves the right to assign that DTYPE to some size register in the future. Therefore, such DTYPEs should not be used as they could break compatibility.

Table 7-18. Which Size Registers Are Loaded

DTYPE (hex)	If PKHA selected	If Class 1 CHA selected, but not PKHA	If Class 2 CHA selected
0	PKHA A Size	None/Reserved	Class 2 Data Size
1	PKHA A Size	AAD Size and Class 1 Data Size	Class 2 Data Size
2	PKHA A Size	IV Size and Class 1 Data Size	Class 2 Data Size
3	PKHA A Size	AAD Size and Class 1 Data Size (but data only counts as Auth data)	Class 2 Data Size
4	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
5	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
6	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
7	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
8	PKHA N Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
9	PKHA E Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
A	None/Reserved	ICV Size	If PKHA selected: Class 2 ICV Size If PKHA not selected: if both Class 1 and Class 2 CHAs selected, Class 2 Data Size else Class 2 ICV Size
B	None/Reserved	None/Reserved	None/Reserved
C	PKHA A Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
D	PKHA B Size	None/Reserved	If PKHA selected, Class 2 Data Size else None/Reserved
E	None/Reserved	None/Reserved	None/Reserved
F	None/Reserved	Class 1 Data Size	Class 2 Data Size

7.11 FIFO LOAD command

NOTE

In the following discussion, the term 'FIFO LOAD command' refers to both the SEQ and non-SEQ forms of the command.

FIFO LOAD commands are used to load message data, PKHA data (other than for the E Memory), IV, AAD, ICV, and bit-length message data into the input data FIFO. The SEQ FIFO LOAD command is identical to the FIFO LOAD command except that no address is specified, the command contains an AIDF bit in place of the IMM bit, and the VLF bit instead of the SGF bit. See [SEQ vs non-SEQ commands](#).

As the only destination is the input data FIFO, this command does not include a DST field. The FIFO INPUT DATA TYPE is used to indicate what type of data is being loaded and whether the length is specified in bits or bytes. The length of data other than message data is measured in bytes. The length of message data can be specified in either bits or bytes (see [Table 7-22](#)). If automatic info FIFO entries are enabled, the FIFO LOAD command writes the appropriate size register(s) and the required info FIFO entry for the specified input data type. This command will block for a variety of reasons:

1. FIFO LOAD IMM will block if the input FIFO is full.
2. FIFO LOAD IMM will block if the DMA is required to move the data but the DMA is busy.
3. If a DMA transaction is required, the FIFO LOAD command will block if the hardware which schedules DMA transactions is in use.
4. If there are external reads destined for the input data FIFO, FIFO LOAD IMM will block until that data arrives.
5. The FIFO LOAD command uses the same logic as the LOAD command does when loading NFIFO entries with LENGTH=8. If an NFIFO entry is required and this logic is busy, the command will block.
6. A SEQ FIFO LOAD SKIP will block if there is a buffer release pending. Once the pending release completes, the command will proceed.

Table 7-19. FIFO LOAD command format

31–27	26–25	24	23	22	21–16
CTYPE = 00100 or 00101	CLASS	SGF or VLF	IMM or AIDF	EXT	INPUT DATA TYPE
15–0					
LENGTH					
<i>Additional words of FIFO LOAD command:</i>					
Pointer (one or two words, see Address pointers) or Value (if immediate, one or more words)					
EXT LENGTH (two words, present if EXT=1)					

Table 7-20. FIFO LOAD command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=00100b : FIFO LOAD command If CTYPE=00101b : SEQ FIFO LOAD command
26-25 CLASS	Class. Cryptographic algorithm class. If CLASS=00b : Used for SEQ FIFO LOAD only. Skips the specified length in memory without scheduling any read transactions and no data is actually read. However, Scatter/Gather Table entries will be read as needed. FIFO INPUT DATA TYPE field is ignored. No info FIFO entry is generated. If CLASS=01b : Load FIFO with data for a Class 1 CHA.

Table continues on the next page...

Table 7-20. FIFO LOAD command field descriptions (continued)

Field	Description
	<p>If CLASS=10b : Load FIFO with data for a Class 2 CHA.</p> <p>If CLASS=11b : Load FIFO with data for both Class 1 and Class 2 CHAs (both In Snoop and Out Snoop; the INPUT DATA TYPE will distinguish between them).</p> <p>NOTE: The CLASS field must be non-zero for FIFO LOAD commands because the 00b case indicates skipping, which is illegal for FIFO LOAD. This is true even when automatic information FIFO entries are disabled.</p>
24 SGF or VLF	<p>Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF).</p> <p>If CTYPE = 00100b (FIFO LOAD), this bit is the Scatter/Gather table Flag (SGF).</p> <p>If SGF=0, the pointer points to actual data.</p> <p>If SGF=1, the pointer points to a Scatter/Gather Table.</p> <p>NOTE: If the IMM bit is set, it is an error for this bit to be set.</p> <p>If CTYPE = 00101b (SEQ FIFO LOAD), this bit is the Variable Length Flag (VLF).</p> <p>If VLF=0, the LENGTH field indicates the length of the data.</p> <p>If VLF=1, the length is variable. SEC uses the length in the Variable Sequence In Length register and ignores the LENGTH field.</p> <p>NOTE: It is an error to set VLF = 1 when the EXT bit = 1.</p>
23 IMM or AIDF	<p>Immediate Flag(IMM) or Already in Input Data FIFO (AIDF)</p> <p>If CTYPE = 00100 (FIFO LOAD), this bit is the Immediate Flag (IMM).</p> <p>If IMM=0, the data begins at the location pointed to by the Pointer field.</p> <p>If IMM=1, the data follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word.</p> <p>NOTE: It is an error if this bit is set when SGF = 1 or EXT = 1.</p> <p>If CTYPE = 00101 (SEQ FIFO LOAD), this is the Already in Data FIFO (AIDF) bit.</p> <p>If AIDF is 0, SEC will read the input sequence data from memory.</p> <p>If AIDF is 1, SEC will not read the input sequence data (because it is already in the Input Data FIFO). This form is convenient since the NFIFO and Data Size Registers will be loaded automatically if Automatic Info FIFO Entries is enabled. As a result, a 1-word command can replace two 2-word commands.</p>
22 EXT	<p>Extended Length</p> <p>If EXT=0 : Input data length is solely determined by the 16-bit LENGTH field,</p> <p>If EXT=1 : Input data length is determined by the 32-bit EXTENDED LENGTH. If the INPUT DATA TYPE indicates a bit length, then the EXTENDED LENGTH field contains the number of full bytes and the right 3 bits of the LENGTH field, if nonzero, indicate the number of valid bits in the last byte. See Bit length data.</p> <p>NOTE: It is an error if this bit is set when IMM is also set.</p>
21-16 INPUT DATA TYPE	<p>FIFO input data type</p> <p>See Table 7-22 for a description of the supported types. When automatic information FIFO entries are disabled, (SEQ) FIFO LOAD Commands ignore the FIFO INPUT DATA TYPE field.</p>
15-0 LENGTH	<p>Length of data</p> <p>If EXT = 0 : LENGTH = number of bytes of input data, or for bit length message data, the number of bits of input data.</p>

Table continues on the next page...

Table 7-20. FIFO LOAD command field descriptions (continued)

Field	Description
	If EXT = 1 : The EXT LENGTH field indicates the number of full bytes of data. The LENGTH field is ignored (unless if FIFO INPUT DATA TYPE = bit length, in which case the least-significant 3 bits of the LENGTH field indicate the number of valid bits in an additional byte of data). See Bit length data .
<i>Additional words of FIFO LOAD command:</i>	
POINTER	If IMM = 0, this field is a pointer to the data to be loaded. If IMM = 1, this field is not present. NOTE: This field is not present for SEQ FIFO LOAD Commands.
31-0 EXT LENGTH	For EXT = 0, this field not present. For EXT = 1, EXTENDED LENGTH specifies number of full bytes of data to load. For bit-length data, the least-significant 3 bits of the LENGTH field indicate the number of valid bits in an additional byte of data. See Bit length data .

7.11.1 Bit length data

If the INPUT DATA TYPE indicates that the input data type being loaded is bit-length message data, the LENGTH field is defined as a bit count, as shown in the "Number of Bits" row in the following figure. This can also be interpreted as a "Number of Full Bytes field" in bits positions 15-3, and a "Number of Additional Valid Bits" field in bit positions 2-0. These additional valid bits are in the next byte after the number of full bytes, starting with the bit on the left. For example, if the LENGTH field is 0101h, SEC loads 33 bytes, with only the leftmost bit of the 33rd byte valid. Note that the entire 33rd byte is read and it is up to the consumer of that last byte to know that only the specified number of bits in the last byte are valid.

The Number of Additional Valid Bits is placed in the NUMBITS field of the Class 1 and/or Class 2 Data Size Register. The NUMBITS field is not visible to any functional logic in SEC other than a subset of the CHAs. (The NUMBITS field may be read via SkyBlue or a store of the Data Size Registers.) The CHAs which receive the NUMBITS field are:

- KFHA
- SNOW F8 and SNOW F9
- ZUCA and ZUCE
- AESA, which will error if it sees a nonzero NUMBITS field.

The following CHAs do not receive the NUMBITS field:

- PKHA
- DES
- CRCA

- MDHA
- RNG

It is possible to use a nonzero NUMBITS field with a CHA which does not receive the NUMBITS field. To do this, add 1 more to the proper Data Size Register. However, note that the remaining bits in the last byte will be whatever values they were at the source of that byte. That is, the remaining bits are not masked to 0.

It is not possible for SEC to automatically concatenate two separate bit fields. For example, if an NFIFO entry for 3 bits is followed by an entry for 5 bits, these entries will NOT result in a one-byte entry. To achieve such concatenation, use the shift operations in the MATH command.

When using automatic NFIFO entries with the FIFO LOAD command to specify bit lengths, C1 must always have Flush or Last set and C2 must always have Last set. Failure to set the Last and Flush bits as stated will result in an error. When manually generating NFIFO entries, no error will be generated if the Flush or Last bits are not set as suggested. However, as stated above, each incomplete byte's remaining bits will have whatever values they had at the source.

If the FIFO LOAD command is used to generate the NFIFO entry, the number of bytes specified in the LENGTH field for the NFIFO entry will be the number of full bytes if the NUMBITS field is zero, and the number of full bytes plus one if the NUMBITS field is nonzero. This ensures that the CHA will consume the last, partial, byte. If the descriptor writer is manually generating the NFIFO entries, care must be taken to handle the length properly.

Table 7-21. Specifying data with residual bit length

15–0	
Number of Bits	
<i>Alternate interpretation:</i>	
15–3	2–0
Number of Full Bytes	Number of Additional Valid Bits

If the input data's bit length is equal to or greater than 2^{16} , set the EXT bit and use the EXTENDED LENGTH field to specify the number of full bytes. The upper 13 bits of LENGTH must be zero, with the rightmost 3 bits specifying the number of additional valid bits as before.

7.11.2 FIFO LOAD input data type

Table 7-22 contains an enumeration of the various built-in input data FIFO data types. This field is ignored if neither of the Class bits are set. Only message data can have a bit length (as opposed to a byte length), and such message data must always have Flush or Last set.

Note that the NFIFO source type is not needed, as it is always inferred from the FIFO LOAD input data type. The length for the NFIFO entry is the amount of data being placed in the data FIFO. (One is added to the length in the case of bit-length data if the number of bits in the last byte is nonzero.) Also, the Last and Flush bits are always sent as 0 except with the last byte of data, in which case the values shown in the table are sent.

Note also that data should not be left in the Input Data FIFO with the expectation that it will be shared with a subsequent shared descriptor executing in the same DECO. This may cause data to be lost if the next shared job executes in a different DECO.

With the exception of IV and AAD, the FIFO LOAD command does not do any padding. This is because all algorithmic padding requires a pad length or a special last byte, which means that at least one byte of padding is required. Therefore, the padding can be sent using a padding NFIFO entry.

Table 7-22. FIFO LOAD input data type field

FIFO Input Type Field Bit #						Meaning
21	20	19	18	17	16	
00b		PKHA				PKHA Register Load All values not specified below are reserved. This data is always flushed. An error is asserted if the length is larger than fits in the PKHA RAM. NOTE: Loading quadrants of a given PKHA register with different-sized values may cause invalid data to be loaded into the quadrants. To avoid this issue, make sure that all quadrants of a given register have the same size values by left-filling short values with zero. If it is necessary to load different-sized values in quadrants of the same register, insert a JUMP command between quadrant loads (which will wait for automatic information FIFO entries to be processed): jump jsl = 1, type = 0, cond = nifp, local offset = 1. NOTE: The PKHA E RAM can not be loaded via the FIFO LOAD command using automatic NFIFO entries. Use the KEY command or get the data into the input data FIFO without an automatic NFIFO entry and then manually create an NFIFO entry and write the PKHA E Size register.
0	0	0	0	0	0	PKHA A0
0	0	0	0	1	0	PKHA A1
0	0	1	0	0	0	PKHA A2
0	0	1	1	0	0	PKHA A3
0	1	0	0	0	0	PKHA B0
0	1	0	1	0	0	PKHA B1
0	1	1	0	0	0	PKHA B2

Table continues on the next page...

Table 7-22. FIFO LOAD input data type field (continued)

FIFO Input Type Field Bit #						Meaning
21	20	19	18	17	16	
		0	1	1	1	PKHA B3
		1	0	0	0	PKHA N
		1	1	0	0	PKHA A
		1	1	0	1	PKHA B
00b		1	1	1	1	Place the data into the input Data FIFO but do not generate an NFIFO entry and do not write any size registers, even if automatic NFIFO entries are enabled.
010b			LC2	LC1	FC1	Message Data
011b			LC2	LC1	FC1	Message Data for Class 1 out-snooped to Class 2
100b			LC2	LC1	FC1	IV. If Last or Flush for Class 1 is set, Class 1 is padded to 16-byte boundary with 0. No padding is done for Class2. No padding is done for Class 1 if the data naturally ends on a 16-byte boundary.
101b			1	LC1	FC1	Bit-length message data. Last of Class 2 treated as set even if not set in the command; either Flush or Last for Class 1 must be set
110b			LC2	LC1	FC1	AAD. If Last or Flush for Class 1 is set. Class 1 is padded to a 16-byte boundary with 0. No padding is done for Class 2. It is an error if Class 2 is specified and Class 1 is not.
111b			LC2	LC1	FC1	ICV
<p>If CLASS 1 is asserted, LC1 means the data defined in the current NFIFO entry is the last data. When LC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, LC1 is ignored.</p> <p>If CLASS 2 is asserted, LC2 means the data defined in the current NFIFO entry is the last data. When LC2 = 1, the last byte of the specified length is made available from the Class 2 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 2 is negated, LC2 is ignored</p> <p>If CLASS 1 is asserted, FC1 means the data defined in the current NFIFO entry is the last data of this type. When FC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, FC1 is ignored.</p> <p>Note that the difference between LC1 and FC1 is only important when the data is going to a CHA. If the data is to be consumed by the CCB DMA, then FC1 should be used as LC1 may confuse a CHA. Similarly, LC2 should not be used if the data will be consumed by the CCB DMA. In such cases, it is better to use a manual NFIFO entry with the FC2 bit set.</p>						
All values not specified are reserved.						

7.12 ECPARAM command

The ECPARAM command is used to load one parameter from a set of built-in elliptic curve parameters into one of the PKHA registers. This command will block until any data already in transit to the Input Data FIFO has been received.

The DEST value 1111b is used to leave the selected parameter in the Input Data FIFO. No NFIFO entry is generated. If automatic NFIFO entries have been disabled, the ECPARAM command will leave the parameter in the Input Data FIFO and no NFIFO entry is generated. Note that when no NFIFO entry is generated, no PKHA size register is

written. Furthermore, the descriptor writer is responsible for the NFIFO entry to get this parameter to its final destination.^{3, 4} In all other legal cases, the ECPARAM command will result in loading the parameter into the selected PKHA register by automatically creating the NFIFO entry and automatically writing the correct PKHA size register with the correct value.

Table 7-23. ECPARAM command, format

31–27		26–20		19–16	
CTYPE = 11100		0100000		DEST	
15	14–8		7–4		3–0
Reserved	DOMAIN		Reserved		PARAMETER

Table 7-24. ECPARAM command, field descriptions

Field	Description
31-27 CTYPE	Command type IF CTYPE=11100b : ECPARAM command
26-20	0100000b. This particular 7-bit value is mandatory. Any other value will generate an error.
19-16 DEST	Destination. This field specifies which PKHA register to load: 0000b - PKHA A0 0001b - PKHA A1 0010b - PKHA A2 0011b - PKHA A3 0100b - PKHA B0 0101b - PKHA B1 0110b - PKHA B2 0111b - PKHA B3 1000b - PKHA N 1100b - PKHA A 1101b - PKHA B 1111b - Input Data FIFO--This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries. All other values are reserved.
15	Reserved
14-8	Elliptic Curve Domain. This field selects one of the built-in elliptic curve domains:

Table continues on the next page...

3. This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries.
4. If parameters of different sizes are to be loaded into different quadrants of the same PKHA register, then it is up to the user to ensure that the first parameter is completely loaded before executing the second ECPARAM command. This is because the same size register is used and can't be changed for the second parameter until the first has been successfully loaded.

Table 7-24. ECPARAM command, field descriptions

Field	Description	
DOMAIN	DOMAIN	ECC Domain Selected
	0h	P-192
	1h	P-224
	2h	P-256
	3h	P-384
	4h	P-521
	5h	brainpoolP160r1
	6h	brainpoolP160t1
	7h	brainpoolP192r1
	8h	brainpoolP192t1
	9h	brainpoolP224r1
	Ah	brainpoolP224t1
	Bh	brainpoolP256r1
	Ch	brainpoolP256t1
	Dh	brainpoolP320r1
	Eh	brainpoolP320t1
	Fh	brainpoolP384r1
	10h	brainpoolP384t1
	11h	brainpoolP512r1
	12h	brainpoolP512t1
	13h	prime192v2
	14h	prime192v3
	15h	prime239v1
	16h	prime239v2
	17h	prime239v3
	18h	secp112r1
	19h	wtls8
	1Ah	wtls9
	1Bh	secp160k1
	1Ch	secp160r1
	1Dh	secp160r2
	1Eh	secp192k1
1Fh	secp224k1	
20h	secp256k1	
40h	B-163	
41h	B-233	
42h	B-283	
43h	B-409	
44h	B-571	

Table continues on the next page...

Table 7-24. ECPARAM command, field descriptions (continued)

Field	Description																						
	45h K-163																						
	46h K-233																						
	47h K-283																						
	48h K-409																						
	49h K-571																						
	4Ah wtls1																						
	4Bh sect113r1																						
	4Ch c2pnb163v1																						
	4Dh c2pnb163v2																						
	4Eh c2pnb163v3																						
	4Fh sect163r1																						
	50h sect193r1																						
	51h sect193r2																						
	52h sect239k1																						
	53h Oakley3a																						
	54h Oakley4a																						
All values not specified are reserved.																							
7-4	Reserved																						
3-0 PARAMETER	<p>Elliptic Curve Parameter. This field specifies which elliptic curve parameter is to be loaded into the PKHA register specified by DEST.</p> <table border="1"> <thead> <tr> <th>PARAMETER</th> <th>EC Domain Parameter Selected</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>q</td> </tr> <tr> <td>1h</td> <td>r^{fn1}</td> </tr> <tr> <td>2h</td> <td>Gx</td> </tr> <tr> <td>3h</td> <td>Gy</td> </tr> <tr> <td>4h</td> <td>a</td> </tr> <tr> <td>5h</td> <td>b</td> </tr> <tr> <td>6h</td> <td>$R^2 \text{ mod } q$</td> </tr> <tr> <td>7h</td> <td>$R^2 \text{ mod } r^{fn1}$</td> </tr> <tr> <td>8h</td> <td>c^{fn2}</td> </tr> <tr> <td>9h</td> <td>k^{fn1}</td> </tr> </tbody> </table> <p style="text-align: center;">All values not specified are reserved.</p> <p>fn1: The "r", "$R^2 \text{ mod } r$" and "k" parameters are not valid for either OAKLEY domain. fn2: The "C" parameter is not valid for Fp domains (i.e. DOMAIN < 40h).</p>	PARAMETER	EC Domain Parameter Selected	0h	q	1h	r^{fn1}	2h	Gx	3h	Gy	4h	a	5h	b	6h	$R^2 \text{ mod } q$	7h	$R^2 \text{ mod } r^{fn1}$	8h	c^{fn2}	9h	k^{fn1}
PARAMETER	EC Domain Parameter Selected																						
0h	q																						
1h	r^{fn1}																						
2h	Gx																						
3h	Gy																						
4h	a																						
5h	b																						
6h	$R^2 \text{ mod } q$																						
7h	$R^2 \text{ mod } r^{fn1}$																						
8h	c^{fn2}																						
9h	k^{fn1}																						

7.13 STORE command

NOTE

In the following discussion, the term 'STORE command' refers to both the SEQ and non-SEQ forms of the command.

STORE commands are used to read data from various registers and write them to a system address. The SEQ STORE command is identical to the STORE command except that no address is specified and the VLF bit replaces the SGF bit. See [SEQ vs non-SEQ commands](#).

The definitions of the OFFSET and LENGTH fields in the STORE command can depend on the CLASS and source (SRC) fields. [Table 7-25](#) shows the command fields, and [Table 7-27](#) defines OFFSET and LENGTH as well as additional behaviors of the command.

As shown in the following table, STORE data sources can be both control and message data registers. Data stored from control data registers are regarded as word-oriented data, whereas data stored from message data registers are regarded as byte strings.

The same swapping operations can be configured independently for:

- each Job Ring (see the Job Ring Configuration Register (JRCFGR))
- the Queue Manager Interface (see the Queue Interface Control Register (QICTL))
- the AIOP Interface (see AIOP Interface Configuration Register (AICR))

Table 7-25. STORE command format

31–27	26–25	24	23	22–16
CTYPE = 01010 or 01011	CLASS	SGF or VLF	IMM	SRC
15–8			7–0	
OFFSET			LENGTH	
<i>Additional words of STORE command:</i>				
Pointer (one or two words, see Address pointers)				
If immediate (IMM = 1), one or more words of data appear here				

Table 7-26. STORE command field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=01010b : STORE command If CTYPE=01011b : SEQ STORE command

Table continues on the next page...

Table 7-26. STORE command field descriptions (continued)

Field	Description
26-25 CLASS	<p>Algorithm class of the data object to be stored</p> <p>See the SRC field for additional explanation. If IMM = 1 a value other than 00b in the CLASS field will cause an error to be generated.</p> <p>If IMM = 0, the following definitions are used:</p> <p>If CLASS=00b : Store class-independent objects from CCB.</p> <p>If CLASS=01b : Store Class 1 objects from CCB.</p> <p>If CLASS=10b : Store Class 2 objects from CCB.</p> <p>If CLASS=11b : Store objects from DECO.</p>
24 SGF or VLF	<p>Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF).</p> <p>If CTYPE = 01010b (STORE), this bit is the Scatter/Gather table Flag (SGF).</p> <p>If SGF=0, the pointer contains the address of the destination for the data to be stored.</p> <p>If SGF=1, the pointer points to a Scatter/Gather Table, which defines the destinations for the data to be stored. Note that SGF should not be set to 1 when using SRC values 41h, 42h, 45h, or 46h. Doing so will cause an error to be generated.</p> <p>NOTE: If the IMM bit is set, it is an error for the SGF bit to be set.</p> <p>If CTYPE = 01011b (SEQ STORE), this bit is the Variable Length Flag (VLF).</p> <p>If VLF=0, the LENGTH field indicates the length of the data.</p> <p>If VLF=1, the data length is variable. SEC uses the length in the Variable Sequence Out Length register rather than the value the LENGTH field. However, an error will be generated if the values in the VSOL register and OFFSET field are not a valid combination as indicated in the table Table 7-27.</p>
23 IMM	<p>Immediate data.</p> <p>If IMM=0 : Data to be stored is found at the location specified by the SRC field.</p> <p>If IMM=1 : Data to be stored follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. For SEQ STORE, the data immediately follows the command; for STORE, the data immediately follows the pointer.</p> <p>NOTE: It is an error if the IMM bit is set when the SGF bit is set. However, the destination of a SEQ STORE can have been defined by a Scatter/Gather Table pointed to by the SEQ OUT PTR Command that initiated the Output Sequence. It is an error if IMM = 1 and the OFFSET field is non-zero.</p>
22-16 SRC	<p>SRC value defines the source (e.g. CONTEXT, ICV, IV) of the data to be stored. See Table 7-27 for a list of supported sources.</p> <p>If IMM = 1 the data to be stored is located as immediate data within the command. Although the SRC field does not specify the source of the data, the SRC field still determines whether the immediate data is treated as message data or control data. When SEC is configured for big endian operation, message data and control data are treated the same. When SEC is configured for little endian operation, control data is byte swapped within words as the immediate data is stored into memory but message data is stored as-is, without byte swapping. SRC = 00h will cause the immediate data to be treated as control data, so when SEC is configured for little-endian operation the data will be byte-swapped within words before it is written to memory. SRC = 7Eh will cause the immediate data to be treated as message data, so the data will be written as-is, without byte swapping. The use of any other SRC value with IMM=1 will cause an error to be generated.</p>
15-8 OFFSET	<p>OFFSET defines the start point for reading within the SRC. For example, if the SRC indicates Class 1 context, the offset can be used to indicate that the data should be read from the fourth byte of context rather than from the beginning. The offset into the descriptor buffer is specified in 4-byte words, but in all other cases the offset is specified in bytes. See Table 7-27 for the legal combinations of OFFSET and LENGTH values.</p>
7-0	<p>Length of the data. For the descriptor buffer, the length is specified in 4-byte words, but in all other cases the length is specified in bytes. See Table 7-27 for the legal combinations of OFFSET and LENGTH values.</p>

Table continues on the next page...

Table 7-26. STORE command field descriptions (continued)

Field	Description
LENGTH	
<i>Additional words of STORE command:</i>	
31-0 POINTER	This field is a pointer to the address in memory where the data is to be stored. NOTE: This field is not present for any SEQ STORE commands or for STORE commands that store the job descriptor (41h and 45h) or shared descriptor (42h and 46h) from the descriptor buffer into memory which use the pointers previously specified for the job and shared descriptors. (Type 40h requires a pointer for the STORE command.)
31-0 VALUE	If IMM = 1, the value is located here. Enough 4-byte words are used to hold the data of size LENGTH.

Table 7-27. STORE command SRC, OFFSET and LENGTH field values

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
00	01	Control	4/0 bytes	MODE1	Class 1 Mode Register	-
	10		8/0 bytes 4/4 bytes	MODE2	Class 2 Mode Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DJQCR	DECO Job Queue Control Register	-
01	01	Control	4/0 bytes	KEYS1	Class 1 Key Size Register	-
	10		8/0 bytes 4/4 bytes	KEYS2	Class 2 Key Size Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DDAR	DECO Descriptor Address Register	-
02	01	Control	4/0 bytes	DATAS1	Class 1 Data Size Register	-
	10		8/0 bytes 4/4 bytes	DATAS2	Class 2 Data Size Register	-
	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DOPSTAT	DECO Operation Status Register	Storing DOPSTAT captures the current "math conditions" (see Table 7-89 , TEST CONDITION field, TEST CONDITION bits when JSL = 0) as well as SEC's current command index. The status is in the left four bytes of this register. The right four

Table continues on the next page...

**Table 7-27. STORE command SRC, OFFSET and LENGTH field values
(continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
						bytes contain the number of bytes written to the SEQ OUT PTR address.
03	01	Control	4/0 bytes	C1ICVS	Class 1 ICV Size Register	-
	10		8/0 bytes 4/4 bytes	C2ICVS	Class 2 ICV Size Register	
03	11	Message	2/0 bytes	DCHKSM	DECO Checksum	An error is generated if length is not 2 or if offset is not 0.
04	11	Control	4/0 bytes 8/0 bytes 4/4 bytes	DICID	DECO ICID Register	Consists of the following fields: CPL/CICID, OPL/OICID, TZ/SDID, IPL/IICID
06	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CCTRL	CHA Control Register	-
07	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	ICTRL	IRQ Control Register	-
08	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CLRW	Clear Written Register	-
	11	Control	0-64 0-7 bytes	MATH0W	DECO Math Register 0 (Words)	1, 2
09	00	Control	4/0 bytes 8/0 bytes 4/4 bytes	CSTA	CCB Status and Error Register	-
	11	Control	0-56 0-7 bytes	MATH1W	DECO Math Register 1 (Words)	1, 2
0A	11	Control	0-48 0-7 bytes	MATH2W	DECO Math Register 2 (Words)	1, 2
0B	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	AADSZR	AAD Size Register	1, 2

Table continues on the next page...

**Table 7-27. STORE command SRC, OFFSET and LENGTH field values
(continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
	11	Control	0-40 0-7 bytes	MATH3W	DECO Math Register 3 (Words)	1, 2
0C	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	C1IVSZ	Class 1 IV Size Register	-
0C	11	Control	0-32/ 0-7 bytes	MATH4W	DECO MATH Register 4 (Words)	1, 2
0D	11	Control	0-24/ 0-7 bytes	MATH5W	DECO MATH Register 5 (Words)	
0E	11	Control	0-16/ 0-7 bytes	MATH6W	DECO Math Register 6 (Words)	
0F	11	Control	0-8/ 0-7 bytes	MATH7W	DECO Math Register 7 (Words)	
10	11	Control	16/0 bytes 32/0 bytes 48/0 bytes 64/0 bytes	GTR	Gather Table Registers	
10	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKASZ	PKHA A Size Register	-
11	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKBSZ	PKHA B Size Register	-
12	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKNSZ	PKHA N Size Register	-
13	01	Control	4/0 bytes 8/0 bytes 4/4 bytes	PKESZ	PKHA E Size Register	-
20	01	Message	0-128/ 0-128 bytes	CTX1	Class 1 Context Register	A STORE from the Class 1 Context Register will automatically block until the Class 1 CHA is done.

Table continues on the next page...

**Table 7-27. STORE command SRC, OFFSET and LENGTH field values
(continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
	10			CTX2	Class 2 Context Register	A STORE from the Class 2 Context Register will automatically block until the Class 2 CHA is done.
20	11	Control	16/0 bytes 32/0 bytes 48/0 bytes 64/0 bytes	STR	Scatter Table Registers	
30	11	Control	0-64 0 bytes	MATH0DW	DECO Math Register 0 (Double Word)	1, 3, 2
31	11	Control	0-56 0 bytes	MATH1DW	DECO Math Register 1 (Double Word)	
32	11	Control	0-48 0 bytes	MATH2DW	DECO Math Register 2 (Double Word)	
33	11	Control	0-40	MATH3DW	DECO Math Register 3 (Double Word)	
34	11	Control	0-32/ 0 bytes	MATH4DW	DECO Math Register 4 (Double Word)	1, 2
35	11	Control	0-24/ 0 bytes	MATH5DW	DECO Math Register 5 (Double Word)	
36	11	Control	0-16/ 0 bytes	MATH6DW	DECO Math Register 6 (Double Word)	
37	11	Control	0-8/ 0 bytes	MATH7DW	DECO Math Register 7 (Double Word)	
38	11	Control	0-64 0-7 bytes	MATH0B	DECO Math Register 0 (Bytes)	1, 4
39	11	Control	0-56 0-7 bytes	MATH1B	DECO Math Register 1 (Bytes)	
3A	11	Control	0-48 0-7 bytes	MATH2B	DECO Math Register 2 (Bytes)	
3B	11	Control	0-40 0-7 bytes	MATH3B	DECO Math Register 3 (Bytes)	

Table continues on the next page...

**Table 7-27. STORE command SRC, OFFSET and LENGTH field values
(continued)**

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
3C	11	Control	0-32/ 0-7 bytes	MATH4B	DECO Math Register 4 (Bytes)	1, 4
3D	11	Control	0-24/ 0-7 bytes	MATH5B	DECO Math Register 5 (Bytes)	
3E	11	Control	0-16/ 0-7 bytes	MATH6B	DECO Math Register 6 (Bytes)	
3F	11	Control	0-8/ 0-7 bytes	MATH7B	DECO Math Register 7 (Bytes)	
40	01	Message	0-128/0-128 bytes	KEY1	Class 1 Key Register	If the corresponding Key Size register has not been written, the STORE or SEQ STORE command may be used to store the key register into memory. After the key size has been written, the key register can be stored to memory only via a FIFO STORE or SEQ FIFO STORE command.
	10	Message	0-128/0-128 bytes	KEY2	Class 2 Key Register	
	11	Control	0-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.
<p>This SRC value can be used to store any portion of the descriptor buffer into memory.</p> <p>The values in the LENGTH and OFFSET field are specified in 4-byte words.</p> <p>offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer. Note that the OFFSET is relative to the start of the descriptor buffer.</p>						
41	11	Control	0-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.
			<p>This SRC value is valid only for STORE Commands, not SEQ STORE Commands. This SRC value is used for writing back modifications to Job Descriptors (including Trusted Descriptors). This overwrites the descriptor in memory, using the address from which the descriptor was fetched. Since no Pointer is used, this is a one-word command. Note that this will result in an error if the descriptor came in via QI or AI. If an In-line descriptor, a replacement job descriptor, or a Non-local JUMP was executed, an error will be generated for a STORE command with SRC=41h or 45h. Note that SGF should not be set to 1. Doing so will cause an error to be generated.</p>			

Table continues on the next page...

Table 7-27. STORE command SRC, OFFSET and LENGTH field values (continued)

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
			The values in the LENGTH and OFFSET field are specified in 4-byte words. offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the Job Descriptor (or Trusted Descriptor) (which will not be the start of the descriptor buffer if there is a Shared Descriptor). See Figure 7-3 .			
42	11	Control	0-64/ offset* words	DESC_BUF	DECO descriptor buffer	See notes below.
			<p>This SRC value is valid only for STORE commands, not SEQ STORE commands. This SRC value is used for writing back modifications to shared descriptors. This overwrites the shared descriptor in memory, using the address from which the shared descriptor was fetched. Note that a STORE with SRC=42h or 46h results in an error if there is no shared descriptor. Even if there is a shared descriptor in the original descriptor, an error is generated if there has been a non-local jump to another descriptor or an in-line descriptor is being executed, and that descriptor attempts a STORE with SRC=42h or 46h. Note that SGF should not be set to 1 for SRC values 41h, 42h, 45h or 46h. Doing so will cause an error to be generated.</p> <p>Since no pointer is used, this is a one-word command. The values in the LENGTH and OFFSET field are specified in 4-byte words.</p> <p>To correctly use sharing flows (wait or serial) in SEC, if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.</p> <p>offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the shared descriptor in both the descriptor buffer and in memory. See Figure 7-3.</p>			
45	11	Control	0-64/ offset* words	DESC_BUF	DECO Descriptor Buffer (Write Efficient)	This SRC value is used to write modifications to job descriptors back to memory using write-efficient bus transactions. See notes below.
			45h is the same as 41h, and 46h is the same as 42h, except that these SRC values cause write-efficient bus transactions (see AXI master (DMA) interface). Since these bus transactions may write more of the descriptor buffer back to memory than is specified by OFFSET and LENGTH, these SRC values should be used only if it is permitted to write back to memory all of the descriptor (See AXI master (DMA) interface). Note that SGF should not be set to 1. Doing so will cause an error to be generated.			

Table continues on the next page...

Table 7-27. STORE command SRC, OFFSET and LENGTH field values (continued)

SRC Value (hex)	Class (binary)	Control Data or Message Data	Legal values in LENGTH/OFFSET Fields	Tag	Source Internal Register	Comment
			If write-safe operations are not implemented or not enabled in this version of SEC, write-efficient operations are also not available, and SRC value 45h operates exactly as 41h, and SRC value 46h operates exactly as 42h.			
46	11	Control	1-64/ offset* words	DESC_BUF	DECO Descriptor Buffer (Write Efficient)	This SRC value is used to write modifications to shared descriptors back to memory like SRC=42h, but using write-efficient bus transactions. See notes above for SRC=45h.
All combinations of SRC and CLASS that do not appear in Table 7-27 are reserved.						

1. Because the math registers are in contiguous addresses, it is possible to store more than one math register simultaneously.
2. When this source is used, the data stored from the Math register will be treated as words.
3. When this source is used, the data stored from the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers.
4. When this source is used, the data stored from the Math register will be treated as bytes.

7.14 FIFO STORE command

NOTE

In the following discussion, FIFO STORE command refers to both the SEQ and non-SEQ forms of the command.

FIFO STORE commands are used to move data from the output data FIFO to external memory by means of the DMA. Because the only source is the output data FIFO, this command does not include a SRC field. The SEQ FIFO STORE command is identical to the FIFO STORE command except that no address is specified and the SGT bit is replaced by the VLF bit. See [SEQ vs non-SEQ commands](#).

Note that data output by means of the output data FIFO is considered message data.

The following types of data can be output from the output data FIFO.

- PKHA registers, other than the E-Memory.
- PKHA E Memory. This data is encrypted as a Black Key prior to being written to memory.
- Class 1 and Class 2 keys.
- RNG data, which can be left in the output data FIFO or stored away.

FIFO STORE command

- Regular data, which is pulled and written as it appears in the output data FIFO. Note that bit length data stores are not available.
- Data in the input sequence or in the input data FIFO

Note that even though this command is not a store check point, it does not start if a prior STORE or FIFO STORE of any type has yet to be scheduled. It is a done checkpoint if asked to encrypt a key because it has to wait until both class CHAs are done. The FIFO STORE command will block if the internal CCB DMA is not available when storing C1 or C2 keys.

The FIFO LOAD command supports bit length data, (see [Bit length data](#)), but the FIFO STORE command does not support bit lengths.

It is occasionally necessary to skip over portions of the output buffer (meaning to advance the output sequence pointer without actually writing data) before writing more output. For instance, in certain networking protocols, portions of the output stream may depend on out-of-order portions of the input stream. This processing can be done in two or more passes through the input and output sequences by:

1. Skipping portions of the input and output data in one pass
2. Restoring the sequences for the next pass by means of the RTO bit in the SEQ IN PTR command and the REW field in the SEQ OUT PTR command
3. Skipping over the portions of the output data that were written in the previous pass

To achieve skipping with SEQ FIFO STORE, use output data type 3Fh. Note that scatter tables may be read while skipping if the sequence was defined with the SGF bit set in the SEQ OUT PTR command. However, no data will be written while skipping.

Table 7-28. FIFO STORE command format

31–27	26–25	24	23	22	21–16
CTYPE = 01100 or 01101	AUX	SGF or VLF	CONT	EXT	OUTPUT DATA TYPE
15–0					
LENGTH					
<i>Additional words of FIFO STORE command:</i>					
Pointer (one or two words, see Address pointers)					
EXT LENGTH (Present if EXT = 1) (one word)					

Table 7-29. FIFO STORE command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=01100b : FIFO STORE command

Table continues on the next page...

Table 7-29. FIFO STORE command field descriptions (continued)

Field	Description
	If CTYPE=01101b : SEQ FIFO STORE command
26-25 AUX	Auxiliary control bits. Used only for certain output data type codes. Set AUX = 00 for all other output data type codes. See Table 7-30 .
24 SGF or VLF	Scatter/Gather table Flag (SGF) or Variable Length Flag (VLF). Meaning depends on CTYPE. If CTYPE = 01100 (FIFO STORE), this bit is the Scatter/Gather table Flag (SGF). If SGT=0 : The pointer points to actual data. If SGT=1 : The pointer points to a scatter/gather table. If CTYPE = 01101 (SEQ FIFO STORE), this bit is the Variable Length Flag (VLF). If VLF=0 : The number of bytes of data to be stored is specified in the LENGTH (if EXT=0) or EXT LENGTH (if EXT=1) field. If VLF=1 : The data length is variable. The number of bytes of data to be stored is specified in the VSOL register. The LENGTH field is ignored. NOTE: It is legal to set VLF=1 when storing a key NOTE: It is illegal to set VLF=1 when EXT=1.
23 CONT	Continue If CONT=0 : If the FIFO STORE pulls data from the output FIFO and finishes at an alignment other than at the end of a dword, the remainder of the last dword is popped and discarded. If the read from the output FIFO ends with the last byte of the dword, that dword is always popped. If CONT=1 : This is not the last FIFO STORE command for this data. The final dword that contributed data is not popped if the data did not end at an 8-byte boundary. This is used to prevent data loss when a store leaves off in the middle of a dword. NOTE: If this bit is set when there is no remaining data, subsequent operations may not work as expected.
22 EXT	Use Extended Length If EXT=0 : Output data length is solely determined by the 16-bit LENGTH field in the first word of the command. If EXT=1 : Output data length is determined by the 32-bit EXT LENGTH field. NOTE: It is illegal to set VLF=1 when EXT=1.
21-16 Output Data Type	This field identifies the type of data that the output data FIFO stores. See Table 7-30 for a list of the supported types.
15-0 LENGTH	The length of the data to be stored. If EXT=0 : The LENGTH field specifies the number of bytes to store. If EXT=1 : The EXT FIELD specifies the number of bytes to store. The LENGTH field is ignored.
<i>Additional words of the FIFO STORE command:</i>	
POINTER	Address pointer where to store the data in memory. NOTE: This field is not present for SEQ FIFO STORE commands, nor is it present for FIFO STORE commands if the data type is for RNG and the data is to be left in the output data FIFO.
EXT LENGTH	Extended length field. If EXT=0 : This field is not present. The LENGTH field specifies the number of bytes of data to be stored. If EXT=1 : The EXT LENGTH field specifies the number of bytes of data to be stored. The LENGTH field in the first word of the command is ignored.

Table 7-30 lists the various built-in FIFO STORE output data types.

Table 7-30. FIFO STORE output data type field

Bits 21-16 (hex)	Meaning	Comment
00	PKHA A0	NOTE: The appropriate size register is automatically written. A FIFO STORE from a PKHA register should never be attempted with size greater than the PKHA register size.
01	PKHA A1	
02	PKHA A2	
03	PKHA A3	
04	PKHA B0	
05	PKHA B1	
06	PKHA B2	
07	PKHA B3	
08	PKHA N	
0C	PKHA A	
0D	PKHA B	
12	PKHA E, AES-CCM encrypted using the job descriptor key encryption key	
13	PKHA E, AES-CCM encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors.
14	Key Register AES-CCM encrypted using the job descriptor key encryption key.	The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.
15	Key register, AES-CCM encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE. <ul style="list-style-type: none"> • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. AUX values 00 and 11 are illegal.
16	Class 2 Key Register MDHA Split Key, AES-CCM encrypted using the job descriptor key encryption key.	For performance and security, use of an MDHA split key is highly recommended. Details about split keys can be found in Using the MDHA Key Register with IPAD/OPAD "split keys" . The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see MDHA use of the Context Register). If the Class 2 Key register was loaded with a split key using a KEY command with PTS=1, or if a key was loaded into the C2 Key register and then the MDHA was run in INIT mode to create a split key, the C2 Key register will be stored in plaintext form.

Table continues on the next page...

Table 7-30. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
17	Class 2 Key Register MDHA Split Key, AES-CCM encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors. The comments for type 16h apply here as well.
22	PKHA E, AES-ECB encrypted using the job descriptor key encryption key	-
23	PKHA E, AES-ECB encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors.
24	Key Register, AES-ECB encrypted using the job descriptor key encryption key.	<p>The AUX field determines the source register for the FIFO STORE.</p> <ul style="list-style-type: none"> • AUX = 01 selects the Class 1 key register to be stored. • AUX = 10 selects the Class 2 key register to be stored. <p>AUX values 00 and 11 are illegal.</p>
25	Key Register, AES-ECB encrypted using the trusted descriptor key encryption key.	<p>Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE.</p> <ul style="list-style-type: none"> • AUX = 01 selects the Class 1 Key Register to be stored. • AUX = 10 selects the Class 2 Key Register to be stored. <p>AUX values 00 and 11 are illegal.</p>
26	Class 2 Key Register MDHA split key, AES-ECB encrypted using the job descriptor key encryption key.	For performance and security, use of an MDHA split key is highly recommended. Details on this split key can be found in Using the MDHA Key Register with IPAD/OPAD "split keys" . The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see MDHA use of the Context Register). If the Class 2 Key register was loaded with a split key using a KEY command with PTS=1, or if a key was loaded into the C2 Key register and then the MDHA was run in INIT mode to create a split key, the C2 Key register will be stored in plaintext form.
27	Class 2 Key Register MDHA split key, AES-ECB encrypted using the trusted descriptor key encryption key.	Available only to trusted descriptors. The comments for type 26h apply here as well.
30	Message Data	If a type 31h has been used, type 30h disables automatic checksum calculation. Any current checksum value will remain. A 0-length command may be used to effect this change.
31	Message Data	The first time this type is used, the running check sum is cleared. The use of this type enables the check sum hardware. A 0-length command may be used to effect this change.
34	Store the specified amount of data to be obtained from RNG to memory.	<p>NOTE: The Class 1 Data Size Register is automatically written and extended lengths are illegal.</p> <p>The different types of random data that can be generated are:</p>

Table continues on the next page...

Table 7-30. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
		<ul style="list-style-type: none"> • Random data with no restriction • Nonzero Random data • Odd Parity Random data. <p>The Mode Register controls the type of random data. Note that the RNG must be selected by writing the Mode register.</p>
35	Obtain the specified amount of data from RNG and leave it in the output data FIFO.	In addition to the comments for type 34h, there is no pointer and it is illegal to use type 35h with SEQ FIFO STORE.
3E	Meta Data	<p>For this output data type, CONT and EXT must both be 0. Either bit set to a 1 generates an error. This type can be used only with SEQ FIFO STORE; an error is generated if this output data type is used with FIFO STORE. Length can be specified in the command (VLF = 0) or in the Variable Sequence Out Length register (VLF = 1). If VLF = 1, the length must fit in the lower 16 bits of the VSIL register or an error is generated.</p> <p>The AUX bits control the behavior of the SEQ FIFO STORE command as follows:</p> <p>00 Use the DECO alignment block to move the specified number of bytes from the input FIFO to the output FIFO and store them to the output frame. This variant of the command is used when handling meta data that has already been read. An example of this would be for a shared descriptor where the RIF bit is set.</p> <p>01 The same as 00, except that the VSIL is decremented by the specified length. This form should be used when the RIF bit is set and the VSIL contains the input frame length of the packet. If the VSIL were not decremented in this case, the descriptor would have to subtract the meta data length from the VSIL register before running a protocol.</p> <p>10 Load the specified number of bytes from the input frame, as defined by a prior SEQ IN PTR command, to the input FIFO and decrement the Sequence In Length by this number of bytes. Move these bytes to the output FIFO by means of the DECO alignment block and store them to the output frame. This variant of the command is used when handling meta data that precedes the packet data.</p> <p>11 The same as 10, except that the Sequence In Length is not decremented. This form should be used when moving meta data that follows the packet data. Normally the length of trailing meta data has to be subtracted from the input frame length prior to running a protocol so that the protocol knows how long the packet is. When using the AUX = 11 variant the descriptor does not have to add the meta data length back to the Sequence In Length before executing the SEQ FIFO STORE Meta Data command.</p>
3F	Skip	Skip over the specified length in memory without using bus cycles. Permitted to be used only by SEQ FIFO STORE.

Table continues on the next page...

Table 7-30. FIFO STORE output data type field (continued)

Bits 21-16 (hex)	Meaning	Comment
NOTE: AUX must be set to 00 except when otherwise specified above. All combinations of output data type and AUX values not specified are reserved.		

7.15 MOVE, MOVEB, MOVEDW, and MOVE_LEN commands

NOTE

In this section "Move Command" is used to refer to the MOVE, MOVEB, MOVEDW, and MOVE_LEN forms of the command.

The MOVE command is used to copy data between two resources internal to a DECO/CCB. This allows data to be put in the proper registers without having to store data to external memory and then load it.

The OFFSET field is used to define an offset into either the source or destination, depending on the values in the SRC, DST, and AUX fields (see table [Table 7-34](#)). The MOVE command has a limited number of sources and destinations as indicated in the SRC and DST field descriptions below.

NOTE

MOVE cautions and restrictions:

- Keys can't be copied from a key register by means of a MOVE command if the corresponding key size register has been written.
- Observe the cautions noted in the "RJD" field of [SEQ IN PTR command](#) if using a MOVE command in a Replacement Job Descriptor.
- Moves may be checkpoints. For example, a move from the Class 2 Context Register to the Input Data FIFO for the Class 1 CHA is a Load Checkpoint and is a Done Checkpoint for the Class 2 CHA.

When moving data to or from the Descriptor Buffer or a MATH register, MOVEB treats data as 32-bit words in those cases that MOVE treats data as bytes, and MOVEB treats data as bytes when MOVE treats data as words. The MOVEDW command and the MOVE_LEN command for dwords always treat data as double words (i.e. 64 bits) and perform word swapping. The MOVEDW command and the MOVE_LEN command for dwords never do byte swapping.

NOTE: For MOVEDW or MOVE_LEN for dwords, with one exception, the offset must be a multiple of dwords (8 bytes). The one exception is when the offset is into the descriptor buffer, in which case the offset is allowed to be a multiple of words (4 bytes). If the source is the Output FIFO dword moves will always result in an error if the OFIFO offset is not zero.

In the MOVE, MOVEB, or MOVEDW command, the LENGTH field specifies the amount of data to be moved. The MOVE_LEN command is identical to the MOVE, MOVEB, and MOVEDW commands except that the length of the data being moved is specified in a MATH register, rather than specified as a constant in the LENGTH field. In the MOVE_LEN command, the MRSEL (Math Register Select) field, the TYPE field, and a reserved field replace the MOVE command's LENGTH field.

The AUX field is used to select among a number of different options, depending on the values in the SRC and DST fields (see the table [Table 7-34](#) below).

The MOVE command will block if the CCB DMA is busy. Other conditions where the MOVE command will block include:

- The SRC is context and the corresponding class CHA is not done or there is a data in flight to either context register.
- The SRC is the Output FIFO but a request for the external DMA to pull data from the output FIFO is pending.
- The DST is a context register and there is data in flight to either context register.
- The DST is the input data FIFO but there is data in flight to the input data FIFO.
- The DST is the input data FIFO for either of the C1 or C2 alignment blocks and an NFIFO entry is to be written and there is a context load pending.
- The DST is the C1 Key Register and a context load is pending and the write is to the extended key range.

NOTE

This device does prevent the byte and word swapping defaults from being overridden.

The Output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the Output FIFO so each can track separately allowing consumption of data at different rates. However, the only time these indices track separately is when the NFIFO is consuming data from the Output FIFO. Therefore, when using a move command to extract data from the Output FIFO, it is critical that the descriptor writer know whether the indices are tracking together and, if not, which index needs to be used to obtain the desired data. Note that this is an extremely unusual circumstance which most descriptor writers will seldom, if ever, encounter.

In prior versions of SEC, different entities handled the move from the Output FIFO depending on alignment. If the OFFSET specified was a multiple of words and the current OFIFO offset was 0, then the CCB DMA handled the move. Otherwise, the external DMA handled the move. The DMA used determined the index that was used to access the data in the Output FIFO. This led to significant confusion about which data was being read. In this version of SEC, the CCB DMA handles all moves from the Output FIFO, eliminating the confusion. However, it is possible for the descriptor to manipulate the index.

NOTE

It is possible to make the earlier behavior forward compatible by careful descriptor construction.

- To ensure that the move reads from the index where the external DMA left off, perform a LOAD IMM to the DECO Control Register to reset the CHA pointer in the Output FIFO. This has the effect of setting the shared index to the same value as the index used by the external DMA. While this will lose the current index of the CHA pointer, the move will get the expected data. (Remember that this is only necessary when the two indices are different. If the amount snooped and the amount read by the external DMA are the same, the indices will be the same.)
- To ensure that the move reads from the index where the NFIFO left off, ensure that the OFIFO offset is 0 and that the OFFSET in the move command is 0.

The OFIFO offset is used in two ways. First, it is used by DECO to tell the external DMA where in the 8-byte interface to the Output FIFO to start reading. Second, it is used by the CCB DMA to know where in the 8-byte interface to the Output FIFO to start reading. However, the two DMAs use different indices to access the Output FIFO so that the offset could be referencing different dwords. While those indices are usually synchronized, they can become different when the NFIFO has pulled data from the Output FIFO. It is therefore critical that the descriptor writer keep track of where each index is when moves from the Output FIFO follow snooping.

Table 7-31. MOVE, MOVEB, MOVEDW, and MOVE_LEN command format

	31–27	26–25	24	23–20	19–16
	CTYPE = 01111, 00110, 00111 or 01110	AUX	WC	SRC	DST
	15–8			7–0	
<i>Fields as they appear in the MOVE, MOVEB, or MOVEDW command:</i>	OFFSET			LENGTH	

Table continues on the next page...

Table 7-31. MOVE, MOVEB, MOVEDW, and MOVE_LEN command format (continued)

	15-8	7-6	5-3	2-0
<i>Fields as they appear in the MOVE_LEN command:</i>	OFFSET	TYPE	Reserved	MRSEL

Table 7-32. MOVE command field descriptions

Field	Description
31-27 CTYPE	Command Type 01111 - MOVE. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the LENGTH field. If byte swapping is enabled, MOVE swaps bytes within words in certain cases (see table Table 7-33). 00111 - MOVEB. When byte swapping is not enabled, the legal MOVEB moves are identical to the corresponding MOVE moves. However, when byte swapping is enabled, the MOVEB moves byte swap within words when the corresponding MOVE moves do not swap and vice versa (see table Table 7-33). 00110 - MOVEDW. Move Double Words. Performs an internal move between two internal DECO/CCB locations. If word swapping is not enabled, the legal MOVEDW moves are identical to the corresponding MOVE moves when byte swapping is disabled. If word swapping is enabled for the descriptor, the MOVEDW command swaps the order of the two words in a double word. No byte swapping is done. 01110 - MOVE_LEN. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the MATH register selected by the MRSEL field.
26-25 AUX	AUX bits are used for some SRC and DST combinations to specify additional options. See table Table 7-34 below.
24 WC	Wait for Completion 0 - Do not Wait for Completion 1 - Wait for Completion. Causes the MOVE command to stall until the move operation completes. This is necessary when the data to be moved must be in place before a subsequent command executes. While it is sometimes possible to know a priori that the MOVE command will complete prior to reaching the subsequent command in question, such completion can not always be guaranteed.
23-20 SRC	Source. This specifies the internal source of data that will be moved. See table Table 7-36 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 7-34 and Table 7-35 indicate which source and destination combinations are permitted.
19-16 DST	Destination. This specifies the internal destination of the data that will be moved. See table Table 7-37 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 7-34 and Table 7-35 indicate which source and destination combinations are permitted.
15-8 OFFSET	Offset. (in bytes) The interpretation of the OFFSET field depends on the source and destination, as shown in table Table 7-35 . The OFFSET is limited to 128 bytes except when the Descriptor Buffer is the source or destination, in which case the OFFSET may be as large as 255 bytes. For MOVEDW and MOVE_LEN for dwords, the OFFSET must be a multiple of 8 bytes unless the OFFSET is into the Descriptor Buffer, in which case the OFFSET must be a multiple of 4 bytes.
7-0 LENGTH	Length for internal move. (in bytes, 128 max) This field appears only in the MOVE, MOVEB, or MOVEDW forms of the command. In the MOVE_LEN form of the command this field is replaced by reserved bits and the MRSEL field and TYPE field, as shown below.
<i>Note that in the MOVE_LEN form of the command the LENGTH field is replaced by the following three fields:</i>	

Table continues on the next page...

Table 7-32. MOVE command field descriptions (continued)

Field	Description
7-6 TYPE	Type of the data items that are to be moved. 00 - Data is treated the same as in the MOVE command 01 - Data is treated as dwords the same way as in the MOVEDW command 10 - Data is treated as bytes the same way as in the MOVEB command 11 - Reserved and reported as an error
5-3	These bits are reserved in the MOVE_LEN form of the command. These bits, the TYPE field and the MRSEL field below replace the LENGTH field that appears in the MOVE form of the command.
2-0 MRSEL	MATH Register Select This field is used only in the MOVE_LEN form of the command. The MRSEL field, TYPE field, and the reserved bits above replace the LENGTH field that appears in the MOVE, MOVEB, and MOVEDW forms of the command. The length (in bytes) of the data to be moved is specified in the MATH Register selected by the MRSEL field. If the move is from the input FIFO or any of the alignment blocks to the Output FIFO, bits 15:0 of the MATH Register are used for the length; otherwise, only bits 7:0 are used. Other bits are simply ignored. 000 - Math Register 0 001 - Math Register 1 010 - Math Register 2 011 - Math Register 3 100 - Math Register 4 101 - Math Register 5 110 - Math Register 6 111 - Math Register 7

Table 7-33. Byte swapping in move commands

When byte swapping is enabled, this table indicates when bytes within a word are swapped. Legend:
M refers to the MOVE command and the MOVE_LEN command when TYPE=00
B refers to the MOVEB command and the MOVE_LEN command when TYPE=10
Swap: indicates the move will swap bytes within words
Not: indicates the move will not swap bytes within words
Err: indicates the move command will generate an error

DST → SRC ↓	0h: C1 Context	2h: Output Output	3h: Descript or Buffer	4h: Math 0 5h: Math 1 6h: Math 2 7h: Math 3	8h: Class 1 Input Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: PKHA A RAM (always flushed)	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
0h: C1 Context	M:Not		M:Swap	M:Not	M:Not					
1h: C2 Context	B:Err		B:Not	B:Swap	B:Err					
2h: Output FIFO	M:Not	M:Err								

Table continues on the next page...

Table 7-33. Byte swapping in move commands (continued)

	B:Err	B:Err						
3h: Descr Buffer	M:Swap B:Not		M:Err B:Err		M:Swap B:Not		M:Not B:Swap	M:Swap B:Not
4h: Math Reg 0 5h: Math Reg 1 6h: Math Reg 2 7h: Math Reg 3	M:Not B:Swap							
8h: DECO Alignment Block (flushed)	M:Not B:Err	M:Swap B:Not	M:Not B:Swap	M:Err B:Err		M:Not B:Err	M:Err B:Err	
9h: Class 1 or Class 2 Alignment Block								
Ah: DECO, Class 1 or Class 2 Alignment Block								
Dh: Class 1 Key Eh: Class 2 Key					M:Not B:Err			

Table 7-34. Usage of the AUX field in move commands

DST → SRC ↓	0h: C1 Context 1h: C2 Context	2h: Output Data FIFO	3h: Descriptor Buffer	4h: Math 0 5h: Math 1 6h: Math 2 7h: Math 3	8h: Class 1 Input Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: PKHA A RAM (automatically flushed)	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
0h: C1 Context 1h: C2 Context			AUX selects offset into Context Register 00: 0 bytes 01: 16 bytes 10: 32 bytes 11: 48 bytes	AUX selects offset into Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes	AUX _{MS} : Flush AUX _{LS} : Last	AUX _{LS} : Last			AUX _{LS} =0 : OFFSET field into Context Reg AUX _{LS} =1 : OFFSET into Key Reg	
2h: Output FIFO		move not allowed								
3h: Descr Buffer	AUX selects offset into Context Register 00: 0 bytes		move not allowed	AUX selects offset into Math Register 00: 0 bytes						

Table continues on the next page...

Table 7-34. Usage of the AUX field in move commands (continued)

	01: 16 bytes 10: 32 bytes 11: 48 bytes			01: 4 bytes 10: 6 bytes 11: 7 bytes						
4h: Math Reg 0 5h: Math Reg 1 6h: Math Reg 2 7h: Math Reg 3	AUX selects offset into Math Register 00: 0 bytes> 01: 4 bytes 10: 6 bytes 11: 7 bytes		AUX selects offset into Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes	AUX selects offset into the source Math Register 00: 0 bytes 01: 4 bytes 10: 6 bytes 11: 7 bytes						
8h: DECO Alignment Block (automatically flushed)					move not allowed					move not allowed
9h: Class 1 or Class 2 Alignment Block	AUX _{MS} : Flush AUX _{LS} =0 : Source is Class 2 Alignment Block AUX _{LS} =1 : Source is Class 1 Alignment Block								AUX _{MS} : Flush AUX _{LS} =0 : Source is C2 Align Block AUX _{LS} =1 : Source is C1 Align Block	
Ah: DECO, Class 1 or Class 2 Alignment Block	AUX field selects Alignment Block 00: DECO Alignment Block 01: Class 1 Alignment Block 10: Class 2 Alignment Block 11: error								AUX field selects Alignment Block 00: DECO Alignment Block 01: C1 AB 10: C2 AB 11: error	
Dh: Class 1 Key Eh: Class 2 Key	Determines SRC/DST offset				AUX _{MS} : Flush AUX _{LS} : Last	AUX _{LS} : Last			Determines SRC/DST offset	

Table 7-35. Usage of the OFFSET field in move commands

DST → SRC ↓	0h: C1 Context 1h: C2 Context	2h: Output Data FIFO	3h: Descriptor Buffer	4h: Math 0 5h: Math 1 6h: Math 2 7h: Math 3	8h: Class 1 Input Data FIFO	9h: Class 2 Input Data FIFO	Ah: Input Data FIFO (no NFIFO entries)	Ch: PKHA A RAM (always flushed)	Dh: C1 Key Eh: C2 Key	Fh: Aux Data FIFO
0h: C1 Context 1h: C2 Context	OFFSET field is used for SRC	OFFSET field is used for SRC	OFFSET field is used for Descriptor Buffer (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC				AUX _{LS} =0: OFFSET field is used for Context Reg AUX _{LS} =1: OFFSET field is used for Key Reg	OFFSET field is used for SRC	
2h: Output FIFO	OFFSET field is used for DST	move not allowed	OFFSET field is used for DST	Error generated if OFFSET≠0			OFFSET field is used for DST	Error generated if OFFSET≠0		
3h: Descr Buffer	OFFSET field is used for SRC (offset into Context Reg is determined by AUX field)	OFFSET field is used for SRC	move not allowed	OFFSET field is used for SRC				OFFSET field is used for SRC	OFFSET field is used for SRC	
4h: Math Reg 0 5h: Math Reg 1 6h: Math Reg 2 7h: Math Reg 3	OFFSET field is used for DST	OFFSET field is used for SRC	OFFSET field is used for DST	OFFSET field is used for SRC				OFFSET field is used for DST	OFFSET field is used for SRC	
8h: DECO Align Block (flushed) 9h: Class 1 or Class 2 Alignment Block Ah: DECO, Class 1 or Class 2 Alignment Block	OFFSET field is used for DST	OFFSET is ignored in MOVE_LEN In MOVE, OFFSET field is prepended to the LENGTH field to form a 16-bit length	OFFSET field is used for DST	move not allowed				OFFSET field is used for DST	move not allowed	

Table continues on the next page...

Table 7-35. Usage of the OFFSET field in move commands (continued)

Dh: Class 1 Key	AUX _{LS} =0 : SRC	SRC	DST	SRC	SRC	AUX _{LS} =0 : SRC;	SRC
Eh: Class 2 Key	SRC; else DST					else DST	

Table 7-36. Move sources

Value	Move Source	Notes
0h	Class 1 Context Reg	—
1h	Class 2 Context Reg	—
2h	Output Data FIFO	—
3h	Descriptor Buffer	—
4h	Math Register 0	A MOVE command that reads past the end of MATH Register 3 will continue reading into Math Registers 4-7 NOTE: Math Registers 4-7 are not available as sources for the MOVE commands.
5h	Math Register 1	
6h	Math Register 2	
7h	Math Register 3	
8h	DECO Alignment Block (always Flushed)	Input to the DECO Alignment Block is specified by an NFIFO entry which is automatically generated if Automatic NFIFO entries are enabled.
9h	Class 1 or Class 2 Alignment Block	The choice between the Class 1 and Class 2 Alignment Blocks is determined by the least-significant bit of the AUX field: <ul style="list-style-type: none"> • A_{LS} = 0 selects C2 Alignment Block • A_{LS} = 1 selects C1 Alignment Block Input to the Class 1 or Class 2 Alignment Block is specified by an NFIFO entry which is automatically generated if Automatic NFIFO entries are enabled. The most-significant bit of the AUX field must be set (causing a FLUSH) only if the destination is the Output Data FIFO.
Ah	DECO, Class 1 or Class 2 Alignment Block, as specified via the AUX field.	no NFIFO entry generated; AUX = 00b: use DECO Alignment Block AUX = 01b: use Class 1 Alignment Block AUX = 10b: use Class 2 Alignment Block AUX = 11b: error
Dh	Class 1 Key	Error if C1 Key Size has been written either directly or by the KEY command and not cleared.
Eh	Class 2 Key	Error if C2 Key Size has been written either directly or by the KEY command and not cleared.
All other values are reserved		

Table 7-37. Move destinations

Value	Move Destination	Notes
0h	Class 1 Context	—
1h	Class 2 Context	—
2h	Output Data FIFO	

Table continues on the next page...

Table 7-37. Move destinations (continued)

Value	Move Destination	Notes
3h	Descriptor Buffer	—
4h	Math Register 0	—
5h	Math Register 1	—
6h	Math Register 2	—
7h	Math Register 3	—
8h	Input Data FIFO (C1)	If Automatic NFIFO entries are enabled, the entries are generated for a Class 1 CHA.
9h	Input Data FIFO (C2)	If Automatic NFIFO entries are enabled, the entries are generated for a Class 2 CHA.
Ah	Input Data FIFO	No NFIFO entry generated
Ch	PKHA A	If Automatic NFIFO entries are enabled, the entries are generated for the PKHA A RAM and the Flush bit is automatically set.
Dh	Class 1 Key	—
Eh	Class 2 Key	
Fh	Auxiliary Data FIFO	Data can be moved to the Auxiliary Data FIFO so that it can later be used as an input to one or more of the Alignment Blocks. (An NFIFO entry with AST = 1 and STYPE = 00 should be created before the MOVE, else DECO may hang.) Note that a LOAD IMM to destination 78 can also be used to supply data to the Auxiliary Data FIFO. If multiple MOVEs and/or MOVEs and LOADs are used to provide data to the Auxiliary Data FIFO, the MOVE commands may need the WC bit set to ensure that the data is not overwritten.
All other values are reserved		

7.16 ALGORITHM OPERATION command

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that SEC performs. Setting OPTYPE = 010 or 100 specifies an ALGORITHM OPERATION. Setting OPTYPE = 000, 110, or 111 specifies a PROTOCOL OPERATION (see [PROTOCOL OPERATION commands](#)). Setting OPTYPE = 001 specifies a PKHA OPERATION (see [PKHA OPERATION command](#)). The operation can range from performing a simple operation using a single CHA to performing a complex operation involving multiple CHAs and multiple steps. More than one OPERATION command can be used in a descriptor, allowing Class 1 and Class 2 operations to be specified separately. For the ALGORITHM OPERATION command, the fields of the command are as shown in the following table. Note that bits 23-0 of the ALGORITHM OPERATION command are automatically written to the appropriate CHA's mode register.

Table 7-38. ALGORITHM OPERATION command format

	31-27	26-24	23-16
	CTYPE = 10000b	OPTYPE = 010b or 100b	ALG

Table continues on the next page...

Table 7-38. ALGORITHM OPERATION command format (continued)

	15-14	13	12-4						3-2	1	0	
<i>format as it appears for CHAs other than RNG:</i>	Reserved	C2K	AAI						AS	ICV	ENC	
	15-13		12	11	10	9	8	7-6	5-4	3-2	1	0
<i>format as it appears for RNG:</i>	Reserved		SK	AI	PS	OBP	NZB	Reserved	SH	AS	PR	TST

Table 7-39. ALGORITHM OPERATION command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in the field descriptions below. If OPTYPE = 010b : Class 1 algorithm operation If OPTYPE = 100b : Class 2 algorithm operation If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command. If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION command.
23-16 ALG	Algorithm This field specifies the algorithm that is to be used for the operations. <ul style="list-style-type: none"> If OPTYPE = 010b (Class 1 algorithm) <ul style="list-style-type: none"> If ALG=10h : AES If ALG=20h : DES If ALG=21h : 3DES If ALG=50h : RNG If ALG=60h : SNOW 3G f8 If ALG=70h : Kasumi f8 or f9 If ALG=B0h : ZUC encryption All other values are reserved. If OPTYPE = 100b (Class 2 algorithm) <ul style="list-style-type: none"> If ALG=10h : AES (only valid for CMAC and XCBC modes) If ALG=40h : MD5 If ALG=41h : SHA-1 If ALG=42h : SHA-224 If ALG=43h : SHA-256 If ALG=44h : SHA-384 If ALG=45h : SHA-512 If ALG=46h : SHA-512/224 If ALG=47h : SHA-512/256 If ALG=90h : CRC If ALG=A0h : SNOW 3G f9 If ALG=C0h : ZUC authentication All other values are reserved.

Table continues on the next page...

Table 7-39. ALGORITHM OPERATION command field descriptions (continued)

Field	Description
15-14	Reserved
13 C2K	Class 2 Key This bit is ignored for all algorithms other than AES. 0: AES uses the Class 1 key for CCM and GCM modes. 1: AES uses the Class 2 key for CCM and GCM modes. Setting this bit = 1 results in a mode error for other AES modes.
12-4 AAI	Additional Algorithm Information This field contains additional mode information that is associated with the algorithm that is being executed. See the tables below for details specific to individual algorithms. See also the section describing the appropriate CHA. Note that some algorithms do not require additional algorithm information and in those cases this field should be all 0s. For RNG OPERATION commands the AAI field is interpreted as shown in the shaded SK, AI, PS, OBP, NZ and SH fields below.
3-2 AS	Algorithm State This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see the shaded AS field below. 00 Update 01 Initialize 10 Finalize 11 Initialize/finalize
1 ICV	ICV Checking For the definition of this bit in RNG commands, see the shaded PR field below. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit is ignored by algorithms that do not support ICV checking.
0 ENC	Encrypt/Decrypt For the definition of this bit in RNG commands, see the shaded TST field below. This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes. However, for performance counting to be done correctly, this bit must be set appropriately even if the CHA or Algorithm does not use it to select cryptographic modes. 0 Decrypt 1 Encrypt
<i>The rows below describe how bits 12-0 are interpreted for RNG commands.</i>	
12 SK (RNG only)	Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS≠00.
11 AI (RNG only)	Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the 256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Uninstantiated).
10 PS	Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional "entropy" input during instantiation. Note that the

Table continues on the next page...

Table 7-39. ALGORITHM OPERATION command field descriptions (continued)

Field	Description
(RNG only)	personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01.
9 OBP (RNG only)	Odd Byte Parity. For RNG OPERATION commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00.
8 NZB (RNG only)	NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00.
7-6 (RNG only)	Reserved. For RNG commands these bits of the AAI field are reserved.
5-4 SH (RNG only)	State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented. 00 State Handle 0 01 State Handle 1 10 Reserved 11 Reserved
3-2 AS (RNG only)	Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown in Table 7-47 .
1 PR (RNG only)	Prediction Resistance. For RNG OPERATION commands this bit is interpreted as shown in Table 7-48 .
0 TST (RNG only)	Test Mode Request. For RNG OPERATION commands this bit is interpreted as shown in Table 7-49 .

Table 7-40. AAI Interpretation for AES modes

AAI Interpretation for AES Modes (See AES accelerator (AESA) functionality)			
Code	Interpretation	Code	Interpretation
00h	CTR (mod 2 ¹²⁸)	80h	CCM (mod 2 ¹²⁸) ¹
10h	CBC	90h	GCM (mod 2 ³²)
20h	ECB	A0h	CBC_XCBC_MAC

Table continues on the next page...

Table 7-40. AAI Interpretation for AES modes (continued)

AAI Interpretation for AES Modes (See AES accelerator (AESA) functionality)				
30h	CFB		B0h	CTR_XCBC_MAC
40h	OFB		C0h	CBC_CMAC
50h	XTS		D0h	CTR_CMAC_LTE
60h	CMAC		E0h	CTR_CMAC
70h	XCBC-MAC			
The codes listed above are mutually exclusive, which means that they cannot be ORed with each other.				
<p>Note that for AES the MSB of AAI is the DK (Decrypt Key) bit. Setting the DK bit (that is, ORing 100h with any AES code above) causes the Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in AES accelerator (AESA) functionality. Note that AES normally acts as a Class 1 CHA, but for CMAC or XCBC-MAC it can also be used as a Class 2 CHA. CBC_XCBC_MAC, CTR_XCBC_MAC, CBC_CMAC, CTR_CMAC_LTE, and CTR_CMAC are modes that use both the Class 1 and Class 2 keys; the Class 1 Key is used for the Confidentiality algorithm, and the Class 2 key is used for the Integrity algorithm. These modes were built to support IPsec and LTE operations but are available for general use. When a Class 2 OPERATION command specifies AES with CMAC or XCBC-MAC, it may be accompanied by a Class 1 OPERATION command specifying AES, if (and only if) the Class 1 OPERATION command specifies a Confidentiality-only mode. Specifying a Class 2 AES OPERATION command in concert with a Class 1 AES Operation command specifying either CCM or GCM is not permitted and will result in an error. Combo modes CBC_XCBC_MAC, CTR_XCBC_MAC, CBC_CMAC, CTR_CMAC_LTE, and CTR_CMAC were specified for the built-in IPsec and LTE protocol but are available for general use. With the extension to AES permitting simultaneous Class 1 Confidentiality-only and Class 2 Integrity OPERATION, these Combo modes are no longer recommended and may be deprecated in the future.</p>				

- Note, if C2K= 0 CCM and GCM use the key in the Class 1 Key Register. If C2K = 1 CCM and GCM use the key in the Class 2 Key Register.

Table 7-41. AAI Interpretation for DES modes

AAI Interpretation for DES modes (See Data encryption standard accelerator (DES) functionality)				
Code	Interpretation		Code	Interpretation
10h	CBC		30h	CFB
20h	ECB		40h	OFB
The codes listed above are mutually exclusive, which means that they cannot be ORed with each other.				
80h ORed with any DES code above: Check odd parity				

Table 7-42. AAI Interpretation for MD5 and SHA modes

AAI Interpretation for MD5, SHA-1, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 (See Message digest hardware accelerator (MDHA) functionality)				
Code	Interpretation		Code	Interpretation
00h	Hash without key		02h	SMAC
01h	HMAC		04h	HMAC with precomputed IPAD and OPAD

Table 7-43. AAI Interpretation for CRC modes

AAI Interpretation for CRC modes (See Cyclic-redundancy check accelerator (CRCA) functionality)				
Code	Interpretation		Code	Interpretation
01h	IEEE 802		10h	DIS
02h	IETF 3385		20h	DOS
04h	CUST_POLY		40h	DOC
			80h	IVZ (initial value zero)

CRC codes in the right column can be ORed with CRC codes in the left column to specify DIS, DOS, DOC and IVZ
(See [CRCA use of the Mode Register](#))

Table 7-44. AAI Interpretation for Kasumi modes

AAI Interpretation for Kasumi 3G modes (See Kasumi f8 and f9 hardware accelerator(KFHA) functionality)				
Code	Interpretation		Code	Interpretation
C0h	Kasumi 3G f8 (encryption/decryption)		10h	GSM
C8h	Kasumi 3G f9 (authentication)		20h	EDGE

Table 7-45. AAI Interpretation for SNOW 3G modes

AAI Interpretation for SNOW 3G modes (See SNOW 3G f8 accelerator functionality and SNOW 3G f9 accelerator functionality)	
Code	Interpretation
C0h	SNOW 3G f8 (encryption/decryption)
C8h	SNOW 3G f9 (authentication)

Table 7-46. AAI Interpretation for ZUC modes

AAI Interpretation for ZUC (See ZUC encryption accelerator (ZUCE) functionality)	
Code	Interpretation
C0h	ZUCE (encryption/decryption)
C8h	ZUCA (authentication)

Table 7-47. AS RNG OPERATION command settings

AS Value	State Handle is already instantiated	State Handle is NOT already instantiated
00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error
01 Instantiate	Error	Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.
10 Reseed	Reseed the state handle.	Error
11 Uninstantiate	Uninstantiate the state handle.	Error

Table 7-48. PR RNG Operation commands setting

AS Value	PR = 0	PR = 1
00 Generate	Do NOT reseed prior to generating new random data. PR bit must be zero.	If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.
01 Instantiate	Instantiate the state handle to NOT support prediction resistance	Instantiate the state handle to support prediction resistance
10 Reseed	Reseed the state handle. PR bit must be zero.	Reseed the state handle. PR bit is ignored.
11 Uninstantiate	Uninstantiate the state handle. PR bit must be zero.	Uninstantiate the state handle. PR bit is ignored.

Table 7-49. TST RNG OPERATION command settings

AS Value	TST = 0	TST = 1
00 Generate	<ul style="list-style-type: none"> If the selected state handle is in nondeterministic mode, generate new random data. If the selected state handle is in deterministic mode, generate a Test error.¹ 	<ul style="list-style-type: none"> If the selected state handle is in deterministic mode, generate new random data. If the selected state handle is in nondeterministic mode, generate a Test error.
01 Instantiate	Instantiate the state handle in normal (nondeterministic) mode.	Instantiate the state handle in test (deterministic) mode.
10 Reseed	<ul style="list-style-type: none"> If the selected state handle is in nondeterministic mode, reseed the state handle. If the selected state handle is in deterministic mode, generate a Test error.² 	<ul style="list-style-type: none"> If the selected state handle is in deterministic mode, reseed the state handle. If the selected state handle is in nondeterministic mode, generate a Test error.
11 Uninstantiate	<ul style="list-style-type: none"> If the selected state handle is in nondeterministic mode, uninstantiate the state handle. If the selected state handle is in deterministic mode, generate a Test error.³ 	<ul style="list-style-type: none"> If the selected state handle is in deterministic mode, uninstantiate the state handle. If the selected state handle is in nondeterministic mode, generate a Test error.

1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a Generate operation requests nondeterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.
2. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test reseed operation is requested State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.
3. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test uninstantiate operation is requested for State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

7.17 PROTOCOL OPERATION commands

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that SEC performs. The OPERATION command's Protocol OpType takes advantage of well-known processing steps for standardized security protocols, so that the user can invoke an existing hard-coded command sequence rather than having to use SEQ commands to create a complex descriptor.

If the OPTYPE specifies a protocol operation (000, 110, 111), the OPERATION command fields are as shown in [Table 7-51](#). If OPTYPE specifies an algorithm operation (OPTYPE = 010: Class 1, OPTYPE = 100: Class 2), see [ALGORITHM OPERATION command](#). If OPTYPE specifies a PKHA operation (OPTYPE = 001), see [PKHA OPERATION command](#).

Table 7-50. PROTOCOL OPERATION command format

31-27	26-24	23-16
CTYPE = 10000	OPTYPE = 000, 110, or 111	PROTID
15-0		
PROTINFO		

Protocols are used to execute complex built-in functions. Before beginning a protocol operation, DECO waits for any pending (SEQ) FIFO STORE operations to complete. When starting the protocol operation, DECO resets the output data FIFO; any data remaining in the output data FIFO from previous operations is lost. It is the responsibility of the programmer to ensure that once the protocol starts, no data is pushed into the output data FIFO as a result of commands executed prior to the protocol operation. It is the responsibility of the programmer to ensure that once the protocol starts, no data is in, or will be pushed into, the input data FIFO or information FIFO as a result of commands executed prior to the protocol operation.

The protocol ID codes and information on PROTINFO encoding are shown in [Table 7-52](#),

Table 7-51. PROTOCOL OPERATION command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 000b, 110b or 111b : PROTOCOL OPERATION; The OPTYPE field indicates the "directionality" of the protocol as shown below. The PROTID field is interpreted as shown in the following PROTID field description table.

Table continues on the next page...

Table 7-51. PROTOCOL OPERATION command field descriptions (continued)

Field	Description
	<p>If OPTYPE=000b : Unidirectional protocol</p> <p>If OPTYPE=110b : Decapsulation protocol</p> <p>If OPTYPE=111b : Encapsulation protocol</p> <p>If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command.</p> <p>If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command.</p> <p>All others: reserved</p>
23-16 PROTID	<p>Protocol Identifier</p> <p>This field represents the specific protocol that this descriptor is implementing. See Table 7-52 for the complete description.</p>
PROTINFO 15-0	<p>This value is protocol-dependent.</p>

Table 7-52. PROTID and PROTINFO field description

PROTID (hex)	Description	PROTINFO Information
01	<p>OPTYPE 000: IKE PRF</p> <p>OPTYPE 11x: IPsec ESP Transport (and legacy ESP Tunnel)</p> <p>else Reserved</p>	<p>For IKE PRF and PRF+, the PROTINFO field is defined in Table 7-55. For further information concerning IKE PRF, refer to IKEv2 PRF overview.</p> <p>For IPsec ESP Transport (and legacy Tunnel) and IPsec ESP Tunnel, the PROTINFO field is defined in Table 7-53. For further information concerning IPsec ESP Transport (and legacy Tunnel), IPsec ESP Tunnel, refer to IPsec ESP encapsulation and decapsulation overview.</p>
02	<p>OPTYPE 000: IKE PRF+</p> <p>OPTYPE 11x: SRTP</p> <p>else Reserved</p>	<p>For IKE PRF and PRF+, the PROTINFO field is defined in Table 7-55. For further information concerning IKE PRF+, refer to Implementation Details for IKE PRF+ function</p> <p>For SRTP, the PROTINFO field is defined in Table 7-53. For further information concerning SRTP, refer to SRTP packet encapsulation and decapsulation.</p>
03	<p>OPTYPE 11x: MACsec (802.1AE)</p> <p>else Reserved</p>	<p>For MACsec, use PROTINFO = 0000_0000_0000_0001 to specify AES_GCM_16. Use PROTINFO = 0000_0000_0000_0002 to specify AES_GCM_32. Use PROTINFO = 0000_0000_0000_0003 to specify AES_GCM_16 with extended PN. Use PROTINFO = 0000_0000_0000_0004 to specify AES_GCM_32 with extended PN. For further information concerning MACsec, refer to IEEE 802.1AE MACsec encapsulation and decapsulation overview.</p>
04	<p>OPTYPE 11x: IEEE 802.11-2012 WPA2 MPDU for WiFi</p> <p>else Reserved</p>	<p>For WPA2, use PROTINFO = 1010_1100_0000_0100 to specify AES_CCM_8 (CCMP).</p> <p>For further information concerning WPA2, refer to IEEE 802.11-2012 WPA2 MPDU encapsulation and decapsulation.</p>

Table continues on the next page...

Table 7-52. PROTID and PROTIINFO field description (continued)

PROTID (hex)	Description	PROTIINFO Information
05	OPTYPE 11x: WiMAX (802.16) else Reserved	For WiMAX OFDM, use PROTIINFO = 0000_0010_0000_0001 to specify AES_CCM_16 with CRC for OFDM. Use PROTIINFO = 0000_0000_0000_0001 to specify CRC-only for OFDM. For further information concerning WiMAX refer to IEEE 802.16 WiMAX encapsulation and decapsulation overview . For WiMAX OFDMa, use PROTIINFO = 0000_0010_0011_0001 to specify AES_CCM_16 with CRC for OFDMa. Use PROTIINFO = 0000_0000_0011_0001 to specify CRC-only for OFDMa. For further information concerning WiMAX refer to IEEE 802.16 WiMAX encapsulation and decapsulation overview .
08	OPTYPE 000: SSL 3.0 PRF NOTE: Descriptors that include a TLS PRF command are limited to 50 words in length. OPTYPE 110: SSL3.0 Decapsulation OPTYPE 111: SSL 3.0 Encapsulation else Reserved	For the SSL/TLS/DTLS protocol family, the PROTIINFO field is defined in Table 7-54 . For information on SSL 3.0 PRF, refer to Process for SSL 3.0 PRF . For further information concerning SSL/3.0T record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview .
09	OPTYPE 000: TLS 1.0 PRF NOTE: Descriptors that include a TLS PRF command are limited to 50 words in length. OPTYPE 110: TLS 1.0 Decapsulation OPTYPE 111: TLS 1.0 Encapsulation else Reserved	For the SSL/TLS/DTLS protocol family, the PROTIINFO field is defined in Table 7-54 . For information on TLS 1.0 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF . For further information concerning TLS 1.0 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview .
0A	OPTYPE 000: TLS 1.1 PRF NOTE: Descriptors that include a TLS PRF command are limited to 50 words in length. OPTYPE 110: TLS 1.1 Decapsulation OPTYPE 111: TLS 1.1 Encapsulation else Reserved	For the SSL/TLS/DTLS protocol family, the PROTIINFO field is defined in Table 7-54 . For information on TLS 1.1 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF . For further information concerning TLS 1.1 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview .
0B	OPTYPE 000: TLS 1.2 PRF using HMAC-SHA-256 except when another HMAC is expressly stated as shown in Table 7-54 NOTE: Descriptors that include a TLS PRF command are limited to 50 words in length. OPTYPE 110: TLS 1.2 Decapsulation OPTYPE 111: TLS 1.2 Encapsulation else Reserved	For the SSL/TLS/DTLS protocol family, the PROTIINFO field is defined in Table 7-54 . For information on TLS 1.2 PRF, refer to Process for TLS 1.2 PRF . For further information concerning TLS 1.2 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview .

Table continues on the next page...

Table 7-52. PROTID and PROTIINFO field description (continued)

PROTID (hex)	Description	PROTIINFO Information
0C	<p>OPTYPE 000: DTLS 1.0 PRF</p> <p>NOTE: Descriptors that include a TLS PRF command are limited to 50 words in length.</p> <p>OPTYPE 110: DTLS 1.0 Decapsulation</p> <p>OPTYPE 111: DTLS 1.0 Encapsulation</p> <p>else Reserved</p>	<p>For the SSL/TLS/DTLS protocol family, the PROTIINFO field is defined in Table 7-54. For information on DTLS 1.0 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF. For further information concerning DTLS 1.0 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview.</p>
0D	<p>For OPTYPE 110 or 111: Blob</p>	<p>For blobs encapsulation or decapsulation, the PROTIINFO field is defined in Table 7-58 and Table 7-59. For further information concerning blobs, see Blobs.</p>
0F	<p>For OPTYPE 110: Anti-Replay</p>	<p>Stand-alone Anti-Replay checking always uses a PROTIINFO code of 0000h, and is described in Anti-Replay built-in checking.</p>
11	<p>OPTYPE 11x: IPsec ESP Tunnel</p> <p>else Reserved</p>	<p>For IPsec ESP Tunnel, the PROTIINFO field is defined in Table 7-53. For further information concerning these protocols, refer to IPsec ESP encapsulation and decapsulation overview.</p>
12	<p>OPTYPE 000: (EC)DSA Verify with Private Key</p> <p>else Reserved</p>	<p>For (EC)DSA Verify using Private Key, the PROTIINFO field is defined in Table 7-61. For further information, see Verifying DSA and ECDSA signatures.</p>
14	<p>OPTYPE 000: DH, DSA, and ECC Key Pair Generation</p> <p>OPTYPE 110: MPPubK generation</p> <p>OPTYPE 111: MPPrivK generation</p> <p>else Reserved</p>	<p>For Key Pair Generation, MPPubK and MPPrivK, the PROTIINFO field is defined in Table 7-61. For further information, see Discrete-log key-pair generation</p>
15	<p>OPTYPE 000: (EC)DSA_Sign</p> <p>OPTYPE 110: MPSign</p> <p>else Reserved</p>	<p>For (EC)DSA Sign, and MPSign, the PROTIINFO field is defined in Table 7-61. For further information, see Generating DSA and ECDSA signatures.</p>
16	<p>OPTYPE 000: (EC)DSA_Verify</p> <p>else Reserved</p>	<p>For (EC)DSA Verify, the PROTIINFO field is defined in Table 7-61. For further information, see Verifying DSA and ECDSA signatures.</p>
17	<p>OPTYPE 000: (EC)Diffie-Hellman</p> <p>else Reserved</p>	<p>For (EC)Diffie-Hellman, the PROTIINFO field is defined in Table 7-61. For further information, see Using the Diffie_Hellman function.</p>
18	<p>OPTYPE 000: RSA_Encrypt</p> <p>else Reserved</p>	<p>For RSA_Encrypt, the PROTIINFO field is defined in Table 7-63. For further information concerning RSA Encrypt see Implementation of the RSA encrypt operation.</p>
19	<p>OPTYPE 000: RSA_Decrypt</p> <p>else Reserved</p>	<p>For RSA_Decrypt, the PROTIINFO field is defined in Table 7-65. For further information concerning RSA Decrypt see Implementation of the RSA decrypt operation.</p>
1A	<p>OPTYPE 000: RSA_Finish_KeyGen</p> <p>else Reserved</p>	<p>For RSA_Finish_Keygen, the PROTIINFO field is defined in Table 7-66. See RSA Finalize Key Generation (RFKG) for further information.</p>
1E	<p>OPTYPE 000: EC Public Key Validation</p> <p>else Reserved</p>	<p>For EC Public Key Validation use bit [0] to select F2m validation. No other PROTIINFO bits are used.</p>

Table continues on the next page...

Table 7-52. PROTID and PROTIINFO field description (continued)

PROTID (hex)	Description	PROTIINFO Information																								
20	OPTYPE 000: Derived Key MD5 else Reserved	For Derived Key MD5, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
21	OPTYPE 000: Derived Key SHA1 else Reserved	For Derived Key SHA1, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
22	OPTYPE 000: Derived Key SHA224 else Reserved	For Derived Key SHA224, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
23	OPTYPE 000: Derived Key SHA256 else Reserved	For Derived Key SHA256, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
24	OPTYPE 000: Derived Key SHA384 else Reserved	For Derived Key SHA384, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
25	OPTYPE 000: Derived Key SHA512 else Reserved	For Derived Key SHA512, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol .																								
2F	else Reserved																									
31	OPTYPE 11x: 3G Double CRC else Reserved	For 3G Double CRC, the PROTIINFO field is defined in Table 7-56																								
32	OPTYPE 11x: 3G RLC else Reserved	For 3G RLC, the PROTIINFO field is defined in Table 7-56 . For further information concerning 3G RLC, see 3G RLC PDU Encapsulation and Decapsulation overview .																								
42	OPTYPE 11x: LTE PDCP User Plane else Reserved	For LTE PDCP, the PROTIINFO field is defined in Table 7-56 . For further information concerning LTE, see 3G RLC PDU Encapsulation and Decapsulation overview .																								
43	OPTYPE 11x: LTE PDCP Control Plane - deprecated in favor of PROTID 44, which supports mixed cipher suites. PROTID 43 does not. else Reserved	For LTE PDCP (PROTID=43h), the PROTIINFO field is defined in Table 7-56 . For further information concerning LTE, see LTE PDCP PDU encapsulation and decapsulation overview																								
44	OPTYPE 11x: LTE PDCP Control Plane - supports mixed cipher suites else Reserved	For LTE Control Plane (PROTID=44h) the PROTIINFO field is interpreted as follows: <table border="1" data-bbox="803 1460 1469 1740"> <thead> <tr> <th colspan="2">(bits 15:8) - Encrypt/Decrypt Algorithm</th> <th colspan="2">(bits 7:0) - Authentication Algorithm</th> </tr> </thead> <tbody> <tr> <td>00h</td> <td>null</td> <td>00h</td> <td>null</td> </tr> <tr> <td>01h</td> <td>SNOW3G</td> <td>01h</td> <td>SNOW3G</td> </tr> <tr> <td>02h</td> <td>AES</td> <td>02h</td> <td>AES</td> </tr> <tr> <td>03h</td> <td>ZUC</td> <td>03h</td> <td>ZUC</td> </tr> <tr> <td colspan="2">all other values are reserved</td> <td colspan="2">all other values are reserved</td> </tr> </tbody> </table> <p>Note: Header meta data is not allowed for the following combinations:</p>	(bits 15:8) - Encrypt/Decrypt Algorithm		(bits 7:0) - Authentication Algorithm		00h	null	00h	null	01h	SNOW3G	01h	SNOW3G	02h	AES	02h	AES	03h	ZUC	03h	ZUC	all other values are reserved		all other values are reserved	
(bits 15:8) - Encrypt/Decrypt Algorithm		(bits 7:0) - Authentication Algorithm																								
00h	null	00h	null																							
01h	SNOW3G	01h	SNOW3G																							
02h	AES	02h	AES																							
03h	ZUC	03h	ZUC																							
all other values are reserved		all other values are reserved																								

Table continues on the next page...

Table 7-52. PROTID and PROTIINFO field description (continued)

PROTID (hex)	Description	PROTIINFO Information	
		(bits 15:8) - Encrypt/Decrypt Algorithm	(bits 7:0) - Authentication Algorithm
		SNOW3G Encrypt and AES Authentication or ZUC Encrypt and AES Authentication Note: Trailing meta data is not allowed for the following combinations: SNOW3G Encrypt/Decrypt and AES Authentication or ZUC Encrypt/Decrypt and AES Authentication	
45	OPTYPE 11x: LTE PDCP PDU User Plane for RN else Reserved	For 3G RLC, the PROTIINFO field is defined as shown in the table above (PROTID=44). For further information concerning 3G RLC, see 3G RLC PDU Encapsulation and Decapsulation overview .	
60	OPTYPE 000: Derived Key MD5 with RIF else Reserved	For Derived Key MD5 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
61	OPTYPE 000: Derived Key SHA1 with RIF else Reserved	For Derived Key SHA1 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
62	OPTYPE 000: Derived Key SHA224 with RIF else Reserved	For Derived Key SHA224 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
63	OPTYPE 000: Derived Key SHA256 with RIF else Reserved	For Derived Key SHA256 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
64	OPTYPE 000: Derived Key SHA384 with RIF else Reserved	For Derived Key SHA384 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
65	OPTYPE 000: Derived Key SHA512 with RIF else Reserved	For Derived Key SHA512 with RIF, the PROTIINFO field is defined in Table 7-57 . For further information concerning Derived Key Protocol, see Implementation of the derived key protocol . For further information concerning the RIF option, see Table 7-11 .	
6F	else Reserved		
All other values reserved.			

Table 7-53. PROTINFO definition when used with IPsec or SRTP protocols

PROTINFO [15:8] (hex)	Encryption algorithm	Notes	PROTINFO [7:0] (hex)	Authentication algorithm
01	DES	For IPsec ESP Transport (and legacy Tunnel), IPsec ESP Tunnel, any encryption algorithm at left can be used with any authentication algorithm at right. For SRTP, these encryption and authentication algorithms are not permitted (with the exception of AES-CTR, see below).	00	NULL authentication, also used with AES-CCM and AES-GCM
02	DES (same as above)		01	HMAC_MD5_96
03	3DES		02	HMAC_SHA1_96
0B	NULL Encryption		05	AES_XCBC_MAC_96
0C	AES-CBC		06	HMAC_MD5_128
0D	AES-CTR		07	HMAC_SHA1_160
			08	AES-CMAC-96
			0C	HMAC_SHA2_256_128
			0D	HMAC_SHA2_384_192
			0E	HMAC_SHA2_512_256
0D	AES-CTR	For SRTP, when using AES_CTR as the encryption algorithm HMAC_SHA1_160 must be selected as the authentication algorithm.	07	HMAC_SHA1_160
0E	AES-CCM-8	For both IPsec and SRTP, when using the algorithms at left PROTINFO[7:0] must be 00. The AES-GCM modes at left are available in the	00	Authentication is incorporated into the encryption algorithm.
0F	AES-CCM-12			
10	AES-CCM-16			
12	AES-GCM-8			
13	AES-GCM-12			
14	AES-GCM-16			
15	AES-NULL-WITH-GMAC			
All other values are reserved.				

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs)

PROT INFO [15:0] (hex)	Description		PROT INFO [15:0] (hex)	Description
0000	NULL_TLS. PRF not supported.			
0001	NULL_MD5. PRF not supported.			
0002	NULL_SHA. PRF not supported.			
0003	Reserved.		002F	AES_128_CBC_SHA
0004	Reserved.		0030	AES_128_CBC_SHA
0005	Reserved.		0031	AES_128_CBC_SHA
			0032	AES_128_CBC_SHA
0008	DES40_CBC_SHA		0033	AES_128_CBC_SHA

Table continues on the next page...

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

PROT INFO [15:0] (hex)	Description	PROT INFO [15:0] (hex)	Description
0009	DES_CBC_SHA	0034	AES_128_CBC_SHA
000A	3DES_EDE_CBC_SHA	0035	AES_256_CBC_SHA
000B	DES40_CBC_SHA	0036	AES_256_CBC_SHA
000C	DES_CBC_SHA	0037	AES_256_CBC_SHA
000D	3DES_EDE_CBC_SHA	0038	AES_256_CBC_SHA
000E	DES40_CBC_SHA	0039	AES_256_CBC_SHA
000F	DES_CBC_SHA	003A	AES_256_CBC_SHA
0010	3DES_EDE_CBC_SHA	003B	NULL_SHA-256. PRF not supported.
0011	DES40_CBC_SHA	003C	AES_128_CBC_SHA-256
0012	DES_CBC_SHA	003D	AES_256_CBC_SHA-256
0013	3DES_EDE_CBC_SHA	003E	AES_128_CBC_SHA-256
0014	DES40_CBC_SHA	003F	AES_128_CBC_SHA-256
0015	DES_CBC_SHA	0040	AES_128_CBC_SHA-256
0016	3DES_EDE_CBC_SHA		
0017	Reserved.	0067	AES_128_CBC_SHA-256
0018	Reserved.	0068	AES_256_CBC_SHA-256
0019	DES40_CBC_SHA	0069	AES_256_CBC_SHA-256
001A	DES_CBC_SHA	006A	AES_256_CBC_SHA-256
001B	3DES_EDE_CBC_SHA	006B	AES_256_CBC_SHA-256
		006C	AES_128_CBC_SHA-256
001E	DES_CBC_SHA	006D	AES_256_CBC_SHA-256
001F	3DES_EDE_CBC_SHA		
0020	Reserved.	008A	Reserved.
		008B	3DES_EDE_CBC_SHA
0022	DES_CBC_MD5	008C	AES_128_CBC_SHA
0023	3DES_EDE_CBC_MD5	008D	AES_256_CBC_SHA
0024	Reserved.	008E	Reserved.
		008F	3DES_EDE_CBC_SHA
0026	DES_CBC_40_SHA		
		0090	AES_128_CBC_SHA
0028	Reserved.	0091	AES_256_CBC_SHA
0029	DES_CBC_40_MD5	0092	Reserved.
		0093	3DES_EDE_CBC_SHA
002B	Reserved.	0094	AES_128_CBC_SHA
002C	NULL_SHA. PRF not supported.	0095	AES_256_CBC_SHA
002D	NULL_SHA. PRF not supported.		
002E	NULL_SHA. PRF not supported.		

Table continues on the next page...

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

PROT INFO [15:0] (hex)	Description	PROT INFO [15:0] (hex)	Description
		009C	Encap: AES_128_GCM; PRF: SHA-256
		009D	Encap: AES_256_GCM; PRF: SHA-384
		009E	Encap: AES_128_GCM; PRF: SHA-256
		009F	Encap: AES_256_GCM; PRF: SHA-384
00A0	Encap: AES_128_GCM; PRF: SHA-256		
00A1	Encap: AES_256_GCM; PRF: SHA-384		
00A2	Encap: AES_128_GCM; PRF: SHA-256		
00A3	Encap: AES_256_GCM; PRF: SHA-384		
00A4	Encap: AES_128_GCM; PRF: SHA-256		
00A5	Encap: AES_256_GCM; PRF: SHA-384		
00A6	Encap: AES_128_GCM; PRF: SHA-256		
00A7	Encap: AES_256_GCM; PRF: SHA-384		
00A8	Encap: AES_128_GCM; PRF: SHA-256		
00A9	Encap: AES_256_GCM; PRF: SHA-384	C020	AES_256_CBC_SHA
00AA	Encap: AES_128_GCM; PRF: SHA-256	C021	AES_256_CBC_SHA
00AB	Encap: AES_256_GCM; PRF: SHA-384	C022	AES_256_CBC_SHA
00AC	Encap: AES_128_GCM; PRF: SHA-256	C023	AES_128_CBC_SHA-256
00AD	Encap: AES_256_GCM; PRF: SHA-384	C024	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
00AE	AES-128-CBC_SHA-256	C025	AES_128_CBC_SHA-256
00AF	AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384	C026	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
00B0	NULL_SHA-256. PRF not supported.		
00B1	NULL_SHA-384. PRF not supported.	C027	AES_128_CBC_SHA-256
00B2	AES-128-CBC_SHA-256	C028	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
00B3	AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384	C029	AES_128_CBC_SHA-256
00B4	NULL_SHA-256. PRF not supported.		
00B5	NULL_SHA-384. PRF not supported.	C02A	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
00B6	AES-128-CBC_SHA-256	C02B	Encap: AES_128_GCM; PRF: SHA-256
00B7	AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384	C02C	Encap: AES_256_GCM; PRF: SHA-384
00B8	NULL_SHA-256. PRF not supported.		
00B9	NULL_SHA-384. PRF not supported.		
C001	NULL_SHA. PRF not supported.	C02D	Encap: AES_128_GCM; PRF: SHA-256
C002	Reserved.	C02E	Encap: AES_256_GCM; PRF: SHA-384
C003	3DES_EDE_CBC_SHA	C02F	Encap: AES_128_GCM; PRF: SHA-256

Table continues on the next page...

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

PROT INFO [15:0] (hex)	Description	PROT INFO [15:0] (hex)	Description
C004	AES_128_CBC_SHA	C030	Encap: AES_256_GCM; PRF: SHA-384
C005	AES_256_CBC_SHA	C031	Encap: AES_128_GCM; PRF: SHA-256
C006	NULL_SHA. PRF not supported.	C032	Encap: AES_256_GCM; PRF: SHA-384
C007	Reserved.	C033	Reserved.
C008	3DES_EDE_CBC_SHA	C034	3DES-CBC_SHA
C009	AES_128_CBC_SHA	C035	AES_128_CBC_SHA-1
C00A	AES_256_CBC_SHA	C036	AES_128_CBC_SHA-1
C00B	NULL_SHA. PRF not supported.	C037	AES_128_CBC_SHA-256
C00C	Reserved.	C038	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
C00D	3DES_EDE_CBC_SHA	C039	NULL_SHA. PRF not supported.
C00E	AES_128_CBC_SHA	C03A	NULL_SHA-256. PRF not supported.
C00F	AES_256_CBC_SHA	C03B	NULL_SHA-384. PRF not supported.
		C09C	AEAD_AES_128_CCM_16.
		C09D	AEAD_AES_256_CCM_16.
		C09E	AEAD_AES_128_CCM_16.
		C09F	AEAD_AES_256_CCM_16.
C010	NULL_SHA. PRF not supported.	C0A0	AEAD_AES_128_CCM_8.
C011	Reserved.	C0A1	AEAD_AES_256_CCM_8.
C012	3DES_EDE_CBC_SHA	C0A2	AEAD_AES_128_CCM_8.
C013	AES_128_CBC_SHA	C0A3	AEAD_AES_256_CCM_8.
C014	AES_256_CBC_SHA	C0A4	AEAD_AES_128_CCM_16.
C015	NULL_SHA. PRF not supported.	C0A5	AEAD_AES_256_CCM_16.
C016	Reserved.	C0A6	AEAD_AES_128_CCM_16.
C017	3DES_EDE_CBC_SHA	C0A7	AEAD_AES_256_CCM_16.
C018	AES_128_CBC_SHA	C0A8	AEAD_AES_128_CCM_8.
C019	AES_256_CBC_SHA	C0A9	AEAD_AES_256_CCM_8.
C01A	3DES_EDE_CBC_SHA	C0AA	AEAD_AES_128_CCM_8.
C01B	3DES_EDE_CBC_SHA	C0AB	AEAD_AES_256_CCM_8.
C01C	3DES_EDE_CBC_SHA		
C01D	AES_128_CBC_SHA		
C01E	AES_128_CBC_SHA		
C01F	AES_128_CBC_SHA		
FF22	NULL_SHA-224. PRF not supported.		
FF23	3DES_EDE_CBC_MD5	FF83	AES_128_CBC_SHA-384 PRF (if TLS 1.2): SHA-384

Table continues on the next page...

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

PROT INFO [15:0] (hex)	Description	PROT INFO [15:0] (hex)	Description
		FF84	AES_128_CBC_SHA-224 (not valid for PRF)
FF30	3DES_EDE_CBC_SHA	FF85	AES_128_CBC_SHA-512 (not valid for PRF)
		FF86	AES_128_CBC_SHA-256
FF33	3DES_EDE_CBC_SHA-384; PRF (if TLS 1.2): SHA-384		
FF34	3DES_EDE_CBC_SHA-224 (not valid for PRF)	FF90	AES_192_CBC_SHA
FF35	3DES_EDE_CBC_SHA-512 (not valid for PRF)		
FF36	3DES_EDE_CBC_SHA-256	FF93	AES_192_CBC_SHA-384; PRF (if TLS 1.2): SHA-384
		FF94	AES_192_CBC_SHA-224 (not valid for PRF)
FF51	NULL_SHA-512. PRF not supported.	FF95	AES_192_CBC_SHA-512 (not valid for PRF)
		FF96	AES_192_CBC_SHA-256
FF60	AES_256_CBC_SHA		
FF63	AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384		
FF64	AES_256_CBC_SHA-224 (not valid for PRF)		
FF65	AES_256_CBC_SHA-512 (not valid for PRF)		
FF66	AES_256_CBC_SHA-256		
FF70	AES_128_CTR_SHA. Not supported for PRF.	FFC0	AEAD_AES_128_CCM_8.
FF71	AES_192_CTR_SHA. Not supported for PRF.	FFC1	AEAD_AES_256_CCM_8.
FF72	AES_256_CTR_SHA. Not supported for PRF.	FFC2	AEAD_AES_128_CCM_16.
FF73	AES_128_CTR_SHA-224. Not supported for PRF.	FFC3	AEAD_AES_256_CCM_16.
FF74	AES_192_CTR_SHA-224. Not supported for PRF.	FFC4	AEAD_AES_128_CCM_8. PRF SHA-384.
FF75	AES_256_CTR_SHA-224. Not supported for PRF.	FFC5	AEAD_AES_256_CCM_8. PRF SHA-384.
FF76	AES_128_CTR_SHA-256. Not supported for PRF.	FFC6	AEAD_AES_128_CCM_16. PRF SHA-384.
FF77	AES_192_CTR_SHA-256. Not supported for PRF.	FFC7	AEAD_AES_256_CCM_16. PRF SHA-384.

Table continues on the next page...

Table 7-54. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

PROT INFO [15:0] (hex)	Description	PROT INFO [15:0] (hex)	Description
FF78	AES_256_CTR_SHA-256. Not supported for PRF.	FFC8	AEAD_AES_128_CCM_8. Not supported for PRF.
FF79	AES_128_CTR_SHA-384. Not supported for PRF.	FFC9	AEAD_AES_256_CCM_8. Not supported for PRF.
FF7A	AES_192_CTR_SHA-384. Not supported for PRF.	FFCA	AEAD_AES_128_CCM_16. Not supported for PRF.
FF7B	AES_256_CTR_SHA-384. Not supported for PRF.	FFCB	AEAD_AES_256_CCM_16. Not supported for PRF.
FF7C	AES_128_CTR_SHA-512. Not supported for PRF.		
FF7D	AES_192_CTR_SHA-512. Not supported for PRF.		
FF7E	AES_256_CTR_SHA-512. Not supported for PRF.		
		FFFE	(SSL, TLS 1.0, TLS 1.1 PRF only): master secret generation using SHA-1 and MD5 (TLS 1.2 PRF only): master secret generation using SHA-384
FF80	AES_128_CBC_SHA	FFFF	(SSL, TLS 1.0, TLS 1.1 PRF only): master secret generation using SHA-1 and MD5 (TLS 1.2 PRF only): master secret generation using SHA-256
All other values reserved. Not all codes permitted with all members of TLS family			

Table 7-55. PROTINFO definition when used with IKE PRF and IKE PRF+ protocols

PROTINFO [15:0] (hex)	Description
0100	PRF_HMAC_MD5
0200	PRF_HMAC_SHA1
0400	PRF_AES128_CBC
0500	PRF_HMAC_SHA2_256
0600	PRF_HMAC_SHA2_384
0700	PRF_HMAC_SHA2_512
0800	PRF_AES128_CMAC
All other values reserved.	

Table 7-56. PROTINFO definition when used with 3G and LTE (PROTID=43h) protocols

PROTINFO [15:0] (hex)	Description
0710	3G Double CRC with 7-bit and 16-bit CRCs
0B10	3G Double CRC with 11-bit and 16-bit CRCs
0000	3G RLC with Null encryption
0001	3G RLC with Kasumi encryption
0002	3G RLC with SNOW3G encryption
0000	LTE PDCP with Null encryption and authentication
0001	LTE PDCP with SNOW3G encryption and authentication
0002	LTE PDCP with AES encryption and authentication
All other values reserved.	

Table 7-57. PROTINFO definition when used with derived key protocol (DKP) for HMACs

PROTINFO	Description
PROTINFO[15:14] SRC	<p>Input Source Control</p> <p>00 - IMM - negotiated key is in words immediately following the DKP Operation Command. This option can only be used with an Immediate Output Destination (OD=00).</p> <p>01 - SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command.</p> <p>10 - PTR - the input key is referenced by the address found immediately following the DKP Operation Command.</p> <p>11 - SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command.</p>
PROTINFO[13:12] DST	<p>Output Destination Control</p> <p>00 - IMM - resulting derived HMAC "split key" will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting derived HMAC key is twice the underlying hash context length. See Table 10-32</p> <p>Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source.</p> <p>01 - SEQ - the resulting derived HMAC "split key" will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source.</p> <p>10 - PTR - the resulting derived HMAC "split key" will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF.</p> <p>11 - SGF - the resulting derived HMAC "split key" will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR.</p>
PROTINFO[11:0] LEN	Length of the negotiated key provided to the DKP Operation command in bytes.

Table 7-58. PROTINFO format when used with Blob Operations

15-10	9	8	7-4	3	2	1-0
Reserved	TK	EKT	K2KR	Reserved	Black_Key	Blob_Format

Table 7-59. PROTINFO field descriptions when used with Blob Operations

Field	Description
15-10	Reserved.
9 TK	Trusted Key Used only for trusted descriptors with black blob encapsulation/decapsulation. Ignored otherwise. 0 Use the JDKEK when encrypting or decrypting black keys. 1 Use the TDKEK when encrypting or decrypting black keys.
8 EKT	Encrypted Key Type Used only for black blob encapsulation/decapsulation. Ignored otherwise. Specifies the encryption/decryption mode for black keys. Also used when deriving the blob key encryption key. Consequently, the same EKT setting must be used when decapsulating a black blob as was used when encapsulating that black blob. This prevents a black key being converted between AES-ECB and AES-CCM by encapsulating it as a blob and then decapsulating the blob in the other encryption mode. 0 Use AES-ECB mode when encrypting/decrypting black keys. 1 Use AES-CCM mode when encrypting/decrypting black keys.
7-4 K2KR	Key to Key Register Specifies the destination for the result of black blob decapsulation. Ignored otherwise. Black blob encapsulation always uses a source from memory. The source and destination for red blob encapsulation and decapsulation is always memory. (See Blob types differentiated by content) 0000 Memory 0001 Class 1 key register 0011 Class 2 key register 0111 Class 2 key register (split key) 1001 PKHA E RAM All other values are reserved.
3	Reserved.
2 Black_key	0 Red Blob. The data encapsulated into the blob or decapsulated from the blob is treated as plaintext. 1 Black Blob. The data encapsulated into the blob or decapsulated from the blob is treated as a black key encrypted with the appropriate KEK (JDKEK or TDKEK). For blob encapsulation operations, the input data is first decrypted using the appropriate KEK and then encrypted using the blob key. For blob decapsulation operations, the data portion of the blob is decrypted using the blob key. If the resulting plaintext is to be written into memory rather than into a key register, the plaintext is encrypted using the appropriate KEK.
1-0 Blob_Format	The format of the blob. 00 Normal Blob. The output is composed of the encrypted blob key, the encrypted data, and MAC tag. 01 Reserved 10 Master Key Verification Blob. This blob type is intended for verifying the master key and the key derivation. The master key is used for key derivation in the Trusted and Secure security states. The test key is used in the Nonsecure state. Only the derived blob key encryption key is output. Note that the Blob_Format value is

Table 7-59. PROTINFO field descriptions when used with Blob Operations

Field	Description
	<p>an input to the BKEK derivation, which ensures that the BKEK value that is exposed in a master key verification blob is different than the BKEK value used for any other blob format. Furthermore, the use of SHA-256 in BKEK derivation ensures that the BKEK values used for other blob formats cannot be learned by analyzing the BKEK values used for master key verification blobs.</p> <p>11 Test Blob. The non-volatile test key is used for key derivation. The output is composed of the derived blob key encryption key, the actual blob key, the encrypted blob key, the encrypted data, and MAC tag. Test blobs can be exported or imported only when SEC is in non-secure mode.</p>

Table 7-60 shows the format of the PROTINFO field for discrete log public key protocols, including:

- Key pair generation (see [Discrete-log key-pair generation](#))
- DSA sign (see [Generating DSA and ECDSA signatures](#))
- DSA verify (see [Verifying DSA and ECDSA signatures](#))
- Diffie-Hellman (see [Using the Diffie_Hellman function](#)).

Table 7-61. describes the bit values of this field.

Table 7-60. PROTINFO format when used with Discrete Log Protocol

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format for Sign function	Reserved			SIGN_NO_T EQ	MES_R EP					SIGN_2 ND_ HALF_ ONLY	SIGN_1 ST_ HALF_ ONLY	EXT_PR I	TEST	ENC_ PRI	ECC/D L	F2M/F p
Format for MPSign functions	Reserved			SIGN_NO_T EQ	MES_R EP					Reserved	Reserved	EXT_PR I	TEST	ENC_ PRI	ECC/D L	F2M/F p
Format for Verify function	Reserved				MES_R EP										ECC/D L	F2M/F p
Format for Keypair Generation functions	Reserved							KPG_I ETF_D H	EKT_Z	ENC_Z	EXT_PR I	KPG_ NO_TE Q	ENC_ PRI	ECC/D L	F2M/F p	
Format for MP Keypair Generation functions	Reserved											KPG_ NO_TE Q	Reserved			

Table 7-61. PROTINFO field descriptions when used with Discrete Log Protocol

Field	Description
MES_REP	Build the message representative from the message

Table continues on the next page...

Table 7-61. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

Field	Description
	<p>0 Use the message representative that is pointed to by the f field in the protocol data block.</p> <p>1 Calculate the message representative from the message (using a SEQ IN PTR command), and the hash function specified by the HASH field. The message representative is calculated using the equivalent of EMSA1 (IEEE-1363).</p>
SIGN_NO_TEQ	<p>For Signature Generation (SIGN) protocol and MPISign: disable Timing Equalization during SIGN.</p> <p>0 Run SIGN using normal Timing Equalization protection.</p> <p>1 Run SIGN with NO Timing Equalization protection.</p>
MES_REP	<p>For Signature Generation (SIGN) and Verification (VERIFY) protocols, this field indicates the format of the message.</p> <p>00 : F input is a message representative.</p> <p>01 : Calculate the message representative from the message (using a SEQ IN PTR command), and the hash function specified by the HASH field. The message representative is calculated using the equivalent of EMSA1 (IEEE-1363).</p> <p>10 : F input is a hashed message, with length specified in the PDB. Protocol will format the message as required.</p> <p>11 : Reserved.</p>
HASH	<p>Hash function used to calculate a message representative from a message; valid when MES_REP=01.</p> <p>000 MD5</p> <p>001 SHA-1</p> <p>010 SHA-224</p> <p>011 SHA-256</p> <p>100 SHA-384</p> <p>101 SHA-512</p>
KPG_IETF_DH	<p>For KPG, this bit enables running IETF_style DH</p> <p>0 No IETF-style DH.</p> <p>1 Run KPG with IETF-style Diffie-Hellman.</p>
PRI_VERIFY_NO_TEQ	<p>For Signature Verification with Private Key (PRI_VERIFY) protocol only. Disable Timing Equalization protection.</p> <p>0 Run PRI_VERIFY with Timing Equalization protection enabled.</p> <p>1 Run PRI_VERIFY with Timing Equalization protection disabled.</p>
SIGN_2ND_HALF_ONLY	<p>For Signature Generation (SIGN) protocol only; otherwise reserved. Run 2nd half (signature "d" generation) only.</p> <p>0 Run full SIGN or 1st half, depending on SIGN_1ST_HALF_ONLY setting.</p> <p>1 Run 2nd half of SIGN only, generating 'd' result. Requires SIGN_1ST_HALF_ONLY = 0.</p>

Table continues on the next page...

Table 7-61. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

Field	Description
SIGN_1ST_HALF_ONLY	For Signature Generation (SIGN) protocol only; otherwise reserved. Run 1st half (signature "c" generation) only. 0 Run full SIGN or 2nd half, depending on SIGN_2ND_HALF_ONLY setting. 1 Run 1st half of SIGN only, generating 'c' result. Requires SIGN_2ND_HALF_ONLY = 0.
EKT_Z	if ENC_Z=1, Key Encryption type (Used only with DH; otherwise reserved.) 0 Secret output is encrypted with AES-ECB mode. 1 Secret output is encrypted with AES-CCM mode.
ENC_Z	Encrypt the DH shared secret (Used only with DH; otherwise reserved.) 0 The DH output is public and is unencrypted. 1 The DH output is secret and encrypted.
EXT_PRI	if ENC_PRI=1, Encrypted key type for private key 0 Private key is encrypted with AES-ECB mode. 1 Private key is encrypted with AES-CCM mode.
KPG_NO_TEQ	KPG_NO_TEQ For KPG, MPPrivK and MPPubK. 0 Key Pair Generation runs with Timing Equalization protection. 1 Key Pair Generation runs with Timing Equalization disabled.
TEST	TEST 0 Signature generation protects the per message secret. 1 Signature generation outputs the per message secret, to aid in testing and verification. This is not allowed in trusted or secure states.
ENC_PRI	Encrypted private key 0 Private key is not encrypted. ENC_PRI must be 0 if SIGN_2ND_HALF_ONLY=1. 1 Private key must be decrypted before use (see KEY command for further information.). For Key Generation, this causes the Private Key to be encrypted.
ECC/DL	Public Key operation type 0 DL: Discrete Log 1 ECC: Elliptic Curve Cryptography
F2M/Fp	Finite Field type 0 Fp: Prime Field 1 F2M: Binary field

[Table 7-62](#) shows the format of the PROTINFO field for the RSA encrypt protocol. [Table 7-63](#) describes the bit values.

Table 7-62. PROTINFO format when used with RSA Encrypt Protocol

15-13	12	1-7	6-4	3-2	1-0
Reserved	FMT	Reserved	fff	Reserved	OP

Table 7-63. PROTINFO field descriptions when used with RSA Encrypt Protocol

Field	Description
15-13	Reserved.
12 FMT	Format of data 0 No formatting 1 EME-PKCS1-v1_5 encryption encoding function
11-7	Reserved.
6-4 fff	Encryption type for f 000b f is not encrypted (This is the only value permitted when OP = 00b). 001b f is to be encrypted with the JDKEK using ECB mode. 011b f is to be encrypted with the JDKEK using CCM mode. 101b f is to be encrypted with the TDKEK using ECB mode. 111b f is to be encrypted with the TDKEK using CCM mode. All other values are reserved.
3-2	Reserved.
1-0 OP	Operation. 00b Public Key n, e, f in - f is a user-supplied value (fff must be 00b) 01b Public Key n, e, f out - f is a random value (f can be encrypted on output, fff can be any non-reserved value) All other values are reserved.

Table 7-64 shows the format of the PROTINFO field for the RSA Decrypt Protocol. Table 7-65 describes the bit values.

Table 7-64. PROTINFO format when used with RSA Decrypt Protocol

15-13	12	11	10-8	7	6-4	3-2	1-0
Reserved	FMT	Reserved	ppp	Reserved	fff	Reserved	Key Form

Table 7-65. PROTINFO field descriptions when used with RSA Decrypt Protocol

Field	Description
15-13	Reserved.
12 FMT	Format of data 0 No formatting 1 EME-PKCS1-v1_5 encryption decoding function

Table continues on the next page...

Table 7-65. PROTINFO field descriptions when used with RSA Decrypt Protocol (continued)

Field	Description
11	Reserved.
10-8 ppp	Type of private key encryption 000 private key is not encrypted 001b private key components are each encrypted with the JDKEK using ECB mode 011b private key components are each encrypted with the JDKEK using CCM mode 101b private key components are each encrypted with the TDKEK using ECB mode 111b private key components are each encrypted with the TDKEK using CCM mode All other values are reserved.
7	Reserved.
6-4 fff	Type of encryption for f. 000b f is not to be encrypted 001b f is to be encrypted with the JDKEK using ECB mode 011b f is to be encrypted with the JDKEK using CCM mode 101b f is to be encrypted with the TDKEK using ECB mode 111b f is to be encrypted with the TDKEK using CCM mode All other values are reserved.
3	No TEQ option. Set to 1 to enable no-TEQ.
2	Reserved.
1-0 Key Form	Form of the Private Key 00b Private Key input in the form #1: n, d 01b Private Key input in the form #2: p, q, d 10b Private Key input in the form #3: p, q, dp, dq, c All other values are reserved.

Table 7-66. PROTINFO format when used with RSA Finish KeyGen

15-8	7	6	5	4	3	2	1-0
Reserved	Reserved	ENC_OUT	Reserved	EKT	SKIP_D	SKIP_PQ	FUNCTION

Table 7-67. PROTINFO field descriptions when used with RSA Finish Keygen Protocol

Field	Description
15-8	Reserved.
7	Reserved.
6 ENC_OUT	Encrypt Outputs 0 Do not encrypt generated private key components 1 Encrypt generated private key (ECB mode, unless EKT=1) (Note that <i>n</i> and <i>d</i> size are not encrypted.)

Table continues on the next page...

Table 7-67. PROTINFO field descriptions when used with RSA Finish Keygen Protocol (continued)

Field	Description
5 Reserved	Reserved. Must be 0.
4 EKT	Encrypted Key Type 0 Do not use CCM-encryption 1 CCM-encrypt private key components (valid only if PROTOCOL command's ENC bit is 1)
3 SKIP_D	Skip length check of d 0 Check that d is at least one bit longer than 1/2 of the bit length of n . 1 Skip length check of d .
2 SKIP_PQ	Skip check of upper 100 bits of p and q 0 Check upper 100 bits of p and q to see whether $ p-q $ is too small 1 Do not check upper 100 bits of p and q
1-0 FUNCTION	Function 00 Compute all key components listed in Key Form, including d 01 Compute all key components listed in Key Form except d , which is an input 10 From p , q , e , compute n , d and d size. 11 Reserved

7.18 PKHA OPERATION command

If OPTYPE = 001 (PKHA), the fields are as shown in [Table 7-68](#). This OPTYPE is used to perform public key operations in the public key hardware accelerator (PKHA). All data for a PKHA operation must already be in place before the function will begin executing. Therefore, this operation does not start until all data transactions have completed and the input data FIFO is empty.

The format of the PKHA MODE field depends on which of the four types of PKHA functions the OPERATION command specifies:

- Clear memory
- Modular arithmetic
- Elliptic curve
- Copy memory

A detailed description of the PKHA MODE fields is found in [Table 7-69](#). The OPERATION command does not complete until the PKHA is done.

When the PKHA operation completes without error, DECO clears the DONE flag and the Mode Register so another operation can be specified.

Table 7-68. PKHA OPERATION command format

31-27	26-24	23-20	19-16
CTYPE = 10000	OPTYPE = 001	ALG = 1000	PKHA_MODE_MS
15-12	11-0		
Reserved	PKHA_MODE_LS		

Table 7-69. PKHA OPERATION command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field)
26-24 OPTYPE	Operation Type If OPTYPE = 001b : PKHA OPERATION: The PKHA_MODE fields are interpreted as shown in the following tables. If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command. If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION command.
23-20 ALG	Algorithm Set ALG=1000b. All other values are reserved.
19-16 PKHA_MODE_MS	PKHA Mode This field contains the value that will be loaded into the upper 4 bits of the PKHA Mode register. Its content depends on which of the four types of PKHA functions, clear memory, modular arithmetic function, or copy memory, is specified in the Function field (bits 5-0). The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function , PKHA OPERATION: Arithmetic Functions , PKHA OPERATION: Elliptic Curve Functions and PKHA OPERATION: copy memory functions .
15-12	Reserved
11-0 PKHA_MODE_LS	PKHA Mode This field contains the value that will be loaded into the lowest 12 bits of the PKHA Mode register. The least-significant six bits of this field is interpreted as a Function field, as shown in the row below. The format of the PKHA_MODE_MS field and the other bits of the PKHA_MODE_LS field depend on the PKHA function specified in the Function field: clear memory, modular arithmetic function, or copy memory. The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function , PKHA OPERATION: Arithmetic Functions , PKHA OPERATION: Elliptic Curve Functions , and PKHA OPERATION: copy memory functions .
5-0 Function	PKHA function to be performed. (Note that the function is encoded in the least-significant six bits of the PKHA_MODE_LS field.) If Function=000001b : Clear Memory. (See Clear Memory (CLEAR_MEMORY) function) If Function=010000b or 010001b : Copy Memory. (See PKHA OPERATION: copy memory functions) If Function=001001b, 001010b, 001011b, or 011100b : Elliptic Curve function. (See PKHA OPERATION: Elliptic Curve Functions)

Table 7-69. PKHA OPERATION command field descriptions

Field	Description
	If Function=000010b - 001111b or 010110b - 011111b : Modular Arithmetic function. (See PKHA OPERATION: Arithmetic Functions) All other values of the Function field are reserved.

7.18.1 PKHA OPERATION: clear memory function

Table 7-70. PKHA Mode register format for clear memory function

19	18	17	16	11-10	9	8	7	6	5-0
Aram	Bram	Eram	Nram	Reserved	Q3	Q2	Q1	Q0	Function
PKHA_MODE_MS				PKHA_MODE_LS					

If the Function field in PKHA MODE specifies the clear memory function, PKHA expects to be in the format shown in [Table 7-70](#). The PKHA RAMs to be cleared may be selected in any combination. Selecting one or more Quadrants for clearing will cause only the specified quadrants (of the specified RAMs) to be cleared. If no Quadrants are selected, then the whole RAM will be cleared.

Table 7-71. PKHA mode register field descriptions for clear memory function

Bits	Description
19	Aram This bit selects the A RAM for zeroization. 0: A not selected 1: A selected.
18	Bram This bit selects the B RAM for zeroization. 0: B not selected 1: B selected
17	Eram This bit selects the E RAM for zeroization. 0: E not selected 1: E selected
16	Nram This bit selects the N RAM for zeroization. 0: N not selected 1: N selected
11-10	Reserved

Table continues on the next page...

Table 7-71. PKHA mode register field descriptions for clear memory function (continued)

Bits	Description
9	Quadrant 3 This bit selects the Quadrant 3 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
8	Quadrant 2 This bit selects the Quadrant 2 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
7	Quadrant 1 This bit selects the Quadrant 1 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid is E RAM is selected.
6	Quadrant 0 This bit selects the Quadrant 0 RAM for zeroization. 0: not selected 1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected.
5-0	Function The Function value for clearmemory is 000001.

7.18.2 PKHA OPERATION: Arithmetic Functions

Table 7-72. PKHA Mode Register Format for Arithmetic Functions

19	18	17	16-12	11	10	9-8	7-6	5-0
inM	outM	F2m	Reser ved	Reser ved	Teq	OutSel	Reserved	Function
<i>PKHA_MODE_MS</i>				<i>PKHA_MODE_LS</i>				

Table 7-73. PKHA Mode register, format for arithmetic operation

Bits	Description
19 inM	Inputs in Montgomery form. Indicates whether the inputs are in Montgomery form. If inM=0 : Normal value representation If inM=1 : Montgomery form. (Not valid for all functions.)
18 outM	Outputs in Montgomery form. Indicates whether the outputs are to be left in Montgomery form or converted to normal values. If outM=0 : Normal value representation If outM=1 : Montgomery form. (Not valid for all functions.)

Table continues on the next page...

Table 7-73. PKHA Mode register, format for arithmetic operation (continued)

Bits	Description
17 F2m	F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function. If F2m=0 : Integer If F2m=1 : Binary polynomial. (Not valid for all functions.)
16-12	Reserved
11	Reserved
10 Teq	Timing Equalized. Indicates that a timing equalized version of the function should be executed. If Teq=0 : No timing equalization If Teq=1 : Timing equalization. (Not valid for all functions.)
9-8 OutSel	Output destination select. Indicates which memory should contain the output of the selected function. If OutSel=00b : B If OutSel=01b : A If OutSel=10b : Reserved If OutSel=11b : Reserved
7-6	Reserved
5-0 Function	Function. Indicates which arithmetic function to execute. If Function=000010b : Modular Addition $(A + B) \bmod N$ If Function=000011b : Modular Subtraction 1 $(A - B) \bmod N$ If Function=000100b : Modular Subtraction 2 $(B - A) \bmod N$ If Function=000101b : Modular Multiplication $(A \times B) \bmod N$ If Function=000110b : Modular Exponentiation $A^E \bmod N$ If Function=000111b : Modular Reduction $A \bmod N$ If Function=001000b : Modular Inversion $A^{-1} \bmod N$ If Function=001100b : Montgomery Radix Constant $R^2 \bmod N$ If Function=001110 : Greatest Common Divisor $\text{GCD}(A,N)$ -see note below If Function=001111 : Miller-Rabin Primality Test -see note below If Function=010110b : Modular Simultaneous Exponentiation $A0^E * A2^B \bmod N$ If Function=011000b : Modular Double A $(A + A) \bmod N$ If Function=011001b : Modular Double B $(B + B) \bmod N$ If Function=011010b : Modular Square A $(A \times A) \bmod N$ If Function=011011b : Modular Cube A $(A \times A \times A) \bmod N$ If Function=011101b : Shift Right A If Function=011110b : Compare A B If Function=011111b : Evaluate A All other values for this field are currently reserved or are Table 7-70 , Table 7-77 , or Table 7-81 . NOTE: When using the GCD function or any ECC function, a divide-by-zero error occurs if the value of the most significant digit of N is all zeros. NOTE: When using the Miller-Rabin primality test function, if the most-significant digit of N is all zeros, the result is composite regardless of the value of N.

NOTE

Note that the arithmetic functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

Table 7-74. List of mode values for PKHA Integer Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
MOD_ADD	Integer Modular Addition	B	0	00002	Integer Modular Addition (MOD_ADD) function
		A	0	00102	
MOD_SUB_1	Integer modular subtraction (A - B)	B	0	00003	Integer Modular Subtraction (MOD_SUB_1) function
		A	0	00103	
MOD_SUB_2	Integer modular subtraction (B - A)	B	0	00004	Integer Modular Subtraction (MOD_SUB_2) function
		A	0	00104	
MOD_MUL MOD_MUL_TEQ	Integer modular multiplication	B	0	00005	Integer Modular Multiplication (MOD_MUL)
		A	0	00105	
	Timing equalized version	B	1	00405	
		A	1	00505	
MOD_MUL_IM MOD_MUL_IM_TEQ	Integer Modular Multiplication with Montgomery Inputs	B	0	80005	Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM)
		A	0	80105	
	Timing equalized version	B	1	80405	
		A	1	80505	
MOD_MUL_IM_OM MOD_MUL_IM_OM_TEQ	Integer Modular Multiplication with Montgomery Inputs and Outputs	B	0	C0005	Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function
		A	0	C0105	
	Timing equalized version	B	1	C0405	
		A	1	C0505	
MOD_EXP MOD_EXP_TEQ	Integer Modular Exponentiation	B	0	00006	Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ)
		A	0	00106	
	Timing equalized version	B	1	00406	
		A	1	00506	
MOD_EXP_IM MOD_EXP_IM_TEQ	Integer Modular Exponentiation with Montgomery Inputs	B	0	80006	Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function
		A	0	80106	
	Timing equalized version	B	1	80406	
		A	1	80506	
MOD_AMODN	Integer Modular Reduction	B	0	00007	Integer Modulo Reduction (MOD_AMODN)
		A	0	00107	
MOD_INV	Integer Modular Inversion	B	0	00008	Integer Modular Inversion (MOD_INV)
		A	0	00108	

Table continues on the next page...

Table 7-74. List of mode values for PKHA Integer Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
MOD_R2	Integer R ² mod N	B	0	0000C	Integer Montgomery Factor Computation (MOD_R2)
		A	0	0010C	
MOD_GCD	Integer Greatest Common Divisor	B	0	0000E	Integer Greatest Common Divisor (MOD_GCD)
		A	0	0010E	
PRIME_TEST	Miller_Rabin primality test	B	0	0000F	Miller_Rabin Primality Test (PRIME_TEST)
		A	0	0010F	
MOD_SML_EXP	Integer Modular Simultaneous Exponentiation	B	0	00016	Integer Simultaneous Modular Exponentiation (MOD_SML_EXP)
		A	0	00116	
MOD_SQRT	Integer Modular Square Root	B	0	00017	Integer Modular Square Root (MOD_SQRT)
		A	0	20017	
MOD_DBL_A	Integer Modular Double A	B	0	20018	
		A	0	20118	
MOD_DBL_B	Integer Modular Double B	B	0	20019	
		A	0	20119	
MOD_SQR MOD_SQR_TEQ	Integer Modular Square A	B	0	0001A	Integer Modular Square (MOD_SQR and MOD_SQR_TEQ)
		A	0	0011A	
	Timing equalized version	B	1	0041A	
		A	1	0051A	
MOD_IM_SQR MOD_IM_SQR_TEQ	Integer Modular Square A, Montgomery Input	B	0	8001A	Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ)
		A	0	8011A	
	Timing equalized version	B	1	8041A	
		A	1	8051A	
MOD_IM_OM_SQR MOD_IM_OM_SQR_TEQ	Integer Modular Square A, Montgomery Input, Montgomery Output	B	0	C001A	Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ)
		A	0	C011A	
	Timing equalized version	B	1	C041A	
		A	1	C051A	
MOD_CUBE MOD_CUBE_TEQ	Integer Modular Cube A	B	0	0001B	Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ)
		A	0	0011B	
	Timing equalized version	B	1	0041B	
		A	1	0051B	
MOD_CUBE_IM MOD_CUBE_IM_TEQ	Integer Modular Cube A, Montgomery input	B	0	8001B	Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ)
		A	0	8011B	
	Timing equalized version	B	1	8041B	
		A	1	8051B	

Table continues on the next page...

Table 7-74. List of mode values for PKHA Integer Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
MOD_CUBE_IM_OM MOD_CUBE_IM_OM_TEQ	Integer Modular Cube A, Montgomery input, Montgomery output	B	0	C001B	Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ)
		A	0	C011B	
	Timing equalized version	B	1	C041B	
		A	1	C051B	

1. PKHA_MODE_MS concatenated with 0000b concatenated with PKHA_MODE_LS

Arithmetic functions on a binary polynomials (characteristic two) (F_2^m). All operate in polynomial basis.

Table 7-75. List of mode values for PKHA Binary Polynomial Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
F2M_ADD	Binary Polynomial Modular Addition	B	0	20002	Binary Polynomial (F_2^m) Addition (F2M_ADD) function
		A	0	20102	
F2M_MUL F2M_MUL_TEQ	Binary Polynomial Modular Multiplication	B	0	20005	Binary Polynomial (F_2^m) Modular Multiplication (F2M_MUL)
		A	0	20105	
	Timing equalized version	B	1	20405	
		A	1	20505	
F2M_MUL_IM F2M_MUL_IM_TEQ	Binary Polynomial Modular Multiplication with Montgomery Inputs	B	0	A0005	Binary Polynomial (F_2^m) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function
		A	0	A0105	
	Timing equalized version"	B	1	A0405	
		A	1	A0505	
F2M_MUL_IM_OM F2M_MUL_IM_OM_TEQ	Binary Polynomial Modular Multiplication with Montgomery Inputs and Output	B	0	E0005	Binary Polynomial (F_2^m) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function
		A	0	E0105	
	Timing equalized version	B	1	E0405	
		A	1	E0505	
F2M_EXP F2M_EXP_TEQ	Binary Polynomial Modular Exponentiation	B	0	20006	Binary Polynomial (F_2^m) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)
		A	0	20106	
	Timing equalized version	B	1	20406	
		A	1	20506	

Table continues on the next page...

Table 7-75. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description	
F2M_AMODN	Binary Polynomial Modular Reduction	B	0	20007	Binary Polynomial (F_{2^m}) Modulo Reduction (F2M_AMODN)	
		A	0	20107		
F2M_INV	Binary Polynomial Modular Inversion	B	0	20008	Binary Polynomial (F_{2^m}) Modular Inversion (F2M_INV)	
		A	0	20108		
F2M_R2	Binary Polynomial $R^2 \text{ mod } n$	B	0	2000C	Binary Polynomial (F_{2^m}) $R^2 \text{ Mod } N$ (F2M_R2) Function	
		A	0	2010C		
F2M_GCD	Binary Polynomial Greatest Common Divisor	B	0	2000E	Binary Polynomial (F_{2^m}) Greatest Common Divisor (F2M_GCD) Function	
		A	0	2010E		
F2M_SQR F2M_SQR_TEQ	Binary Polynomial Modular A Square	B	0	2001A		
		A	0	2011A		
	Timing equalized version		B	1		2041A
			A	1		2051A
F2M_IM_SQR F2M_IM_SQR_TEQ	Binary Polynomial Modular A Square. Montgomery input	B	0	A001A		
		A	0	A011A		
	Timing equalized version		B	1		A041A
			A	1		A051A
F2M_IM_OM_SQR F2M_IM_OM_SQR_TEQ	Binary Polynomial Modular Square A, Montgomery input, Montgomery output	B	0	E001A		
		A	0	E011A		
	Timing equalized version		B	1		E041A
			A	1		E051A
F2M_CUBE F2M_CUBE_TEQ	Binary Polynomial Modular Cube A	B	0	2001B		
		A	0	2011B		
	Timing equalized version		B	1		2041B
			A	1		2051B
F2M_CUBE_IM F2M_CUBE_IM_TEQ	Binary Polynomial Modular Cube A. Montgomery input	B	0	A001B		
		A	0	A011B		
	Timing equalized version		B	1		A041B
			A	1		A051B
F2M_CUBE_IM_OM F2M_CUBE_IM_OM_TEQ	Binary Polynomial Modular Cube A, Montgomery input, Montgomery output	B	0	E001B	Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_O	
		A	0	E011B		
	Timing equalized version		B	1		E041B

Table continues on the next page...

Table 7-75. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
		A	1	E051B	M and F2M_CUBE_IM_OM_TEQ)
F2M_SML_EXP	Binary Polynomial Modular Simultaneous Exponentiation	B	0	20016	Binary Polynomial (F_{2^m}) Simultaneous Modular Exponentiation (F2M_SML_EXP)
		A	0	20116	

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

These functions are grouped here because they do not fall into one of the previous categories of PKHA functions.

Table 7-76. List of mode values for Miscellaneous Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
Shift Right A	Right Shift	B	0	0001D	Right Shift A (R_SHIFT) function
		A	0	0011D	
Compare A B	Comparison	(no output)	0	0001E	Compare A B (COMPARE) function
Evaluate A	Compute sizes	B	0	0001F	Evaluate A (EVALUATE) function
		A	0	0011F	

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

7.18.3 PKHA OPERATION: copy memory functions

Table 7-77. PKHA Mode register, format for copy memory functions

19-17	16	11-10	9-8	7-6	5-0
Source Register	Destination Register	Source Segment	Destination Segment	Function	
PKHA_MODE_MS		PKHA_MODE_LS			

Table 7-78. PKHA Mode register, field descriptions for copy memory functions

Bits	Description
19-17 Source Register	Source Register. Specifies the register to be copied from. If Source Register=000 : A If Source Register=001 : B If Source Register=011 : N All other values are currently reserved.
16 Destination Register	Destination Register. Specifies the register to be copied to. If Destination Register=000 : A If Destination Register=001 : B
11-10	If Destination Register=010 : E If Destination Register=011 : N All other values are currently reserved. NOTE: The source register and destination register fields must not be the same.
9-8 Source Segment	Source Segment. Used when copying a register segment to specify which segment in the source register to copy from. If Source Segment=00 : Segment 0 If Source Segment=01 : Segment 1 If Source Segment=10 : Segment 2 If Source Segment=11 : Segment 3 NOTE: These bits must be zero when E is the destination register.
7-6 Destination Segment	Destination Segment. Used when copying a register segment to specify which segment in the Destination Register to copy to. If Destination Segment=00 : Segment 0 If Destination Segment=01 : Segment 1 If Destination Segment=10 : Segment 2 If Destination Segment=11 : Segment 3 NOTE: These bits must be zero when E is the destination register.
5-0 Function	Function. Indicates which copy function to execute. If Function=010000 : Copy Memory N-Size (copies the same number of words as are in the modulus.) If Function=010001 : Copy Memory SRC-Size (copies the number of words specified in the source's size register)

This table gives the encodings for the PKHA memory-to-memory copy functions. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size ([Copy memory, N-Size and Source-Size \(COPY_NSZ and COPY_SSZ\)](#)).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

Table 7-79. Mode values for PKHA copy memory functions

Source Memory	Destination Memory			
	A	B	N	E
A		00410	00C10	00810
		00411	00C11	00811
B	20010		20C10	20810
	20011		20C11	20811
N	60010	60410		60810
	60011	60411		60811

This table gives the encodings for the PKHA memory-to-memory copy functions, when segments are involved. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size (*Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)*).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

Table 7-80. Mode values for PKHA copy memory by segment functions

Source Quadrant	Destination Quadrant											
	A0	A1	A2	A3	B0	B1	B2	B3	N0	N1	N2	N3
A0					00410	00450	00490	004D0	00C10	00C50	00C90	00CD0
					00411	00451	00491	004D1	00C11	00C51	00C91	00CD1
A1					00510	00550	00590	005D0	00D10	00D50	00D90	00DD0
					00511	00551	00591	005D1	00D11	00D51	00D91	00DD1
A2					00610	00650	00690	006D0	00E10	00E50	00E90	00ED0
					00611	00651	00691	006D1	00E11	00E51	00E91	00ED1
A3					00710	00750	00790	007D0	00F10	00F50	00F90	00FD0
					00711	00751	00791	007D1	00F11	00F51	00F91	00FD1
B0	20010	20050	20090	200D0					20C10	20C50	20C90	20CD0
	20011	20051	20091	200D1					20C11	20C51	20C91	20CD1
B1	20110	20150	20190	201D0					20D10	20D50	20D90	20DD0
	20111	20151	20191	201D1					20D11	20D51	20D91	20DD1
B2	20210	20250	20290	202D0					20E10	20E50	20E90	20ED0
	20211	20251	20291	202D1					20E11	20E51	20E91	20ED1
B3	20310	20350	20390	203D0					20F10	20F50	20F90	20FD0
	20311	20351	20391	203D1					20F11	20F51	20F91	20FD1
N0	60010	60050	60090	600D0	60410	60450	60490	604D0				
	60011	60051	60091	600D1	60411	60451	60491	604D1				
N1	60110	60150	60190	601D0	60510	60550	60590	605D0				
	60111	60151	60191	601D1	60511	60551	60591	605D1				
N2	60210	60250	60290	602D0	60610	60650	60690	606D0				

Table continues on the next page...

Table 7-80. Mode values for PKHA copy memory by segment functions (continued)

Source Quadrant	Destination Quadrant											
	A0	A1	A2	A3	B0	B1	B2	B3	N0	N1	N2	N3
	60211	60251	60291	602D1	60611	60651	60691	606D1				
N3	60310	60350	60390	603D0	60710	60750	60790	607D0				
	60311	60351	60391	603D1	60711	60751	60791	607D1				

7.18.4 PKHA OPERATION: Elliptic Curve Functions

NOTE

The elliptic curve functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

Table 7-81. PKHA Mode Register Format for Elliptic Curve Functions

19	18	17	16	11	10	9-8	7-6	5-0
Reserved	Reserved	F2m	R2	Reserved Coord	Teq	OutSel	Reserved	Function
PKHA_MODE_MS				PKHA_MODE_LS				

Table 7-82. PKHA Mode register, format for elliptic curve operation

Bits	Description
17 F2m	F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function. If F2m=0 : Integer (prime) curve If F2m=1 : Binary polynomial curve. (Not valid for all curve types.)
16 R2	(R2 mod N). Indicates whether the term (R2 mod N) must be supplied as an input or will be calculated by the routine. If R2=0 : (R ² mod N) is calculated and applied, if needed If R2=1 : (R ² mod N) is an input. (Not valid for all functions.)
11 Reserved	Reserved
10 Teq	Timing Equalized. Indicates that a timing equalized version of the function should be executed. If Teq=0 : No timing equalization If Teq=1 : Timing equalization. (Not valid for all functions.)
9-8 OutSel	Output destination select. Indicates which memory should contain the output of the selected function. If OutSel=00b : B

Table continues on the next page...

Table 7-82. PKHA Mode register, format for elliptic curve operation (continued)

Bits	Description
	If OutSel=01b : A If OutSel=10b : Reserved If OutSel=11b : Reserved
7-6	Reserved
5-0 Function	Function. Indicates which elliptic curve function to execute. If Function=001001b : ECC Point Add (P1 + P2) If Function=001010b : ECC Point Double (P2 + P2) If Function=001011b : ECC Point Multiply (E x P1) if Function=011100b : ECC Check Point All other values for this field are currently reserved or are Table 7-70 , Table 7-77 , or Table 7-72 .

Elliptic Curve Functions over a prime field (ECC_MOD), where prime $p > 3$. For a general discussion, see [ECC_MOD: Point math on a standard curve over a prime field \(\$F_p\$ \)](#).

Table 7-83. List of mode values for Prime Field (F_p) Elliptic Curve Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
ECC_MOD_ADD	ECC prime field point add - affine coordinates	B	0	00009	ECC F_p Point Add, Affine Coordinates (ECC_MOD_ADD) Function
		A	0	00109	
ECC_MOD_ADD_R2	ECC prime field point add - affine coordinates, R2 input	B	0	10009	ECC F_p Point Add, Affine Coordinates, R ² Mod N Input (ECC_MOD_ADD_R2) Function
		A	0	10109	
ECC_MOD_DBL	ECC prime field point double - affine coordinates	B	0	0000A	ECC F_p Point Double, Affine Coordinates (ECC_MOD_DBL) Function
		A	0	0010A	
ECC_MOD_MUL ECC_MOD_MUL_TEQ	ECC prime field point multiply - affine coordinates	B	0	0000B	ECC F_p Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function
		A	0	0010B	
	Timing equalized version of ECC prime field point multiply - affine coordinates	B	1	0040B	
		A	1	0050B	
ECC_MOD_MUL_R2	ECC prime field point multiply - affine coordinates, r2 mod n input	B	0	1000B	ECC F_p Point Multiply, R ² Mod N Input, Affine
		A	0	1010B	

Table continues on the next page...

Table 7-83. List of mode values for Prime Field (F_p) Elliptic Curve Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
ECC_MOD_MUL_R2_TEQ	Timing equalized version of ECC prime field point multiply - affine coordinates, r2 mod n input	B	1	1040B	Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function
		A	1	1050B	
ECC_MOD_CHECK_POINT	ECC Prime Field Point Validation	-	0	0001C	ECC F_p Check Point (ECC_MOD_CHECK_POINT) Function
ECC_MOD_CHECK_POINT_R2	ECC Prime Field Point Validation, R2 input	-	0	1001C	ECC F_p Check Point, R^2 Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

Elliptic Curve Functions over a binary field (ECC_F2M). All operate in polynomial basis. For a general discussion, see [ECC_F2M: Point math on a standard curve over a binary field \(\$F_{2^m}\$ \)](#).

Table 7-84. List of mode values for Binary Field (F_{2^m}) Elliptic Curve Arithmetic Functions

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
ECC_F2M_ADD	ECC binary field point add - affine coordinates	B	0	20009	ECC F_{2^m} Point Add, Affine Coordinates (ECC_F2M_ADD) Function
		A	0	20109	
ECC_F2M_ADD_R2	ECC binary field point add - affine coordinates, R2 input	B	0	30009	ECC F_{2^m} Point Add, Affine Coordinates, R^2 Mod N Input (ECC_F2M_ADD_R2) Function
		A	0	30109	
ECC_F2M_DBL	ECC binary field point double - affine coordinates	B	0	2000A	ECC F_{2^m} Point Double - Affine Coordinates (ECC_F2M_DBL) Function
		A	0	2010A	
ECC_F2M_MUL ECC_F2M_MUL_TEQ	ECC binary field point multiply - affine coordinates	B	0	2000B	ECC F_{2^m} Point Multiply, Affine Coordinates (ECC_F2M_MUL
		A	0	2010B	

Table continues on the next page...

Table 7-84. List of mode values for Binary Field (F_{2^m}) Elliptic Curve Arithmetic Functions (continued)

Function name	Brief description	Output reg	Teq	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)	Detailed description
	Timing equalized version of ECC binary field point multiply - affine coordinates	B	1	2040B	and ECC_F2M_MUL_TEQ) Function
		A	1	2050B	
ECC_F2M_MUL_R2	ECC binary field point multiply - affine coordinates, r2 mod n input	B	0	3000B	ECC F_{2^m} Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function
		A	0	3010B	
ECC_F2M_MUL_R2_TEQ	Timing equalized version of ECC binary field point multiply - affine coordinates, r2 mod n input	B	1	3040B	
		A	1	3050B	
ECC_F2M_CHECK_POINT	ECC Binary Polynomial Point Validation	-	0	A001C	ECC F_{2^m} Check Point (ECC_F2M_CHECK_POINT) Function
ECC_F2M_CHECK_POINT_R2	ECC Binary Polynomial Field Point Validation, R2 input	-	0	B001C	ECC F_{2^m} Check Point, R^2 (ECC_F2M_CHECK_POINT_R2) Function

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

7.19 SIGNATURE command

Trusted descriptors end with a SIGNATURE command, which requires the descriptor's signature (HMAC) to be validated before allowing it to run. SIGNATURE commands also support regeneration of the signature if the trusted descriptor modifies itself.

Trusted descriptors can be created and signed with a signature (a keyed hash) when executed from a specially privileged Job Ring. (See [Trusted descriptors](#).) Trusted descriptors can be used to integrity protect the descriptor and to bind a key to a descriptor.

The SIGNATURE command that generates and verifies the keyed hash is always the last command of a trusted descriptor, although additional SIGNATURE commands can appear within the descriptor. The signature (HMAC) immediately follows the last SIGNATURE command in the trusted descriptor. When the descriptor is created:

SIGNATURE command

- Room must be left at the end of the buffer for the 32-byte signature
- The length of the descriptor must include the signature.

DECO does not read the signature when creating the signature, so any initial value can be placed there.

If a trusted descriptor has a shared descriptor, the shared descriptor is part of the keyed hash computation. The shared descriptor is hashed first, followed by the descriptor; this is the order in which they appear in the descriptor buffer. The final hash is the value computed for both.

NOTE

It is an error for a SIGNATURE command to be in a descriptor that is not trusted or being made trusted.

NOTE

Because the SIGNATURE command must be the last command executed in the descriptor, trusted descriptors cannot have the REO bit set in their header. Doing so results in an error.

SIGNATURE types are available that allow a portion of the following command to not be included in the keyed hash. This provides flexibility in changing the address or the immediate data specified by a command. For example, the following command may a LOAD command, which contains the command word itself followed by a pointer. These SIGNATURE types would allow the command word to be part of the keyed hash but would exclude the pointer from the calculation. The writer of the trusted descriptor is responsible for using these SIGNATURE types only when the skipped information does not need to be integrity protected, meaning any immediate data or any address is permissible.

NOTE

Skipping the signature over immediate data would allow a malicious user to shorten the length of the immediate data and insert additional commands that would not be included in the keyed hash. Note that this could be done without altering the overall length of the descriptor. To prevent this, it is recommended that the first four bytes of an immediate command always be protected by the keyed hash. Because the length of the immediate data is included in the keyed hash, the length cannot be altered such that additional commands can substitute for a portion of the immediate data.

Table 7-85. SIGNATURE command format

31–27	26–20	19–16
CTYPE = 10010	Reserved	TYPE
15–0		
Reserved		
<i>Additional words of SIGNATURE command</i>		
8 Words to hold the Signature (these are used in types 0000, 0001 and 0010 only)		

Table 7-86. SIGNATURE command field descriptions

Field	Description
31-27 CTYPE	Command type IF CTYPE=10010b : Signature command
26-20	Reserved
19-16 TYPE	See Table 7-87
15-0	Reserved

Table 7-87. TYPE field description

Type	Meaning	Instructions
0000	SIGNATURE command types 0000, 0001, or 0010 must be the last command that is executed in a trusted descriptor. If one of these types is used, the trusted descriptor signature (the keyed hash value) immediately follows the command. It is an error for a SIGNATURE command with one of these types to appear anywhere other than at the end of the descriptor.	Type 0000, when executed, terminates execution of the descriptor normally.
0001		Type 0001 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution. This type is used in cases where the descriptor could modify itself during execution. Note that the rehash and update is always done whether the descriptor was modified or not. Following the rehash and update, descriptor execution terminates normally.
0010		Type 0010 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution if, upon completion, the MATH_Z bit is set. This type is used in cases where the descriptor could modify itself during execution but updating the keyed hash should be conditional. This version allows the rehash and update to be skipped when no change has been made to the descriptor. If MATH_Z is 0, descriptor execution immediately terminates normally. Otherwise, descriptor execution terminates normally after the rehash and update.
1010	SIGNATURE command types 1010, 1011, and 1100 are used to include only a portion of the following command in the keyed hash calculation, omitting the remainder of the command from the calculation. There is no hash value associated with this type, so it is an error for this type to	Type 1010 instructs SEC to hash only the first 2 bytes of the next command.
1011		Type 1011 instructs SEC to hash only the first 3 bytes of the next command.

Table continues on the next page...

Table 7-87. TYPE field description (continued)

Type	Meaning	Instructions
1100	appear at the end of the descriptor. These types allow the trusted descriptor to be modified with other offsets, addresses and lengths without invalidating the signature. Note that the SIGNATURE command is, itself, included in the hash so that it cannot be added later.	Type 1100 instructs SEC to hash only the first 4 bytes of the next command.
Others	Reserved	

Two types of the final SIGNATURE command, 0001 and 0010 described in the above table, will recompute and update the signature in memory. These types are used when the trusted descriptor modifies itself and the modified version is to be used thereafter. Note that it is up to the descriptor writer to ensure that the copy of the descriptor in memory is updated using a STORE command. However, this update should only be done once all other commands in the descriptor have completed successfully. That is, the penultimate command should be the STORE to update the descriptor and the final command must be the SIGNATURE command. (If the update was done earlier and an error was detected prior to the SIGNATURE command running, the trusted descriptor could never be run again since the signature won't match.) The final signature command will wait to run until all reads have completed, all write data has been taken by the DMA, and all internal moves have completed. However, the final signature command is not a Done Checkpoint, which means that it will not wait for CHAs to complete.

7.20 JUMP (HALT) command

The JUMP command has the following uses:

- Alters the execution order of descriptor commands
- Pauses execution until specified conditions are satisfied
- Halts the execution of the descriptor if specified conditions are satisfied

[JUMP command format](#) shows the format of the JUMP command, and [Table 7-89](#) describes the JUMP command field definitions.

The JUMP command may or may not be a checkpoint depending on its conditions and type.

7.20.1 Jump type

The JUMP command has eight different types, distinguished by the value in the JUMP TYPE field. All of these types specify a tested condition and take some execution flow action if the tested condition evaluates as true, and simply continue with the next command if the tested condition is false. See [Test type](#) for an explanation of what it means for the tested condition to be true.

Four of these jump types are true conditional jumps, another two are conditional halts, and the last two are a conditional subroutine call and a conditional subroutine return. Regardless of the jump type, the execution of the command waits for any specified wait conditions to be satisfied before the conditional action (jump, call, return, or halt) is taken. Some wait conditions can be specified in the CLASS field (wait for the Class 1 CHA to be done, wait for the Class 2 CHA to be done, or both), and additional wait conditions can be specified with the TEST CONDITION field (if JSL = 1).

7.20.1.1 Local conditional jump

The local conditional jump works as follows:

- If the tested condition is true, a JUMP command of the local conditional jump type continues the execution sequence at a new point within the descriptor buffer.
- If the tested condition is false, the jump is not taken and execution continues with the command that follows the JUMP command.

Local jumps are relative. The LOCAL OFFSET field is treated as an 8-bit 2's complement number that is added to the position of the JUMP command within the descriptor buffer. For example, a jump of one goes to the next 32-bit word and a jump of two skips one 32-bit word. Backward jumps are performed using 2's complement negative numbers.

A LOCAL OFFSET of 0 is a shorthand means of jumping back to the start of the descriptor buffer, which is either the start of the job descriptor if there is no shared descriptor or the start of the shared descriptor, if there is one. (see [Figure 7-3](#))

7.20.1.2 Local conditional increment/decrement jump

The local conditional increment/decrement jump is simply a local conditional jump that either increments or decrements a specified register, updates the math conditions based upon the result, and then evaluates the selected math conditions to determine whether or not the jump should be taken:

- If the tested condition is satisfied, command execution continues at a new point within the descriptor buffer.
- If the tested condition is not satisfied, the jump is not taken and execution continues with the command that follows the JUMP command.

Note that the increment and decrement jump types use a different JUMP command format than the other jump types. The four most-significant bits of the TEST CONDITION field are replaced with a SRC_DST field that specifies the register that is to be incremented or decremented. The least significant four bits of the TEST CONDITION field constitute the MATH CONDITION field, which specifies the tested conditions that are evaluated to determine whether the jump is taken or not.

Any of the legal choices for the SRC0 field of the MATH command which are also legal choices for the DEST field of the MATH command may be selected as the register to increment or decrement. Use the same value to select the register as is used for the MATH command.

7.20.1.3 Non-local conditional jump

The non-local conditional jump is just like the local conditional jump except that the target of the jump must be the header of a job descriptor or trusted descriptor. Note that the target descriptor may not be a shared descriptor nor may the target descriptor have a shared descriptor. The pointer to the target descriptor is in the one or two words following the JUMP command.

- If the tested condition evaluates to true, the jump is taken.
- If the tested condition evaluates to false, the jump is not taken and execution continues with the command following the pointer.

NOTE

It is permissible to JUMP from a job descriptor to another job descriptor or from a job descriptor or a trusted descriptor to another trusted descriptor, but jumping from a trusted descriptor to a job descriptor results in an error.

7.20.1.4 Conditional halt

This JUMP command is actually a conditional halt, meaning it stops the execution of the current descriptor if the tested condition evaluates to true. In this case the PKHA/Math condition bits (see the "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field in [JUMP command format](#)) are written out right-justified in the SSED field of the job termination status word (see [Job termination status/error codes](#)).

If the tested condition evaluates to false, the descriptor is not halted and execution instead continues with the command that follows the jump.

NOTE

If the specified conditions evaluate as true, this command will always result in a nonzero status being returned for this job. Therefore, such a job will always appear to have encountered an error. The 8-bit error code will, as described above, be the PKHA and Math status flags rather than one of the predefined error codes.

7.20.1.5 Conditional halt with user-specified status

A JUMP command with the user-specified status option is another type of conditional halt. If the tested condition is true, it stops execution of the descriptor but instead of writing the PKHA/Math condition bits, this conditional halt writes out the value in the LOCAL OFFSET field (again, right-justified in the SSED field of the job termination status word). The interpretation of the code in the LOCAL OFFSET field is user-specified, so it could be used during debugging to indicate that execution reached a certain point in a particular descriptor. If the tested condition evaluates to false, execution continues with the command following the jump.

NOTE

If the specified conditions evaluate as true, and the LOCAL OFFSET field is nonzero, this command will result in a nonzero status being returned for this job. That is, it will appear that such a job encountered an error. The 8-bit error code will, as described above, be a copy of the LOCAL OFFSET field rather than one of the predefined error codes.

NOTE

If the specified conditions evaluate as true, and the LOCAL OFFSET is zero, this command will terminate execution of the descriptor with normal status. That is, it will appear that such a job terminated normally. This is a convenient way to terminate

execution in the middle of a descriptor when it can be determined that all work is done rather than having to jump to the end of the descriptor.

7.20.1.6 Conditional subroutine call

A JUMP command with the subroutine call option is another type of local conditional jump. If the tested condition is true, it jumps to the specified location in the descriptor buffer but also saves the return address. The return address is the location immediately following the JUMP command. If the tested condition evaluates to false, execution continues with the command following the JUMP.

Note that only one return address can be saved, so subroutine calls cannot be nested. The descriptor writer is responsible for enforcing this as no error will be thrown if subroutine calls are nested.

NOTE

A built-in protocol is, in fact, also a special subroutine call. The return address is used to note where execution should resume following the execution of the built-in protocol. Therefore, while a protocol may be called from within a subroutine, a subsequent subroutine return will return to the command following the protocol command rather than the command following the subroutine call.

Each time a conditional subroutine call is taken or a built-in protocol is started, the return address is saved. That return address will be maintained until it is overwritten by another conditional subroutine call or built-in protocol. Therefore, it is possible to have one subroutine call which corresponds to multiple subroutine returns. It is also possible to match subroutine returns with calls to built-in protocols.

7.20.1.7 Conditional subroutine return

A JUMP command with the subroutine return option is another type of local conditional jump. In this case the local offset is ignored because the target of the jump is taken from the previously saved return address. If the tested condition is true, the subroutine return jumps to the saved return address. This address is the location immediately following the most recently executed command which updated the return address. One command which updates the return address is a conditional subroutine call in which the tested condition

evaluated as true. The other command which updates the return address is a built-in protocol. If the tested condition evaluates to false, execution continues with the command following the subroutine return command.

NOTE

See the previous section, Conditional subroutine call, for important details on the use of the conditional subroutine return.

7.20.2 Test type

The TEST TYPE field is used to specify when the conditional jump/halt tested condition is considered to be met. The test type options are:

- 00—All specified test conditions are true. (Logical AND of all conditions.)
- 01—All specified test conditions are false. (Logical NOR of all conditions.)
- 10—Any specified test condition is true. (Logical OR of all conditions.)
- 11—Any specified test condition is false. (Logical NAND of all conditions.)

To create an unconditional jump, use TEST TYPE = 00 (all specified conditions true) and clear all TEST CONDITION bits because the tested condition is considered to be true if no test condition bits are set.

To create an unconditional jump/halt with a JSL = 1 conditional wait condition, use TEST TYPE = 10 (any specified condition is true). This always jumps or halts once the wait is completed because the selected conditional wait condition(s) are always true after the wait is completed.

A local conditional jump with offset 1 (signifying a jump to the following command) is a no-op because the next command in sequence is executed whether or not the jump is taken. This is true regardless of the TEST TYPE and TEST CONDITION settings. However, a wait condition can be specified to prevent the next command from executing until the conditions are satisfied. This is a common use case for the local conditional jump.

7.20.3 JSL and TEST CONDITION fields

The JSL field selects between two different interpretations of the TEST CONDITION field:

JUMP (HALT) command

- When $JSL = 0$, the conditional jump/halt bits select various MATH and PKHA status conditions. These are used to jump or halt if the tested condition is satisfied.
- When $JSL = 1$, the bits in the TEST CONDITION field can affect the action taken by the JUMP Command in two ways.
 - Some of the TEST CONDITION bits are conditional jump/halt bits. The JQP, SHRD, and SELF test conditions are typically used to avoid storing data that the next descriptor might change or to prevent reloading data that is already available because it was left by the previous descriptor.
 - The remainder of the TEST CONDITION bits are conditional wait bits. The CALM, NIP, NIFP, NOP, and NCP conditional wait bits are used to time loads, moves, and stores properly. If conditional wait bits are set the JUMP command stalls until all of the specified wait conditions become true. All of the conditional wait bits must evaluate to true independent of the TEST TYPE specified. In other words, you can't wait for one of two conditional wait conditions to become true; you must wait for both. Once all the conditional wait conditions are true, the jump or halt either occurs or not, depending upon whether all of the specified conditions are satisfied. Note that once the wait has completed, the selected conditional wait conditions are always true; because they are evaluated as part of the tested condition, they can affect whether the jump or halt action is taken. Note that the CLASS bits are, in fact, conditional wait bits even though they are not used in the decision on whether to take the JUMP.

For example, if a JUMP command is executed with $JSL = 1$ and the TEST CONDITION bits NIP, NIFP, JQP, and SELF are set, the JUMP command stalls until both of the following are true:

- No input to the input data FIFO is pending (NIP).
- No input to the information FIFO is pending (NIFP).

Because these are conditional wait bits, the command waits until all of the wait conditions are true before evaluating the remaining conditions. The evaluation depends upon the test conditions that are selected, the state of the selected conditions, and the value in the TEST TYPE field:

- TEST TYPE = 00 (if all conditions are true): the jump or halt occurs if another job wants to share this shared descriptor (JQP) and this shared descriptor is running in the same DECO (SELF) as the one from which it was shared.
- TEST TYPE = 01 (if all conditions are false): the jump or halt never occurs because the NIP and NIFP conditions are true after the wait completes.

- TEST TYPE = 10 (if any condition is true): the jump or halt always occurs because the NIP and NIFP conditions are true after the wait completes.
- TEST TYPE = 11 (if any condition is false): the jump or halt occurs if no job wants to share this shared descriptor (JQP) or this shared descriptor is not running in the same DECO (SELF) as the one from which it was shared.

7.20.4 JUMP command format

Table 7-88. JUMP command format

	31-27	26-25	24	23-20	19-18	17-16
	CTYPE = 10100	CLASS	JSL	JUMP TYPE	Reserved	TEST TYPE
	15-8			7-0		
Format used with all jump types except 0001 and 0011	TEST CONDITION			LOCAL OFFSET		
	15-12	11-8		7-0		
Format used with jump types 0001 and 0011	SRC_DST	MATH CONDITION		LOCAL OFFSET		
	<i>Additional words of JUMP command</i>					
	Pointer (one or two words); see Address pointers (this field is present for non-local JUMPs only)					

Table 7-89. JUMP command field descriptions

Field	Description
31-27 CTYPE	Command type If CTYPE=10100 : JUMP command
26-25 CLASS	Class Wait until specified class type CHA(s) is done before evaluating jump/halt conditions. For CLASS != 00, this makes the JUMP command a DONE checkpoint. If CLASS=00 : None If CLASS=01 : Class 1 If CLASS=10 : Class 2 If CLASS=11 : Both Class 1 and Class 2
24 JSL	Jump Select Type Selects which definition of the TEST CONDITION field to use. If JSL=0 : MATH and PKHA status conditions If JSL=1 : Various jump/halt and wait conditions (Note that JSL=1 is prohibited with jump types 0001 and 0011 and such usage will result in an error.)
23-20 JUMP TYPE	Jump Type Specifies the action taken by the JUMP Command. See Jump type for more information.

Table continues on the next page...

Table 7-89. JUMP command field descriptions (continued)

Field	Description
	<p>If JUMP TYPE=0000 : Local conditional jump. Evaluates the specified TEST CONDITION to determine whether the local jump should be taken.</p> <p>If JUMP TYPE=0001 : Local conditional increment jump. Increments the register specified in SRC_DST before evaluating the specified MATH CONDITION.</p> <p>If JUMP TYPE=0010 : Conditional subroutine call. Evaluates the specified TEST CONDITION to determine whether the local subroutine call should be taken.</p> <p>If JUMP TYPE=0011 : Local conditional decrement jump. Decrements the register specified in SRC_DST before evaluating the specified MATH CONDITION.</p> <p>If JUMP TYPE=0100 : Non-local conditional jump. Evaluates the specified TEST CONDITION to determine whether the non-local jump should be taken.</p> <p>If JUMP TYPE=0110 : Conditional subroutine return. Evaluates the specified TEST CONDITION to determine whether the subroutine return should be taken.</p> <p>If JUMP TYPE=1000 : Conditional Halt. If the specified TEST CONDITION is true, this returns the PKHA/MATH bits as status. (see "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field) and halts descriptor execution with error status.</p> <p>If JUMP TYPE=1100 : Conditional Halt with user-specified status. If the specified TEST CONDITION is true, this returns the value in the LOCAL OFFSET field as status and halts descriptor execution with error status unless the LOCAL OFFSET is zero, in which case descriptor execution terminates normally.</p> <p>All other codes are reserved, and will generate an error.</p>
19-18	Reserved
17-16 TEST TYPE	<p>Test Type. This field defines how the condition code bits (see TEST CONDITION field) should be interpreted. See Test type for more information.</p> <p>If TEST TYPE=00 : Jump/halt if ALL selected conditions are true. That is, jump or halt if all the status conditions are true for all TEST CONDITION bits that are 1. Note that if JSL = 1 and one or more conditional wait bits is set, the command waits for all selected conditional wait conditions to be true before the conditional jump/halt conditions are evaluated. The jump/halt then takes place if these conditions are all true.</p> <p>If TEST TYPE=01 : Jump/halt if ALL selected conditions are false. That is, jump or halt if all the status conditions are false for all TEST CONDITION bits that are 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and the jump/halt will not take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if all of the selected Conditional Jump/Halt conditions are false.</p> <p>If TEST TYPE=10 : Jump/halt if ANY selected condition is true. That is, jump or halt if any status condition is true for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then the jump/halt will take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if any of the selected Conditional Jump/Halt conditions are true.</p> <p>If TEST TYPE=11 : Jump/halt if ANY selected condition is false. That is, jump or halt if any status condition is false for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then Tested Condition will be evaluated. Whether a wait occurs or not, the jump/halt will take place if any selected Conditional Jump/Halt condition is false.</p>
15-8 TEST CONDITION	<p>Test Condition. This 8-bit field is used with all jump types except 0001 and 0011. The interpretation of the TEST CONDITION field depends upon the value of the JSL field, a shown in Table 7-90. See JSL and TEST CONDITION fields for more information.</p>
15-12 SRC_DST	<p>Source/Destination. This four-bit field is used only with jump types 0001 and 0011. It replaces the most-significant four bits of the TEST CONDITION field. This field is used to select the register that will be incremented (jump type 0001) or decremented (jump type 0011) before the selected math condition is</p>

Table continues on the next page...

Table 7-89. JUMP command field descriptions (continued)

Field	Description												
	<p>evaluated to determine whether the local jump will be taken. For 8-byte registers, only the least-significant 4 bytes are used. (That is, the length of the math operation is restricted to 4 bytes.)</p> <p>If SRC_DST=0000 : Math Register 0</p> <p>If SRC_DST=0001 : Math Register 1</p> <p>If SRC_DST=0010 : Math Register 2</p> <p>If SRC_DST=0011 : Math Register 3</p> <p>If SRC_DST=0101 : Math Register 4</p> <p>If SRC_DST=0110 : Math Register 5</p> <p>If SRC_DST=0111 : DECO Protocol Override Register</p> <p>If SRC_DST=1000 : Sequence In Length (SIL)</p> <p>If SRC_DST=1001 : Sequence Out Length (SOL)</p> <p>If SRC_DST=1010 : Variable Sequence In Length (VSIL)</p> <p>If SRC_DST=1011 : Variable Sequence Out Length (VSOL)</p> <p>If SRC_DST=1101 : Math Register 6</p> <p>If SRC_DST=1110 : Math Register 7</p> <p>All other values are reserved.</p>												
11-8 MATH CONDITION	<p>Math condition. This four-bit field is used only with jump types 0001 and 0011. It is identical with the least-significant four bits of the TEST CONDITION field and uses the same definitions. This field is used to select the math conditions that will be evaluated to determine whether the local jump should be taken.</p> <table border="1"> <thead> <tr> <th>bit 11</th> <th>bit 10</th> <th>bit 9</th> <th>bit 8</th> </tr> </thead> <tbody> <tr> <td>MATH N</td> <td>MATH Z</td> <td>MATH C</td> <td>MATH NV</td> </tr> <tr> <td>The result is negative.</td> <td>The result is zero.</td> <td>The operation resulted in a carry or borrow.</td> <td>Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.</td> </tr> </tbody> </table>	bit 11	bit 10	bit 9	bit 8	MATH N	MATH Z	MATH C	MATH NV	The result is negative.	The result is zero.	The operation resulted in a carry or borrow.	Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.
bit 11	bit 10	bit 9	bit 8										
MATH N	MATH Z	MATH C	MATH NV										
The result is negative.	The result is zero.	The operation resulted in a carry or borrow.	Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.										
7-0 LOCAL OFFSET	<p>For local jumps this field specifies the offset of the JUMP target from the JUMP command's address in the descriptor buffer. This field is ignored for non-local JUMPs. If the LOCAL OFFSET is 0, the target is the start of the Descriptor Buffer. For non-zero values, the target address is relative to the JUMP Command. That is, the field is interpreted as an 8-bit 2's complement number that is added to the index of the JUMP Command to yield the 32-bit word of the target. For Halt with status, the LOCAL OFFSET will be returned in Descriptor status. This will show up in the Output Job Status as the USTA field. If nonzero on halt with status, an error is reported.</p>												
31-0 POINTER	<p>Pointer (32-bit or 64-bit). This field is present only for non-local jumps. This is the address of the Descriptor to which to jump if the jump is taken.</p>												

Table 7-90. TEST CONDITION bit settings

Bit #	TEST CONDITION bits when JSL=0			TEST CONDITION bits when JSL=1		
15	PKHA IS_ZERO	For Finite Field operations the result of a PKHA operation is zero. For ECC operations, the result is a Point at Infinity.	Conditional Jump/Halt	JQP	Job Queue Pending. The Job Queue Controller has identified that another job wants to share this Shared Descriptor. This bit can be used to avoid storing data that the next Shared Descriptor would just refetch. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
14	PKHA GCD_1	The greatest common divisor of two numbers is 1 (that is, the two numbers are relatively prime).	Conditional Jump/Halt	SHRD	SHARED. This Shared Descriptor was shared from a previously executed Descriptor. Depending on the type of sharing, this bit can be tested to conditionally jump over commands. For example, if the keys are shared they will already be in the Key Registers so decrypting and placing them in the Key Registers must be skipped. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
13	PKHA IS_PRIME	The given number is probably prime (that is, it passes the Miller-Rabin primality test).	Conditional Jump/Halt	SELF	The SELF bit indicates that this Shared Descriptor is running in the same DECO as the one from which it was shared. Hence, the Shared Descriptor may be able to assume that Context Registers, CHAs, and other items are still valid or available. This condition is false if this is not a Shared Descriptor.	Conditional Jump/Halt
12	Reserved	Must be 0.	—	CALM	All pending bus transactions for this DECO, whether internal or external, have completed.	Conditional Wait
11	MATH N	The negative math flag is set.	Conditional Jump/Halt	NIP	No input pending. No external loads, whether from LOAD, FIFO LOAD, SEQ LOAD, or SEQ FIFO LOAD, are pending.	Conditional Wait
10	MATH Z	The zero math flag is set.	Conditional Jump/Halt	NIFP	No iNformation FIFO entries pending. The NFIFO is empty and no data is waiting in the C1 or C2 alignment blocks.	Conditional Wait
9	MATH C	The carry/borrow math flag is set.	Conditional Jump/Halt	NOP	No output pending. No external stores, whether from STORE, FIFO STORE, SEQ STORE, or SEQ FIFO STORE, are pending.	Conditional Wait
8	MATH NV	The NV math flag is set. This is the XOR of the sign bit and 2's complement overflow.	Conditional Jump/Halt	NCP	No context load pending. There is no data in flight toward the context registers via the internal or external DMA.	Conditional Wait

7.21 MATH and MATHI Commands

The MATH and MATHI commands compute simple mathematical functions of values in registers or specified via immediate data. The result can be written to a specified destination register or the result can be dropped. The commands set MATH condition bits that reflect the result of the mathematical operation (see MNV, MN, MC, and MZ). These condition bits can be tested with the JUMP commands, providing SEC with the flexibility to implement conditional processing constructs, including loops. In addition, the MC bit can be used to perform addition and subtraction of values larger than the math registers via borrow or carry.

Length must always be specified in the command, as it determines the size of the arguments used to set the MATH status bits. Note that the LENGTH field is used to mask off results after the math operation, not before, so the user must present properly sized data.

The MATHI command is useful when a one-byte immediate value is to be used. Since this immediate value is contained within the MATHI command word, this allows the MATHI command to be a single word rather than forcing the use of a two-word MATH command. This is useful since one-byte arguments are common. In some large descriptors, saving this one word several times can make the difference between fitting in the descriptor buffer and having to use multiple descriptors.

Table 7-91. MATH and MATHI Commands, format

	31–27	26	25	24	23–20	19–16
MATH:	CTYPE = 10101	IFB	NFU	STL	FUNCTION	SRC0
MATHI:	CTYPE = 11101	Reser ved	NFU	SSEL	FUNCTION	SRC
	15–12	11–8		7–4		3–0
MATH:	SRC1	DEST		Reserved		LEN
MATHI:	DEST	IMM_VALUE				LEN

Table 7-92. MATH command, field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=10101b : MATH command (fields defined in this table) If CTYPE=11101b : MATHI command (fields defined in table Table 7-93 below)
26 IFB	Immediate Four Bytes If IFB=0 : Include full length immediate data in descriptor (length specified in LEN field) If IFB=1 : Use only four bytes of immediate data even if LEN is 8. This shortens the descriptor by one word when 1, 2, or 4-byte immediate data is to be used in an 8-byte operation. The immediate data will automatically be zero padded out to 8 bytes. This bit has no effect if the LEN is less than 8.

Table continues on the next page...

Table 7-92. MATH command, field descriptions (continued)

Field	Description																																		
25 NFU	No Flag Update If NFU=0 : Math flags will be updated as appropriate. If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags.																																		
24 STL	Stall. If STL=0 : Don't stall the execution of the MATH command. If STL=1 : Stall MATH command. Causes the MATH command to take one extra clock cycle.																																		
23-20 FUNCTION	This field specifies which function to perform, as listed in the table below titled Table 7-94 . The operands are specified in the SRC0 and SRC1 fields and the result is written to the destination specified in the DEST field.																																		
19-16 SRC0	The SRC0 field indicates the source of operand 0. <table border="1"> <thead> <tr> <th>Source of Operand 0</th> <th>SRC0 Field Value</th> </tr> </thead> <tbody> <tr><td>Math Register 0</td><td>0h</td></tr> <tr><td>Math Register 1</td><td>1h</td></tr> <tr><td>Math Register 2</td><td>2h</td></tr> <tr><td>Math Register 3</td><td>3h</td></tr> <tr><td>Math Register 4</td><td>5h</td></tr> <tr><td>Math Register 5</td><td>6h</td></tr> <tr><td>Math Register 6</td><td>Dh</td></tr> <tr><td>Math Register 7</td><td>Eh</td></tr> <tr><td>Immediate data from descriptor words following the MATH command¹</td><td>4h</td></tr> <tr><td>Protocol Override (DPOVRD), left-extended with 0s</td><td>7h</td></tr> <tr><td>Sequence In Length (SIL), left-extended with 0s</td><td>8h</td></tr> <tr><td>Sequence Out Length (SOL), left-extended with 0s</td><td>9h</td></tr> <tr><td>Variable Sequence In Length (VSIL)</td><td>Ah</td></tr> <tr><td>Variable Sequence Out Length (VSOL)</td><td>Bh</td></tr> <tr><td>ZERO (the value 0000 0000h) is used as operand 0</td><td>Ch</td></tr> <tr><td>ONE (the value 0000 00001) is used as operand 0</td><td>Fh</td></tr> </tbody> </table>	Source of Operand 0	SRC0 Field Value	Math Register 0	0h	Math Register 1	1h	Math Register 2	2h	Math Register 3	3h	Math Register 4	5h	Math Register 5	6h	Math Register 6	Dh	Math Register 7	Eh	Immediate data from descriptor words following the MATH command ¹	4h	Protocol Override (DPOVRD), left-extended with 0s	7h	Sequence In Length (SIL), left-extended with 0s	8h	Sequence Out Length (SOL), left-extended with 0s	9h	Variable Sequence In Length (VSIL)	Ah	Variable Sequence Out Length (VSOL)	Bh	ZERO (the value 0000 0000h) is used as operand 0	Ch	ONE (the value 0000 00001) is used as operand 0	Fh
Source of Operand 0	SRC0 Field Value																																		
Math Register 0	0h																																		
Math Register 1	1h																																		
Math Register 2	2h																																		
Math Register 3	3h																																		
Math Register 4	5h																																		
Math Register 5	6h																																		
Math Register 6	Dh																																		
Math Register 7	Eh																																		
Immediate data from descriptor words following the MATH command ¹	4h																																		
Protocol Override (DPOVRD), left-extended with 0s	7h																																		
Sequence In Length (SIL), left-extended with 0s	8h																																		
Sequence Out Length (SOL), left-extended with 0s	9h																																		
Variable Sequence In Length (VSIL)	Ah																																		
Variable Sequence Out Length (VSOL)	Bh																																		
ZERO (the value 0000 0000h) is used as operand 0	Ch																																		
ONE (the value 0000 00001) is used as operand 0	Fh																																		
15-12 SRC1	The SRC1 field indicates the source of operand 1. <table border="1"> <thead> <tr> <th>Source of Operand 1</th> <th>SRC1 Field Value</th> </tr> </thead> <tbody> <tr><td>Math Register 0</td><td>0h</td></tr> <tr><td>Math Register 1</td><td>1h</td></tr> <tr><td>Math Register 2</td><td>2h</td></tr> <tr><td>Math Register 3</td><td>3h</td></tr> <tr><td>Math Register 4</td><td>5h</td></tr> </tbody> </table>	Source of Operand 1	SRC1 Field Value	Math Register 0	0h	Math Register 1	1h	Math Register 2	2h	Math Register 3	3h	Math Register 4	5h																						
Source of Operand 1	SRC1 Field Value																																		
Math Register 0	0h																																		
Math Register 1	1h																																		
Math Register 2	2h																																		
Math Register 3	3h																																		
Math Register 4	5h																																		

Table continues on the next page...

Table 7-92. MATH command, field descriptions (continued)

Field	Description	
	Source of Operand 1	SRC1 Field Value
	Math Register 5	6h
	Math Register 6	Dh
	Math Register 7	Eh
	Immediate data from descriptor words following the MATH command ¹	4h
	Protocol Override (DPOVRD), left-extended with 0s	7h
	Variable Sequence In Length (VSIL)	8h
	Variable Sequence Out Length (VSOL)	9h
	Input Data FIFO ^{2,3}	Ah
	Output Data FIFO ^{3,4}	Bh
	ONE (the value 0000 0001h) is used as operand 1	Ch
	ZERO (the value 0000 0000h) is used as operand 1	Fh
11-8 DEST	The DEST field specifies the destination for the result of the command as follows:	
	Destination for MATH operation result	DEST Field Value
	Math Register 0	0h
	Math Register 1	1h
	Math Register 2	2h
	Math Register 3	3h
	Math Register 4	5h
	Math Register 5	6h
	Math Register 6	Dh
	Math Register 7	Eh
	Protocol Override	7h
	Sequence In Length	8h
	Sequence Out Length	9h
	Variable Sequence In Length	Ah
	Variable Sequence Out Length	Bh
	No Destination. The result should not be written anywhere. ⁵	Fh
		All other values for this field are reserved.
7-4	This field is reserved. All bits must be 0.	
3-0 LEN	LEN denotes the length, in bytes, of the operation and the immediate value, if there is one. 1h : 1 byte 2h : 2 bytes	

Table continues on the next page...

Table 7-92. MATH command, field descriptions (continued)

Field	Description
	4h : 4 bytes 8h : 8 bytes 9h : 8 bytes, with word swapping performed prior to use. ⁶ All other values are reserved. NOTE: If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h may yield unexpected results. Also note that the IFB bit in the command can be used to override the LEN field for the immediate value. When set, the IFB (Immediate Four Bytes) bit allows the MATH command to use a 1, 2, or 4-byte immediate value (0 padded to the left) in the descriptor even though it is doing an 8-byte operation. NOTE: If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD.
<p>1. If the data is less than 8 bytes, it is left-extended with 0s. If the data is less than 8 bytes it must be right-aligned. If SRC0 and SRC1 both specify Immediate data, the SRC0 data is in the first word following the MATH command and the SRC1 data is in the second word, and either the LEN field must be set to 4 bytes or the IFB field must be set to 1, else an error is generated.</p> <p>2. The input data FIFO is popped when the MATH command executes unless the function is shld (shift and load). Note that this means a final pop may have to be done if the data consumed by the shld is the end of the data. If this is the last data to be consumed by DECO, then it is not necessary to pop the data, because leaving it there is not a problem if the input data FIFO is reset. The input FIFO is not automatically reset between job descriptors with the same shared descriptor unless the CIF bit in the Shared Descriptor is set. The input data FIFO is always reset between jobs without, or with different, shared descriptors. Note that the descriptor must have already created an NFIFO entry to get data to the DECO alignment block, from which the MATH command will pop it.</p> <p>3. If SRC1 specifies either input data FIFO or output data FIFO, the MATH command does not execute until the corresponding FIFO has valid data. It is up to the user to ensure that a sufficient amount of data is present. The user must also realize that data comes out of the FIFOs left aligned. This means that if there are only five bytes, the data is in the left 5 bytes, not in the right 5 bytes, of the 8-byte source word.</p> <p>4. The output data FIFO is popped when the MATH command executes unless the function is shld (shift and load). Note that this means a final pop may have to be done if the data consumed by the shld is the end of the data. If this is the last data to be consumed by DECO, then it is not necessary to pop the data, because leaving it there is not a problem if the output data FIFO is reset. The output FIFO is always cleared between descriptors whether shared or not.</p> <p>5. No Destination is useful for setting flags when the actual result is not needed. An error will be generated if No Destination is selected when the FUNCTION is shift_l or shift_r.</p> <p>6. An error will be generated for LEN=9h if IFB=1 or if both or neither of the operands is Immediate.</p>	

Table 7-93. MATHI command, field descriptions

Field	Description
31-27	Command Type
CTYPE	If CTYPE=10101b : MATH command (see field definitions in table Table 7-92 above) If CTYPE=11101b : MATHI command (fields defined in this table)
26	Reserved. Must be 0.
25	No Flag Update
NFU	If NFU=0 : Math flags will be updated as appropriate. If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags.

Table continues on the next page...

Table 7-93. MATHI command, field descriptions (continued)

Field	Description															
24 SSEL	<p>SSEL. Source Select. Selects the type and order of the operands to the math function: operand 0 <function> operand 1 -> destination</p> <table border="1"> <thead> <tr> <th>SSEL value</th> <th>operand 0 specified by</th> <th>math function specified by</th> <th>operand 1 specified by</th> <th>destination specified by</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>SRC0 for MATH command</td> <td>FUNCTION</td> <td>IMM_VALUE</td> <td>DEST</td> </tr> <tr> <td>1</td> <td>IMM_VALUE</td> <td>FUNCTION</td> <td>SRC1 for MATH command</td> <td>DEST</td> </tr> </tbody> </table> <p>NOTE: If FUNCTION=Ah (Fbyt) it is illegal to set SSEL to 1.</p>	SSEL value	operand 0 specified by	math function specified by	operand 1 specified by	destination specified by	0	SRC0 for MATH command	FUNCTION	IMM_VALUE	DEST	1	IMM_VALUE	FUNCTION	SRC1 for MATH command	DEST
SSEL value	operand 0 specified by	math function specified by	operand 1 specified by	destination specified by												
0	SRC0 for MATH command	FUNCTION	IMM_VALUE	DEST												
1	IMM_VALUE	FUNCTION	SRC1 for MATH command	DEST												
23-20 FUNCTION	This field specifies which function to perform, as listed in the table below titled Table 7-94 . The operands are specified in the SRC field and the IMM_VALUE field, and the result is written to the destination specified in the DEST field.															
19-16 SRC	<p>The SRC field indicates the source of one of the operands. The SRC field has two definitions, selected via the SSEL field:</p> <ul style="list-style-type: none"> • If SSEL=0: the SRC field is defined the same as the MATH command's SRC0 field (see SRC0) except that IMM (4h) is not supported and will result in an error. • If SSEL=1: the SRC field is defined the same as the MATH command's SRC1 field (see SRC1) except that IMM (4h) is not supported and will result in an error. 															
15-12 DEST	The destination for the result of the math operation. The MATHI command DEST field is defined the same as the MATH command's DEST field (see DEST), but is shifted to the left 4 bits to make room for the IMM_VALUE field.															
11-4 IMM_VALUE	The IMM_VALUE field contains an 8-bit immediate value that is left-extended with 0s. This is used as either operand 0 or operand 1, as specified in the SSEL field.															
3-0 LEN	<p>LEN denotes the length, in bytes, of the operation (and the amount by which the IMM_VALUE is left-extended with 0s).</p> <p>1h : 1 byte 2h : 2 bytes 4h : 4 bytes 8h : 8 bytes</p> <p>All other values are reserved.</p> <p>NOTE: If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h may yield unexpected results.</p> <p>NOTE: If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD.</p>															

Table 7-94. FUNCTION field values

Value	Type	Description	Result
0h	add	Perform addition operation on operand 0 and operand 1.	operand 0 + operand 1
1h	add_w_carry	Perform addition with a carry bit operation on operand 0 and operand 1.	operand 0 + operand 1 + MC

Table continues on the next page...

Table 7-94. FUNCTION field values (continued)

Value	Type	Description	Result
2h	sub	Perform subtraction operation on operand 0 and operand 1.	operand 0 - operand 1
3h	sub_w_borrow	Perform subtraction with borrow operation on operand 0 and operand 1.	operand 0 - operand 1 - MC
4h	or	Perform bitwise OR operation on operand 0 and operand 1	operand 0 operand 1
5h	and	Perform bitwise AND operation on operand 0 and operand 1	operand 0 & operand 1
6h	xor	Perform bitwise XOR operation on operand 0 and operand 1	operand 0 ^ operand 1
7h	shift_l	Perform shift left operation. operand 0 should be shifted left by operand 1 bits; can't be used with "No Destination"	operand 0 << operand 1
8h	shift_r	Perform shift right operation. operand 0 should be shifted right by operand 1 bits; can't be used with "No Destination"	operand 0 >> operand 1
9h (MATH command only)	shld	Perform 32-bit left shift of DEST and concatenate with left 32 bits of operand 1. shld is only meaningful when DEST specifies Math Registers 0-7. For all other destinations, this function will work like an ADD with operand 0 set to 0. (That is, operand 1 will be placed into DEST.) Note that if operand 1 and DEST are the same Math Register, then shld would do a word swap. Function type shld is prohibited for the MATHI command.	{DEST[31:0], operand 1[63:32]}
Ah	zbyt or fbyt	MATH command: zbyt. Find zero bytes in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is all zeros. MATHI command: fbyt. Find the immediate byte in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is equal to IMM_VALUE. For the fbyt function it is illegal to set SSEL=1.	result is shown at left
Bh	swap_bytes	Swap the order of the four bytes in the ms half of operand 0, and independently swap the order of the four bytes in the ls half of operand 0. operand 0[39:32], operand 0[47:40], operand 0[55:48], operand 0[63:56], operand 0[7:0], operand 0[15:8], operand 0[23:16], operand 0[31:24] If this is used in conjunction with shld, the result of the two MATH operations will be an "end-for-end" swap of all 8 bytes. Function swap_bytes is prohibited for the MATHI command.	result is shown at left
All other values for this field are reserved.			
NOTE: A Compare operation is accomplished by selecting FUNCTION=sub, with DEST=No Destination and then doing a JUMP based on the CZ and/or CN flags.			

All MATH and MATHI commands take one clock cycle to execute except for the shift_l and shift_r functions. For the shift_l and shift_r functions, the number of bit positions that the data is shifted is specified in operand 1.

- If the data is to be shifted 64 or more bit positions, the shift command takes two clocks. One clock decodes the command, and one clock stores all zeros in the DEST register. Because all bits are shifted off the end, the result is all zeros.
- If the data is to be shifted 63 or fewer bit positions, the shift_l and shift_r functions take at most two clocks more than the number of bits in operand that are 1. The shifter can shift any power-of-2 number of bit positions in one cycle, and there are up to two additional cycles of overhead. If an intermediate shift result is all 0, the remaining shifts are skipped, resulting in fewer clock cycles than the maximum.

Note that the shift_l and shift_r functions first copy the data specified by operand 0 into the register specified by DEST and then shift the data in the DEST register. If the source is 64 bits but the destination is a 32-bit register, the 64-bit source value is truncated to its least-significant 32 bits before the shifting begins. A shift_l works as expected, but a shift_r of data from a 64-bit source to a 32-bit destination shifts in 0s rather than shifts in bits from the most-significant 32-bits of the source.

When one source, operand 0 or operand 1, is immediate, then the length may be any legal value. If 1, 2, or 4 bytes, the value is right-aligned in the word following the command. If the value is 8 bytes, then the value is in the two words that follow the command. Note that the immediate data can be 4 bytes even if the LEN is 8 bytes if the IFB bit is set.

7.22 SEQ IN PTR command

The Sequence In Pointer (SEQ IN PTR) command is used to specify the starting address for an input sequence and the length of that sequence (see [SEQ vs non-SEQ commands](#)). Only one input sequence may be active within the DECO at any one time. An input sequence is initiated by executing a SEQ IN PTR command with PRE = 0. This causes the following:

- Starting address of the input sequence to be set to the value in the Pointer field or to the original pointer if RTO=1 or to the original output sequence pointer if SOP=1.
- The Sequence In Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT_LENGTH field (if EXT = 1). If rewinding, the LENGTH or EXT_LENGTH field is added to the current length.

Note that if the EXT bit is 0, the EXT_LENGTH field is omitted from the SEQ IN PTR command.

The input sequence terminates when one of the following occurs:

- All input data is utilized.

SEQ IN PTR command

- An error occurs.
- A new input sequence is started by executing a SEQ IN PTR command with PRE = 0.

An error is flagged if a SEQ command attempts to input data if the execution of that command would cause the remaining length to go below 0. To extend the length of the sequence any number of additional SEQ IN PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current Sequence In Length register value, but the address for the input sequence is unaffected. In this case the SEQ IN PTR command does not include a Pointer field. Additional length may also be added via the MATH and MATHI commands.

If the same input data needs to be processed again, the input pointer can be restored to the original starting address by executing a SEQ IN PTR with RTO = 1. However, if the job descriptor using this input sequence was submitted through the Queue Manager Interface or the AIOP Interface, scatter tables are active, and input buffers are being released, it is not possible to back up if buffers have already been released. The SEQ IN PTR command does not include a Pointer field in this case.

The SEQ IN PTR command, in conjunction with the SEQ OUT PTR command, can reuse the input frame as the output frame. To achieve this, both commands must be executed with the IFR bit set. While the two commands may be executed in any order, they must both be executed prior to any reads from the input frame or any writes to the output frame. In addition, the input frame must be scattered. (If the frame isn't scattered, just provide the pointers as appropriate. If the job came in through QI or AI, the reuse is handled there rather than in DECO.) Furthermore, only one input frame reuse can be done per job. An offset to differentiate the starting point for the output frame from that of the input frame is provided via the SEQ OUT PTR command.

Table 7-95. SEQ IN PTR command, format

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CTYPE = 11110					RBS	INL	SGF	PRE	EXT	RTO	RJD	SOP	CTRL	Reserved	IFR
	15–0															
	LENGTH (used if EXT = 0)															
	<i>Optional words of SEQ IN PTR command:</i>															
This pointer is omitted if PRE=1 or RTO=1	Pointer (one or two words; see Address pointers)															
This field is omitted if EXT=0	EXT_LENGTH (present if EXT = 1) (one word)															

Table 7-96. SEQ IN PTR command, field descriptions

Field	Description
31-27 CTYPE	Command Type If CTYPE=11110b : SEQ IN PTR command
26 RBS	Release Buffers If RBS=0 : Do not release buffers. If RBS=1 : Release buffers from input frame. DECO releases buffers from the input frame after using them. It is an error if RBS = 1 when the Queue Manager Interface or AIOP Interface is not the job source. DECO releases only data buffers and scatter/gather tables referenced, directly or indirectly, by a top-level scatter/gather table but does not release the top-level scatter/gather table. If the table was specified by the original job descriptor's SEQ IN PTR command, then the top-level scatter/gather table will be released following the completion of the job by the Queue Manager Interface or the AIOP Interface. If the input frame is not from a scatter/gather table, it is handled in the same manner as the top-level scatter/gather table.
25 INL	In-Line Descriptor. This specifies that a new descriptor is to be found at the start of the data pointed to by the sequence. If INL=0 : No in-line descriptor If INL=1 : In-line descriptor present. An in-line descriptor is found at the start of the data pointed to by the sequence. DECO reads that descriptor (which must not have a shared descriptor) and then executes it. Therefore, a SEQ IN PTR with INL = 1 is the last command that is executed in the current descriptor. If the INL bit is 1 and the current Input Sequence length is not as large as the in-line descriptor, an error is flagged. Note that it is an error for INL and RJD to both be 1. See Using in-line descriptors for more information.
24 SGF	Scatter/Gather Table Flag. If SGF=0 : Pointer points to actual data. If SGF=1 : Pointer points to a scatter/gather table. SGF is ignored if RTO=1.
23 PRE	Previous. Add more length to the previously specified length of the input sequence. If PRE=0 : The sequence pointer is set to the value of the Pointer and the input sequence length is set to the value specified in the LENGTH or EXT LENGTH field. If PRE=1 : Command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current input sequence length. It is an error for the PRE bit and the RTO bit to both be set.
22 EXT	Extended Length If EXT=0 : Input data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command. If EXT=1 : Input data length value is in the 32-bit EXT LENGTH following the pointer. The 16-bit LENGTH field is ignored.
21 RTO	Restore. Used to restore an input sequence. If RTO=0 : Do not restore. If RTO=1 : Restore. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current Input Sequence length. The original sequence address and RBS and SGF bits are automatically restored. The intended use is to be able to go back to the beginning of a sequence to reprocess some or all of the data. It is an error for the PRE bit and the RTO bit to both be set. SGF is ignored if RTO=1.
20 RJD	Replacement Job Descriptor If RJD=0 : Don't replace job descriptor

Table continues on the next page...

Table 7-96. SEQ IN PTR command, field descriptions (continued)

Field	Description	
	<p>If RJD=1 : Replace job descriptor. If there is no shared descriptor, and CTRL=0, this is synonymous with the INL bit (that is, setting either bit yields the same result). However, if there is a shared descriptor, setting the RJD bit causes the job descriptor to be replaced without affecting the shared descriptor, which will have already been loaded. See Using replacement job descriptors for more information. It is an error if both RJD = 1 and INL = 1.</p> <p>NOTE: See the description below for the CTRL bit to understand how that bit can modify the above behavior.</p>	
19 SOP	<p>Sequence Out Pointer</p> <p>If SOP=0 : This bit has no effect.</p> <p>If SOP=1 : Start a new input sequence using the pointer and SGF bit used when the current output sequence was defined. (If there was no previous sequence, behavior is undefined.) The length used is the length that has already been written to the current output sequence. This functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. By using the SOP bit, the SEQ IN PTR command allows the second pass to reference the results of the first pass.</p> <p>It is an error to assert SOP if RBS, PRE, EXT or RTO are set. SGF and LENGTH are ignored.</p>	
18 CTRL	<p>CTRL. This bit is used in conjunction with the RJD bit to differentiate between a normal RJD and a control RJD. See Using replacement job descriptors for more information.</p> <p>If CTRL=0 and RJD=0 : This bit has no effect.</p> <p>If CTRL=0 and RJD=1 : The new descriptor is the next data to be read from the input frame.</p> <p>If CTRL=1 and RJD=0 : An error will be thrown.</p> <p>If CTRL=1 and RJD=1 : The new descriptor is found following the shared descriptor in memory. If this job was submitted via a Job Ring, and if there is no shared descriptor, an error will be thrown. If this job is from QI or AI, and if there is no shared descriptor, the control RJD is found in memory following the FLC.</p>	
17	Reserved	
16 IFR	<p>IFR. Input Frame Reuse.</p> <p>If IFR=0 : Do not use the output frame as the input frame.</p> <p>If IFR=1 : Use the output frame as the input frame. The descriptor should provide the input frame in the SEQ OUT PTR command. DECO will use that pointer as both the input and output frame. Note that QI and AI will automatically handle the pointer correctly when requested to perform input frame reuse. It is an error if IFR=1 and SGF=0, RTO=1, SOP=1, PRE=1, RJD=1, or INL=1. The correct input frame length must still be specified and a pointer must still be included (even though the pointer from the SEQ OUT PTR command will be used).</p>	
15-0 LENGTH	<p>LENGTH. This is the length of the input frame.</p> <p>If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the input sequence. The Extended Length word is omitted.</p> <p>If EXT = 1 : The number of bytes in (or to be added to) the input sequence is specified in the Extended Length field. The LENGTH field is ignored.</p>	
<i>Optional words of SEQ IN PTR command:</i>		
31-0 POINTER	<p>Pointer</p> <p>Specifies the starting address for an Input Sequence. See Address pointers.</p>	<p><i>If PRE = 1, RTO = 1, or SOP = 1, this field is omitted.</i></p>
31-0 EXT_LENGTH	<p>Extended Length Field</p> <p>If EXT = 0 : This field not present.</p>	<p><i>If EXT = 0, this field is omitted.</i></p>

Table 7-96. SEQ IN PTR command, field descriptions

Field	Description
	If EXT = 1 : The EXT LENGTH field specifies the number of bytes in (or to be added to) the Input Sequence.

7.23 SEQ OUT PTR command

The Sequence Out Pointer (SEQ OUT PTR) command is used to specify the starting address for an output sequence and the length of that sequence (see [SEQ vs non-SEQ commands](#)). Only one output sequence may be active within the DECO at any one time.

An output sequence is initiated by executing a SEQ OUT PTR command with PRE = 0. This causes the following:

- The starting address of the output sequence to be set to the value in the Pointer field or to the original pointer if rewinding.
- The Sequence Out Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1). If rewinding, the LENGTH or EXT_LENGTH field is added to the current length if REW = 10b and is ignored if REW = 11b.

If the EXT bit is 0, the EXT LENGTH field is omitted from the SEQ OUT PTR command.

The output sequence terminates when one of the following occurs:

- An error
- A new output sequence is started by executing a SEQ OUT PTR command with PRE = 0.

To extend the length of the sequence, any number of additional SEQ OUT PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current value in the Sequence Out Length register, but the address for the output sequence is unaffected. In this case, the SEQ OUT PTR command does not include a Pointer field. Additional length may also be added via the MATH and MATHI commands.

If the same output data needs to be processed again, the output pointer can be restored to the original starting address by executing a SEQ OUT PTR using the REW field. The SEQ OUT PTR command does not include a Pointer field in this case.

SEQ OUT PTR command

The SEQ OUT PTR command, in conjunction with the SEQ IN PTR command, can reuse the input frame as the output frame. To achieve this, both commands must be executed with the IFR bit set. While the two commands may be executed in any order, they must both be executed prior to any reads from the input frame or any writes to the output frame. In addition, the input frame must be scattered. (If the frame isn't scattered, just provide the pointers as appropriate. If the job came in through QI or AI, the reuse is handled there rather than in DECO.) Furthermore, only one input frame reuse can be done per job. If IFR=1 in the SEQ OUT PTR command, the EXT must also be set to 1. The EXT_LENGTH specified for the SEQ OUT PTR command may be different from the LENGTH or EXT_LENGTH specified for the SEQ IN PTR command. In order to allow the output frame to start at a different location than the input frame, use the OFFSET MODIFIER field to specify an offset. The offset is a 12-bit signed value (2's-complement) so that the output frame can start prior to the input frame (if the offset is negative), or after the start of the input frame (if the offset is positive). The descriptor writer should ensure that the output does not overwrite the input before the input has been read.

Table 7-97. SEQ OUT PTR command, format

	31–27	26	25	24	23	22	21–20	19	18–17	16
	CTYPE = 11111	Reserved		SGF	PRE	EXT	REW	EWS	Reserved	IFR
If IFR=0	15–0									
	LENGTH (used if EXT = 0)									
If IFR=1	15–12			11–0						
	Reserved			OFFSET MODIFIER						
<i>Additional words of SEQ OUT PTR command:</i>										
<i>This pointer is omitted if PRE=1 or if rewinding</i>	Pointer (one or two words; see Address pointers)									
<i>This word is omitted if EXT=0</i>	EXT LENGTH (used if EXT = 1)									

Table 7-98. SEQ OUT PTR command, field descriptions

Field	Description
31-27	Command Type.
CTYPE	If CTYPE=11111b : SEQ OUT PTR command
26-25	Reserved
24	If SGF=0 : Pointer points to actual data.
SGF	If SGF=1 : Pointer points to a scatter/gather table.
23	Previous. Add more length to the previously specified length of the Output Sequence.
PRE	If PRE=0 : The sequence pointer is set to the value of the pointer and the Output Sequence Length is set to the value specified in the LENGTH or EXT LENGTH field.

Table continues on the next page...

Table 7-98. SEQ OUT PTR command, field descriptions (continued)

Field	Description
	<p>If PRE=1 : The SEQ OUT PTR command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current Output Sequence Length.</p> <p>Note that it is an error if PRE = 1 and REW = 10b or 11b.</p>
22 EXT	<p>Extended Length</p> <p>If EXT=0 : The output data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command.</p> <p>If EXT=1 : The output data length value is in the 32-bit EXT LENGTH field following the pointer. The 16-bit LENGTH field is ignored.</p>
21-20 REW	<p>Rewind. Used to rewind an Output Sequence.</p> <p>If REW = 00b : Do not rewind.</p> <p>If REW = 01b : Error</p> <p>If REW = 10b : Rewind. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current sequence output length. The original sequence address and SGF bit are automatically restored. This allows returning to the beginning of a sequence to reprocess some or all of the data. DECO automatically disables the counting of bytes written to the output frame. In order to re-enable counting, use a write to the DECO CTRL Register.</p> <p>If REW = 11b : Rewind and Reset. The same as 10b, except that any length provided is ignored, the current output frame length is added back to the SOL (sequence output length) register and the tracking length ¹ of bytes written to the output frame is reset to 0. Care must be taken if the descriptor has modified the SOL register other than as a result of decrements caused by SEQ STORE and SEQ FIFO STORE commands. Since the number of bytes written to the output frame has been reset, counting such bytes remains enabled in this case.</p> <p>The REW = 10b or 11b functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. Executing the SEQ OUT PTR command with REW = 11b allows the second pass to start from the beginning of the output frame as if this were the original output stream. That way the final status reported back contains the correct length.</p>
19 EWS	<p>Enable Write Safe.</p> <p>When this bit is set, write-safe bus transactions are permitted for this output sequence. See AXI master (DMA) interface.</p>
18-17	Reserved
16	<p>IFR</p> <p>If IFR = 0 : Do not use the output frame as the input frame.</p> <p>If IFR = 1 : Use the output frame as the input frame. The descriptor must still have a SEQ IN PTR command. Note that QI and AI will automatically handle the pointer correctly when requested to perform input frame reuse. It is an error if IFR=1 and SGF=0, REW!=00b, PRE=1, or EXT=0.</p>
15-0 (If IFR=0) LENGTH	<p>If IFR=1, this field is not present.</p> <p>If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the output sequence.</p> <p>If EXT = 1 : The LENGTH field is ignored.</p>
15-12 (If IFR=1)	Reserved
11-0 (If IFR=1) OFFSET MODIFIER	<p>If IFR = 0 : This field is not present</p> <p>If IFR = 1 : This field is a 2's complement number and is used to modify the output frame relative to the input frame.</p>

Table continues on the next page...

Table 7-98. SEQ OUT PTR command, field descriptions (continued)

Field	Description	
<i>Optional words of SEQ OUT PTR command:</i>		
POINTER	Pointer. Specifies the starting address for an Output Sequence. See Address pointers .	If PRE = 1 or REW != 00b, this field is omitted.
One word EXT LENGTH	If EXT = 0 : The EXT LENGTH field is omitted. If EXT = 1 : The EXT LENGTH field specifies the number of bytes in (or to be added to) the output sequence.	If EXT = 0, this field is omitted.

1. DECO tracks how many bytes have been written to the output frame so that this number can be part of the status reported when a job completes.

Chapter 8

Public Key Cryptography Operations

SEC implements, through protocol commands, a number of public (and private) key functions. These are:

- DSA and ECDSA sign/verify
- Diffie-Hellman (DH) and ECDH key agreement
- ECC key generation (for ECDH, ECDSA, etc.)
- ECC public key validation
- DLC key generation for DH, DSA
- RSA public-key and private-key primitives, for use with RSA encryption/decryption and RSA signature generation/verification
- RSA key-generation finalization

SEC also contains a hardware block, [Public-key hardware accelerator \(PKHA\) functionality](#), which can be programmed directly for public key calculations.

8.1 Conformance considerations

The DSA and ECDSA key-generation, signing, verification, and Diffie-Hellman functions described are intended to conform to the following specifications (except where noted). For more information refer to the [NIST Cryptographic Algorithm Validation Program \(CAVP\) Certifications whitepaper](#), www.nxp.com/security, or consult these standards:

- FIPS PUB 186-4, *Digital Signature Standard (DSS)*, July 2013
- NIST SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012
- IEEE1363-2000, *IEEE Standard Specifications for Public-Key Cryptography*, January 30, 2000
- ANSI X9.42-2003, *Public Key Cryptography for the Financial Services Industry, Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*, November 19, 2003

- ANSI X9.63-2001, *Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography*, November 20, 2001
- ANSI X9.62-2005, *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 16, 2005

The notation used is from IEEE 1363-2000 because only that document provides a set of variable names and definitions consistent between both DSA and ECDSA.

Private keys for DSA and ECDSA, (as well as per-message secrets), are generated using the method of extra random bits, equivalent to that described in FIPS 186-4, (Appendix B.1.1). In B.1.1, c is a string of random bits, 64 bits longer than requested.

Then $x = (c \bmod (q - 1)) + 1$

SEC uses the following equivalent version.

$x = c \bmod q$; if $(x = 0)$, choose another c

In both cases, x is uniformly distributed in the range $[1, q-1]$.

Binary (aka Characteristic 2 or F_{2^m}) Elliptic Curves inputs and outputs are in polynomial basis and in affine (x, y) coordinates.

Assurances for the validity of all domain parameters and public keys must be obtained before invoking any of these functions. These functions assume that all domain parameters and public keys are valid and are associated with each other.

8.2 Discrete-log key-pair generation

Some important characteristics and requirements of discrete-log key-pair generation is as follows:

- DL KEY PAIR GEN is used to generate public key-pairs. There are four variations to generate either prime field or binary field keys for either DSA or ECDSA.
- Each of the public key functions writes out the private key, followed by the public key.
- DL KEY PAIR GEN requires the parameters listed in the following table. Note that $G_{x,y}$ and $W_{x,y}$ are pointers to input buffers containing both an x and y coordinate. The two coordinates must be the same length.
- There are two parameter lengths, size of the field (L), and size of the group or private key modulus (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that

the size of the buffers for $G_{x,y}$, $W_{x,y}$ and a,b must be twice L, as each holds two values of size L.

Table 8-1. Public key-generation parameters

Parameter	Input/output	Length	Definition
q	Input	L	Prime number or irreducible polynomial that creates the field
r	Input	N	Order of the field of private keys or modulus for creating private keys
a,b	Input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.)
g or $G_{x,y}$	Input	L or 2L	Generator or generator point (ECC)
s	Output	N	Private key
w or $W_{x,y}$	Output	L or 2L	Public key

8.2.1 Inputs to the discrete-log key-pair generation function

- For DSA, the domain parameters q , r , and g
- For ECDSA, the domain parameters q , a , b or b',r , and $G_{x,y}$

8.2.2 Assumptions of the discrete-log key-pair generation function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC bit of the Protocol Command register is set, s is treated as an encrypted key and is encrypted before being written out. When generating an encrypted key, the buffer must be large enough to hold the black key, i.e., the encrypted version of the key.

8.2.3 Outputs from the discrete-log key-pair generation function

- The signer's private key s
- For DSA, the signer's public key w
- For ECDSA, the signer's public key $W_{x,y}$

8.2.4 Operation of the discrete-log key-pair generation function

- Generate a private key s , in the range $1 \leq s < r$. (Generate a random number k , 64 bits larger than r , and find $s = k \bmod r$. If $s = 0$, generate a new k .)

- Compute $w = g^s \text{ mod } q$, or $W_{x,y} = sG_{x,y}$.
- Output (s, w) or $(s, W_{x,y})$ as the private and public keys.

8.2.5 Notes associated with the discrete-log key-pair generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \text{ mod } q$ must be given, rather than b .

For IETF DH involving domains like MODP Groups 5, 14, 15, and 16, there is no published r value. However, a value is necessary for this function, as it is the modulus used to create the private key, where $1 (< \text{private_key} < \text{mod})$. The value of N should be determined based upon the desired strength of the private key; there are recommendations in the IETF RFCs and elsewhere. Both r and the private key will be N bytes long. A typical value for r would be a string containing N bytes of FFh.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSSEL field and their meanings are listed in [Table 8-5](#).

Table 8-2. Public-key generation protocol data block (PD=0 version)

SGF (6 bits)	PD=0 (1 bit)	Reserved (8 bits)	L (10 bits)	N (7 bits)
Pointer to q				
Pointer to r				
Pointer to g (DSA) or G _{x,y} (ECC)				
Pointer to s				
Pointer to w (DSA) or W _{x,y} (ECC)				
Pointer to a,b (ECC only)				
(The protocol data block for DSA is shorter than for ECDSA, because the pointer to a,b is not required.)				

Table 8-3. Public-key generation protocol data block (PD=1 version)

SGF (6 bits)	PD=1 (1 bit)	Reserved (8 bits)	Reserved d (3 bits)	ECDSSEL (7 bits)	Reserved (7 bits)
Pointer to s					
Pointer to w or W _{x,y}					

For both PDB versions the format of the SGF field is illustrated in this figure.

Table 8-4. Public key-generation PDB - format of the SGF field

	31	30	29	28	27	26
Version when PD=0	ref q	ref r	ref G _{x,y}	ref s	ref w or W _{x,y}	a,b (ECC) reserved (DSA)
Version when PD=1	Reserved			ref s	ref W _{x,y}	Reserved

NOTE: If the SGF bit is set the argument is referenced via a scatter/gather table. If the SGF bit is not set the argument is referenced via a direct-address pointer.

When PD=1, a valid value must appear in the ECDSEL (Elliptic Curve Domain Selection) field. A list of the values that may be used in the ECDSEL field, and their meanings, is given in the table below.

The following variable definitions apply to the following table. Variable names (q , r , b , c) follow the conventions of IEEE Std 1363.

Name

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org
- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

q

This is the *field-defining* value for the elliptic curve. For F_p curves, it is the prime number used as the modulus for all point arithmetic; it is named p in some other publications. For F_{2^m} curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

L

Discrete-log key-pair generation

This is the number of bytes needed to hold q and each of its associated values: a, b, c , the point coordinates x and y , the result of an ECDH key agreement, etc.

r

This is the (usually prime) number which is the order of G , the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named n in some other publications.

N

This is the number of bytes needed to hold r and each of its associated values: private keys, each of the two components of an ECDSA signature, etc.

a

This variable, along with q and b , define the elliptic curve. For F_p , a is the coefficient for the x term. For F_{2^m} , it is the coefficient for the x^2 term.

b / c (b')

b is the coefficient for the x^0 (ones) term in an F_{2^m} elliptic curve equation. Its relationship with c is $b = c^4$. c is sometimes referred to as b' in NXP documentation.

Table 8-5. ECDSEL field values for built-in ECC F_p domains

When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size.	
Value	Name(s)
ECC F_p domains	
00h	P-192, secp192r1, ansix9p192r1, prime192v1, ECPRGF192Random
01h	P-224, secp224r1, ansix9p224r1, wtls12, ECPRGF224Random
02h	P-256, secp256r1, ansix9p256r1, prime256v1, ECDSA-256, ecp_group_19, ECPRGF256Random
03h	P-384, secp384r1, ansix9p384r1, ECDSA-384, ecp_group_20, ECPRGF384Random
04h	P-521, secp521r1, ansix9p521r1, ECDSA-521, ecp_group_21, ECPRGF521Random
05h	brainpoolP160r1
06h	brainpoolP160t1
07h	brainpoolP192r1
08h	brainpoolP192t1
09h	brainpoolP224r1
0Ah	brainpoolP224t1
0Bh	brainpoolP256r1
0Ch	brainpoolP256t1
0Dh	brainpoolP320r1
0Eh	brainpoolP320t1
0Fh	brainpoolP384r1
10h	brainpoolP384t1
11h	brainpoolP512r1
12h	brainpoolP512t1
13h	prime192v2

Table continues on the next page...

Table 8-5. ECDSEL field values for built-in ECC F_p domains (continued)

When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size.	
Value	Name(s)
ECC F_p domains	
14h	prime192v3
15h	prime239v1
16h	prime239v2
17h	prime239v3
18h	secp112r1, wtls6
19h	wtls8
1Ah	wtls9
1Bh	secp160k1, ansix9p160k1
1Ch	secp160r1, ansix9p160r1, wtls7
1Dh	secp160r2, ansix9p160r2
1Eh	secp192k1, ansix9p192k1
1Fh	secp224k1, ansix9p224k1
20h	secp256k1, ansix9p256k1
ECC F_{2^m} domains	
40h	B-163, ansix9t163r2, sect163r2, EC2NGF163Random
41h	B-233, sect233r1, ansix9t233r1, EC2NGF233Random, wtls11
42h	B-283, sect283r1, ansix9t283r1, EC2NGF283Random
43h	B-409, sect409r1, ansix9t409r1, EC2NGF409Random
44h	B-571, sect571r1, ansix9t571r1, EC2NGF571Random
45h	K-163, ansix9t163k1, sect163k1, EC2NGF163Koblitz, wtls3
46h	K-233, sect233k1, ansix9t233k1, EC2NGF233Koblitz, wtls10
47h	K-283, sect283k1, ansix9t283k1, EC2NGF283Koblitz
48h	K-409, sect409k1, ansix9t409k1, EC2NGF409Koblitz
49h	K-571, sect571k1, ansix9t571k1, EC2NGF571Koblitz
4Ah	wtls1
4Bh	sect113r1, wtls4
4Ch	c2pnb163v1, wtls5
4Dh	c2pnb163v2
4Eh	c2pnb163v3
4Fh	sect163r1, ansix9t163r1
50h	sect193r1, ansix9t193r1
51h	sect193r2, ansix9t193r2
52h	sect239k1, ansix9t239k1
53h	Oakley 3, ec2n_group_3
54h	Oakley 4, ec2n_group_4

8.3 Using the Diffie_Hellman function

Diffie-Hellman is used in key exchange and key agreement schemes. For example, Internet Key Exchange (IKE) specifies a pseudo-random function (PRF) that takes the result of a Diffie-Hellman operation as an input. Because the output of Diffie-Hellman is a secret value, it is advisable to store the output in encrypted form. SEC's Diffie-Hellman protocol provides this option, and SEC's IKE PRF protocol can read the secret in encrypted form.

Diffie-Hellman is defined for both discrete log (DH) and elliptic-curve (ECDH) forms. SEC provides acceleration support for both forms.

8.3.1 Diffie_Hellman requirements

Diffie-Hellman requires the parameters listed in this table.

Table 8-6. Required Diffie-Hellman parameters

Parameter	Input/Output	Length	Definition
L	input	10 bits	Number of bytes of the the field
N	input	7 bits	Number of bytes of the private key
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	-	Unused for Diffie-Hellman
a,b	input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.)
w' or $W'_{x,y}$	input	L (DH) or 2L (ECDH)	Other party's public key
s	input	N	Own private key
z	output	L	Shared secret value

NOTE: $W_{x,y}$ is a pointer to an input buffer containing both an x and a y coordinate. The two coordinates must be the same length.

There are two parameter lengths, size of the field (L), and the size of the private key (N). These represent the size of the buffers, in bytes, required to hold the input and output data.

The size of the buffers for $G_{x,y}$, $W_{x,y}$, and a,b must be twice L, as each holds two values of size L.

8.3.2 Inputs to the Diffie-Hellman function

- For discrete logs, the domain parameters q , s (own private key), and w' (other's public key).
- For elliptic curve, the domain parameters q , s (own private key), and $W'_{x,y}$ (other's public key), a and b (or b').

Note that the domain parameters r and g (or $G_{x,y}$) are not used.

8.3.3 Assumptions of the Diffie-Hellman function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC_PRI bit of the Protocol Information register is set, s is treated as an encrypted key and is decrypted after being read. If the ENC_PUB bit of the protocol information is set, then z is encrypted before being written.

8.3.4 Outputs from the Diffie-Hellman function

The shared secret value z

8.3.5 Operation of the Diffie-Hellman function

- Read in the private key pointed to by s .
- For DL, compute $z = w^s \bmod q$.
- For ECC, compute $\text{new_point} = s * W$, and output $z = x$ coordinate of new_point
- Output z as the shared secret.

8.3.6 Notes associated with the Diffie-Hellman function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given rather than b . For a detailed explanation, see [ECC_F2M: Point math on a standard curve over a binary field \(\$F_{2^m}\$ \)](#)

Table 8-7. Diffie-Hellman protocol data block

SGF (6 bits)	Reserved (9 bits)	L (10 bits)	N (7 bits)
Pointer to q			

Table continues on the next page...

Table 8-7. Diffie-Hellman protocol data block (continued)

Pointer to r (unused)
Pointer to w or $W_{x,y}$
Pointer to s
Pointer to z
Pointer to a,b (ECC only)

For discrete log Diffie-Hellman, the pointer to a,b is not required. The following figure illustrates the format of the SGF field.

Table 8-8. Diffie-Hellman PDB-format of the SGF field

31	30	29	28	27	26
ref q	ref r (unused)	ref w or $W_{x,y}$	ref s	ref z	a,b (ECC only)
NOTE: If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.					

8.4 Generating DSA and ECDSA signatures

DSA_SIGN is SEC's hardware implementation of NIST's DSA (Digital Signature Algorithm) and ECDSA digital signing functions. It supports DSA and ECDSA in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MSG_REP bit in the OPERATION Command register.

There are two parameter lengths: size of the field (L), and size of the group (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that the size of the buffers for $G_{x,y}$ and a,b must be twice L , as each holds two values of size L .

This table lists the DSA and ECDSA sign protocol parameters.

Table 8-9. DSA and ECDSA sign parameters

Parameter	Input/Output	Length	Definition
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	N	Order of the field of private keys
a, b	input	$2L$	ECC curve parameters. For binary field curves, b' rather than b is given. (ECC only.)
g or $G_{x,y}$	input	L (DSA), $2L$ (ECDSA)	Generator or generator point (ECC)

Table continues on the next page...

Table 8-9. DSA and ECDSA sign parameters (continued)

Parameter	Input/Output	Length	Definition
s	input	N	Private key
f (or m)	input	N	Message representative (typically the hash of the message) or the actual message
c	output	N	First part of digital signature
d	output	N	Second part of digital signature. The buffer for d must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding.
u	output	N	Per message random number, only in TEST mode

8.4.1 Inputs to the DSA and ECDSA signature generation function

- For DSA, the domain parameters q , r , and g associated with key s .
- For ECDSA, the domain parameters q , r , g , a and b associated with key s .
- The signer's private key s .
- The message representative, which is an integer $f \geq 0$, or the message itself (which is hashed to form a message representative).

8.4.2 Assumptions of the DSA and ECDSA signature generation function

- The private key s is in the range $1 \leq s < r$, and the domain parameters are valid and are associated with each other, (that is, parameter validation must be done prior to using this function).
- The message representative, f , is generated using an approved hashing function of the appropriate security strength.
- If the ENC bit of the Protocol Command is set, then s is treated as an encrypted key, and is decrypted before it is used.

8.4.3 Outputs from the DSA and ECDSA signature generation function

When running the full signature operation, the output is a pair of integers (c, d) , where $1 \leq c < r$ and $1 \leq d < r$

When just the first part of the signature is generated the output is the integer c , as above, and the encrypted version of the (inverted) temporary key used in the creation of the signature. It is stored at d , and the memory there must have space for the ECB-encrypted version (i.e, rounded up to the nearest 16 bytes).

When just the second part of the signature is generated the output is d , as in the output of the complete signature operation.

8.4.4 Operation of the DSA and ECDSA signature generation function

- Generate a per message private key u , in the range $1 \leq u < r$. (Generate a random number k , 64 bits larger than r , and find $u = k \text{ mod } r$. If $u = 0$, generate a new k .)
- Compute $c = (g^u \text{ mod } q) \text{ mod } r$, or $V_{x,y} = uG_{x,y}$, $c = V_x \text{ mod } r$. If $c = 0$, try again with a new u .
- Compute $d = u^{-1}(f + sc) \text{ mod } r$. If $d = 0$, try again with a new u .
- Output (c, d) as the signature.
- If the TEST bit of the Protocol Command is set, then also output u . This test mode is not accessible in the Trusted or Secure states.

8.4.5 Notes associated with the DSA and ECDSA Signature Generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \text{ mod } q$ must be given, rather than b .

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Generation function and pointers to those arguments. Each pointer occupies one word of the PDB if MCFGR[PS]=0, or two words of the PDB if MCFGR[PS]=1.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSSEL field and their meanings are listed in [Table 8-5](#). Note that if PD=1 for a DSA operation, a PDB error will be generated.

Table 8-10. DSA and ECDSA Signature Generation protocol data block

The format of PDB Word 1 depends on the value in the PD field.					
PDB Word 1	SGF (see format in table below) (9 bits)	PD=0 (1 bit)	Reserved (5 bits)	L (10 bits)	N (7 bits)

Table continues on the next page...

Table 8-10. DSA and ECDSA Signature Generation protocol data block (continued)

SGF (see format in table below) (9 bits)	PD=1 (1 bit)	Reserved (5 bits)	Reserved (3 bits)	ECDSSEL (7 bits)	Reserved (7 bits)
---	-----------------	----------------------	----------------------	---------------------	----------------------

Table 8-11. DSA and ECDSA Signature Generation protocol data block, continued

Following PDB Word 1 the PDB contains a series of address pointers for arguments to the Sign Protocol operation. Each pointer occupies two words. If the pointer's SGF bit is 0 this is the 49-bit address of the argument data itself. If the pointer's SGF bit is 1 this is the 49-bit address of a Scatter Gather Table that references the argument data. The format of the SGF field is illustrated in the table below. Note that different arguments pointers are included or omitted depending upon the value in the PD field and the type of Sign operation (full, 2nd half, 1st half). The six possible arrangements of the pointers are shown in the rightmost six columns below.

	PD=0			PD=1		
	Full Sign operation (1st Half Sign and 2nd Half Sign)	2nd Half Sign operation but not 1st Half Sign	1st Half Sign operation but not 2nd Half Sign	Full Sign operation (1st Half Sign and 2nd Half Sign)	2nd Half Sign operation but not 1st Half Sign	1st Half Sign operation but not 2nd Half Sign
PDB pointer 1	<i>q</i>	<i>r</i>	<i>q</i>	<i>s</i>	<i>s</i>	<i>c</i>
PDB pointer 2	<i>r</i>	<i>s</i>	<i>r</i>	<i>f</i> (MSG_REP=0) or <i>m</i> (MSG_REP=1)	<i>f</i> (MSG_REP=0) or <i>m</i> (MSG_REP=1)	<i>d</i>
PDB pointer 3	<i>g</i> (DSA) or <i>G_{x,y}</i> (ECDSA)	<i>f</i> (MSG_REP=0) or <i>m</i> (MSG_REP=1)	<i>g</i> (DSA) or <i>G_{x,y}</i> (ECDSA)	<i>c</i>	<i>c</i>	See note 4
PDB pointer 4	<i>s</i>	<i>c</i>	<i>c</i>	<i>d</i>	<i>d</i>	
PDB pointer 5	<i>f</i> (MSG_REP=0) or <i>m</i> (MSG_REP=1)	<i>d</i>	<i>d</i>	See notes 2 and 7	See notes 3 and 7	
PDB pointer 6	<i>c</i>	See notes 3 and 7	See note 5	See notes 3 and 7		
PDB pointer 7	<i>d</i>		See note 6			
PDB pointer 8	See notes 1 and 7					
PDB pointer 9	See notes 8 and 7					
PDB pointer 10	See notes 9 and 7					

Note 1: If ECDSA, this is a pointer to *a,b*, else if TEST=1, this is a pointer to *u*, else if MSG_REP=1, this is the message length, else the previous word is the last word of the PDB.

Note 2: If TEST=1, this is a pointer to *u*, else if MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.

Note 3: If MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.

Note 4: If TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDBt.

Table 8-11. DSA and ECDSA Signature Generation protocol data block, continued

Note 5: If ECDSA, this is a pointer to *a,b*, else if TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDB.

Note 6: If ECDSA **and** TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDB.

Note 7: MSG_REP=1 means calculate the message representative from the message *m*. In this case the message length must be provided in one word of the PDB. MSG_REP=0 means don't calculate a message representation, use *f* as the message representative. In this case the word that would contain the message length is omitted from the PDB.

Note 8: If ECDSA **and** TEST=1, this is a pointer to *u*, else if (ECDSA **or** TEST=1) **and** MSG_REP=1, this word contains the message length, else the previous word is the last word of the PDB.

Note 9: If ECDSA **and** TEST=1 **and** MSG_REP=1, this word contains the message length, else the previous word is the last word of the PDB.

This table shows the format of the SGF field.

Table 8-12. DSA and ECDSA Signature Generation protocol data block - Format of the SGF Field

Formats for	Format for		bit number								
			31	30	29	28	27	26	25	24	23
DSA or for ECDSA with PD=0	Full Sign	SGF bit for:	q	r	g (DSA) or G _{x,y} (ECDSA)	s	f (MSG_REP=0) or m (MSG_REP=1)	c	d	a,b (ECDSA) or reserved (DSA)	u (TEST=1) or reserved (TEST=0)
	1st Half Sign		q	r	g (DSA) or G _{x,y} (ECDSA)	reserved	reserved	c	d	a,b (ECDSA) or reserved (DSA)	u (TEST=1) or reserved (TEST=0)
	2nd Half Sign		reserved	r	reserved	s	f (MSG_REP=0) or m (MSG_REP=1)	c	d	reserved	reserved
ECDSA with PD=1	Full Sign	SGF bit for:	reserved	reserved	reserved	s	f (MSG_REP=0) or m (MSG_REP=1)	c	d	reserved	u (TEST=1) or reserved (TEST=0)

Table continues on the next page...

Table 8-12. DSA and ECDSA Signature Generation protocol data block - Format of the SGF Field (continued)

	1st Half Sign	reserved	reserved	reserved	s	reserved	c	d	reserved	u (TEST=1) or reserved (TEST=0)
	2nd Half Sign	reserved	reserved	reserved	reserved	f (MSG_REP=0) or m (MSG_REP=1)	c	d	reserved	reserved
<i>If the SGF bit for an argument is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.</i>										

8.5 Verifying DSA and ECDSA signatures

DSA_VERIFY is the digital signature algorithm (DSA) verification function. It supports both DSA and ECDSA, in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MSG_REP bit in the OPERATION command.

There are two parameter lengths:

- Size of the field (L)
- Size of the subgroup (N)

These are given in bytes, and denote the size of the buffer required to hold each parameter. Note that the size of the buffers for $G_{x,y}$, $W_{x,y}$ and a,b must be twice L, as each holds two values of size L.

Table 8-13. DSA and ECDSA Verify parameters

Parameter	Input/Output	Length (bytes)	Definition
q	input	L	Prime number or irreducible polynomial that creates the field
r	input	N	Order of the subgroup of private keys
a, b	input	2L	ECC curve parameters. For binary field curves, b' rather than b is used. (ECDSA only.)
g or $G_{x,y}$	input	L (DSA), 2L (ECDSA)	Generator or generator point (ECDSA)
w or $W_{x,y}$	input	L (DSA),	Public key

Table continues on the next page...

Table 8-13. DSA and ECDSA Verify parameters (continued)

Parameter	Input/Output	Length (bytes)	Definition
		2L (ECDSA)	
f (or m)	input	N	Message representative (typically the hash of the message) or the actual message
c	input	N	First part of digital signature
d	input	N	Second part of digital signature
Temp	input/output	L (DSA) 2L (ECDSA)	Temporary storage for intermediate results

8.5.1 Inputs to the DSA and ECDSA signature verification function

- For DSA, the domain parameters q , r , and g associated with key w
- For ECDSA, the domain parameters q , r , $G_{x,y}$, a and b associated with key $W_{x,y}$
- The signer's public key w or $W_{x,y}$
- The received message representative, which is an integer $f \geq 0$, or the message itself (which is hashed to form a message representative)
- The received signature to be verified, which is a pair of integers (c, d)

8.5.2 Assumptions of the DSA and ECDSA signature verification function

- The public key (w or $W_{x,y}$) and the domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- The message representative, f , is generated using an approved hashing function of the appropriate security strength.

8.5.3 Outputs from the DSA and ECDSA signature verification function

- If the signature is correct, this function terminates normally.
- If the signature is not correct, this function terminates with an error code.

8.5.4 Operation of the DSA and ECDSA signature verification function

- Check that c is in the range $[1, r-1]$. If not, terminate with error code invalid signature.
- Check that d is in the range $[1, r-1]$. If not, terminate with error code invalid signature.
- For DSA, compute $c' = ((G^{d^1f} \bmod q)(w^{d^1c} \bmod q)) \bmod r$.
- For ECDSA, compute $P_{x,y} = d^1fG_{x,y} + d^1cW_{x,y}$, and then if $P_{x,y}$ is the point at infinity, terminate with error code invalid signature, else let $c' = P_x \bmod r$.
- If $c' \neq c$, then terminate with error code invalid signature.
- Continue as valid.

8.5.5 Notes associated with the DSA and ECDSA Signature Verification function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given, rather than b .

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Verification function and pointers to those arguments. Each pointer occupies one word of the PDB if MCFGR[PS]=0, or two words of the PDB if MCFGR[PS]=1.

Parameter information is as follows:

- L is the number of bytes in various data buffers. L the size of the prime number or irreducible polynomial representing the cryptographic field.
- N is another length, in number of bytes in data buffers. N is the size of the number representing the order of the subgroup of private keys within the field.
- All parameters are pointers to data buffers of size L or N, (or 2L for elliptic curve points and a,b).

The protocol data block for DSA is shorter than for ECDSA, as the pointer to a,b is absent.

A temporary buffer is required during the verification of the signature.

- For DSA, the temporary buffer must be at least L bytes.
- For ECDSA, the temporary buffer must be at least 2L bytes.

The ENC bit of PROTOINFO is ignored for signature verification because only public keys are used. If the MSG_REP bit of the OPERATION command is set to 1, the pointer to f (or m) points to the message to be signed, rather than to a message representative. The message length field of the protocol data block (only used when the MSG_REP bit of the OPERATION command is set to 1) defines the length of the message to be signed.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 8-5. Note that if PD=1 for a DSA operation, a PDB error will be generated.

Table 8-14. DSA and ECDSA Signature Verification protocol data block

The format of PDB Word 1 depends on the value in the PD field.						
PDB Word 1	SGF (see format in table below) (9 bits)	PD=0 (1 bit)	Reserved (5 bits)	L (10 bits)		N (7 bits)
	SGF (see format in table below) (9 bits)	PD=1 (1 bit)	Reserved (5 bits)	Reserved (3 bits)	ECDSEL (7 bits)	Reserved (7 bits)

Table 8-15. DSA and ECDSA Signature Verification protocol data block, continued

Following PDB Word 1 the PDB contains a series of address pointers for arguments to the Signature Verification function. Each pointer occupies two words. If the pointer's SGF bit is 0 this is the 49-bit address of the argument data itself. If the pointer's SGF bit is 1 this is the 49-bit address of a Scatter Gather Table that references the argument data. The format of the SGF field is illustrated in the table below. Note that different arguments pointers are included or omitted depending upon the value in the PD field and the type of Signature Verification operation (verification with public key, verification with private key). The four possible arrangements of the pointers are shown in the rightmost four columns below.

	PD=0 (don't use a predefined domain)		PD=1 (use a predefined domain)	
	Signature Verification with public key	Signature Verification with private key	Signature Verification with public key	Signature Verification with private key
PDB pointer 1	q	q	$W_{x,y}$	s
PDB pointer 2	r	r	f (MSG_REP=0) or m (MSG_REP=1)	f (MSG_REP=0) or m (MSG_REP=1)
PDB pointer 3	g (DSA) or $G_{x,y}$ (ECDSA)	g (DSA) or $G_{x,y}$ (ECDSA)	c	c
PDB pointer 4	w (DSA) or $W_{x,y}$ (ECDSA)	s	d	d
PDB pointer 5	f (MSG_REP=0) or m (MSG_REP=1)	f (MSG_REP=0) or m (MSG_REP=1)	Temp	See notes 2 and 3
PDB pointer 6	c	c	See notes 2 and 3	
PDB pointer 7	d	d		
PDB pointer 8	Temp	See notes 1 and 3		

Table continues on the next page...

Table 8-15. DSA and ECDSA Signature Verification protocol data block, continued (continued)

PDB pointer 9	See notes 1 and 3	See notes 2 and 3		
PDB pointer 10	See notes 2 and 3			

Note 1: If ECDSA, this is a pointer to *a,b*, else if MSG_REP=1, this is the message length, else the previous word is the last word of the PDB.

Note 2: If ECDSA and MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.

Note 3: MSG_REP=1 means calculate the message representative from the message *m*. In this case the message length must be provided in one word of the PDB. MSG_REP=0 means don't calculate a message representation, use *f* as the message representative. In this case the word that would contain the message length is omitted from the PDB.

This table shows the format of the SGF field.

Table 8-16. DSA and ECDSA Signature Verification protocol data block - Format of the SGF Field

Formats for	Format for	SGF bit for:	bit number								
			31	30	29	28	27	26	25	24	23
DSA or ECDSA with PD=0	Verify with public key	SGF bit for:	q	r	g (DSA) or G _{x,y} (ECDSA)	w (DSA) or W _{x,y} (ECDSA)	f (MSG_REP=0) or m (MSG_REP=1)	c	d	Temp	a,b (ECDSA) or reserved (DSA)
	Verify with private key		q	r	g (DSA) or G _{x,y} (ECDSA)	s	f (MSG_REP=0) or m (MSG_REP=1)	c	d	Temp	a,b (ECDSA) or reserved (DSA)
ECDSA with PD=1	Verify with public key	SGF bit for:	reserved	reserved	reserved	W _{x,y} (ECDSA)	f (MSG_REP=0) or m (MSG_REP=1)	c	d	Temp	reserved
	Verify with private key		reserved	reserved	reserved	s	reserved	c	d	Temp	reserved
			<i>If the SGF bit for an argument is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.</i>								

8.6 RSA Finalize Key Generation (RFGK)

SEC is able to complete RSA key generation given primes p and q and the public exponent e . It can be configured to compute, or skip computation of, the remaining elements of the public and private key.

The computations performed are:

- $n = p \cdot q$
- $d = 1/e \text{ mod LCM}(p-1, q-1)$
- $d1 = d \text{ mod } (p-1)$
- $d2 = d \text{ mod } (q-1)$
- $c = 1/q \text{ mod } p$

it will also

- check p and q to determine whether they are 'too close' (per FIPS 186-3). This will occur if they are long enough. It will not be effective if they are not of the same bit length (that is, high order bits of p and q are not the same).
- Compute $\#d$ if d is being computed
- Check that $\#d > \#p$.
- Check that $\#n$ and the computed $\#n$ are the same

RSA Finalize Key Generation PDB

Table 8-17. RSA Finalize Key Generation PDB

Descriptor header (one or two words)			
SGF (9 bits)	Reserved (23 bits)		
Reserved (23 bits)			#p (9 bits)
Rsv (6 bits)	#n (10 bits)	Reserved (6 bits)	#e (10 bits)
Reference to p			
Reference to q			
Reference to e			
Reference to n			
Reference to d			
Reference to #d			
Reference to d1 (not required if FUNCTION=10b)			
Reference to d2 (not required if FUNCTION=10b)			
Reference to c (not required if FUNCTION=10b)			

The fields #d and #n contain right-aligned 10-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. Note that the size of d must be at least as large as the size of an encrypted n.

A reference is a pointer, either to the data or to a scatter-gather table. The pointer is one or two words long, depend upon the platform.

The references to d1, d2, and c may be omitted if those values are not to be generated.

This figure shows the format of the SGF field.

Table 8-18. RSA Finalize Key Generation PDB - SGF field

31	30	29	28	27	26	25	24	23	22	21
ref p	ref q	ref e	ref n	ref d	ref #d	ref d1	ref d2	ref c	Reserved	Reserved
If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.										

8.7 Implementation of the RSA encrypt operation

SEC implements an RSA encrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. It is the "RSA public key primitive" and it is commonly used to encrypt a secret or to verify a signature.

When used for signature verification, it is actually "unscrambling" the signature so that its contents may be verified. The input must be passed "raw" to the RSA function.

SEC implements the RSA encrypt operation in the following form:

$g = \text{RSA-Encrypt}(n, e, \text{FORMAT}, \#f, f, \text{fff})$

The variables have the following definitions:

- n, e represent the public key
- Before the RSA math is performed, FORMAT specifies the format to be used for encoding f (none or PKCS #1 v1.5 encryption)
- f is the value to be RSA-encrypted (input value; will be output value if random data ('f out') is selected)
- #f is the size in bytes of f
- fff represents the type of encryption applied to f if it is output by SEC
- g is the RSA-encrypted value of (the possibly formatted version of) f

Implementation of the RSA encrypt operation

The RSA Encrypt function is implemented with the OPERATION command. See [PROTOCOL OPERATION commands](#) for details on selecting this operation. See [Table 7-62](#) and [Table 7-63](#) for details about the PROTINFO field in the OPERATION command.

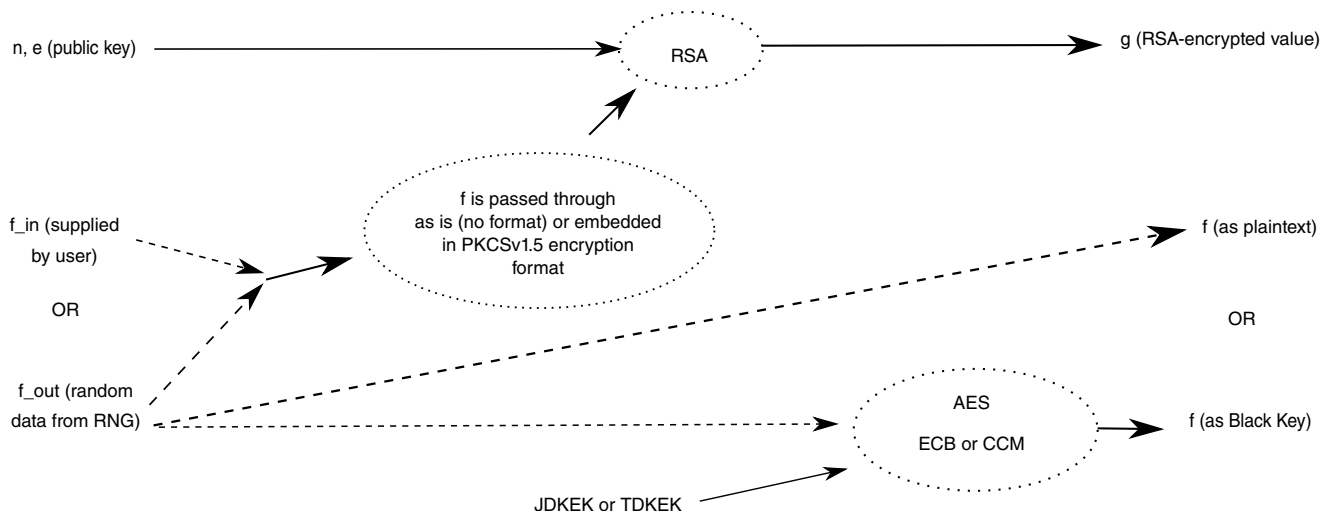


Figure 8-1. RSA encrypt operation

The user may either supply a plaintext value to be RSA encrypted (f_{in}) or may opt to have SEC generate $\#f$ bytes of random data from the RNG (f_{out}). The latter option allows f to be stored encrypted as a black key, It is then protected; encrypted or not, it can be used as an input to a PRF operation.

Once the value of f is known and possibly wrapped in PKCSv1.5 encoding, it is RSA-encrypted and the result stored as g .

The PDB for the RSA encrypt operation is shown below.

Table 8-19. RSA Encrypt PDB

SGF (4 bits)	Rsv (4 bits)	#e (12 bits)	#n (12 bits)
Reference to f			
Reference to g			
Reference to n			
Reference to e			
Reserved (20 bits)		#f (12 bits)	

All references are either 32-bit, 36-bit or 40-bit address pointers.

The fields #e, #n and #f contain right-aligned 12-bit values that indicate the size of the e, n and f inputs, respectively. The format of the SGF field is shown below.

Table 8-20. RSA Encrypt PDB; SGF field

31	30	29	28
ref f	ref g	ref n	ref e
If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.			

8.8 Implementation of the RSA decrypt operation

SEC implements an RSA decrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. This is the "RSA private key primitive" and it is commonly used either to decrypt a secret or to create a signature (sign a message).

When used for signing a message, it is actually "scrambling" the signature; the output must be allowed to pass "raw" from the RSA function.

SEC implements the RSA decrypt operation in the form:

$f = \text{RSA-Decrypt}(\text{(private key)}, \text{FORMAT}, g, \text{fff})$

The variables have the following definitions:

- (private key) represents the private key, in one of three forms
- After the RSA math is performed on g, FORMAT specifies the format to be used for decoding f (none or PKCS #1 v1.5 encryption)..
- g is the input value
- fff represents the type of encryption applied to f when it is output by SEC
- f is the RSA-decrypted output value.

This operation leaves #f (size of the plaintext f, in bytes) in the MATH0 register. This may be important if using a FORMAT of PKCS #1 v1.5 encryption.

The RSA Decrypt function is implemented via the OPERATION Command. See [PROTOCOL OPERATION commands](#) for details on selecting this operation. See [Table 7-64](#) and [Table 7-64](#) for details concerning the PROTINFO field in the OPERATION Command.

Implementation of the RSA decrypt operation

As the private key is an input and is considered sensitive, it may be supplied in Black Key form. The components, individually, of the private key would then be decrypted using the appropriate key encryption key and cryptographic mode prior to use. Note that n is never encrypted. SEC allows the private key input to be provided in three different forms: #1 (n, d), #2 (p, q, d), and #3 (p, q, dp, dq, c).

The RSA Decrypt function is implemented via the OPERATION Command. See [PROTOCOL OPERATION commands](#) for details on selecting this operation. See [Table 7-64](#) and [Table 7-65](#) for details concerning the PROTINFO field in the OPERATION Command.

The operation of RSA decrypt when using form #1, in which the private key is input as (n, d), is illustrated below.

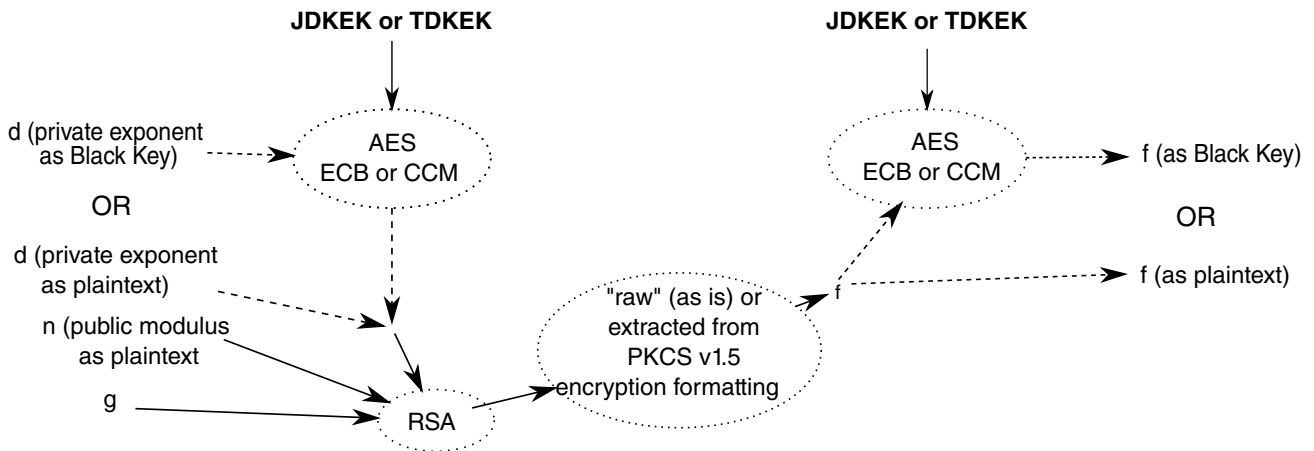


Figure 8-2. RSA decrypt operation - private key form #1

The PDB for private key form #1 is shown below. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 8-21. RSA decrypt PDB - private key form #1

SGF (4 bits)	Rsv (4 bits)	#d (12 bits)	#n (12 bits)
		Reference to g	
		Reference to f	
		Reference to n	
		Reference to d	

The fields #d and #n contain right-aligned 12-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. This figure shows the format of the SGF field.

Table 8-22. RSA decrypt PDB - private key form #1; SGF field

31	30	29	28
ref g	ref f	ref n	ref d
If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.			

The RSA decrypt operation also accepts the private key in the form (p, q, d). This form (form #2) of the RSA decrypt operation is illustrated in this figure.

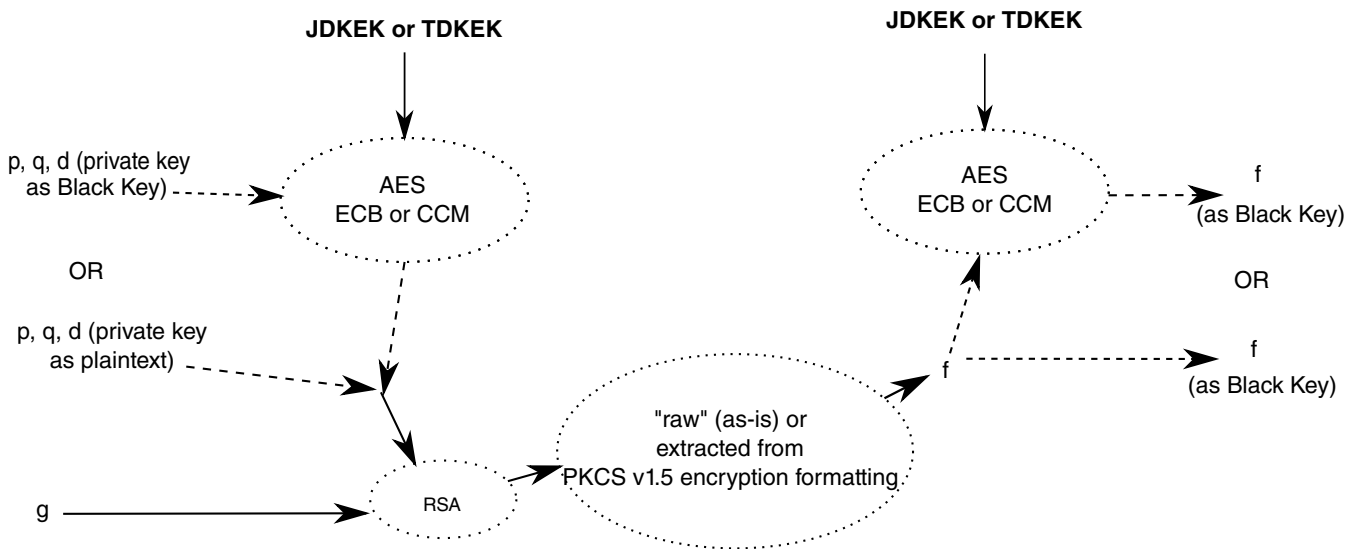


Figure 8-3. RSA decrypt operation - private key form #2

This figure shows the PDB for private key form #2. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 8-23. RSA decrypt PDB - private key form #2

SGF (7 bits)	Rsv (1 bit)	#d (12 bits)	#n (12 bits)
		Reference to g	
		Reference to f	
		Reference to d	
		Reference to p	
		Reference to q	
		Reference to tmp1	
		Reference to tmp2	
Reserved (8 bits)		#q (12 bits)	#p (12 bits)

Implementation of the RSA decrypt operation

The fields #d, #n, #q and #p contain right-aligned 12-bit values that indicate the size (plaintext sizes, in bytes) of d, n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. tmp1 needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp2 needs to be as long as q (either #q, or, if q is encrypted, as big as the encrypted value of q). This figure shows the format of the SGF field.

Table 8-24. RSA decrypt PDB - private key form #2; SGF field

31	30	29	28	27	26	25
ref g	ref f	ref d	ref p	ref q	ref tmp1	ref tmp2
If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.						

The RSA decrypt operation also accepts the private key in form #3, (p, q, dp, dq, c). dp, dq, and c are

- $dp = d \bmod p-1$
- $dq = d \bmod q-1$
- $c = q^{-1} \bmod p$

The operation this form of RSA decrypt is illustrated in this figure.

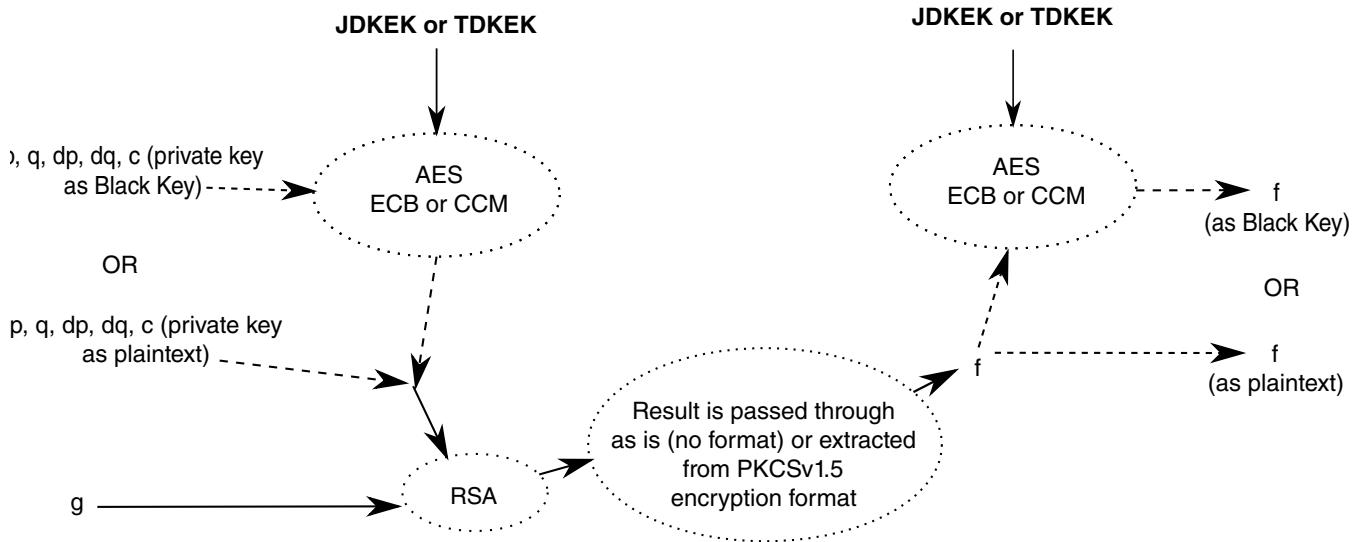


Figure 8-4. RSA Decrypt Operation - private key form #3

This figure shows the PDB for private key form #3. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 8-25. RSA decrypt PDB - private key form #3

SGF	Reserved	#n
-----	----------	----

Table continues on the next page...

Table 8-25. RSA decrypt PDB - private key form #3 (continued)

(9 bits)	(11 bits)	(12 bits)
	Reference to g	
	Reference to f	
	Reference to c	
	Reference to p	
	Reference to q	
	Reference to d_p	
	Reference to d_q	
	Reference to tmp1	
	Reference to tmp2	
Reserved (8 bits)	#q (12 bits)	#p (12 bits)

The fields #n, #q and #p contain right-aligned, 12-bit values that indicate the size (plaintext sizes, in bytes) of n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. Note that #dp and #c are assumed to be #p, and #dq is assumed to be #q. tmp1 needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp2 needs to be as long as q (either #q, or, if q is encrypted, as big as the encrypted value of q).

Table 8-26. RSA Decrypt PDB - private key form #3; SGF Field

31	30	29	28	27	26	25	24	23
ref g	ref f	ref c	ref p	ref q	ref dp	ref dq	ref tmp1	ref tmp2
NOTE: If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.								

Chapter 9

Protocol acceleration

SEC is designed to accelerate the cryptographic operations associated with various network protocols. These cryptographic operations can be implemented using the descriptor commands described in sections [KEY commands](#) through [SEQ OUT PTR command](#), but SEC also implements specialized descriptor commands for particular networking protocols. Each such command performs a sequence of operations that are equivalent to a series of the more general descriptor commands; for example, all the protocols in this section manage the input data and output FIFOs directly -- a SEQ FIFO LOAD command is required in a descriptor only if there is data in the input frame that is not to be handled by the protocol. These protocols often require that state information (for example, sequence numbers) be maintained per security association.

The specialized protocol commands implemented by SEC use data structures called protocol data blocks (PDBs) embedded within the descriptor to specify protocol options and hold state information. Typically these protocol commands and their associated PDBs are contained in shared descriptors, so that the same protocol options and state information can be shared among all the job descriptors that identify the PDUs within a particular security association. The PDB is embedded within the shared descriptor immediately following the header, and the START INDEX field in the header is used to skip over the PDB to continue executing the commands within the shared descriptor.

If the protocol requires that state information be updated, SEC writes the updated information back to the PDB in the shared descriptor located in system memory.

Sharing is described in [Shared descriptors](#). Sharing significantly impacts Protocol operation in particular, because SEC protocols tend to use a shared descriptor PDB to share state across many jobs within a flow. One example is a sequence or packet number -- it is important that only one packet be encapsulated with a given sequence number. Therefore sharing type as described in [Table 7-1](#) is to be carefully considered when crafting a descriptor.

SEC is most efficient when using serial sharing -- there is extra time associated with moving shared descriptor material from one DECO to another. On the other hand, an idle DECO resulting from overuse of serial sharing is a greater cost. Therefore the overall set of applications should be considered.

SEC protocols maintain a lock called OK to Share in order to allow for wait sharing. For example, consider IPsec encapsulation using CBC mode. CBC requires every packet include an Initialization Vector (an IV). For IPsec, either the IV can be the final block of ciphertext from the previous packet (a Chained IV), or it can be a random value (a Random IV). The IPsec protocol state machine will block WAIT sharing of a shared descriptor until a Chained IV has been prepared and OK to Share is signalled. If instead a Random IV is used, OK to Share can be set as soon as the IPsec protocol state machine has updated Sequence Number in the PDB. It is probably not useful to use WAIT sharing with a Chained IV; two jobs from a single flow can only be present in multiple DECOs for a very limited period of time.

Never and Always Sharing should be used with extreme care. Selecting share type of Always will cause a shared descriptor to be shared between DECOs without consideration of state. In the IPsec encapsulation example, Always sharing can result in packets with duplicate sequence numbers. Duplicate sequence numbers can also result from using Never sharing, as a DECO will get a fresh copy of a Shared Descriptor from system memory, without any consideration for any pending writes to update the Shared Descriptor PDB from another DECO.

SEC includes built-in descriptor programming shortcuts for the following functions:

- IPsec ESP Encapsulation and Decapsulation
- SSL 3.0 Record Encapsulation and Decapsulation
- TLS 1.0, 1.1, and 1.2 Record Encapsulation and Decapsulation
- DTLS 1.0 and 1.2 Record Encapsulation and Decapsulation
- SRTP Packet Encapsulation and Decapsulation
- IEEE 802.1AEbw-2013 MACsec Encapsulation and Decapsulation
- IEEE 802.11-2012 WPA2 Encapsulation and Decapsulation for WiFi
- IEEE 802.16 WiMAX Encapsulation and Decapsulation
- 3G Double CRC
- 3G RLC Encryption and Decryption
- LTE PDCP Encapsulation and Decapsulation
- Cryptographic Blob Encapsulation and Decapsulation

Each detailed description of the function includes color-coded diagrams. [Figure 9-1](#) shows the color coding key. Note that in the diagrams, processing order is reflected top-to-bottom, and PDU content is reflected left-to-right.

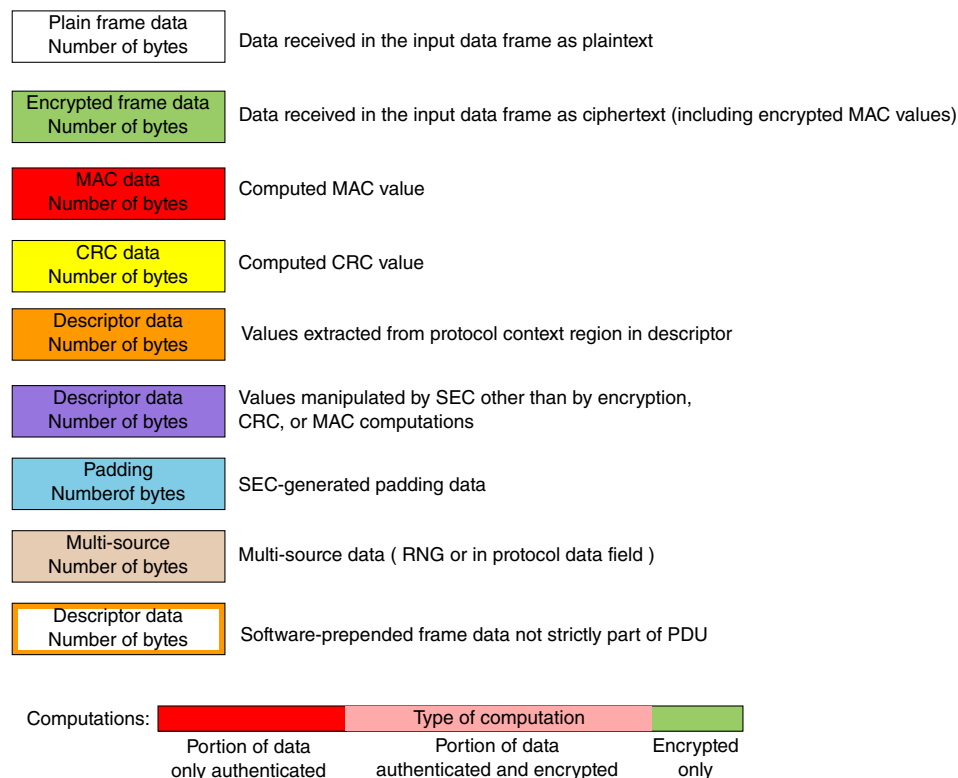


Figure 9-1. Protocol diagram color-coding key

9.1 IPsec ESP encapsulation and decapsulation overview

SEC's built-in IPsec protocol supports data encapsulation, encryption, and data integrity checking for the following cipher suites:

- DES-CBC or 3DES-CBC with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CBC with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CTR (AES-Counter) with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CCM
- AES-GCM

The PROTINFO field codes enumerated in [Table 7-52](#) are used to define the specific encryption and data integrity algorithms to be used by the protocols.

SEC supports different IP versions as follows:

- For IPv4, SEC supports zero or one 32-bit option field.

- For IPv6, SEC supports up to 96 bytes of extension headers.
- For all modes, SEC treats the option field and extension headers as part of the IP header.

SEC supports IPsec with null encryption. For the most part IPsec with null encryption processes the packet in a very similar fashion to how IPsec processing occurs with encryption. Null encryption is not supported when AES-CCM is selected. When AES-GCM is selected, null encryption uses the AES-GMAC algorithm. In this case, the datagram is processed identically to packet processing for AES-GCM (including the use of an IV), except that the payload is not encrypted. For IPsec null encryption when using any HMAC algorithm, no IV is included in the encapsulated frame. Otherwise, the encapsulated packet contains all the other fields, including padding and pad length.

SEC also supports IPsec with null authentication (with selected cipher algorithms). For null authentication, SEC does not produce an ICV; an ICV is not written to the encapsulation output frame, and no ICV check is performed during decapsulation. Null Authentication may be combined with Null encryption, in which case SEC produces an output frame that consists of an IP header followed by an ESP header, payload, and then an ESP trailer with minimum padding.

The shared descriptor protocol data block (PDB) includes a field that indicates the byte length of the IP header with options/extensions.

The following table summarizes the IPsec protocol descriptors.

Table 9-1. IPsec protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block includes next header, SPI, sequence number, IV (if not from RNG)		Protocol data block includes anti-replay information
Class 2 key data block		Class 2 key data block
Class 1 key data block		Class 1 key data block
Protocol = IPsec encrypt		Protocol = IPsec decrypt

NOTE

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a [split key](#). Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the [Derived Key Protocol](#) may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

9.1.1 IPsec ESP encapsulation and decapsulation mode support

SEC supports two modes: transport and tunnel.

- During tunnel mode encapsulation, SEC treats the entire input frame as payload except for the optional Outer IP header. During tunnel mode decapsulation, the original IP header (with options and extensions) is uncovered.
- During tunnel mode encapsulation, SEC can prepend a new IP header (with options and extensions). For decapsulation SEC must be given a pointer to the start of the encapsulated data, skipping over the Outer IP header.
- For transport mode, the pre-encapsulation IP header and post-encapsulation IP header are virtually identical, except for the 2-byte length field and the header checksum.
- For both tunnel and transport modes, SEC recomputes the length field in the header, can optionally recompute the header checksum.

The Protocol Data Block, or PDB, is used to store relevant parameters within a descriptor. Parts of the PDBs are specific to particular cipher suites, but most is common to either encapsulation or decapsulation. The common PDB for encapsulation is described in [PDB format for IPsec ESP Transport \(and Legacy Tunnel\) encapsulation](#), and for decapsulation in [Common PDB format descriptions for IPsec ESP Transport \(and Legacy Tunnel\) decapsulation](#).

Prior to SEC Era 8, what is now called ESP Transport (and legacy tunnel) covered both tunnel and transport modes. More recent developments to ESP Tunnel support, such as the addition of direct support for NAT, have necessitated the addition of the new ESP Tunnel encapsulation and decapsulation threads. For the foreseeable future, Tunnel mode IPsec encapsulation and decapsulation can be achieved by either the ESP Transport (and legacy tunnel) or the ESP Tunnel processing threads.

The PDBs for ESP Tunnel are similar to those for ESP Transport (and Legacy Tunnel) -- in particular that both share that there is a segment of the PDB specific to the chosen cipher suite. The common PDB for ESP Tunnel encapsulation is described in [PDB format for IPsec ESP Tunnel encapsulation](#) and for decapsulation in [Common PDB format descriptions for IPsec ESP Tunnel decapsulation](#).

9.1.2 IPsec ESP error codes

This table lists the conditions under which IPsec encapsulation or decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure in decapsulation can also produce an ICV check error.

Table 9-2. IPsec ESP encapsulation and decapsulation error conditions for both Tunnel and Transport operation

Condition	Error status	Applies to encapsulation/decapsulation
Input frame too long <ul style="list-style-type: none"> For IPsec ESP Transport (and legacy tunnel), the input frame is longer than 2¹⁶ bytes For IPsec ESP Tunnel, the input frame is longer than 2²⁰ bytes Note this length restriction applies to the input frame received at the start of the protocol itself. Any bytes that are part of the input frame that are dispositioned by commands prior to either IPsec ESP protocol are not governed by the size restriction.	Protocol size error	Both
Reserved bit set to 1 in the PDB options byte	Protocol PDB error	Both
ARS in PDB set to reserved setting	Protocol PDB error	Decapsulation only
Tun/Trsp = 1 and NH_OFFSET != 0	Protocol PDB error	Both
Inc IPHdr = 1 and PDB IP Hdr Length == 0	Protocol PDB error	Both
Inc IPHdr = 1 and PDB IP Hdr Length is not a multiple of 4	Protocol PDB error	Both
Tun/Trsp = 0, and NH_OFFSET points to a byte beyond PDB IP Hdr Length	Protocol PDB error	Both
OPERATION Command PROT ID selects IPsec Encap/Decap, and PROTINFO is not a valid protocol	Protocol Command Error	Both
ESN option = 0, and PDB SEQNUM is FFFFFFFF	Protocol Sequence Number Overflow	Encapsulation only
ESN option = 1, both ESN and SEQNUM are FFFFFFFF	Protocol Sequence Number Overflow	Encapsulation only
ESN option = 0, and SEQNUM overflows	Protocol Sequence Number Overflow	Decapsulation only
ESN option = 1, and [ESN and SEQNUM] overflows	Protocol Sequence Number Overflow	Decapsulation only
Anti-Replay detects a LATE packet	Protocol LATE error	Decapsulation only
Anti-Replay detects a REPLAY packet	Protocol REPLAY error	Decapsulation only
Output option 0 selected and AOFL option bit == 1	Protocol PDB error	Decapsulation only
UDP-encapsulated-ESP CE Drop See Manipulation of the Inner IP Header during ESP Tunnel decapsulation	CE DROP Error	Decapsulation only

9.1.3 Programming for IPsec

IPsec in SEC is designed to provide stateful encapsulation and decapsulation of IPsec ESP packets. The SEC representation of the ESP security association is the PDB -- the Protocol Data Block. SPI and Sequence Number are maintained in the PDB, as are other parameters and selectable options. The format of the PDB varies depending upon cipher suite and options selected, and are shown in sections following.

In some circumstances, it may be desirable to override some contents of the PDB on a frame-by-frame basis. Some capability is provided for that by programming the DECO Protocol Override register. Use and capability of that functionality is also shown in sections following.

9.1.3.1 PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation

IPsec ESP Transport (and Legacy Tunnel) encapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete encapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used), plus some cipher-suite-specific material, such as the IV used by AES-CBC and DES-CBC.

The PDB can, optionally, also provide for a common outer IP header to be written to the output frame prior to the tunnel-mode encapsulated header. The header material need not consist only of the outer IP header, but if extra material is included prior to the IP header (such as an ethernet header), then the several options that can manipulate the outer header (such as DSC and Cksm) will not work right.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follow afterwards. Details for the cipher-suite-specific portion are found:

- For AES-CBC and DES-CBC specific IV format, refer to [IPsec ESP encapsulation CBC-specific PDB segment format descriptions](#)
- For AES-CTR specific Counter and IV format, refer to [IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions](#)
- For AES-CCM specific data format, refer to [IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions](#)
- For AES-GCM specific Salt and IV format, refer to [IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions](#)

All fields shown here should be programmed as appropriate per the negotiated tunnel parameters. NH Offset is used only for transport mode. For more detail on fields other than HMO and Options, see [Process for IPsec ESP Transport \(and Legacy Tunnel\) encapsulation](#). HMO and Options are described below the PDB diagram.

Table 9-3. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB

		Descriptor Header (1 or 2 words)				
PDB Word 0	HMO (4 bits)	Reserved (4 bits)	Next Header (8 bits)	NH Offset (8 bits)	Options (8 bits)	
PDB Word 1	Optional Extended Sequence Number (ESN)					DECO writes back to PDB as needed
PDB Word 2	Sequence Number					
PDB Word 3	Cipher-suite-specific portion of PDB					for CBC-mode see CBC IV Format for CTR-mode see CTR IV Format for CCM-mode see CCM IV Format for GCM-mode see GCM IV Format
PDB Word 4						
PDB Word 5						
PDB Word 6						
PDB Word 7	SPI					
PDB Word 8	Reserved			Opt IP Header Length		
PDB Word 9	Optional IP Header (bytes 0-3)					
PDB Word 10	Optional IP Header (bytes 4-7)					ID field is incremented and written back as needed
PDB Word 11	Optional IP Header (bytes 8-11)					
PDB Word 12	Optional IP Header (bytes 12-15)					
PDB Word 13 +	Optional IP Header (bytes 16+)					

Table 9-4. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Cksm	DSC	IVsrc	ESN	IPHdr Src	Inc IPHdr	IPvsn	Tun/ Trsp

Table 9-5. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the options byte

Field	Description
7 Cksm	Enable Checksum Update 0 - Do not perform any checksum computations 1 - For any changes performed to the IP header, perform computations to update the header checksum
6 DSC	DiffServ Copy 0 - Do not copy 1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the inner IP header to the IP header copied from the PDB.

Table continues on the next page...

Table 9-5. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the options byte (continued)

Field	Description
5 IVsrc	IV Source 0 - Chained IV Stored in PDB 1 - Random IV fetched from RNG prior to starting encapsulation
4 ESN	Extended Sequence Number 0 - ESN not included in ICV computation. Optional ESN field of PDB is reserved in this case. 1 - ESN is copied from PDB and used for ICV computation but is not written to the output frame. ESN is incremented as if part of the sequence number and written back to the PDB as required.
3 IPHdrSrc	IP Header source 0 - IP header in input frame 1 - IP header in PDB
2 Inc IPHdr	Include Optional IP Header 0 - Do not prepend IP header 1 - Prepend IP Header to output frame
1 IPvsn	This field indicates which version of IP is being used. 0 - IPv4 1 - IPv6
0 Tun/Trsp	Selects Tunnel or Transport Mode with respect to handling of the N (Next Header) byte 0 - Transport mode 1 - Tunnel mode

This figure shows the format of the HMO field.

Table 9-6. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, format of the HMO field

31	30	29	28
Reserved	DFC	DTTL	SNR

Table 9-7. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the HMO field

Field	Description
31	Reserved
30 DFC	Copy DF bit 0 - Do not copy DF bit 1 - If an IPv4 tunnel mode outer IP header is coming from the PDB, copy the DF bit from the inner IP header to the outer IP header. If not in tunnel mode or the outer IP header from the PDB is not included, setting DFC = 1 causes a Protocol PDB error.
29	Decrement TTL (Hop Limit)

Table continues on the next page...

Table 9-7. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the HMO field (continued)

Field	Description
DTTL	0 - Do not decrement 1 - Based on the value of the Options Byte IPvsN field: if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated.
28 SNR	Sequence Number Rollover enable 0 - Sequence Number (as extended by ESN) does not rollover; an error is generated if a rollover is attempted 1 - Sequence Number (as extended by ESN) permitted to roll over

9.1.3.2 Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation

IPsec ESP Transport (and Legacy Tunnel) decapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete decapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used). SPI and Sequence Number can be used as extracted from the input frame, but the Extended Sequence Number is not included in the encapsulated datagram. PDB words 1 and 2 are reserved for some cipher-suite-specific material, such as the Salt used by AES-GCM. These words are reserved for CBC-based cipher suites because no other information is required.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follows afterwards. For the cipher-suite-specific portion, additional details can be found as follows:

- For AES-CBC and DES-CBC specific IV format, refer to [IPsec ESP decapsulation CBC-specific PDB segment format descriptions](#)
- For AES-CTR specific Counter and IV format, refer to [IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions](#)
- For AES-CCM specific data format, refer to [IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions](#)
- For AES-GCM specific Salt and IV format, refer to [IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions](#)

Fields shown here should be programmed as appropriate per the negotiated tunnel parameters. NH Offset is used only for transport mode. The Anti-replay scorecard should be initialized with zeros. For more detail on fields other than HMO and Options, see [IPsec ESP Transport \(and Legacy Tunnel\) decapsulation procedure overview](#). HMO and Options are described below the PDB diagram.

Table 9-8. IPsec ESP Transport (and Legacy Tunnel) decapsulation PDB

		Descriptor Header (1 or 2 words)			
PDB Word 0	HMO (4 bits)	IP Header Length (12 bits)	NH Offset (8 bits)	Options (8 bits)	
PDB Word 1	Cipher-suite-specific portion of PDB				unused for CBC-mode for CTR-mode see CTR IV Format for CCM-mode see CCM IV Format for GCM-mode see GCM IV Format
PDB Word 2					
PDB Word 3	Optional Extended Sequence Number (ESN)				DECO writes back to PDB as needed
PDB Word 4	Sequence Number				
PDB word 5	anti-replay scorecard 1 [present if ARS not 00b]				
PDB word 6	anti-replay scorecard 2 [present if ARS either 10b or 11b]				
PDB word 7	anti-replay scorecard 3 [present if ARS= 10b]				
PDB word 8	anti-replay scorecard 4 [present if ARS= 10b]				

Table 9-9. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, format of the options byte

7-6	5	4	3	2	1	0
ARS	CKSM	ESN	OUT_FMT	AOFL	IPVSN	TUN/TRSP

Table 9-10. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the options byte

Field	Description
7-6 ARS	Anti-replay window size. 00 - No anti-replay window 01 - 32-entry anti-replay window 10 - 128-entry anti-replay window 11 - 64-entry anti-replay window
5 CKSM	Checksum Enable 0 - Checksum calculation/verification disabled 1 - Checksum calculation/verification enabled
4 ESN	Include Extended Sequence Number: 0 - ESN not used. Optional ESN field of PDB is reserved in this case.

Table continues on the next page...

Table 9-10. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the options byte (continued)

Field	Description
	1 - ESN is used for ICV and anti-replay computations. The ESN is not found in the input frame
3 OUT_FMT	Output Frame format: 0 - All Input Frame fields copied to Output Frame 1 - Output Frame is just the decapsulated PDU
2 AOFL	Adjust Output Frame Length 0 - Don't adjust output frame length -- output frame length reflects output frame actually written to memory, including the padding, Pad Length, and Next Header fields. 1 - Adjust output frame length -- subtract the length of the padding, the Pad Length, and the Next Header byte from the output frame length reported to the frame consumer. If outFmt==0, this bit is reserved and must be zero.
1 IPVSN	This field indicates which version of IP is being used. 0 - IPv4 1 - IPv6
0 TUN/TRSP	Selects Tunnel or Transport Mode with respect to handling the N (Next Header) byte 0 - Transport mode 1 - Tunnel mode

Table 9-11. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, format of the HMO field

31	30	29	28
ODF	DFV	DTTL	DSC

Table 9-12. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the HMO field

Field	Description
31 ODF	ODF -- Override DF bit in IPv4 header of decapsulated output frame 0 - DF bit the IPv4 header in the output frame is identical to that in the input frame (encapsulated header in the input frame if in tunnel mode) 1 - DF bit in the IPv4 header in the output frame is replaced with the DFV value as shown below. If IPv6 is selected, then this bit is reserved and must be zero.
30 DFV	DFV -- DF bit Value If ODF (see above) is set, this bit replaces whatever value would have otherwise been placed in the IPv4 Header DF bit field. If ODF is not set, then this bit is reserved and must be zero.
29 DTTL	Decrement TTL (Hop Limit) 0 - Do not decrement

Table continues on the next page...

Table 9-12. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the HMO field (continued)

Field	Description
	1 - Based on the value of the Options Byte IPvsn field if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated.
28	DiffServ Copy
DSC	0 - Do not copy 1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the outer IP header to the inner IP header.

9.1.3.3 Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For IPsec ESP Transport (and Legacy Tunnel) encapsulation and decapsulation, several fields are maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired values in the job descriptor. For more information, see [Job Ring interface](#). When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field in the FD. For more information, see [Queue Manager Interface \(QI\)](#). When using the Advanced IO Processor interface, AI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field provided in the FD. For more information, see [Advanced IO Processor Interface \(AI\)](#).

When DPOVRD is selected for use, SEC then uses the following values as specified in DPVORD instead of as specified in the PDB:

- IP Header Length
- NH OFFSET,
- Next Header (encapsulation only)
- ECN

Note that the values in DPOVRD in no way affect the values stored in the descriptor's PDB. DPOVRD works the same way regardless of the format of the IPsec encapsulation and decapsulation PDB, although the PDB's format varies depending on the chosen cipher suite. For details on how each of these fields are used, please refer to the

appropriate section -- either [PDB format for IPsec ESP Transport \(and Legacy Tunnel\) encapsulation](#) or [Common PDB format descriptions for IPsec ESP Transport \(and Legacy Tunnel\) decapsulation](#).

Table 9-13. IPsec ESP Transport (and legacy Tunnel) format of the DPOVRD register for encapsulation and decapsulation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OVRD	Reserved			ECN				IP Header Length							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NH OFFSET								Next Header [Encapsulation only] Reserved for decapsulation							

Table 9-14. IPsec ESP Transport (and Legacy Tunnel) description of the DPOVRD register for encapsulation and decapsulation

Field	Description
31 OVRD	Selects whether DPOVRD overrides the values specified in the PDB. 0: Use the PDB as provided 1: Override values in PDB by using values in DPOVRD
30-28	Reserved
27-24 ECN	If OVRD = 1 and the MSB of the ECN field (bit 27) = 1, the two LSBs of the ECN field (bits 25 and 24) replace the ECN bits in the IP header written to the output frame as part of encapsulation or decapsulation.
23-16 IP Header Length	The length, in bytes, of the portion of the IP header that is not encrypted. If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame.
15-8 NH OFFSET	The location of the next header field within the IP header of the transport mode packet. This location is indicated by the number of bytes from the beginning of the IP header. For encapsulation, the value at this location is replaced with the contents of the Next Header byte (bits 7-0), and the value at the NH OFFSET location is used as the N byte in constructing the IPsec encapsulation trailer. For decapsulation, the reverse occurs; the N byte value decrypted and extracted from the ESP trailer replaces the Next Header byte at this offset into the IP header.
7-0 Encapsulation: Next Header Decapsulation: Reserved	For encapsulation, this value is placed in the transport mode header at location NH OFFSET, replacing the value that was in the Next Hdr field of that particular header or extension header. For decapsulation, this field is reserved.

9.1.3.4 PDB format for IPsec ESP Tunnel encapsulation

The PDB for IPsec ESP Tunnel encapsulation descriptors is very similar to the PDB used by IPsec ESP Transport (and Legacy Tunnel) encapsulation. However, the use of the HMO and Options fields has changed significantly.

The Transport encapsulation PDB field NH Ofst is used only for transport-mode applications; the Tunnel encapsulation PDB replaces that field with AOIPHO - Actual Outer IP Header Offset. Tunnel encapsulation can permit the so-called Outer IP Header material to include extra material, such as an ethernet header, which will be ignored for the purposes of modifying the Outer IP header based on inner IP header values.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follow afterwards. Details for the cipher-suite-specific portion are found:

- For AES-CBC and DES-CBC specific IV format, refer to [IPsec ESP encapsulation CBC-specific PDB segment format descriptions](#)
- For AES-CTR specific Counter and IV format, refer to [IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions](#)
- For AES-CCM specific data format, refer to [IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions](#)
- For AES-GCM specific Salt and IV format, refer to [IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions](#)

All fields shown here should be programmed as appropriate per the negotiated tunnel parameters. For more detail on fields other than HMO and Options, see [IPsec ESP Tunnel encapsulation overview](#). HMO and Options are described below the PDB diagram.

Table 9-15. IPsec ESP Tunnel encapsulation PDB

Descriptor Header (1 or 2 words)						
PDB Word 0	HMO (4 bits)	Reserved (4 bits)	Next Header (8 bits)	AOIPHO (8 bits)	Options (8 bits)	
PDB Word 1	Optional Extended Sequence Number (ESN)					DECO writes back to PDB as needed
PDB Word 2	Sequence Number					
PDB Word 3	Cipher-suite-specific portion of PDB					for CBC-mode see CBC IV Format
PDB Word 4						for CTR-mode see CTR IV Format
PDB Word 5						for CCM-mode see CCM IV Format
PDB Word 6						for GCM-mode see GCM IV Format
PDB Word 7	SPI					

Table continues on the next page...

Table 9-15. IPsec ESP Tunnel encapsulation PDB (continued)

PDB Word 8	Reserved	Opt IP Header Length	
PDB Word 9	Optional IP Header (bytes 0-3)		
PDB Word 10	Optional IP Header (bytes 4-7)		ID field is incremented and written back as needed
PDB Word 11	Optional IP Header (bytes 8-11)		
PDB Word 12	Optional IP Header (bytes 12-15)		
PDB Word 13 +	Optional IP Header (bytes 16+)		

Table 9-16. IPsec ESP Tunnel encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved		IVsrc	ESN	OIHI		NAT	NUC

Table 9-17. IPsec ESP Tunnel encapsulation, description of the options byte

Field	Description
7-6	Reserved
5 IVsrc	IV Source 0 - Chained IV Stored in PDB 1 - Random IV fetched from RNG prior to starting encapsulation
4 ESN	Extended Sequence Number 0 - ESN not included in ICV computation. Optional ESN field of PDB is reserved in this case. 1 - ESN is copied from PDB and used for ICV computation but is not written to the output frame. ESN is incremented as if part of the sequence number and written back to the PDB as required.
3-2 OIHI	Outer IP Header Included 00 - No Outer IP Header Provided 01 - Outer IP Header from input frame -- The input frame provided will include <i>two</i> IP headers; first the outer IP header, then the IP header of the datagram being encapsulated. The output frame will include the Outer IP Header (modified as necessary), followed by the ESP Header, followed by the encrypted datagram 10 - Outer IP Header Referenced by PDB - The output frame will include an outer IP header fetched from the address in the appropriate PDB field. Note the extra memory reads associated with fetching the outer IP header from independent memory cannot be prefetched or otherwise optimized, so a significant performance penalty results from using this option. 11 - Outer IP Header from PDB - The output frame will include the Outer IP Header (modified as necessary) extracted from the appropriate fields of the PDB.
1 NAT	Enable UDP-encapsulated-ESP as defined by RFC 3948 for traversing Network Address Translators (NATs). 0 - no UDP-encapsulated-ESP 1 - UDP-encapsulated-ESP enabled.
0	enables NAT checksum

Table 9-17. IPsec ESP Tunnel encapsulation, description of the options byte

Field	Description
NUC	0 - If NAT is enabled, then no UDP checksum is computed, and the resulting UDP header uses a zero-value checksum. 1 - If NAT is enabled, then a UDP checksum is computed and included in the UDP header. If NAT is not enabled, NUC must be zero, else an error will be generated.

This figure shows the format of the HMO field.

Table 9-18. IPsec ESP Tunnel encapsulation PDB, format of the HMO field

31	30	29	28
Reserved	DFC	DTTL	SNR

Table 9-19. IPsec ESP Tunnel encapsulation PDB, description of the HMO field

Field	Description
31	Reserved
30 DFC	Copy DF bit 0 - Do not copy DF bit 1 - If an IPv4 tunnel mode outer IP header is coming from the PDB, copy the DF bit from the inner IP header to the outer IP header. If not in tunnel mode or the outer IP header from the PDB is not included, setting DFC = 1 causes a Protocol PDB error.
29 DTTL	Decrement TTL (Hop Limit) 0 - Do not decrement 1 - Based on the value of the Options Byte IPvsN field: if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated.
28 SNR	Sequence Number Rollover enable 0 - Sequence Number (as extended by ESN) does not roll over; an error is generated if a rollover is attempted 1 - Sequence Number (as extended by ESN) permitted to roll over

9.1.3.5 Common PDB format descriptions for IPsec ESP Tunnel decapsulation

IPsec ESP Tunnel decapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete decapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used). SPI and Sequence Number can be used as extracted from the input frame, but the Extended Sequence Number is not included in the encapsulated datagram. PDB words 1 and 2 are reserved for some cipher-suite-specific material, such as the Salt

used by AES-GCM. These words are reserved for CBC-based cipher suites because no other information is required. All of this is common with the IPsec ESP Transport (and legacy tunnel) form of the decapsulation PDB. What is not common is the field AOIPHO (Actual Outer IP Header Offset), which provides an offset to the actual outer IP header in the input frame; allowing material preceding the IP header (such as an ethernet header) to be passed from input frame to output frame. This field replaces the ESP Transport PDB field called NH Offset, which is used only for Transport mode processing.

This PDB diagram shows the common form, and the ESP Tunnel definitions of the Options and HMO bits follows afterwards. For the cipher-suite-specific portion, additional details can be found as follows:

- For AES-CBC and DES-CBC specific IV format, refer to [IPsec ESP decapsulation CBC-specific PDB segment format descriptions](#)
- For AES-CTR specific Counter and IV format, refer to [IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions](#)
- For AES-CCM specific data format, refer to [IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions](#)
- For AES-GCM specific Salt and IV format, refer to [IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions](#)

Fields shown here should be programmed as appropriate per the negotiated tunnel parameters. The Anti-replay scorecard should be initialized with zeros. For more detail on fields other than HMO and Options, see [IPsec ESP tunnel decapsulation overview](#). HMO and Options are described below the PDB diagram.

Table 9-20. IPsec ESP Tunnel decapsulation PDB

	Descriptor Header (1 or 2 words)				
PDB Word 0	HMO (4 bits)	IP Header Length (12 bits)	AOIPHO (8 bits)	Options (8 bits)	
PDB Word 1	Cipher-suite-specific portion of PDB				unused for CBC-mode for CTR-mode see CTR IV Format for CCM-mode see CCM IV Format for GCM-mode see GCM IV Format
PDB Word 2					
PDB Word 3	Optional Extended Sequence Number (ESN)				DECO writes back to PDB as needed
PDB Word 4	Sequence Number				
PDB word 5	anti-replay scorecard 1 [present for any size of anti-replay window]				
PDB word 6	anti-replay scorecard 2 [present if anti-replay window size exceeds 32]				
PDB word 7	anti-replay scorecard 3 [present if anti-replay window size exceeds 64]				
PDB word 8	anti-replay scorecard 4 [present if anti-replay window size exceeds 96]				

Table 9-21. IPsec ESP Tunnel Common decapsulation PDB, format of the options byte

7-6	5	4	3	2	1	0
ARS	rsv	ESN	rsv	rsv	rsv	ETU

Table 9-22. IPsec ESP Tunnel Common decapsulation PDB, description of the options byte

Field	Description
7-6 ARS	Anti-replay window size. 00 - No anti-replay window 01 - 32-entry anti-replay window 10 - 128-entry anti-replay window 11 - 64-entry anti-replay window
5 rsv	reserved
4 ESN	Include Extended Sequence Number: 0 - ESN not used. Optional ESN field of PDB is reserved in this case. 1 - ESN is used for ICV and anti-replay computations. The ESN is not found in the input frame
3	Reserved
2-1	Reserved
0 ETU	EtherType update 0 - AOIPHO-defined material preceeding outer IP header copied as is to output frame 1 - AOIPHO-defined material preceeding outer IP header copied, except last two bytes are updated to proper EtherType value for the IP header following -- 0800h for IPv4, and 86ddh for IPv6

Table 9-23. IPsec ESP Tunnel Common decapsulation PDB, format of the HMO field

31	30	29	28
ODF	DFV	DTTL	DSC

Table 9-24. IPsec ESP Tunnel Common decapsulation PDB, description of the HMO field

Field	Description
31 ODF	ODF -- Override DF bit in IPv4 header of decapsulated output frame 0 - DF bit the IPv4 header in the output frame is identical to that in the input frame (encapsulated header in the input frame if in tunnel mode) 1 - DF bit in the IPv4 header in the output frame is replaced with the DFV value as shown below. If IPv6 is selected, then this bit is reserved and must be zero.
30 DFV	DFV -- DF bit Value If ODF (see above) is set, this bit replaces whatever value would have otherwise been placed in the IPv4 Header DF bit field. If ODF is not set, then this bit is reserved and must be zero.

Table continues on the next page...

Table 9-24. IPsec ESP Tunnel Common decapsulation PDB, description of the HMO field (continued)

Field	Description
29 DTTL	Decrement TTL (Hop Limit) 0 - Do not decrement 1 - Based on the value of the Options Byte IPVsn field if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated.
28 DSC	DiffServ Copy 0 - Do not copy 1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the outer IP header to the inner IP header.

9.1.3.6 Overriding ESP Tunnel PDB content with the DECO Protocol Override Register

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For IPsec ESP Tunnel encapsulation and decapsulation, several fields are maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired values in the job descriptor. For more information, see [Job Ring interface](#). When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field in the FD. For more information, see [Queue Manager Interface \(QI\)](#). When using the Advanced IO Processor interface, AI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field provided in the FD. For more information, see [Advanced IO Processor Interface \(AI\)](#).

When DPOVRD is selected for use, SEC then uses the following values as specified in DPVORD instead of as specified in the PDB for both encapsulation and decapsulation:

- Outer IP Header Material Length
- AOIPHO
- OIMIF (encapsulation only)
- Next Header (encapsulation only)

Note that the values in DPOVRD in no way affect the values stored in the descriptor's PDB. DPOVRD works the same way regardless of the format of the IPsec encapsulation and decapsulation PDB, although the PDB's format varies depending on the chosen

cipher suite. For details on how each of these fields are used, please refer to the appropriate section -- either [PDB format for IPsec ESP Tunnel encapsulation](#) or [Common PDB format descriptions for IPsec ESP Tunnel decapsulation](#).

The following table shows the form of the DPOVRD register used for encapsulation. A separate format is used for decapsulation, and is shown below.

Table 9-25. IPsec ESP Tunnel format of the DPOVRD register for encapsulation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OVRD	Reserved			Outer IP Header Material Length											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OIMIF	AOIPHO							Next Header							

Table 9-26. IPsec ESP Tunnel description of the DPOVRD register for encapsulation

Field	Description
31 OVRD	Selects whether DPOVRD overrides the values specified in the PDB. 0: Use the PDB as provided 1: Override values in PDB by using values in DPOVRD
30-28	Reserved
27-16 Outer IP Header Material Length	The length, in bytes, of the portion of the IP header that is not encrypted. If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame.
15 OIMIF	Outer IP Header Material in Input Frame 0: Use Outer IP Header Material as specified by PDB 1: For encapsulating this datagram, use Outer IP Header material found in input frame. Length specified by DPOVRD register value Outer IP Header Material Length; AOIPHO specifies offset into material for actual Outer IP Header.
14	Reserved
13-8 AOIPHO	Actual Outer IP Header Offset This allows the Outer IP Header Material to be preceeded by some additional content, such as an Ethernet header. The value in this field indicates where the actual Outer IP Header starts in the material provided.
7-0	Next Header Used in the Next Header field of the encapsulated payload.

The following table shows the form of the DPOVRD register used for decapsulation. A separate format is used for encapsulation, and is shown above.

Table 9-27. IPsec ESP Tunnel format of the DPOVRD register for decapsulation

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OVRD	Reserved											AOIPHO			

Table continues on the next page...

Table 9-27. IPsec ESP Tunnel format of the DPOVRD register for decapsulation (continued)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AOIPHO (continued)				Outer IP Header Material Length											

Table 9-28. IPsec ESP Tunnel description of the DPOVRD register for decapsulation

Field	Description
31 OVRD	Selects whether DPOVRD overrides the values specified in the PDB. 0: Use the PDB as provided 1: Override values in PDB by using values in DPOVRD
30-20	Reserved
19-12 AOIPHO	Actual Outer IP Header Offset This allows the Outer IP Header Material to be preceeded by some additional content, such as an Ethernet header. The value in this field indicates where the actual Outer IP Header starts in the material provided.
15 OIMIF	Outer IP Header Material in Input Frame 0: Use Outer IP Header Material as specified by PDB 1: For encapsulating this datagram, use Outer IP Header material found in input frame. Length specified by DPOVRD register value Outer IP Header Material Length; AOIPHO specifies offset into material for actual Outer IP Header.
11-0 Outer IP Header Material Length	The length, in bytes, of the portion of the IP header that is not encrypted. If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame.

9.1.3.7 IPsec ESP encapsulation CBC-specific PDB segment format descriptions

These figures show the format of the segment of the IPsec ESP encapsulation PDB used with CBC-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

Table 9-29. CBC-specific segment of IPsec ESP encapsulation PDB

	common PDB above (common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15	
PDB Word 3	IV bytes 0-3	DECO writes back to PDB as needed
PDB Word 4	IV bytes 4-7	
PDB Word 5	IV bytes 8-11 (AES only - reserved for DES)	
PDB Word 6	IV bytes 12-15	

Table continues on the next page...

Table 9-29. CBC-specific segment of IPsec ESP encapsulation PDB (continued)

	(AES only - reserved for DES)	
	more common PDB below	

9.1.3.8 IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-CTR-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

Table 9-30. AES-CTR specific segment of IPsec ESP encapsulation PDB

	common PDB above (common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15		
PDB Word 3	Counter Nonce		
PDB Word 4	Counter Initial Count		
PDB Word 5	IV bytes 0-3		DECO writes back to PDB as needed
PDB Word 6	IV bytes 4-7		
	more common PDB below		

9.1.3.9 IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-CCM cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

Table 9-31. AES-CCM specific segment of IPsec ESP encapsulation PDB

	common PDB above (common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15			
PDB Word 3	Reserved (8 bits)	Salt (24 bits)		
PDB Word 4	B ₀ Flags (8 bits)	CTR ₀ Flags (8 bits)	Counter Initial Count (16 bits)	

Table continues on the next page...

Table 9-31. AES-CCM specific segment of IPsec ESP encapsulation PDB (continued)

PDB Word 5	IV bytes 0-3	
PDB Word 6	IV bytes 4-7	
	more common PDB below	

9.1.3.10 IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-GCM-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

Table 9-32. AES-GCM specific segment of IPsec ESP encapsulation PDB

	common PDB above (common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15	
PDB Word 3	Salt	
PDB Word 4	Reserved	
PDB Word 5	IV bytes 0-3	
PDB Word 6	IV bytes 4-7	
	more common PDB below	

9.1.3.11 IPsec ESP decapsulation CBC-specific PDB segment format descriptions

For both IPsec ESP decapsulation PDBs -- ESP Tunnel and ESP Transport (and legacy tunnel) -- the two words reserved for cipher-specific additions are unused and therefore reserved.

Table 9-33. CBC-specific segment of IPsec ESP decapsulation PDB

	descriptor header above	
PDB Word 1	Reserved	
PDB Word 2	Reserved	
	more common PDB below (common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20	

9.1.3.12 IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions

The initial counter value required for decapsulation using counter mode is built using the Nonce value assigned to the security association and the IPsec IV extracted from the ESP header. The Nonce value is provided in the AES-CTR-specific portion of the decapsulation PDB in word 1. The lower 4 bytes of the initial counter value is specified using PDB word 2. RFC 3686 calls this segment the Block Counter, and specifies an initial value of 00000001h.

This PDB segment is common to both ESP Tunnel and ESP Transport PDBs.

Table 9-34. AES-CTR specific segment of IPsec ESP decapsulation PDB

	descriptor header above		
PDB Word 1	Counter Nonce		
PDB Word 2	Counter Initial Count		
	more common PDB below (common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20		

9.1.3.13 IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP decapsulation PDB used with AES-CCM cipher suites. This form is common to both the IPsec ESP Tunnel and ESP Transport PDBs.

Table 9-35. AES-CCM specific segment of IPsec ESP decapsulation PDB

	descriptor header above			
PDB Word 1	Reserved (8 bits)	Salt (24 bits)		
PDB Word 2	B ₀ Flags (8 bits)	CTR ₀ Flags (8 bits)	Counter Initial Count (16 bits)	
	more common PDB below (common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20			

9.1.3.14 IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions

The GCM IV required for decapsulation is built using the Salt value assigned to the security association and the IPsec IV extracted from the ESP header. The salt value is provided in the GCM-specific portion of the decapsulation PDB in word 1 for both ESP Tunnel and ESP Transport PDB forms.

Table 9-36. AES-GCM specific segment of IPsec ESP decapsulation PDB

	descriptor header above	
PDB Word 1	Salt	
PDB Word 2	Reserved	
	more common PDB below (common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20	

9.1.4 IPsec ESP Transport (and Legacy Tunnel) encapsulation overview

SEC supports tunnel and transport encapsulation of either complete IP packets or of IP payloads, per IPsec ESP requirements. The IPsec ESP Transport hardware (as specified in the PROTOCOL OPERATION COMMAND with PROTID=01) is designed to support Transport mode ESP datagram encapsulation. Limited Tunnel mode support is also provided, but may be deprecated in future SEC hardware revisions; additional ESP Tunnel mode support is (and will continue to be) supported by the [IPsec ESP Tunnel](#) threads.

Once the payload is identified, cryptographic encapsulation proceeds identically for Transport mode datagrams, for Tunnel mode datagrams when using the ESP Transport (and legacy tunnel) encapsulation thread, and for Tunnel mode datagrams when using the ESP Tunnel encapsulation thread. This procedure is described in [IPsec ESP Cryptographic Encapsulation](#).

As part of encapsulation, SEC updates the length field of the IP header. This field has different characteristics depending on whether your device uses IPv4 or IPv6. The following table summarizes these differences:

Table 9-37. Differences in the length field by IPrev

Characteristic	IPv4	IPv6
Name	IP Total Length	Payload Length
Byte range	2:3	4:5

Table continues on the next page...

Table 9-37. Differences in the length field by IPrev (continued)

Characteristic	IPv4	IPv6
What includes	Length of IP header itself	Length of all extension headers; does not include the length of the IP header

NOTE

To handle the differences between IP revs in terms of what the length field includes, the ESP Tunnel protocol assumes that the length field provided with the IP header is appropriate for IPv4 or IPv6. In other words, it assumes that the IP header Total Length field (IPv4) includes the length of the IP header and options whereas the IP header Payload Length field (IPv6) does not include the length of the IP header and extensions. (ESP Transport continues to use the PDB option bit that indicates IPversion and ignores the IPversion field within the IP header altogether.)

Note that SEC can optionally update the header checksum based on other changes made to the header. This update is valid only if the checksum was correct for the header prior to any other changes made.

9.1.4.1 Encapsulating the IP header in tunnel mode

For tunnel mode, the IP header is encapsulated as part of the payload. SEC can prepend a new IP header from the PDB or the input frame after it computes the proper length value. It computes this value as follows:

SEC-added ESP Header (SPI, Seq Num, and IV) length + the payload + ESP Trailer (padding, Pad Len, N, and ICV) length = proper length value

Note that for the final length field to be correct, the correct length value must have been programmed into the PDB field titled "Opt IP Hdr Length".

For IPv4, prior to RFC 6864, the IP Header identification field was required to be unique for each packet transmitted. Therefore, if SEC is prepending an IP Header from the PDB, it increments by 1 the bytes corresponding to the IP Header Identification field in the PDB, per RFC 791.

9.1.4.2 Encapsulating the IP header in transport mode

No new IP header is prepended in transport mode. Instead, SEC inserts all these added fields after the received IP header and updates the length to account for all the bytes SEC adds: those in the ESP Header and the ESP Trailer. Note that SEC does not add bytes for the payload length because it should be included in the original IP header length field.

9.1.4.3 Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation

This figure shows an example of the IPsec ESP Transport (and Legacy Tunnel) encapsulation processing-sequence.

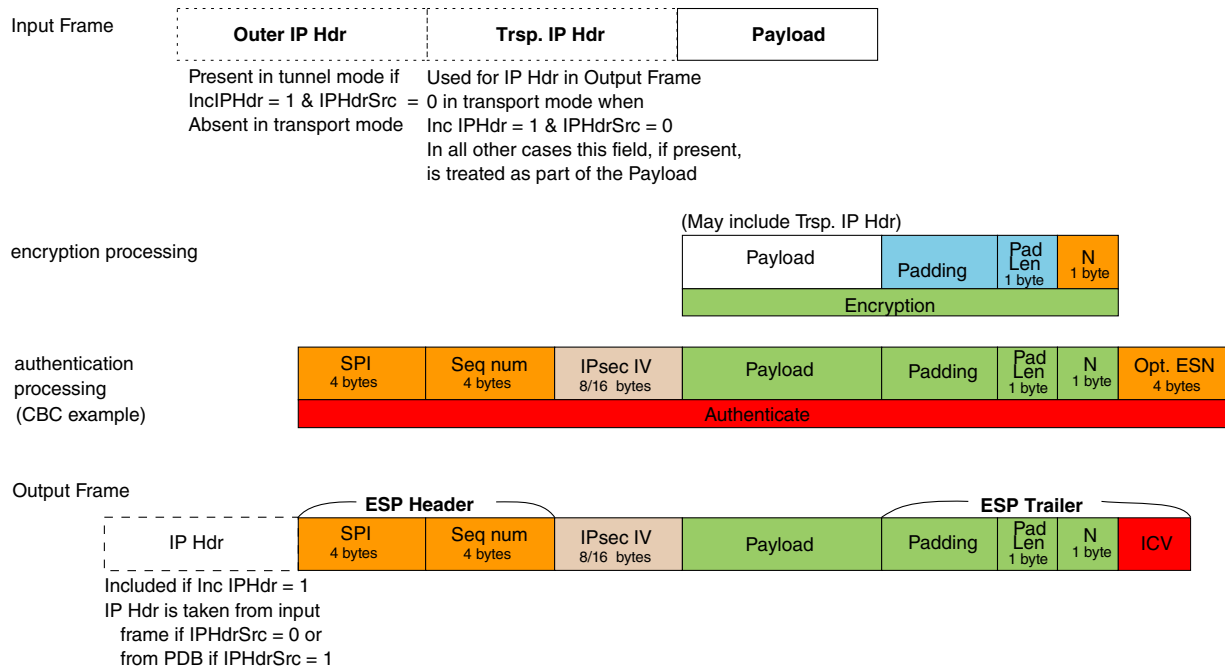


Figure 9-2. Example IPsec ESP Transport (and Legacy Tunnel) encapsulation processing sequence (DES-CBC, AES-CBC, or AES-CTR)

As shown above, processing begins when SEC receives an input frame. Note that the Transport IP Header is optional for transport mode because SEC does not authenticate or encrypt the IP header. The transport IP header may be omitted in transport mode by clearing IncIPHdr, setting IPHdrSrc, or clearing IP Header Length. For tunnel mode, the input frame must include an IP header because SEC must authenticate and encrypt this header as part of the payload.

When a IP header is included in transport mode, the N (Next Header) byte receives special treatment if the NH_OFFSET byte of the PDB is set to a non-zero value, as follows:

- If transport mode ESP with IPv4 for any non-zero value of NH_OFFSET (typically set to 01h), the N byte used in the ESP trailer comes from byte 9 of the IP header, and byte 9 of the IP header is replaced with the Next Header byte from the PDB.
- If transport mode ESP with IPv6 and NH_OFFSET = 01h, the N byte used in the ESP trailer comes from byte 6 of the IP header, and byte 6 of the IP header is replaced with the Next Header byte from the PDB.
- If transport mode ESP with IPv6 and NH_OFFSET > 01h, the N byte used in the ESP trailer comes from byte (NH_OFFSET x 8) of the IP header, and byte (NH_OFFSET x 8) of the IP header is replaced by the Next Header byte from the PDB.
- In all cases, if NH_OFFSET contains zero, no swap occurs and the ESP trailer N byte is copied directly from the PDB Next Header byte, as in tunnel mode.

As shown in [Process for IPsec ESP Transport \(and Legacy Tunnel\) encapsulation](#), the output frame results from SEC IPsec encapsulation regardless of the cipher suite used.

9.1.5 IPsec ESP Cryptographic Encapsulation

Construction of the ESP Header, Payload and ESP trailer is common for both ESP Tunnel and Transport modes, as are the cryptographic processes involved in encapsulating. However, the procedure and set up of cryptographic context differs, depending upon the cipher suite chosen.

9.1.5.1 Process for IPsec encapsulation when using AES-CBC or DES-CBC

This figure shows stages of IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation.

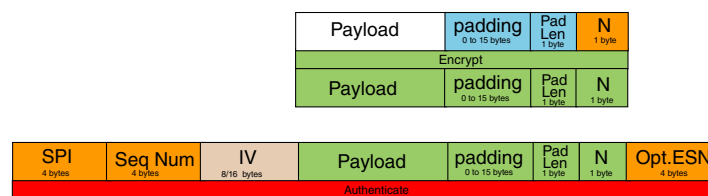


Figure 9-3. Stages of IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CBC or DES-CBC

When using the AES-CBC or DES-CBC cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Begins by prepending authenticate-only data (the SPI and Seq Num found in the PDB) to the payload. The authenticate-only data is passed to the output frame for transmission and in parallel is passed to the authentication CHA.
2. Appends 8- or 16-byte IV (IV size is the cipher block size) after the sequence number and sends it to the authentication CHA (typically a Class 2 CHA); in parallel, it sends it to the output frame.

NOTE

The IV can come from two possible sources, depending on the setting of the IVsrc field in the PDB Options Byte:

- The RNG can generate it randomly for a packet (IVsrc = 1).
- The IV can be a chained IV, meaning that the final block of ciphertext from the previous packet is used as the IV for the next packet (IVsrc = 0).

3. Writes the IV also into the first locations of the Context Register for the Class 1 CHA
4. After encryption is completed, writes the last block of ciphertext back to the PDB and Seq Num is incremented and updated in the PDB in memory

After the IV is in place, the Payload itself is fetched from the Input Frame and encrypted. The result of the encryption process is pushed onto the output frame, and is also pushed into the authentication CHA (normally Class 2). SEC generates the appropriate padding of monotonically increasing bytes, the first byte having the value 01h. SEC computes the Pad Length (padding such that Next Header is the last byte in a cryptographic block), and it appends Next Header (N) as found in the PDB. The padding, Pad Length, and Next Header are encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. If an ESN is found in the PDB, it is the last thing given to the authentication CHA. The ESN is incremented whenever the Seq Num rolls over.

For AES-CBC and DES-CBC, the update to the header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8 or 16-byte IV
- Padding (0-7 bytes for DES, 0-15 bytes for AES)
- 1-byte pad length
- 1-byte Next (N) field
- the ICV (the length of which is dependent on the cipher suite chosen)

Null Authentication may be chosen along with AES-CBC or DES-CBC confidentiality. If null authentication is chosen, then encryption proceeds as described before, steps that cause data to be authenticated are skipped. As no ICV is generated, writing of ICV to the output frame is also skipped.

9.1.5.2 Process for IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CTR

This figure shows IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CTR.

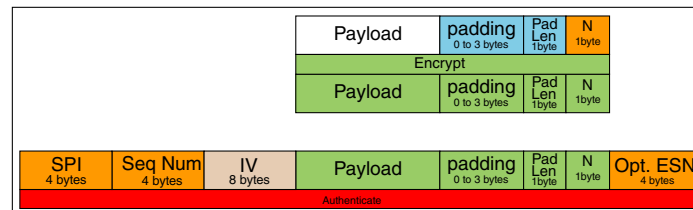


Figure 9-4. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CTR

When using an AES-CTR-based cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Begins by prepending authenticate-only data to the payload
2. Passes the SPI and sequence number found in the PDB to the output frame for transmission and to the authentication CHA
3. Appends the IV after the sequence number for authentication (typically a Class 2 operation) and the Output Frame.

NOTE

The IV can come from two possible sources, depending on the setting of the IVsrc field in the PDB Options byte

- The RNG can generate it for a packet (IVsrc = 1)
- The IV value within the PDB can be treated as a pseudo-sequence number (IVsrc = 0).

If the second option is chosen, the actual sequence number and the pseudo-sequence number are incremented and updated after use in the PDB in memory.

4. Does not write the 8-byte IV to the Class 1 Context Register directly, but combines it with a 4-byte nonce value and a 4-byte initial count constant (0000 0001h per RFC 3686). Both constants are static in the PDB.



Figure 9-5. Initial counter construction

5. Writes the 16-byte counter value into the counter value segment of the Class 1 Context Register, at offset 16.
6. Following generation of the authentication-only data prepended to the payload, fetches the payload itself from the input frame and encrypts it. The result of the encryption process is pushed onto the output frame, and is also pushed into the authentication CHA (normally Class 2).
7. Applies monotonically increasing padding bytes, followed by a Pad Len byte, such that the end of the frame after ICV ends on a 4-byte boundary. SEC next appends Next Header (N) as found in the PDB. The Next Header is encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. If an ESN is found in the PDB, it is the last thing given to the authentication CHA. The ESN is incremented whenever the Seq Num rolls over.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8-byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- Next byte
- The ICV (the length of which is dependent on the cipher suite chosen).

Null Authentication may be chosen along with AES-CTR confidentiality. If null authentication is chosen, then encryption proceeds as described before, steps that cause data to be authenticated are skipped. As no ICV is generated, writing of ICV to the output frame is also skipped.

9.1.5.3 Process for IPsec encapsulation when using AES-CCM

When using the AES-CCM cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Constructs the CCM B_0 and Initial Counter (CTR_0) values from Flag bytes, the Salt, and the IPsec IV as follows:
 - A nonce is constructed by appending the 8-byte IPsec IV to a 3-byte salt value. The salt value is a static value stored in the PDB. The IPsec IV is either a

pseudo-sequence number in the PDB that is incremented after use or a random number.

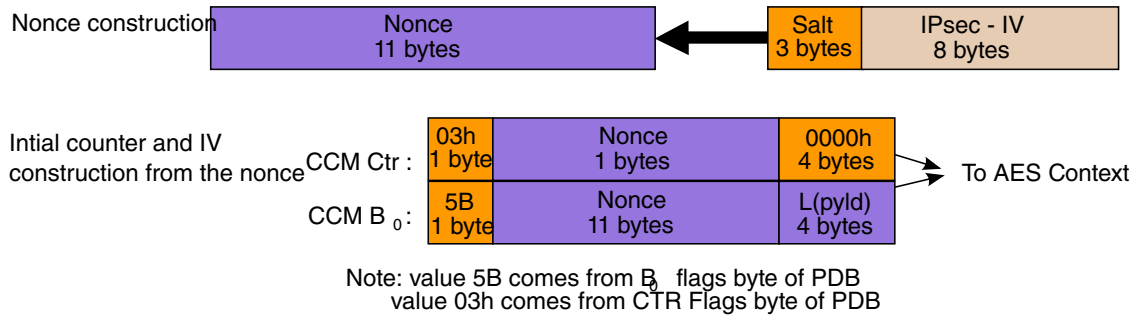


Figure 9-6. IPsec AES-CCM context construction

- To construct the Initial Counter value, SEC prepends the nonce with the counter field flags, which is a static byte from the PDB (with a value of 03h), and appends another 4 bytes that remain static in the PDB.
 - To construct the CCM B₀, SEC uses the B₀ flags byte of the PDB, whose value must be selected by the user according to the size of ICV transmitted.
 - For an 8-byte ICV, select a value of 5Bh.
 - For a 12-byte ICV, select a value of 6Bh.
 - For a 16-byte ICV, select a value of 7Bh.
 - Note that the cipher suite value in the Protocol Command selects the ICV size.
- Writes the CCM B₀ and Initial Counter values to the Class 1 Context Register (16 bytes starting with offset zero receive the CCM B₀; the CCM Ctr immediately follows).
 - After programming the Class 1 Context Registers, prepends authenticate-only data (AAD) to the payload (see the following figure).

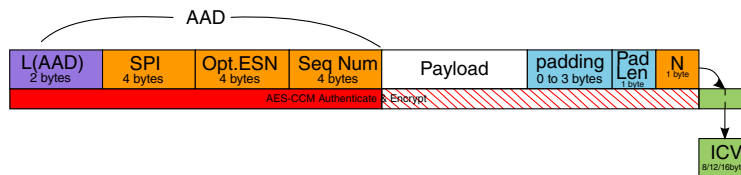


Figure 9-7. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CCM

- The AES-CCM mode encapsulation data starts with the formatted AAD, which consists of a 2-byte field indicating the length of AAD field, and the ESP Header, which consists of:
 - SPI and Seq Num from the PDB (which are also passed to the output frame)
 - ESN from the PDB and if enabled (ESN is not passed to the output frame).
- The formatted AAD includes zero-padding to the nearest block (16-byte boundary) and is passed into the input-data FIFO, but not into the output frame.

- c. The 8-byte IPsec IV is appended after the sequence number to the output frame, but is not written to the input-data FIFO.
4. Following generation of the formatted AAD, fetches the payload from the input frame and encrypts it.
5. The result of the encryption process is pushed onto the output frame; applies monotonically increasing padding bytes, followed by a Pad Len byte, such that the end of the frame after ICV ends on a 4-byte boundary.
6. Appends Next Header (N) as found in the PDB; N is encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. For combination algorithms (that is, unlike the AES-CBC), the ESN is appended to the AAD after the SPI. ESN, which is found in the PDB, is incremented whenever the sequence number rolls over.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8-byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- The Next byte
- The ICV (the length of which is dependent on the cipher suite chosen. The ICV may be 8, 12, or 16 bytes, depending upon the PROTINFO code provided.)

9.1.5.4 Process for IPsec encapsulation when using AES-GCM

When using the AES-GCM cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Constructs the nonce, which is a 12-byte GCM IV.
 - The GCM IV is created by appending the IPsec IV value to the 4-byte Salt value (see [Figure 9-8](#)).
 - The Salt is a static value found in the PDB.
 - The IPsec IV value is either a pseudo-sequence number in the PDB that is incremented after use or it is a random number.
 - Per RFC 4106, Salt concatenated with IV creates a value called the Nonce. This same value is called an IV (a GCM-IV) by specifications defining AES-GCM, such as NIST document SP800-38D.

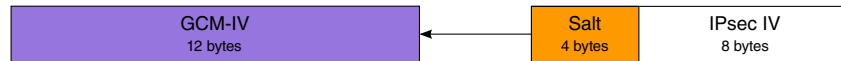


Figure 9-8. IPsec GCM-IV building

2. DECO writes the GCM-IV to the input-data FIFO after padding it with zeros to the nearest block (16-byte) boundary.
3. The 8- or 12-byte authentication-only data (AAD) is zero padded by the DECO to the nearest block (16-byte) boundary and then written to the input-data FIFO
 - The AAD consists of the 4-byte SPI, the 4-byte ESN (if present), and the 4 byte Seq Num, concatenated together (see [Figure 9-9](#)).
 - The SPI and Seq Num are written to the output frame (in addition to the input-data FIFO) and followed by the IPsec IV.
 - ESN is written to the input-data FIFO only and not to the output frame.

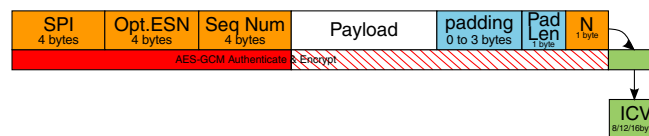


Figure 9-9. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-GCM

4. Inputs the payload, monotonically increasing padding bytes, a Pad Len byte and the next header (N) to the input-data FIFO, and encrypts these.
5. Pops the output-data FIFO off the encrypted result and writes it to the output frame.
6. Writes AES-GCM ICV to the output frame.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8 byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- the Next byte
- ICV (the length of which is dependent on the cipher suite chosen by the PROTINFO field -- see [Table 7-52](#))

9.1.6 IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure overview

SEC is designed to decapsulate received IPsec ESP Transport (and limited Tunnel mode) packets. The following figure provides an illustration of the procedure for both modes. For mode-specific details, see [IPsec ESP Transport Mode outer IP header decapsulation](#)

procedure or IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode). For details on cryptographic processing for both Tunnel and Transport modes, see [IPsec ESP Cryptographic Decapsulation](#).

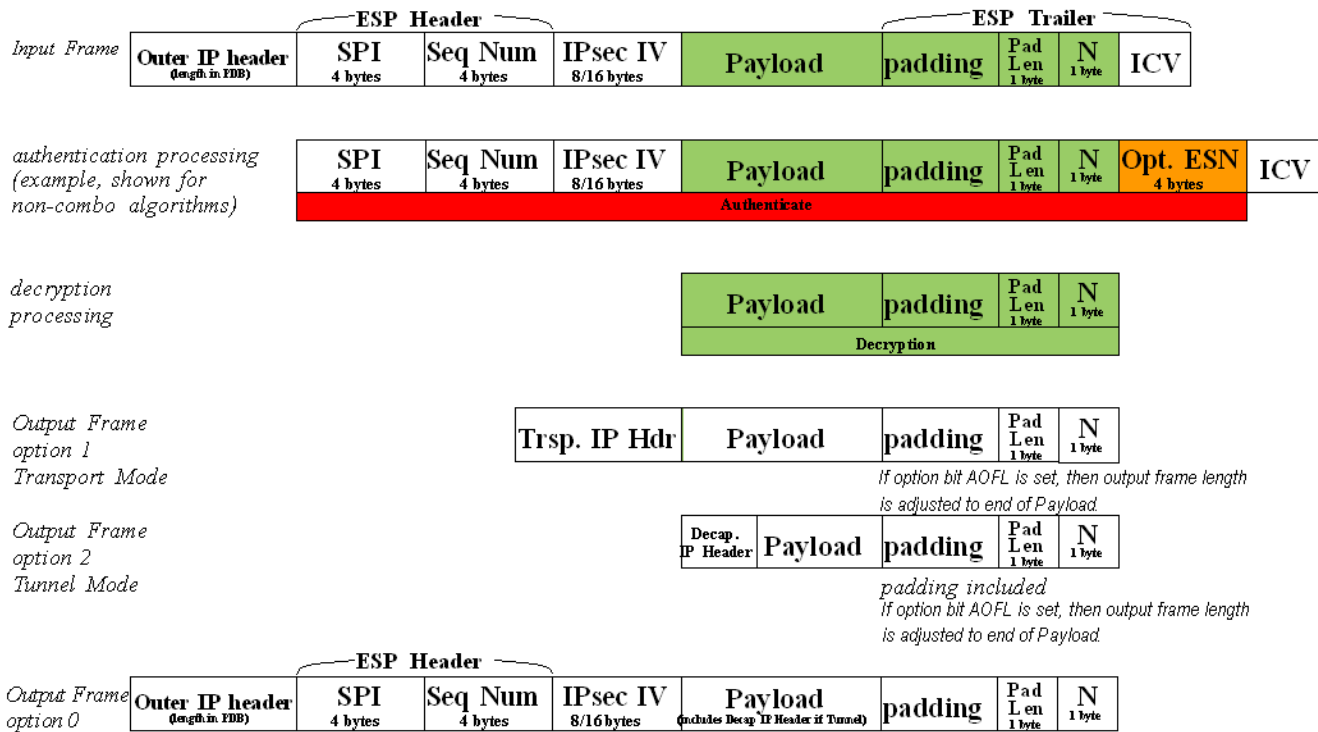


Figure 9-10. IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure for both transport and tunnel modes

SEC does the following to decapsulate an IPsec ESP datagram:

1. Receives an input frame as illustrated in [Figure 9-10](#).
2. Processes the input frame (minus the outer IP header) for authentication. Each mode handles the outer IP header differently; see their individual sections for details.
3. Optionally updates the checksum of the valid IP header being written to the output frame, if other changes are being made to the IP header. Note the validity of the updated checksum depends upon the original checksum being valid to the IP header before any changes were made.
4. Decrypts the payload, padding, pad length field (Pad Len), and next header (N) field. Note that these fields are marked with green shading in [Figure 9-10](#). For details on cryptographic processing for both Tunnel and Transport modes, see [IPsec ESP Cryptographic Decapsulation](#).
5. Outputs the resulting output frame based on the selected mode and options.

Note that SEC does not have enough available buffering to decide how much padding to drop for all possible values of the Pad Len byte at the end of the frame. As a result, the output frame in options 1 and 2 includes the padding, the pad length byte, and the N byte.

If the AOFL Options bit is set in the PDB for output options 1 and 2, SEC uses the post-decryption Pad Length byte to compute the effective frame length after the fact. It adjusts the output frame length to tell consumers of the output frame to ignore padding, pad length, and N. These fields are written to the output frame, so the provided output frame must be long enough to receive these fields.

Selecting option 0 for the output frame format (see [Figure 9-10](#)) causes SEC to decrypt the payload, padding, pad length and next header fields, but leave the Outer IP header, ESP header and IPSEC IV as is.

9.1.6.1 IPsec ESP Transport Mode outer IP header decapsulation procedure

In transport mode, SEC processes the outer IP header by doing the following:

1. Finds the PDB field "IP Header Length"; this header length, in bytes, includes the length of all option fields and extension headers.
2. Recomputes the length field by subtracting the length of the ESP Header, the IPsec IV, and the complete ESP trailer, which consists of the ICV, the padding, the 1-byte pad length (Pad Len) field, and the 1-byte next header (N) field.
3. Replaces the byte at the location in the outer IP header indicated by the PDB field "NH Offset" with the decrypted Next Header value from the ESP Trailer (shown as N in [Figure 9-10](#)).
4. Passes the modified outer IP header from the input frame to the output frame, where it becomes the first field.

Note that SEC only recomputes the checksum field in the transport mode IP header if the Cksm bit in the PDB Options Byte is set to 1 (see [Table 9-4](#)).

[Figure 9-10](#) shows the output frame resulting from decapsulation in transport mode as option 1.

9.1.6.2 IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode)

In tunnel mode, SEC processing does not modify the outer unencapsulated IP header. The decapsulated IP header is included in the output frame, and normally the outer IP header is discarded. The format of the output frame is controlled by PDB Options bit outFMT.

- If outFMT=0, Output Frame option 0 as shown in [Figure 9-10](#) is selected. Unencrypted fields in the input frame are copied as-is to the output frame, and after decryption, all other fields are copied to the output frame, with the exception of the

ICV. It is then the responsibility of another processing element to discard the outer IP header, the ESP header, and the ESP trailer.

- If outFMT=1, Output Frame option 1 as shown in [Figure 9-10](#) is selected. The outer IP header and ESP header are used as required for decapsulation processing but are not written to the output frame. When outFMT=1, PDB Options bit AOFL (Adjust Output Frame Length) comes into play in determining the contents of the output frame:
 - If AOFL= 0, then the entire decrypted ESP trailer is written to the output frame.
 - If AOFL= 1, then SEC adjusts the output frame length after decryption has completed. SEC cannot know where payload ends until the N byte is decrypted, and goes to no special effort to decrypt it early. As a result, padding may be written to the output frame. However, once the end of payload is known, SEC adjusts the output frame length to reflect the end of proper payload, and rewinds the output frame pointer to that point.

Because the original (inner) IP header was encrypted and authenticated during the encapsulation procedure, it is uncovered during decapsulation.

Other PDB Options bits control how the inner IP header is presented:

- Setting DSC will cause SEC to copy the IPv4 TOS byte or the IPv6 Traffic Class byte from the outer IP header to the inner IP header before writing the inner header to the output frame.
- Setting DTTL will cause SEC to decrement the IPv4 TTL or IPv6 hop limit byte before writing the inner IP header to the output frame.
- Setting ODF will cause SEC to overwrite the IPv4 DF bit in the inner IP header with the value in DFV.
- Setting Cksm causes the IPv4 header checksum to be updated as a result of any changes made to the inner IPv4 header.

9.1.7 IPsec ESP Cryptographic Decapsulation

The IPsec ESP cryptographic processes associated with decapsulation are common for both ESP Tunnel and Transport modes, but vary depending upon cipher suite. In all cases, the ICV is computed to cover both the ESP Header and the encrypted payload and ESP Trailer. Payload and ESP Trailer are decrypted as well.

9.1.7.1 IPsec decapsulation procedure when using AES-CBC or DES-CBC

This figure shows IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when the cipher is AES-CBC or DES-CBC.

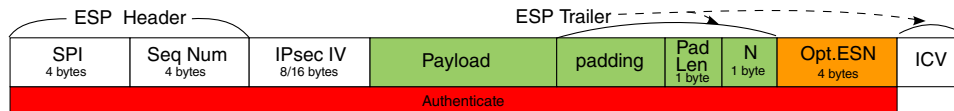


Figure 9-11. IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when cipher is AES-CBC or DES-CBC

When using the AES-CBC or DES-CBC cipher suite, SEC performs IPsec decapsulation by doing the following:

1. Receives the packet from the Input Frame serially, first receiving the outer IP header, then the ESP header, which consists of the SPI and the sequence number, and finally the IPsec IV.
2. Pushes the SPI and sequence number into a CHA, typically a Class 2 CHA, for authentication.
3. Pushes the IPsec IV into a CHA for authentication and then copies it to the Class 1 CHA Context Register (register offset 0).
4. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above in red. If the comparison fails, then an ICV CHECK FAIL is signalled in the Job Completion Status Word.
5. Receives payload data (see [Figure 9-12](#)) from the input frame and pushes it into the input-data FIFO.

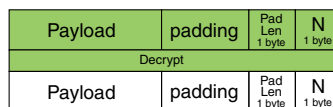


Figure 9-12. IPsec ESP Transport (and Legacy Tunnel) AES-CBC/3DES-CBC decryption for decapsulation

6. Tags the payload data for authentication and decryption.
7. If the decapsulated output frame option 0 is selected, pushes the results of payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

9.1.7.2 Process for IPsec decapsulation when using AES-CTR

This figure shows the IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when the cipher is AES-CTR.

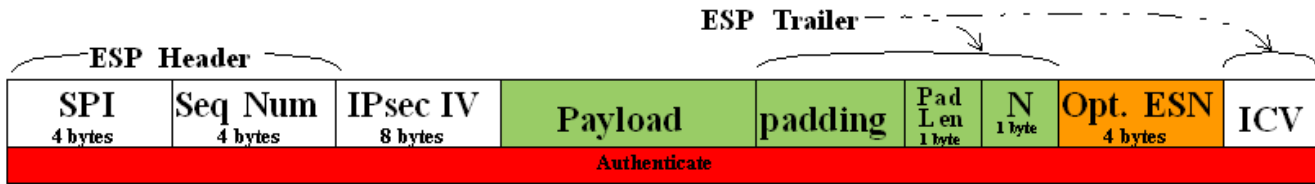


Figure 9-13. IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when cipher is AES-CTR

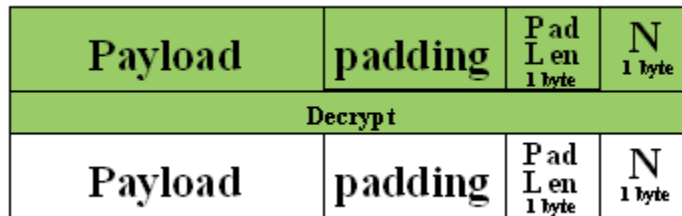


Figure 9-14. IPsec ESP Transport (and Legacy Tunnel) AES-CTR decryption for decapsulation

When using the AES-CTR cipher suite, SEC performs IPsec decapsulation occurs by doing the following:

1. Receives the packet from the input frame serially, first the outer IP header, then the ESP header, which consists of the SPI and the sequence number, and finally the IPsec IV.
2. Pushes the SPI and sequence number into a CHA, typically a Class 2 CHA, for authentication.
3. Constructs the counter value by prepending the 8-byte IPsec IV with a 4-byte Nonce and appending IPsec IV with a 4-byte initial count value (typically 0000 0001h); both the Nonce and initial count values are extracted as is from the PDB.
4. Writes the constructed counter value to the Class 1 Context Register, offset 16.



Figure 9-15. Initial counter construction

5. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above. If the comparison fails, then an ICV CHECK FAIL is signalled in the Job Completion Status Word.

6. Receives payload data (see [Figure 9-12](#)) from the input frame and pushes it into the input-data FIFO
7. Tags the payload data for authentication and decryption
8. If the decapsulated output frame option 0 is selected, pushes the payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

9.1.7.3 Process for IPsec decapsulation when using AES-CCM

This figure shows the IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation when the cipher is AES-CCM.

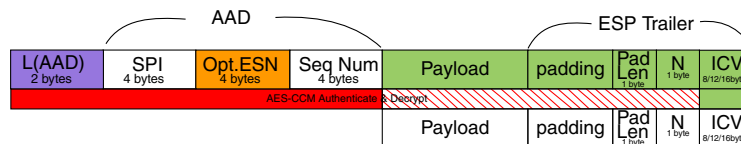


Figure 9-16. IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation with AES-CCM

When using the AES-CCM cipher suite, SEC performs IPsec decapsulation by doing the following. Note that this cipher suite can receive packets out of order:

1. Computes the length of AAD, and creates the Formatted AAD from a 2-byte representation of the length concatenated to the AAD, which consists of SPI, and optional ESN, and the sequence number
2. Writes this portion of the input frame to the input-data FIFO
3. Optional: If, the ESN is included, determines the correct ESN value and inserts it between SPI and Seq Num as the input frame is inserted to the input-data FIFO.

NOTE

Because packets may be received out of order and because the ESN increments upon Seq Num rollover, SEC may have to adjust the ESN. SEC does not increment the ESN value until the entire anti-replay window reflects post-rollover Seq Num.

4. Constructs an 11-byte nonce value by first extracting the 8-byte IPsec IV from the input frame and then prepending a 3-byte Salt value obtained from the PDB (see the following figure).

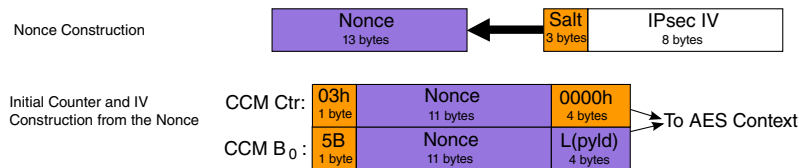


Figure 9-17. IPsec AES-CCM context construction

5. Constructs the CCM B₀ value by prepending the B₀ flags, which are a single-byte constant found in the PDB, and postpending the 4-byte SEC-computed payload length to create a 16-byte IV (see [Figure 9-17](#)).
6. Writes the 16-byte IV to the Class 1 Context Register (offset 0).

NOTE

Software must properly provision the PDB with the B₀ flags, per RFC 3610. Typical values matching the Nonce configuration (as specified in RFC 4309) are as follows:

- For an 8-byte ICV, use 5b.
- For a 12-byte ICV, use 6b.
- For a 16-byte ICV, use 7b.

7. Constructs the initial count value by prepending the counter field flags (another single-byte constant extracted from the PDB, shown as CTR₀) to the nonce and appending a four-byte value, also extracted from the PDB (typically 4 bytes of zeros) (see [Figure 9-17](#)).
8. Writes the resulting 16-byte initial count to the Class 1 Context Register immediately following the IV (that is, at offset 16).

NOTE

Software must properly provision the PDB with the counter field flags, per RFC 3610. The expected value matching the nonce configuration, as specified in RFC 4309, is 03h.

9. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above. If the comparison fails, then an ICV CHECK FAIL is signaled in the Job Completion Status Word.
10. Receives payload data from the input frame and pushes it into the input-data FIFO
11. Tags the payload data for authentication and decryption
12. If the decapsulated output frame option 0 is selected, pushes the results of payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

9.1.7.4 Process for IPsec decapsulation when using AES-GCM

These figures show IPsec decapsulation when using AES-GCM.

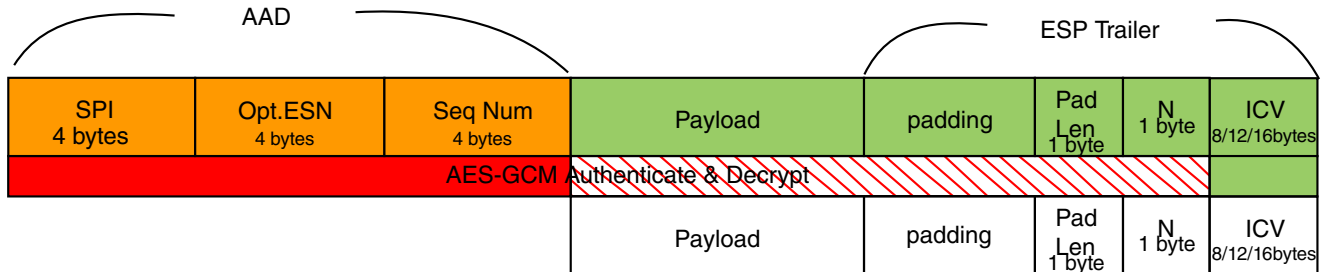


Figure 9-18. IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation with AES-GCM

When using the AES-GCM cipher suite, SEC performs IPsec decapsulation by doing the following:

1. Computes the GCM-IV by extracting the 4-byte Salt value from the PDB and appending to that the 8-byte IPsec IV received in the input frame (see the following figure).

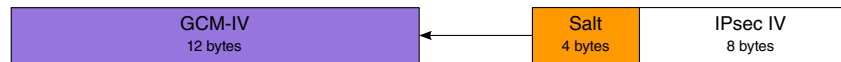


Figure 9-19. IPsec GCM IV building

2. Pads this GCM-IV with 4-bytes of zeros and writes the resulting 16-byte value to the input-data FIFO.
3. Pushes the AAD (the 4-byte SPI, an optional 4-byte ESN, and the 4-byte Seq Num) onto the input-data FIFO, zero padded to a block (16-byte) boundary if necessary (see [Figure 9-18](#)).

NOTE

Because packets may be received out-of-order and because the ESN increments upon Seq Num rollover, SEC may have to adjust the ESN. SEC does not increment the ESN value until the entire anti-replay window reflects post-rollover Seq Nums.

4. Pushes the encrypted payload and the ESP Trailer (padding, Pad Len, N and ICV) onto the input-data FIFO for decryption and authentication.

9.1.7.5 Use of SPI and the sequence number in decapsulation

When decapsulating, SEC ignores the SPI except when performing authentication.

SEC uses the sequence number for anti-replay checking (see [Anti-replay checking in IPsec ESP decapsulation](#)). Using fields in the PDB, SEC maintains a replay window of up to 128 packets.

- Any packet received and observed to be a duplicate of a previously received packet (within the window) has REPLAY indicated in the Job Completion Status Word.
- Any packet received and observed to fall prior to a value permitted by the replay window is tagged as LATE in the Job Completion Status Word.

9.1.7.6 Optional use of ESN in ESP decapsulation

After the encrypted data, the packet may infer an extended sequence number (ESN). If so configured:

1. The ESN is copied from the PDB, with a possible modification (see below).
2. The ESN is pushed into the authentication CHA, which is typically the Class 2 CHA.
3. Upon rollover of the sequence number, the packet encapsulator increments the ESN.

SEC needs to detect which sequence numbers correspond to the pre-incremented ESN and which correspond to the post-increment ESN. SEC adjusts the ESN as necessary when authenticating the packet. Once the entire anti-replay window reflects the rollover of the sequence number, SEC increments the ESN values stored in the PDB.

9.1.7.7 Anti-replay checking in IPsec ESP decapsulation

The IPsec decapsulation protocol uses SEC also performs anti-replay checking by doing the following:

1. Confirms the received ICV against the computed ICV
2. Stores a Seq Num value in the PDB; this value indicates the newest packet still within the window
3. Stores a bit array that supports an anti-replay window of up to 128 packets; this array indicates which packets have been received, with the least significant bit representing the packet with the aforementioned Seq Num, and which packets have not been received.

The ARS bits in the PDB Options field controls the size of the anti-replay window, as follows:

- If 01b is programmed into ARS, then a 32-bit window is selected and SEC only uses word 6 of the PDB for implementing the anti-replay scorecard.
- When ARS contains 11b, then an anti-replay window of 64 is selected, and the LS bit of word 7 represents the PDU immediately older than that represented by the MS bit of word 6.
- When an anti-replay window of 128 is selected by programming ARS to 10b, then the LS bit of word 8 represents the PDU immediately older than that represented by the MS bit of word 7 and the LS bit of word 9 represents the PDU immediately older than the MS bit of word 8.
- For any size anti-replay window, the LS bit of word 6 of the entire scorecard is used to indicate the status of the PDU represented by the values of ESN and Seq Num stored in the PDB, and each bit to the left represents a packet earlier in time.

In this version of SEC, the IPsec protocol actually uses the Anti-replay built-in protocol described in [Anti-Replay built-in checking](#).

9.1.7.7.1 When anti-replay checking is enabled

If anti-replay checking is enabled, the anti-replay scorecard (ARS) is updated with each PDU that has passed its ICV check.

If the current PDU represents a more recent packet than any previously received, then the ESN/Seq Num fields in the PDB are updated to match the received PDU, and the anti-replay scorecard is shifted left so that the LS bit of word 6 represents the current packet. For 64-bit windows, any bits shifted left out of PDB word 6 are shifted into the right end of PDB word 7. For 128-bit windows, words 8 and 9 are handled in a similar fashion to word 7.

If the current PDU is older than at least one PDU received before (as represented by the input state of the PDB ESN/Seq Num fields), then the scorecard is not shifted, but the bit representing the received packet is set to 1.

In either case, if that bit was already set to 1, then the received packet is considered a REPLAY packet, and the return status for the frame indicates REPLAY. If the bit representing the PDU has fallen off the end of the anti-replay scorecard (that is, if the difference between the received packet's Seq Num is greater than the anti-replay size), then the received packet is considered LATE, and the return status for the frame indicates LATE.

9.1.7.7.2 When anti-replay checking is disabled

If anti-replay checking is disabled, the PDB is always updated to match the ESN or Seq Num values for the last PDU received.

9.1.7.8 ICV checking during IPsec ESP decapsulation

The last segment (normally 12 bytes) of the input frame is the Integrity Check Value, or ICV. Once computed, the CHA responsible for computing the ICV on the received decapsulated packet will compare the computed ICV to the received ICV. If PDB Options indicates presence of ESN (Extended Sequence Number), the ESN value is taken out of the PDB and is used as part of ICV computation. For cryptographic modes CBC and CTR, the ICV is computed as if the ESN were received between the ESP Trailer's Next Header byte and the ICV. For cryptographic modes CCM and GCM, the ICV is computed as if the ESN were received between the SPI and the Sequence Number. If the computed ICV and the Received ICV do not match, then an ICV ERROR is signalled, and processing halts.

NOTE

The ICV is checked before sequencing and padding is checked. That is, an ICV ERROR takes precedence and will mask LATE, REPLAY, and BAD ESP PADDING errors. Only after the ICV compares favorably is the sequencing and the padding checked.

9.1.8 IPsec ESP Tunnel encapsulation overview

The IPsec ESP Tunnel Protocol-thread, when selected for encapsulation, encapsulates the contents of the Input Frame, using the encryption and authentication functions selected in the Protocol Operation command PROTINFO field. The Input Frame must not be so long such that when an Outer IP Header is added, a Jumbo Datagram is constructed. In other words, the maximum length after encapsulation must not exceed 65535 bytes (including an outer IPv4 header but not including a 40 byte outer IPv6 header).

Note that the IPsec ESP Transport (and legacy tunnel) Protocol thread can also be used for IPsec tunnel-mode encapsulation. Some of the details differ; in particular the IPsec ESP Transport (and legacy tunnel) Protocol thread cannot be used in conjunction with UDP-encapsulated-ESP, and the use of the Outer IP Header is more greatly constrained. Both threads perform cryptographic encapsulation identically; for further details, see [IPsec ESP Cryptographic Encapsulation](#).

The details of how the IPsec ESP Tunnel encapsulation Protocol thread handles the outer IP header are described below.

9.1.8.1 Handling the Outer IP Header during ESP Tunnel encapsulation

During ESP Tunnel mode encapsulation, an outer IP header is applied to the new datagram, and the IP header of the datagram being encapsulated is encrypted. SEC provides several options for providing this outer IP header, as controlled by the PDB Options field OIHI and DPOVRD bit OIMIF:

- OIHI = 11b - Outer IP Header Material is copied from the PDB.
- OIHI = 10b - Outer IP Header Material is copied from memory -- from the location addressed by the PDB
- OIHI = 01b - Outer IP Header Material is copied from input frame.
- OIHI = 00b - No Outer IP Header is applied
- OIMIF = 1b overrides the OIHI-specified source for the Outer IP Header Material, instead using the input frame. Whereas the scope of OIHI is for every frame in the flow, the scope of OIMIF is to the particular frame to which OIMIF applies.

The description of ESP Tunnel mode encapsulation refers to "Outer IP Header Material" because specific provision is made for the material to contain more than just the Outer IP Header. PDB / DPOVRD field AOIPHO specifies a number of additional bytes that precede the actual Outer IP Header. This additional material could consist of additional outer headers, such as an Ethernet header. The additional material is copied to the output frame as is. The Outer IP Header, on the other hand, is subject to additional processing.

The PDB is designed such that if OIHI selects Outer IP header material from PDB, that the Outer IP header Material starts offset by some multiple of 8 bytes within the descriptor. If using AOIPHO, then both the Outer IP header Material and the actual Outer IP header must start offset on an 8-byte multiple within the descriptor. In such a circumstance, AOIPHO reflects the number of bytes to be used that is part of the overall material but is not part of the actual Outer IP Header. For example, if AOIPHO is used to provide for a 14B Ethernet header (two mac addresses and ethertype) in addition to a standard 20B IPv4, then AOIPHO is programmed to 14, and Outer IP Header Material length is programmed to 34. However, in constructing the PDB, there must be two extra bytes of padding between the Ethernet header and the actual Outer IP header that SEC will skip. The Outer IP Header Material will then occupy a total of 36 bytes in the PDB, including the 2 bytes of padding.

NOTE

The 8-byte alignment rule only applies when OIHI selects Outer IP header material from the PDB. If OIHI selects outer IP header material from external memory or from the input frame, then no padding material is required to align the actual Outer IP header.

9.1.8.2 Outer IP Header handling with UDP-encapsulated-ESP

The ESP Tunnel encapsulation thread is capable of handling RFC 3948 UDP-encapsulated-ESP. If PDB Options bit NAT is set, then the ESP Tunnel encapsulation procedure includes special handling as follows:

- The last 4 bytes of the Outer IP Header Material are treated as the Source and Destination Port fields of the UDP header
- SEC inserts two bytes after the Destination Port field for the UDP Packet Length. The length written in this field to the output frame includes the 8-byte UDP header, the ESP header, the encrypted datagram, the ESP trailer, and the ICV.
- SEC inserts two bytes after the UDP Packet Length field for the UDP Packet Checksum.
 - If PDB Options bit NUC = 0, then no proper UDP Packet Checksum is computed, and the field is left as two bytes of zeros.
 - If PDB Options bit NUC = 1, then a proper UDP Packet Checksum is computed across all the bytes written to the output frame as accounted for by the UDP Packet Length field, *plus* an IP Pseudo-Header covering the appropriate fields of the Outer IP Header. Note that these fields differ, depending upon whether the Outer IP Header is IPv4 or IPv6.

NOTE

DECO includes hardware for computing a 16-bit one's complement checksum. Normally, the use of the hardware is controlled by SEQ FIFO STORE commands; that is SEQ FIFO STORE Source Field values can be used to determine which bytes written to the output frame are included in a checksum computation. However, for IPsec ESP Tunnel encapsulation, if NAT and NUC are selected in the PDB Options byte, then those controls are overridden, and the checksum hardware is used for computing the UDP checksum.

9.1.8.3 ESP Tunnel Outer IP Header manipulation

The ESP Transport (and legacy tunnel) PDB Options byte contains several bits that control how that thread might handle the Outer IP Header. These bits include:

- Cksm: if enabled, and if IPv4 is selected, update the outer header checksum per any manipulations made to the outer IP header
- DSC: if enabled, copy the IPv4 TOS or IPv6 Traffic Class byte from the inner IP header to the outer IP header
- IPvsn, selecting the version of IP header handled

For the ESP Tunnel thread, these are not options. The version of IP header is known based on the Version field of the inner and outer IP headers, and these need not be the same. As long as an outer IP header is provided, SEC will copy the TOS or Traffic Class byte from the inner header to the outer header, and if the outer header is IPv4, SEC will update the checksum resulting from this and any other changes to the header.

In addition to manipulations described above, SEC will decrement the outer IP header Time-to-Live field (for IPv4) and the Hop Limit field (for IPv6). SEC will also compute the appropriate length field value for the Outer IP Header.

The PDB HMO field for both the ESP Transport (and legacy tunnel) and ESP Tunnel encapsulation threads is identical. Copying the DF bit from inner IPv4 header to outer IPv4 header is optional, as is decrementing the Time-to-Live (IPv4) / Hop Limit (IPv6) field.

9.1.8.4 ESP Tunnel handling of Next Header

The last byte encrypted during encapsulation is the Next Header byte. The unencrypted value is designed to indicate the type of payload that has been encapsulated. The value SEC uses during encapsulation comes either from the Next Header field of the PDB, or from Next Header field of the DPOVRD register, if the most significant bit of DPOVRD is set.

9.1.9 IPsec ESP tunnel decapsulation overview

The IPsec ESP Tunnel Protocol-thread, when selected for decapsulation, decapsulates the contents of the Input Frame, using the decryption and authentication functions selected in the Protocol Operation command PROTINFO field. The input frame must contain an ESP header, an encrypted payload with appropriate padding, and an ICV, and in total must not exceed 65535 bytes in length (including an outer IPv4 header but not including a 40 byte outer IPv6 header). The Input Frame may also contain Outer IP Header Material at the front of the Input Frame. The primary purpose for inclusion of an Outer IP Header as an input to the decapsulation process is to allow the Protocol-thread to copy selected fields from the Outer IP Header to the decapsulated Inner IP Header. The Descriptor Protocol Data Block (PDB) contains fields and control bits to specify the precise processing performed. This is described in sections below.

Note that the IPsec ESP Transport (and legacy tunnel) Protocol thread can also be used for IPsec tunnel-mode decapsulation. Some details differ; in particular the IPsec ESP Transport (and legacy tunnel) Protocol thread cannot be used in conjunction with UDP-

encapsulated-ESP, and the use of the Outer IP Header is more greatly constrained. Both threads perform cryptographic decapsulation identically; for further details, see [IPsec ESP Cryptographic Decapsulation](#).

After decryption of the ESP trailer has completed, SEC performs a check of the cryptographic padding field to ensure it conforms to requirements and was not corrupted during transmission or decapsulation. The detection of corrupted Cryptographic padding will result in the signalling of a BAD ESP PADDING error, and will halt processing.

9.1.9.1 Input material preceding the outer IP header

Unlike the IPsec ESP Transport (and legacy tunnel) thread, the IPsec ESP Tunnel thread is designed to accept lower level headers or other material that may precede the actual Outer IP Header. PDB / DPOVRD field AOIPHO defines the number of bytes that precede the actual Outer IP Header. The PDB value is used normally; if DPOVRD is used to override the PDB, the the AOIPHO field in DPOVRD is used instead.

Normally, any additional material preceding the actual Outer IP Header is not included in the output frame, like the Outer IP Header. PDB Options bit ETU enables copying of the preceding material, under the assumption that it represents an Ethernet header.

Because the last two bytes of an Ethernet header are the EtherType field, SEC does not copy the last two bytes of the preceding material (as defined by AOIPHO), but instead replaces them with a proper EtherType value, depending on the version of the decapsulated Inner IP Header: if IPv4, then 0x0800 is put into the Output Frame immediately prior to the Inner IP Header. If IPv6, then 0x86DD is put into the Output Frame.

9.1.9.2 Handling the Outer IP Header during ESP Tunnel decapsulation

That Outer IP Header Material may consist of some segment of data prior to the actual Outer IP Header, the Outer IP Header, and a UDP header suitable for NAT. The resulting Output Frame will consist of the decapsulated payload, with extra material (including cryptographic padding) removed. Note that the Decapsulation thread is not designed to detect and remove TFC padding. In some circumstances, Input Frame material outside the decapsulated payload that was prior to the actual Outer IP Header in the Input Frame may be copied with adjustment to the Output Frame.

The primary purpose for inclusion of an Outer IP Header as an input to the decapsulation process is to allow the Protocol-thread to copy selected fields from the Outer IP Header to the decapsulated Inner IP Header. The Descriptor Protocol Data Block (PDB) contains fields and control bits to specify the precise processing performed. This is described below.

9.1.9.3 Manipulation of the Inner IP Header during ESP Tunnel decapsulation

The PDB Options field for ESP Tunnel decapsulation contains several bits to control how the Inner IP Header is manipulated after decryption.

In an IPv6 Header, DS and ECN are carried in 8 bits labelled Traffic Class, and straddle the lower half of the first byte, and the upper half of the second byte. An IPv4 header uses DS and ECN in the Type of Service (TOS) field, found in the second byte of an IPv4 Header. (The upper 6 bits is DS, and the lower 2 bits is ECN.) If the PDB options select DSC then the entire TOS / TC byte is copied from the Outer IP Header to the decapsulated Inner IP Header.

If the Inner IP Header is IPv4, then the PDB ODF Option applies as follows: the DF bit in the Inner IP Header is replaced with the value of the DFV PDB Options bit before the Inner IP header is written to the Output Frame.

Upon decapsulation, the IPv4 inner Header TTL field is decremented (by one) before the decapsulated Inner Header is written to the Output Frame. If an IPv6 Header instead, then the decrement by one is applied to the Hop Limit field.

All changes made to the decapsulated Inner IPv4 Header result in a update of the Header Checksum. Note that if the Header Checksum was incorrect to begin with, the resulting Checksum will also be incorrect.

9.1.9.4 Decapsulation Output Frame Length

The length of the Output Frame depends upon the version of the Inner IP Header as follows:

- For IPv4, the Inner IP Header contains a Total Length field, and that field is used to specify the length of the Output Frame. As a result, only the intended encapsulated

payload is written to the Output Frame; the remainder can be reliably prevented from being written to memory.

- For IPv6, the Inner IP Header contains a Payload Length that indicates the length of the encapsulated payload (including the Extension Headers), but not the length of the encapsulated base IPv6 header (which is always 40 bytes in length).

9.2 SSL/TLS/DTLS record encapsulation and decapsulation overview

SEC supports the following versions of the TLS family of security protocols:

- SSL 3.0
- TLS 1.0
- TLS 1.1
- TLS 1.2
- DTLS 1.0 (a variant of TLS 1.1)
- DTLS 1.2 (a variant of TLS 1.2)

The variants of the protocol are similar, but have the following key differences:

- Handling of IVs for block ciphers has evolved over the versions
- The list of supported cipher suites has evolved over the variants.
- DTLS 1.0 is a variant of TLS 1.1 with an explicit sequence number.
- TLS 1.2 and DTLS 1.2 replace the MD5/SHA-1-based pseudo-random function (PRF) with a PRF computed using only SHA-256 or SHA-384

The PROTINFO field codes enumerated in [Table 7-54](#) define the cipher suites used by the protocol, and SEC's built-in protocol processing sequences handles the remaining details. Detailed processing descriptions must be described differently for different versions. PDB and PDB Override programming, decapsulation output frame options, and finding the last byte of the encrypted payload during decapsulation processing are all common to the different protocol versions, and are described first.

Table 9-38. SSL/TLS/DTLS protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block includes next header, SPI, sequence number, IV (if not from RNG)		Protocol data block includes sequence number, anti-replay information (DTLS only)
Class 2 key data block		Class 2 key data block
Class 1 key data block		Class 1 key data block
Protocol = <protocol> encrypt		Protocol = <protocol> decrypt

NOTE

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a [split key](#). Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the [Derived Key Protocol](#) may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

NOTE

Sharing MD5 SMAC and HMAC keys is restricted. SEC will prevent mis-sharing of MD5 Keys if shared descriptor SHARE is set to NEVER, WAIT, or SERIAL. ALWAYS should not be used. For more information on sharing, please refer to [Table 7-1](#).

9.2.1 Programming and processing details common to all versions of SSL, TLS, and DTLS

Certain details of processing, and how to program the shared descriptor, are common to all supported versions of SSL, TLS, and DTLS:

- Protocol Data Block Programming formats vary by cipher suite, but are common across protocol versions, including encapsulation and decapsulation.
- Using the Datapath Override (DPOVRD) register to provide a non-default Type field for encapsulation
- Finding the last byte of payload and determining the pre-encapsulation record header
- Decapsulation Output Frame Formats

9.2.1.1 PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation

Unlike other protocols' PDBs, the SSL/TLS/DTLS PDB varies in content and field size based on PROTINFO and Options settings. In particular, for CBC-mode cipher suites, the IV field is only 2 words if the PROTINFO field of the Operation Command selects DES or 3DES. Also, the ICV Len field is present only if necessary per the TrICV bit in the PDB's Options field. The format of the PDB is, as much as possible, kept common

between all different versions. One notable exception is the Sequence Number: Rather than using a 64-bit sequence number, DTLS uses a 16-bit epoch and a 48-bit sequence number.

SSL and all versions of TLS use identical PDBs for both encapsulation and decapsulation. DTLS PDBs for decapsulation are almost the same as all the others, with the addition of the Anti Replay Scorecard prior to the ICV Length word. The Options byte is somewhat different for decapsulation, with the addition of a field to control the size of the Anti Replay window

9.2.1.1.1 PDB for SSL, TLS, and DTLS when a Block Cipher is used

Block ciphers in SSL family require an initialization vector -- an IV. The IV randomizes the payload prior to encryption. For SSL and TLS version 1.0, the IV is the final cipher block of the previous record. TLS 1.2 uses a random IV. TLS 1.1 allows the final cipher block of the previous record to be masked with a random or fixed mask. The IV field is designed to store IV state, as required, between a previous record and a next record. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

Table 9-39. Block cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation

PDB Word 0	Descriptor Header (1 or 2 words)			
	Type (8 bits)	Version (16 bits)	Options (8 bits)	
PDB Word 1 <i>for SSL and TLS</i>	Seq Num 1			DECO writes back to PDB as needed
PDB Word 1 <i>for DTLS</i>	Epoch	Seq Num 1		
PDB Word 2 <i>all protocols</i>	Seq Num 2			
PDB Word 3 <i>either-CBC and I/E=1</i>	IV word 1			
PDB Word 4 <i>either-CBC and I/E=1</i>	IV word 2			
PDB Word 5 <i>AES-CBC and I/E=1</i>	IV word 3			
PDB Word 6 <i>AES-CBC and I/E=1</i>	IV word 4			
PDB Word 7 or 5 or 3	Anti-Replay Scorecard word 1			

Table continues on the next page...

Table 9-39. Block cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation (continued)

PDB Word 8 or 6 or 4	Anti-Replay Scorecard word 2		for DTLS Decap only (First PDB word identifying number used when AES-CBC and I/E=1) (Second PDB word identifying number used when DES-CBC and I/E=1) (Third PDB word identifying number used when I/E=0)
PDB Word 9 or 7 or 5	Anti-Replay Scorecard word 3 (when anti-replay window is 128)		
PDB Word 10 or 8 or 6	Anti-Replay Scorecard word 4 (when anti-replay window is 128)		
Last PDB Word	ICV Len (8 bits)	Reserved (24 bits)	

9.2.1.1.2 PDB for SSL, TLS, and DTLS when AES-Counter mode is used

The TLS implementation of AES-Counter is based on a draft RFC that was permitted to expire. The PDB requires a 48-bit WRITE_IV for constructing the initial counter value, which per the draft is a product of the 48 "rightmost" bits of either CLIENT_WRITE_IV or SERVER_WRITE_IV -- whichever was generated by this side of the negotiation. The draft RFC for TLS with AES-counter specifies initial lower 16 bits are to be programmed as zeros, but a different can be used by programming the PDB field "Constant 0000h" otherwise. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

Table 9-40. AES-Counter cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation

	Descriptor Header (1 or 2 words)			
PDB Word 0	Type (8 bits)	Version (16 bits)	Options (8 bits)	
PDB Word 1 <i>for SSL and TLS</i>	Seq Num 1			DECO writes back to PDB as needed
PDB Word 1 <i>for DTLS</i>	Epoch	Seq Num 1		
PDB Word 2 <i>all protocols</i>	Seq Num 2			

Table continues on the next page...

Table 9-40. AES-Counter cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation (continued)

PDB Word 3	WRITE_IV (Upper 32 bits)		
PDB Word 4	Write IV (lower 16 bits)	Constant 0000h	
PDB Word 5	Anti-Replay Scorecard word 1		DECO writes back to PDB as needed <i>for DTLS Decap only</i>
PDB Word 6	Anti-Replay Scorecard word 2		
PDB Word 7	Anti-Replay Scorecard word 3 (when anti-replay window is 128)		
PDB Word 8	Anti-Replay Scorecard word 4 (when anti-replay window is 128)		
Last PDB Word	ICV Len (8 bits)	Reserved (24 bits)	<i>for DTLS Decap only</i>

9.2.1.1.3 PDB for TLS and DTLS when AES-GCM is used

AES GCM state required in the PDB that required for all cipher suites, plus a 4 byte Salt value that is essentially extra key material. Salt, and the 8-byte sequence number, are concatenated to form a 12-byte GCM IV. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

Table 9-41. AES-GCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation

	Descriptor Header (1 or 2 words)			
PDB Word 0	Type (8 bits)	Version (16 bits)	Options (8 bits)	
PDB Word 1 <i>for SSL and TLS</i>	Seq Num 1			DECO writes back to PDB as needed
PDB Word 1 <i>for DTLS</i>	Epoch	Seq Num 1		
PDB Word 2	Seq Num 2			
PDB Word 3	Salt			
PDB Word 4	Anti-Replay Scorecard word 1			DECO writes back to PDB as needed <i>for DTLS Decap only</i>
PDB Word 5	Anti-Replay Scorecard word 2			
PDB Word 6	Anti-Replay Scorecard word 3 (when anti-replay window is 128)			
PDB Word 7	Anti-Replay Scorecard word 4 (when anti-replay window is 128)			
Last PDB Word	ICV Len (8 bits)	Reserved (24 bits)		<i>for DTLS Decap only</i>

9.2.1.1.4 PDB for TLS and DTLS when AES-CCM is used

AES-CCM uses the most complex PDB of all. Besides Type, Version, and Sequence Number, Several constants get pulled out of the PDB in order to create B_0 and CTR_0 that must be written into Class 1 context for AESA to perform CCM mode encapsulation properly. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

Table 9-42. AES-CCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation

	Descriptor Header (1 or 2 words)			
PDB Word 0	Type (8 bits)	Version (16 bits)	Options (8 bits)	
PDB Word 1 <i>for SSL and TLS</i>	Seq Num 1			DECO writes back to PDB as needed
PDB Word 1 <i>for DTLS</i>	Epoch	Seq Num 1		
PDB Word 2 <i>all protocols</i>	Seq Num 2			
PDB Word 3	WRITE_IV32 <i>(write IV generated by this endpoint)</i>			
PDB Word 4	B_0 Flags	CTR_0 Flags	Reserved	
PDB Word 5	Reserved	CTR_0 lower 3 bytes 000000h		
PDB Word 6	Anti-Replay Scorecard word 1			
PDB Word 7	Anti-Replay Scorecard word 2			DECO writes back to PDB as needed for DTLS Decap only
PDB Word 8	Anti-Replay Scorecard word 3 (when anti-replay window is 128)			
PDB Word 9	Anti-Replay Scorecard word 4 (when anti-replay window is 128)			
Last PDB Word	ICV Len (8 bits)	Reserved (24 bits)		<i>for DTLS Decap only</i>

9.2.1.1.5 Programming the Options byte with the PDB for SSL, TLS and DTLS

The encapsulation options byte contains three control bits, described below. Note that W/B and I/E are used only for CBC-based cipher suites, and are NOT for use with SSL or TLS 1.0.

Table 9-43. SSL, TLS, DTLS encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved			TrICV	Reserved		w/b	e/i

Table 9-44. SSL, TLS, and DTLS encapsulation PDB, description of the options byte

Field	Description
7-5	Reserved
4 TrICV	Truncate ICV 0 Normal ICV as defined per cipher suite. 1 ICV length is determined by ICV Len field in PDB.
3-2	Reserved
1 w/b	IV writeback 0 IV field in PDB held constant. 1 IV field in PDB written back with last block of ciphertext. <i>NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.</i> <i>NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.</i>
0 e/i	e/i: Explicit/Implicit random IV. 0 Implicit Random IV field transmitted as part of encrypted payload. 1 Explicit Random IV field transmitted as plaintext IV. <i>NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.</i> <i>NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.</i>

The decapsulation options byte for SSL and TLS contains five control bits, described below. DTLS decapsulation options add two ARS control bits. Note that W/B and I/E are used only for CBC-based cipher suites, and are NOT for use with SSL or TLS 1.0.

Table 9-45. SSL, TLS, DTLS decapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
DTLS: ARS SSL, TLS: Reserved		rsv	TrICV	outFMT		w/b	e/i

Table 9-46. SSL, TLS, and DTLS decapsulation PDB, description of the options byte

Field	Description
7-6 ARS	Anti-replay window size 00 - No anti-replay window 01 - 32 entry anti-replay window 10 - 128 entry anti-replay window 11 - 64 entry anti-replay window

Table continues on the next page...

Table 9-46. SSL, TLS, and DTLS decapsulation PDB, description of the options byte (continued)

Field	Description
	<i>Note ARS is used only with DTLS, not with TLS or SSL</i>
5	Reserved
4 TriCV	Truncate ICV 0 Normal ICV as defined per cipher suite. 1 ICV length is determined by ICV Len field in PDB.
3-2 outFMT	Decapsulation Output Frame format 00 - Option 1: Output is payload only 01 - Option 3: Output frame consists of Header and Payload 10 - Option 2: Output frame consists of entire input frame, decrypted 11 - Reserved -- results in PDB error
1 w/b	IV writeback 0 IV field in PDB held constant. 1 IV field in PDB written back with last block of ciphertext. <i>NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.</i> <i>NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.</i>
0 e/i	e/i: Explicit/Implicit random IV. 0 Implicit Random IV field transmitted as part of encrypted payload. 1 Explicit Random IV field transmitted as plaintext IV. <i>NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.</i> <i>NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.</i>

9.2.1.2 Overriding the PDB for SSL, TLS, and DTLS Encapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For SSL, TLS, and DTLS, the header TYPE field is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired TYPE value in the job descriptor. For more information, see [Job Ring interface](#). When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field in the FD. For more information, see [Queue Manager Interface \(QI\)](#). When using the Advanced IO Processor interface, AI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field provided in the FD. For more information, see [Advanced IO Processor Interface \(AI\)](#).

Table 9-47. SSL/TLS/DTLS encapsulation-DECO Protocol Override Register format

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	OVR D	Reserved														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								Type							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-48. SSL/TLS/DTLS encapsulation-DECO Protocol Override Register description

Field	Description
31 OVRD	Indicates whether to use the contents of DPOVRD to override values specified in the PDB 0 Use the PDB as provided. 1 Override values in PDB by using values in DPOVRD.
30-8	Reserved
7-0 Type	This value is used for constructing the SSL or TLS packet header (instead of the Type field in the PDB) if OVRD = 1.

9.2.1.3 Computing the pre-encrypted record length during decapsulation

TLS was developed such that the record is encapsulated using the pre-encryption length in the header, but the length field of the encapsulated record includes additions such as the ICV, any nonce or IVs, and padding. Because SEC performs decryption and authentication processing simultaneously, it must pre-compute the pre-encryption length to use for authentication processing. This is particularly challenging for block ciphers, which during encapsulation adds some number of bytes of padding, that number being unknown to decryption processing until the last byte of ciphertext has been decrypted.

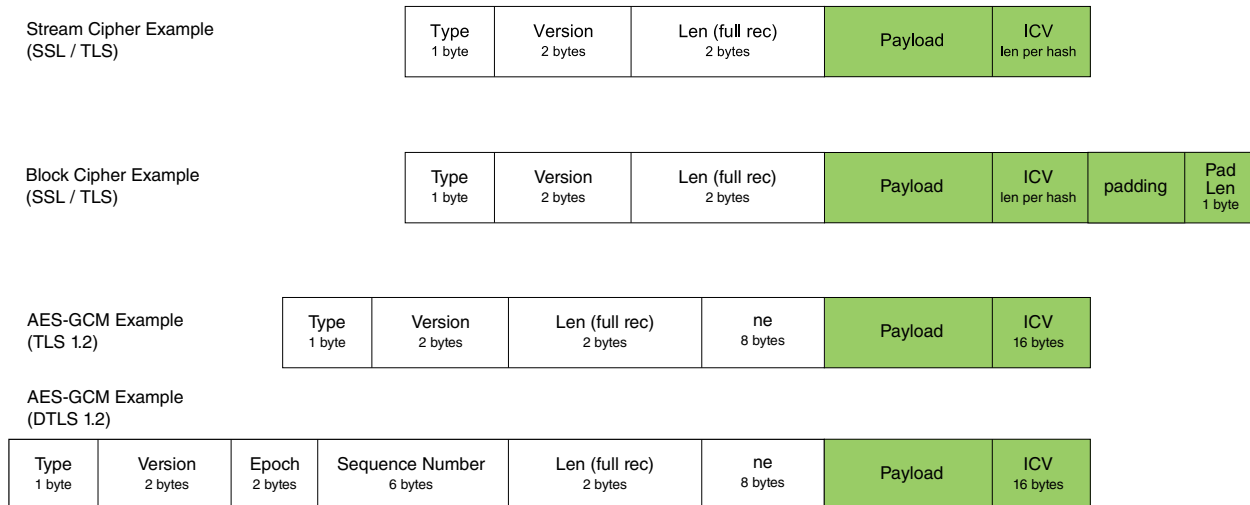


Figure 9-20. Some examples of encapsulated records

For stream ciphers, the required computation is relatively simple: subtract the length of the ICV. The ICV length in the case of all supported stream cipher suites turns out to be the size of the underlying hash unless TLS extension Truncated_HMAC has been negotiated, and PDB option TrICV has been set. In that case, the ICV length subtracted is the value found in the PDB ICV Len field.

Some AEAD ciphers are like stream ciphers, in that the ICV Length must be subtracted from the record length prior to beginning decapsulation. For DTLS, a second eight bytes needs to be subtracted from the record length to account for Epoch (two bytes) and Sequence Number (six bytes).

AES-GCM is an AEAD cipher that requires a slightly different encapsulated record -- it is like stream ciphers, except for the addition of ne (nonce-explicit). For this form, the length of the unprotected record is found by subtracting both the ICV length (16 bytes) and the ne length (8 bytes)

DTLS includes an explicit 8-byte Epoch / Sequence Number pair, in both the encapsulated and unencapsulated forms of the record, regardless of the cipher suite selected. These eight bytes are always included in the record length.

During encapsulation using block ciphers (CBC mode), some number of bytes of padding, plus one byte of pad length, are added to the encrypted payload to ensure the encrypted payload is an integral multiple of the underlying cipher block size. The total subtracted from the encapsulated record length is the ICV length (described above for stream ciphers), any IVs added during encapsulation, the length of padding, and 1 for the pad length byte.

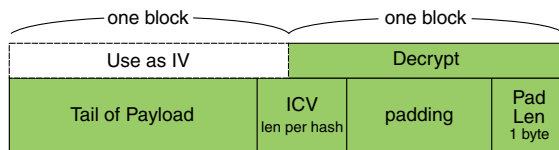


Figure 9-21. Example of last two blocks of ciphertext

To compute the length of padding, prior to starting decapsulation, SEC jumps to the end of the input frame, grabs the two last block of ciphertext, and performs a quick decryption of the last block, using the penultimate block as an IV. The last byte of the decrypted block *is* the padlength.

9.2.1.4 SSL, TLS, DTLS Decapsulation Output frame options

Programming the outFMT field of the PDB Options byte offers three options for providing the decapsulated record:

- Option 1 is the bare record encapsulated record, with everything but the payload removed.
 - For jobs submitted through QI, the payload length can be determined by the output buffer length returned as part of the frame description
 - For jobs submitted through a Job Ring, the actual record length will be returned only if INCL_SEQ_OUT is set in the JRCFGR_JRx_MS register associated with the particular Job Ring.
- Option 2 returns the entire decrypted record, with the modification that the length field reflects the length of the plaintext-payload. This option allows software inspection of padding (which is recommended by standards but is not implemented by SEC).
- Option 3 returns the record header (length-adjusted) and payload.

Some example diagrams follow for reference. First is an example showing option 1, where the returned output frame consists of just the record payload.

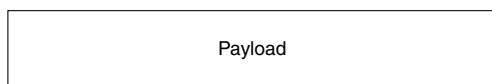


Figure 9-22. Decapsulation output frame Option 1

Next, some examples of option 2. For option 2, all fields are found in the input frame are returned; albeit with the record length field modified to reflect the decapsulated record length, and the record payload decrypted. The simplest form is used by stream ciphers and most AEAD ciphers. AES-GCM is the exception for AEAD ciphers, including the

nonce_explicit field. The block cipher forms include the padding and the pad-length byte, with either the random IV or the masked IV included for TLS 1.1 and TLS 1.2. Finally, for any permitted cipher, DTLS varies from the TLS 1.2 form by inclusion of the Epoch and Sequence Number.

AEAD cipher AES-GCM Output frame

Type 1 byte	Version 2 bytes	Len (preICV) 2 bytes	ne 8 bytes	Payload	ICV 16 bytes
----------------	--------------------	-------------------------	---------------	---------	-----------------

Output frame for stream ciphers and other AEAD ciphers

Type 1 byte	Version 2 bytes	Len (preICV) 2 bytes	Payload	ICV len per hash
----------------	--------------------	-------------------------	---------	---------------------

SSL 3.0 and TLS 1.0 output frame for block ciphers

Type 1 byte	Version 2 bytes	Len (preICV) 2 bytes	Payload	ICV 16 / 20 bytes	padding	Pad Len 1 byte
----------------	--------------------	-------------------------	---------	----------------------	---------	----------------------

TLS 1.1 and TLS 1.2 output frame for block ciphers

Type 1 byte	Version 2 bytes	Len (preICV) 2 bytes	Opt IV 0/8/16 bytes	Opt IVM 0/8/16 bytes	⋮	Payload	ICV len per hash	padding	Pad Len 1 byte
----------------	--------------------	-------------------------	------------------------	-------------------------	---	---------	---------------------	---------	----------------------

DTLS output frame for block ciphers

Type 1 byte	Version 2 bytes	Epoch 2 bytes	Sequence Number 6 bytes	Len (preICV) 2 bytes	Opt IV 0/8/16 bytes	Opt IVM 0/8/16 bytes	⋮	Payload	ICV len per hash	padding	Pad Len 1 byte
----------------	--------------------	------------------	----------------------------	-------------------------	------------------------	-------------------------	---	---------	---------------------	---------	----------------------

Figure 9-23. Examples of decapsulation output frames for output format Option 2

For Option 3, all versions of SSL, TLS, and DTLS return the same unprotected TLS record, regardless of cipher suite. This consists of Type, Version, Record Length, and Payload. For DTLS, Epoch and Sequence Number are also included. DTLS is virtually the same, including Epoch and Sequence Number.

SSL / TLS example output frame

Type 1 byte	Version 2 bytes	Len (pre ICV) 2 bytes	Payload
----------------	--------------------	--------------------------	---------

DTLS example output frame

Type 1 byte	Version 2 bytes	Epoch 2 bytes	Sequence Number 6 bytes	Len (pre ICV) 2 bytes	Payload
----------------	--------------------	------------------	----------------------------	--------------------------	---------

Figure 9-24. Decapsulation example output frames for Option 3

9.2.1.5 SSL / TLS / DTLS error codes

This table lists the conditions under which SSL, TLS, or DTLS encapsulation or decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure in decapsulation can also produce an ICV check error.

Table 9-49. SSL, TLS, and DTLS encapsulation and decapsulation error conditions

Condition	Error status	Applies to:
PDB Options Field outFMT programmed to 11b	Invalid Setting in PDB	Decapsulation
Bad Protocol Operation command (often caused by protocol version and cipher suite mismatch)	undefined protocol command	both encapsulation and decapsulation
Required keys not present when protocol starts execution	Key not written before start of protocol	both encapsulation and decapsulation
Sequence Number rolls over back to zero	Sequence Number Overflow	both encapsulation and decapsulation, TLS only
Received a sequence number far enough below the latest sequence number it "fell off" the window	Anti-replay LATE error	DTLS decapsulation, if anti-replay is turned on
Received a recent, repeated sequence number	Anti-replay REPLAY error	DTLS decapsulation, if anti-replay is turned on

9.2.2 Process for SSL 3.0 and TLS 1.0 record encapsulation

SEC performs SSL 3.0/TLS 1.0 encapsulation by doing the following:

1. Receives an input frame containing the payload.
2. Examines the contents of the DECO Protocol Override Register (DPOVRD)
 - If the MS bit (OVRD) is set, the job descriptor has selected a record-type override
 - If the MS bit is cleared, no record-type override is selected
3. Optional: If a record-type override has been selected, extracts the sequence number and the version field from the PDB and uses the least significant byte of DPOVRD for the record type
4. Optional: If no record-type override has been selected, extracts the sequence number, the record type and version fields from the PDB
5. Concatenates the sequence number, the record type (from whichever source is selected) and the version together, and pushes them into the Class 2 CHA for authentication (Note that version is excluded for SSL)
6. Pushes the record type and version fields onto the output frame
7. Increments the sequence number prior to writing it back to memory
8. Extracts and pushes the payload length, which is part of the frame description, as a 2-byte field into the Class 2 CHA for authentication

9. Adds the ICV length to the payload length, along with the length of any padding (including Pad Len) SEC adds; note that padding and Pad Len are added only for block ciphers (see [Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers](#))
10. Pushes the encrypted-payload length onto the output frame

Note that while the pre-ICV length is authenticated, the full record length (the length after ICV and padding is appended) is what is transmitted.

SEC supports two output frame formats: one for block ciphers and one for stream ciphers. See [Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers](#) for more details.

9.2.2.1 Differences between SSL 3.0 and TLS 1.0 (record encapsulation)

For record encapsulation, the only differences between SSL 3.0 and TLS 1.0 are:

- For SSL:
 - A custom SSL-MAC is used for message authentication
 - Version field in header is 0300h
 - Version field is not part of authentication computation
 - AES is not part of any valid cipher suite
- For TLS 1.0:
 - A HMAC is used for message authentication
 - Version field in header is 0301h
 - Version field is included in authentication computation
 - Cipher suites including AES are supported

9.2.2.2 Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers

This figure shows the process of SSL 3.0/TLS 1.0 when a block cipher (such as AES-CBC) is used.

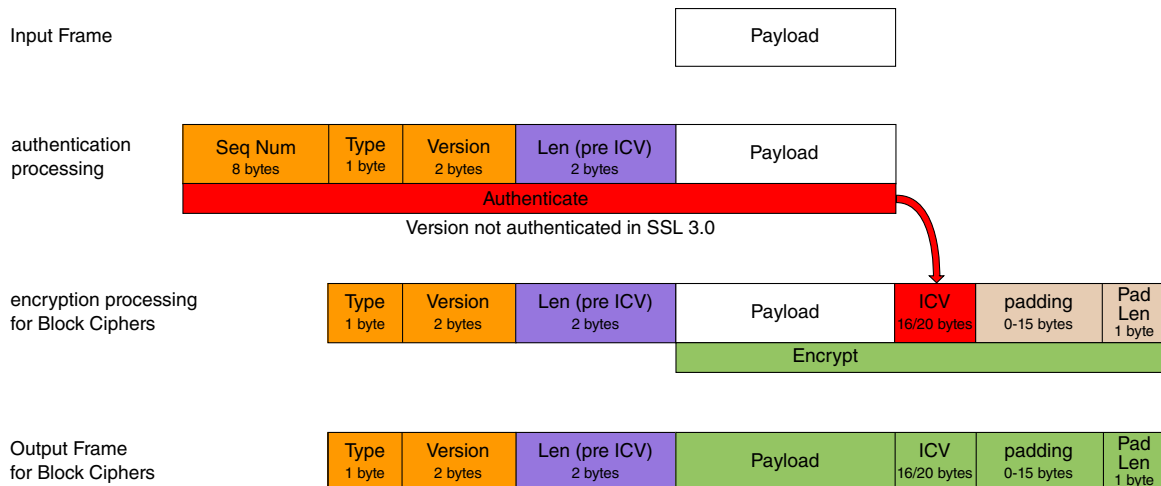


Figure 9-25. SSL 3.0/TLS 1.0 block cipher encapsulation

1. If the selected cipher suite includes a block cipher, the IV is extracted from the PDB and written to the CCB Class 1 Context Register (offset 0) as appropriate for the chosen block cipher. For AES, this IV is 16 bytes; for DES, it is 8 bytes.
2. The IV in the PDB is overwritten with the first block in the Class 1 Context Register (containing the final block of ciphertext) after encryption has completed.
3. As payload is extracted from the input frame, it is pushed onto the input-data FIFO and tagged for both encryption and authentication.
4. The last byte of the payload is the last byte authenticated. As a result, the ICV computed is also encrypted.
5. Following the ICV, SEC adds the minimal padding and a pad length byte such that the pad length byte is the last byte in a cipher block. Per the standard for TLS, the value of every byte of padding is the same as the value of the pad length byte.

Example: Using an AES-128-CBC-SHA cipher suite with a plaintext-payload length of 32 bytes

When using an AES-128-CBC-SHA cipher suite with a plaintext-payload length of 32 bytes, the ICV is 20 bytes.

Payload (32) + ICV (20) = 52 bytes.

The next multiple of the AES-CBC block size of 16 is 64.

Therefore, SEC appends 12 bytes, each with value 0Bh, immediately after the ICV.

6. This SEC-generated padding gets pushed into the Class 1 CHA for encryption, as shown in the following figure.
7. The resulting encrypted payload (which includes the encrypted ICV and the encrypted padding and pad length) is pushed onto the output frame, as shown in the following figure.

9.2.3 Process for SSL 3.0 and TLS 1.0 record decapsulation

SEC performs SSL 3.0/TLS 1.0 decapsulation by doing the following:

1. Receives an input frame consisting of the record header and the protected record.
2. Computes the length of the unprotected record, as described in [Computing the pre-encrypted record length during decapsulation](#).
3. Decrypts the record payload, MAC (ICV), and any padding.
4. Extracts the implicit sequence number from the PDB, and updates the incremented number back to the PDB.
5. Computes a MAC on the implicit sequence number, the record header (using the unprotected record length), and the decrypted record payload. When SSL 3.0 is selected, the version field of the record header is excluded from the MAC computation.
6. Compares the computed MAC against the decrypted ICV. If the two do not match, SEC returns an ICV Check Fail value in the Job Completion Status Word, and updated PDB is not written back to memory.
7. Returns an output frame per the selected output format, as described in [SSL, TLS, DTLS Decapsulation Output frame options](#).

CAUTION

For SSL and TLS, SEC does not perform any form of replay checking. Records are required to arrive in order. The use of an implicit sequence number guarantees that any records that arrive out of order result in an ICV failure.

9.2.3.1 SSL 3.0 and TLS 1.0 Record Decapsulation for block ciphers

This figure shows SSL 3.0/TLS 1.0 decapsulation when a block cipher is used.

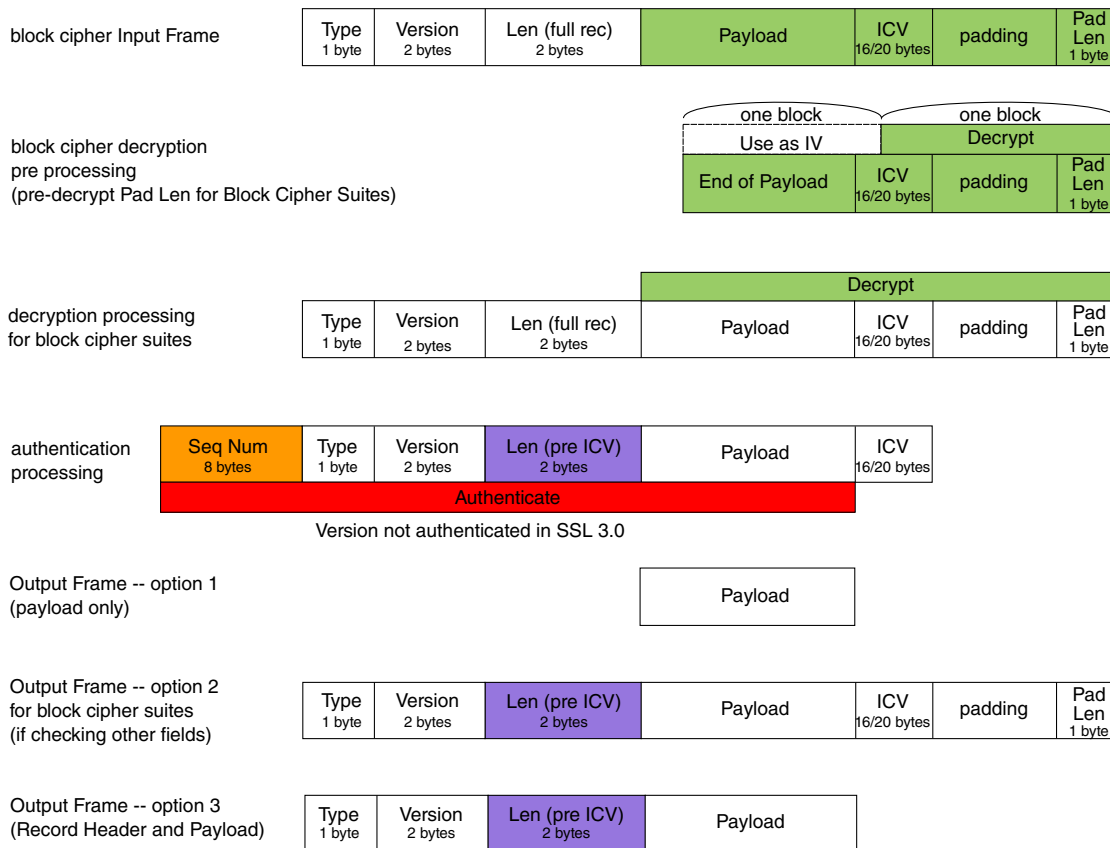


Figure 9-26. SSL 3.0 / TLS 1.0 Block Cipher Decapsulation

Block cipher decapsulation follows the process set forth in [Process for SSL 3.0 and TLS 1.0 record decapsulation](#), but with a few extra steps added. In particular:

1. SEC supports block cipher decapsulation only with CBC mode. CBC mode requires an IV. Prior to beginning of decryption, the IV is extracted from the PDB and written to the Class 1 Context Register.
2. After decryption is complete, the final block of ciphertext is written back into the PDB to be used as the IV for the next record in sequence.

9.2.3.2 Differences between SSL 3.0 and TLS 1.0 (record decapsulation)

For record decapsulation, the only differences between SSL 3.0 and TLS 1.0 are the message authentication code and the version code applied.

- For SSL, a custom SSL-MAC is used for message authentication, and the version code is set to 0300h.

- For TLS, standard HMAC is used for message authentication, and the version code is set to 0301h.
- For SSL, the Version field is not included in the SSL-MAC Computation. It is included in the TLS HMAC computation.

9.2.4 Process for TLS 1.1 and TLS 1.2 record encapsulation

In general, SEC performs TLS 1.1/1.2 encapsulation by doing the following:

1. Begins encapsulation when it receives an input frame containing the payload
2. Extracts the sequence number (which is incremented and written back to memory) and version fields from the PDB
3. Extracts the record type from the PDB unless the MS bit of DPOVRD = 1, indicating that the record type field comes from the least significant byte of DPOVRD
4. Use the selected keyed MAC function to authenticate the sequence number and the record header (Type, Version, Record Length). Note that the Record Length value that is part of the authentication function does not include any additions to the record due to payload protection, such as the ICV.
 - For an HMAC, MDHA is used to perform the authentication computations, so the sequence number and record header are passed to the input Data FIFO tagged as Class 2 message data.
 - For AEAD using AES (CCM or GCM), the sequence number and record header are passed to the input Data FIFO tagged as AAD.
5. The record type and version fields are also pushed onto the output frame
6. The length of the protected record is computed and is pushed onto the output frame.

For more specifics, please refer to the appropriate section as follows:

- For encapsulation using either the AES-CBC or DES-CBC confidentiality algorithms, see [Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers \(AES or DES\)](#).
- For encapsulation using the AES-Counter confidentiality algorithms, see [Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers](#).
- For encapsulation using either AEAD algorithms AES-CCM or AES-GCM, see [Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers](#).

9.2.4.1 Differences between TLS 1.0, TLS 1.1, and TLS 1.2 Record Encapsulation

The main difference between TLS 1.0 and TLS 1.1 is in how block ciphers handle IVs. Because of security concerns, TLS 1.1 adopted the use of a random mask instead of a purely implicit IV, chained from the previous record. If a TLS 1.0 style implicit IV is used, the IV, which is the final block of ciphertext from the previously encapsulated record, is XORed with a random mask that is prepended to payload and treated as payload through the encryption process. The diagrams in this section represent this as IVM for IV Mask.

For cipher suites supported by TLS version 1.1, record encapsulation is identical between TLS 1.1 and TLS 1.2. For block ciphers, options bits IE and WB must be set to select a pure random IV.

TLS 1.2 specifies how to use AEAD (authenticated encryption with additional authenticated data) algorithms with TLS. Therefore SEC now supports AES-GCM and AES-CCM with TLS 1.2.

9.2.4.2 Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation

An Initialization Vector (IV) is used to provide per-packet randomization. For CBC-Mode (Ciphers AES or DES), This value is used to randomize the input in a reproducible but unpredictable manner. This randomization prevents attacks based upon knowing the structure of the plaintext. The TLS standard has evolved to provide more cryptographically secure IVs. As such, SEC supports IV generation in three ways: Explicit IV, Implicit IV with Mask, or TLS 1.0 compatibility IV.

- If an Explicit IV is chosen, RNG generates a random IV (shown as Opt IV in [Figure 9-28](#)), which is written to the Class 1 Context Register and also to the output frame.
- If an Implicit IV with Mask is chosen, the IV is extracted from the PDB and written to the Class 1 Context Register. RNG also generates a random IV Mask (shown as Opt IVM in [Figure 9-28](#)), which is encrypted but not authenticated.
- For TLS 1.0 Compatibility Mode, IV is extracted from the PDB, and is written to the Class 1 Context Register. The final block of the encrypted record is saved back to memory for use as the next IV. This truly chained IV is supplemented with a RNG-generated IV Mask, which is prepended to the payload and encrypted, but not authenticated.

Other cipher suites supported by SEC that use an IV are AEAD algorithms, and AES-Counter based cipher suites. In all these ciphers, the IV does not randomize the data, but instead randomizes the key-stream generation process.

- For AES-CCM and AES-GCM, RNG generates a random nonce_explicit (shown in Figure 9-30 and in Figure 9-27 as ne), which is combined with SALT from the PDB to form the nonce. SALT is a form of Write_IV, generated by the PRF as part of key generation.
- For AES-GCM the 12-byte nonce is passed into the input Data FIFO tagged as IV.
- For AES-CCM, the nonce is formed from the appropriate 4 byte Write IV (server or client), concatenated with the 8 byte Sequence number. The nonce is in turn used to create CTR_0 and B_0 .

AES-CCM mode has a much more complex use model for the IV than other modes. As said above, the IV is combined with WRITE_IV32 that has been programmed into the PDB to form a nonce identical to that generated for AES-GCM. This nonce is used differently for AES-CCM, and the SEC TLS state machine contains special instructions to explicitly construct B_0 and CTR_0 , and to write them to the Class 1 Context Register.

B_0 is constructed:

- The first byte is taken from the PDB B_0 Flags field
- The next 12 bytes consist of the Nonce
- The final 3 bytes are constructed by the state machine and reflect the length of the payload being encapsulated

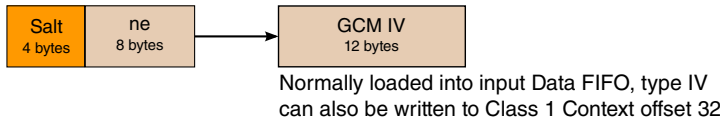
CTR_0 is constructed:

- The first byte is taken from the PDB CTR_0 Flags field
- The next 12 bytes consist of the Nonce
- The final 3 bytes are taken from the CTR_0 Constant field in the PDB.

NOTE

Proper CCM-mode encapsulation relies upon proper programming of B_0 Flags, CTR_0 Flags, and CTR_0 Constant fields into the Shared Descriptor PDB. Per RFCs 3610 and 6655, the CTR_0 Constant should be programmed with zeros, CTR_0 Flags should be programmed with 0x02, and B_0 Flags should be programmed to 0x7A for ciphersuites requiring a 16-byte authentication tag, and to 0x5A for cipher suites requiring an 8-byte authentication tag.

GCM IV (Nonce)



CCM Nonce

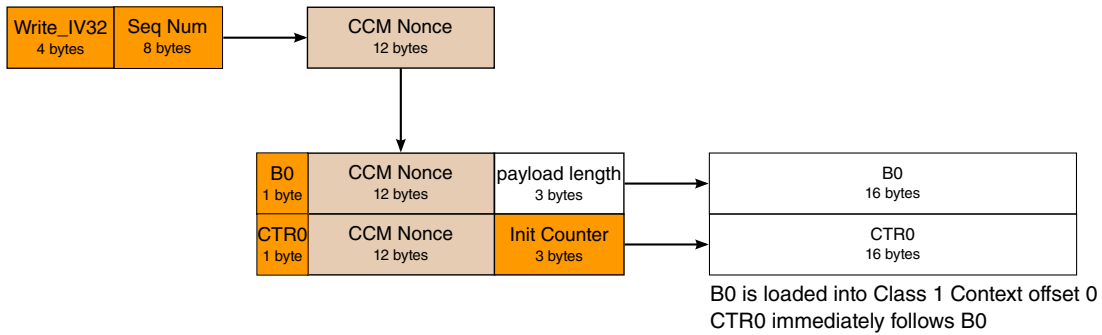


Figure 9-27. TLS 1.1 /1.2 Nonce Generation and use for AEAD ciphers

9.2.4.3 Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers (AES or DES)

This figure shows TLS 1.1/1.2 authentication for encapsulation using block ciphers (AES or DES).

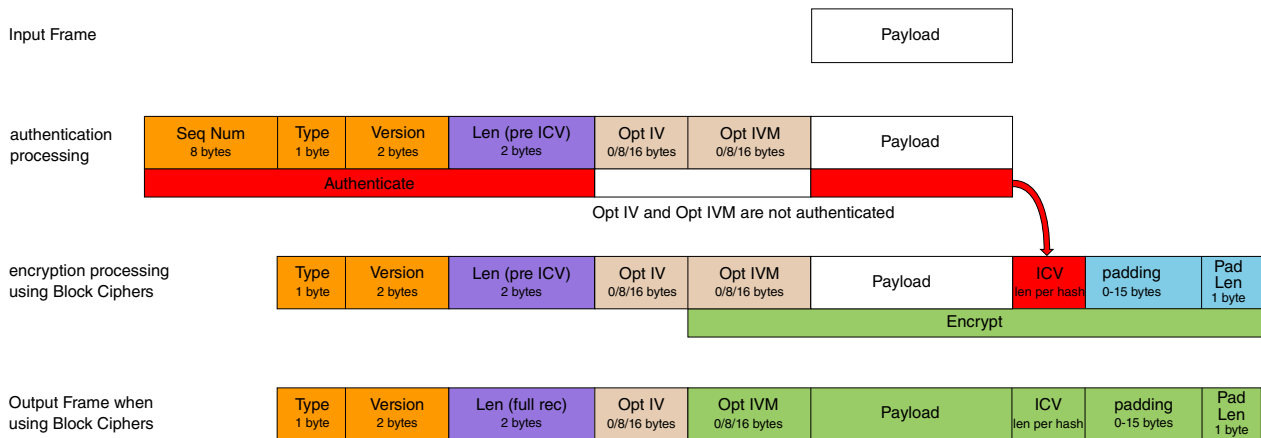


Figure 9-28. TLS 1.1/1.2 authentication for encapsulation using block ciphers (AES or DES)

For block cipher processing, the length of either the IV or the IV Mask is also added to the record header length field. Although the transmitted length field is of the record (except the TLS record header), the length field used during authentication is the length of the payload itself.

Payload is processed as follows:

1. As payload is extracted from the Input Frame, it is pushed onto the input-data FIFO, tagged for both encryption and authentication.
2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the information FIFO.
3. The ICV computed as a result of authentication is pushed back into the Class 1 CHA, and is also encrypted.
4. Following the ICV, SEC adds the minimal padding and a pad length byte such that the pad length byte is the last byte in a cipher block. The value of every byte of padding is the same as the value of the pad length byte.

Example: An AES-128-CBC-SHA ciphersuite with a plaintext-payload length of 32 bytes

Using an AES-128-CBC-SHA ciphersuite with a plaintext-payload length of 32 bytes, the ICV is 20 bytes.

Payload (32) + ICV (20) = 52 bytes.

The next multiple of the AES-CBC block size of 16 is 64.

Therefore, SEC appends 12 bytes, each with value 0Bh, immediately after the ICV.

5. This SEC-generated padding is pushed into the Class 1 CHA for encryption.
6. The resulting encrypted payload, which includes the encrypted ICV and the encrypted padding and pad length, is pushed onto the output frame.

9.2.4.4 Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers

IVs are not transmitted for stream ciphers.

Payload is processed as follows:

1. As payload is extracted from the Input Frame, it is pushed onto the input-data FIFO, tagged for both encryption and authentication.

2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the iNformation FIFO.
3. The ICV computed as a result of authentication is pushed back into the Class 1 CHA, and is also encrypted.
4. The resulting encrypted payload (which includes the encrypted ICV is pushed onto the output frame.

Note that for stream ciphers, padding is not required, so the padding and Pad Len fields are skipped altogether.

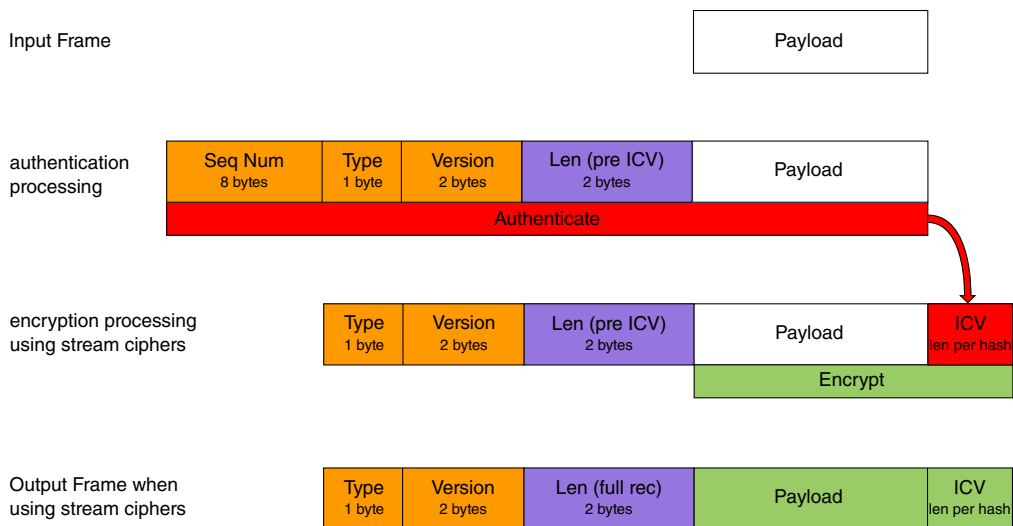


Figure 9-29. TLS 1.1 /1.2 encapsulation for stream ciphers

9.2.4.5 Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers

AEAD stands for Authenticated Encryption with Additional Data. Introduced to TLS with version 1.2, it provides a new structure: an independently vetted algorithm that combines encryption and authentication in one.

For both AES-CCM and AES-GCM, payload is processed as follows:

1. As payload is extracted from the input frame, it is pushed into the input-data FIFO and tagged for both encryption and authentication.
2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the iNformation FIFO.
3. The ICV computed as a result of authentication is also encrypted.

4. The resulting encrypted payload, which includes the ICV, is pushed onto the output frame, as shown in the following figure.

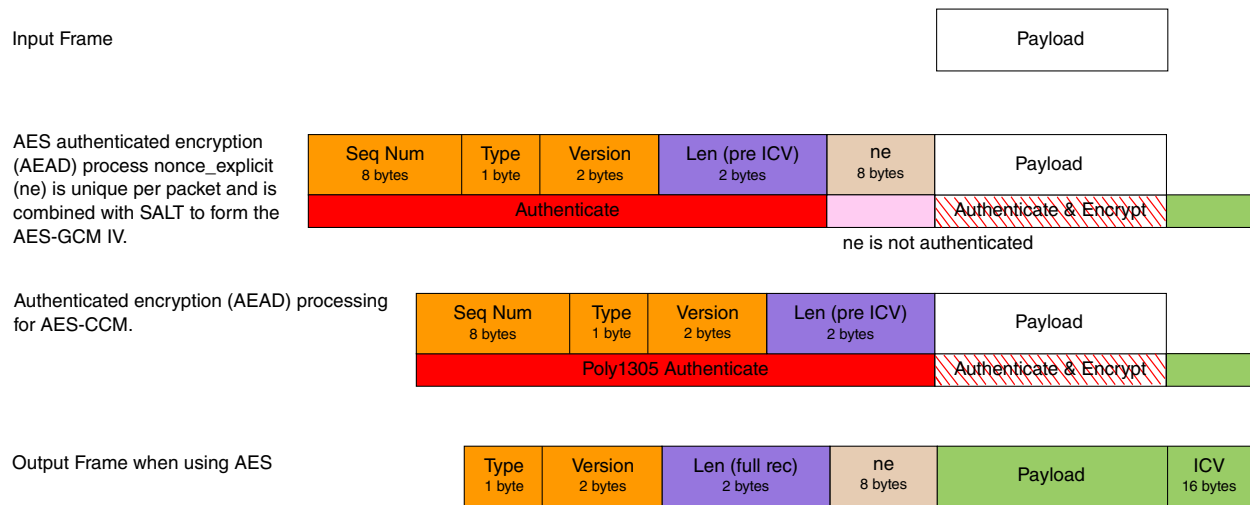


Figure 9-30. TLS 1.2 encapsulation for AEAD ciphers

For AES-GCM ciphers, TLS 1.2 defines a special value: the nonce. The default nonce is 12 bytes and is comprised of two values from two sources: the salt and the nonce_explicit (ne). Nonce generation is discussed in [Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation](#).

- Salt is generated as key material and remains constant throughout the lifetime of the keys.
- nonce_explicit is generated randomly by RNG for each frame.

In the AES-GCM algorithm, the 12-byte nonce is used as the GCM-IV, which sets the first counter value used for encryption.

The TLS 1.2 protocol thread also supports encapsulation using AEAD algorithm AES-CCM. AES-CCM uses a 12-byte nonce generated from the 4 byte Write IV and the 8-byte Sequence Number. However the usage is different, as is the method for programming AESA with the nonce. Section [Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation](#) provides more details.

9.2.5 Process for TLS 1.1 and TLS 1.2 record decapsulation

For TLS record decapsulation, SEC must authenticate:

- All the plaintext fields

- Implicit sequence number
- Decrypted payload

The details depend upon the type of cipher suite used. For cipher suites based on block ciphers (using either AES-CBC or DES-CBC), refer to [Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used](#). For cipher suites based on stream ciphers, refer to [Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used](#). For cipher suites based on AEADs (Authenticated Encryption with Additional Data), refer to [Decapsulation of TLS 1.2 records when an AEAD is used](#).

In general, processing consists of two different computations: decryption, and integrity checking. During encapsulation, the record and the original header was integrity protected; the record length reflecting the header and payload, but not any other additions resulting from encapsulation. As a result, for decapsulation, the record header has to be modified to reflect the original length.

For block and stream cipher encapsulation, the integrity computation was performed on plaintext. So for decapsulation, the payload must be decrypted before being integrity checked. AEAD ciphers define an algorithm (or pair of algorithms) that perform both encryption and integrity computations. TLS performs the algorithm as specified by the algorithm definition.

In all cases, the sequence number is extracted from the PDB and is included in the integrity computation, prior to the record header.

9.2.5.1 Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used

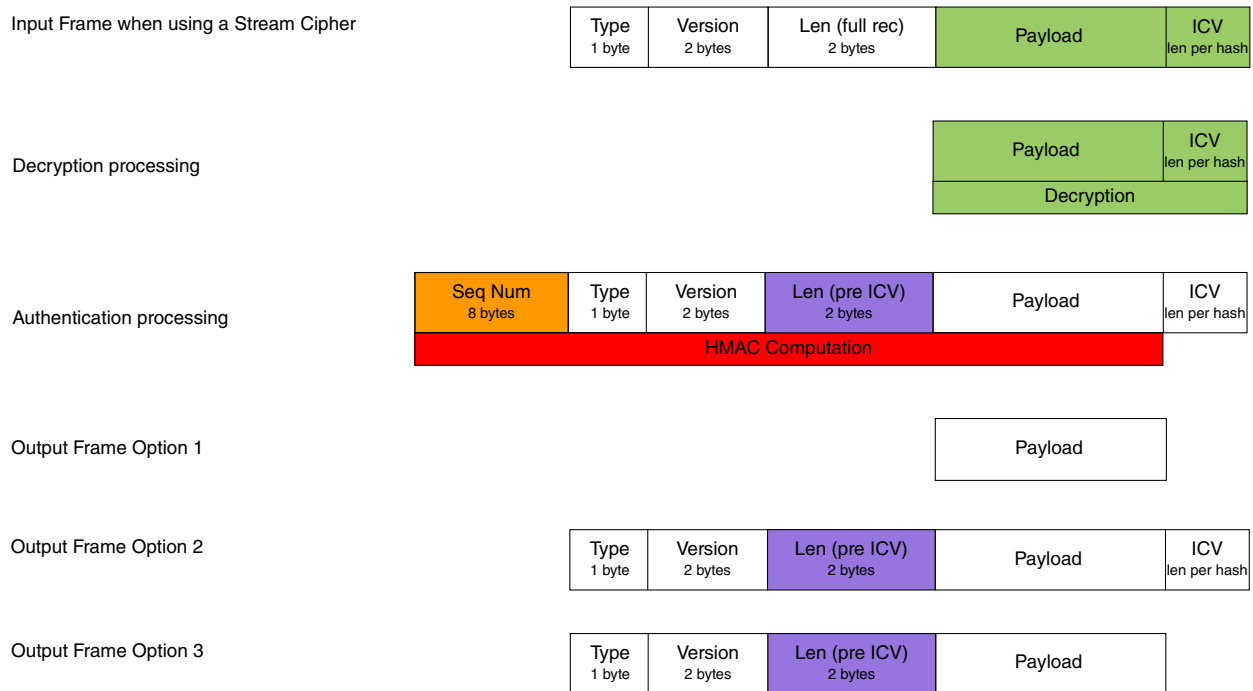


Figure 9-31. TLS 1.1 / 1.2 Decapsulation when using a Stream Cipher

A stream cipher is the simplest TLS construction for SEC to decapsulate. The record header gets pre-processed, so that the length of the ICV can be removed from the record length. The modified header is then passed into MDHA to be the first segment of input frame to be integrity checked.

Subsequent bytes are all encrypted during encapsulation, thereby requiring decapsulation. All encrypted bytes up to but not including the encrypted ICV are tagged "class 1 outsnop to class 2", thereby automatically shunting the decrypted bytes into MDHA for integrity checking.

Decrypted payload is written out to the memory location specified. If the PDB Options outFMT field selects, the record header precedes the payload, and if the outFMT selects, the decrypted ICV will be written out as well. See [Table 9-46](#) for details on programming outFMT.

The ICV, after decryption, is put into MDHA as an ICV. MDHA, after completing the integrity computation, compares the computed ICV against the decrypted ICV, and signals an ICV error if the match fails.

9.2.5.2 Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used

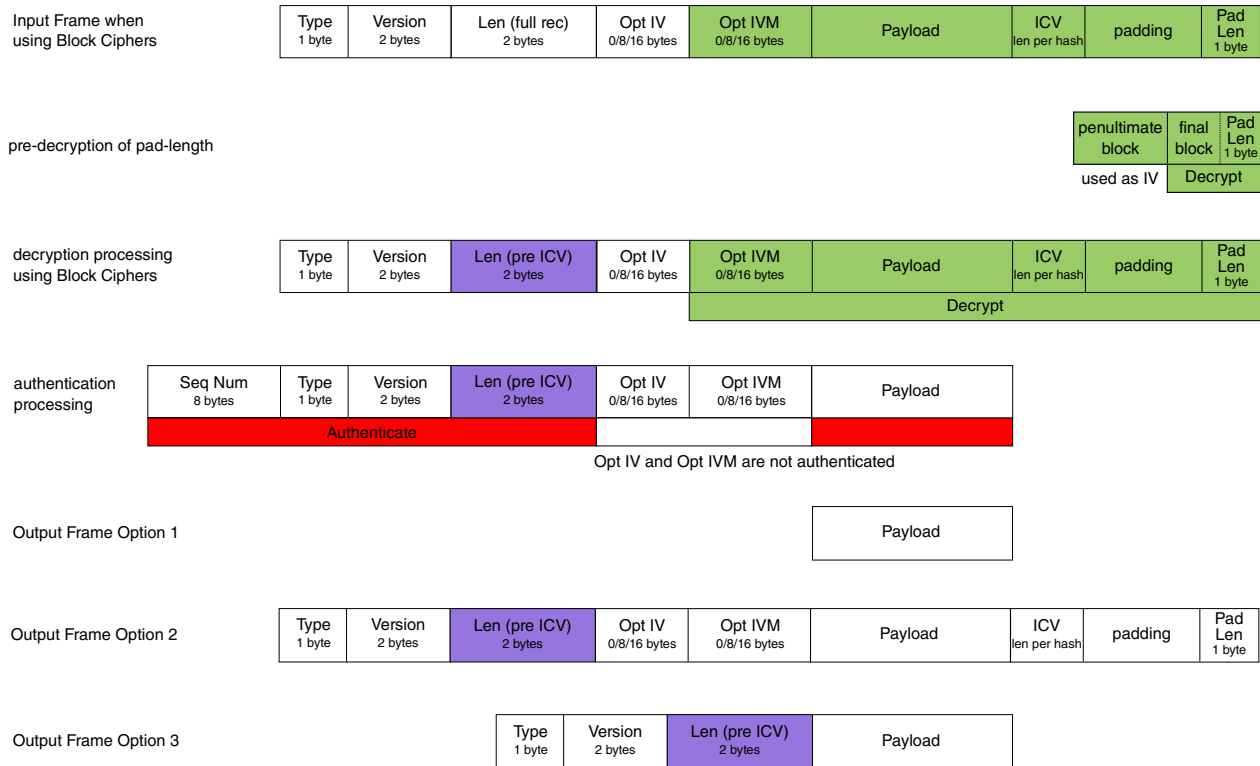


Figure 9-32. TLS 1.1 / 1.2 Decapsulation when using a Block Cipher

A block cipher (AES-CBC, DES-CBC) requires that encryption be performed on multiples of the block size. TLS specifies padding when a block cipher is used, and a pad-length byte is the last byte of the encrypted record. The integrity computation includes the header, which includes the *original* record length. That is, the length of all fields added as part of the encapsulation process need to be subtracted before authentication can begin. In order to compute the HMAC in parallel with decryption, the final byte of the record -- the pad-length byte, must be decrypted *first*. As a result, SEC will start by loading the final two blocks of the record (16 bytes for DES, 32 bytes for AES), and use the first half as IV for decrypting the final block. Once the pad-length byte is decrypted, this material is thrown away, the pre-encapsulation record length is computed, and used for the integrity check process.

From this point, block cipher decapsulation proceeds much like stream cipher decapsulation -- the modified header is passed into MDHA to be the first segment of the input frame to be integrity checked.

Block Ciphers use an IV to randomize the input, to thwart cryptanalysis. TLS 1.0 uses a chained IV -- literally the final block of ciphertext in record *i* becomes the IV for record *i* +1. TLS 1.2 specifies use of a random IV; setting PDB options WB=0 and IE=1 is used to specify a pure random IV that is explicitly part of the encapsulated record. For the IV, TLS 1.1 is a transitional standard that allows for an implicit IV or an explicit IV, and allows a random mask to be used. IE and WB can be programmed as required for TLS 1.1. The IV, whether implicit or explicit, is not included in the integrity computation. More info on programming WB and IE can be found at [Table 9-46](#).

Subsequent bytes are all encrypted during encapsulation, thereby requiring decapsulation. All encrypted bytes up to but not including the encrypted ICV are tagged "class 1 outsnop to class 2", thereby automatically shunting the decrypted bytes into MDHA for integrity checking.

Decrypted payload is written out to the memory location specified. If the PDB Options outFMT field selects, the record header precedes the payload, and if the outFMT selects, the decrypted ICV and padding will be written out as well. SEC does not check the contents of the padding bytes after decryption, so outFMT should be programmed to 01b to select output type 2 if software checking of the padding bytes is required. Note that the ICV computation does cover the padding bytes. See [Table 9-46](#) for details on programming outFMT.

The ICV, after decryption, is put into MDHA as an ICV. MDHA, after completing the integrity computation, compares the computed ICV against the decrypted ICV, and signals an ICV error if the match fails.

9.2.5.3 Decapsulation of TLS 1.2 records when an AEAD is used

TLS 1.2 adds the capability to use AEAD ciphers for record protection.

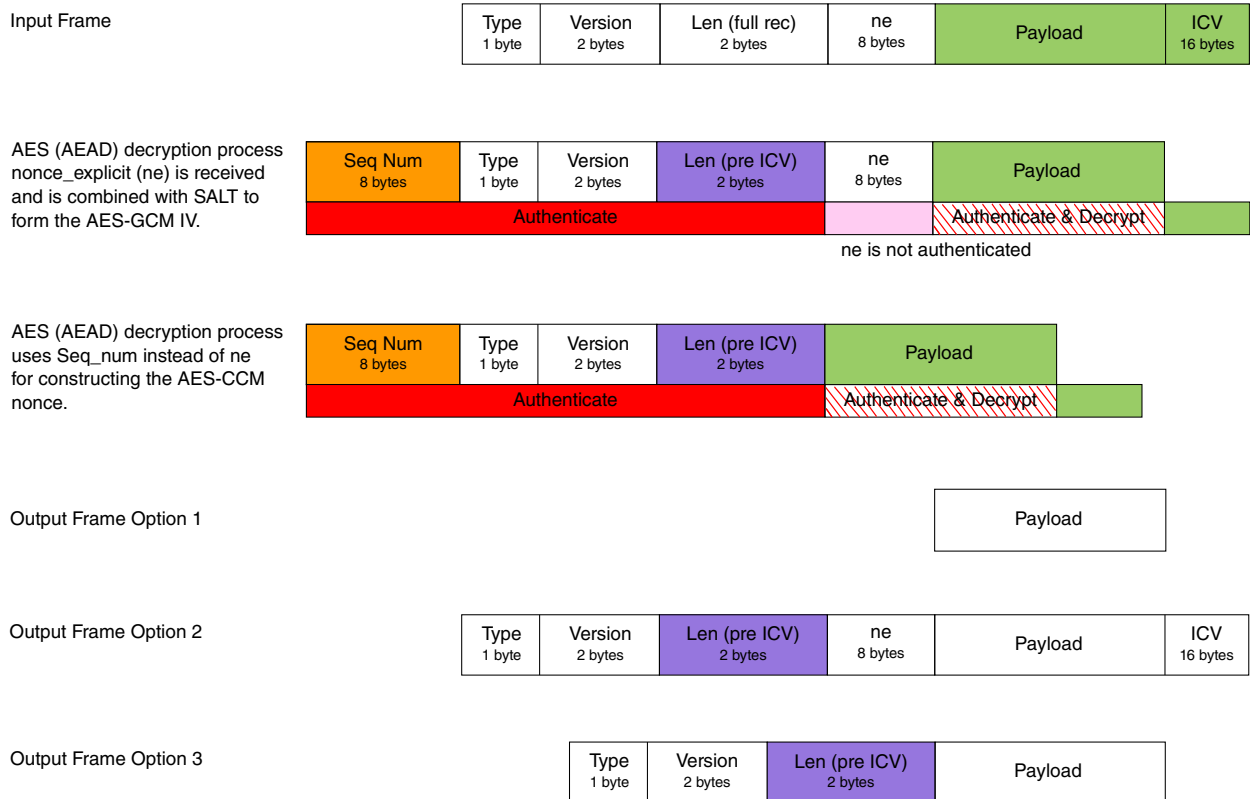


Figure 9-33. TLS 1.1 / 1.2 Decapsulation when using an AEAD Cipher

AEAD ciphers often include use of a nonce or an IV. Please refer to [Figure 9-27](#) for more information about constructing the nonce or IV for encapsulation. Note that in the case of decapsulation, the received nonce_explicit is used instead of generated.

AES-GCM and AES-CCM processes the record very similarly to a stream cipher, that nonce_explicit (ne in figures) is not included in either decryption nor the integrity check. The nonce_explicit is used as a parameter to randomize the key-stream generated by AES-Counter mode. (AES-Counter is the confidentiality portion of AES-GCM and AES-CCM).

AES-CCM, like AES-GCM, is a AEAD cipher using AES-Counter for confidentiality. However SEC implements AES-CCM a little differently, in that it creates a 12-byte NONCE from the 4-byte Write IV and the 8-byte Sequence Number. The 12-byte nonce is then used as shown in [Figure 9-27](#) to construct the AES-CCM IV (B_0) and Initial Counter (CTR_0).

9.2.6 Process for DTLS record encapsulation

This version of SEC supports DTLS versions 1.0 and 1.2, and supports several cipher suites, including:

- Triple-DES-CBC with HMAC-SHA-1
- AES-128-CBC with HMAC-SHA-1
- AES-256-CBC with HMAC-SHA-256
- AES-128-CTR with HMAC-SHA-1
- AES-256-GCM
- AES-128-CCM-8

Note this list is not complete. For a complete list, please see [Table 7-54](#).

9.2.6.1 Differences between DTLS and TLS

DTLS 1.0 is a variant of TLS 1.1, and DTLS 1.2 is a variant of TLS 1.2. The DTLS procedure for record encapsulation is different from TLS in that:

- DTLS requires the insertion of an explicit sequence number.
- DTLS authenticates the header fields in a different order than transmitted. The field order for authentication matches how TLS performs authentication.

The explicit Sequence Number is necessary to support the DTLS-specific requirement of support for out-of-order reception of records. Because TLS uses implicit sequence numbers, it cannot support out-of-order reception.

9.2.6.2 Process of DTLS Record Encapsulation when using a Block Cipher

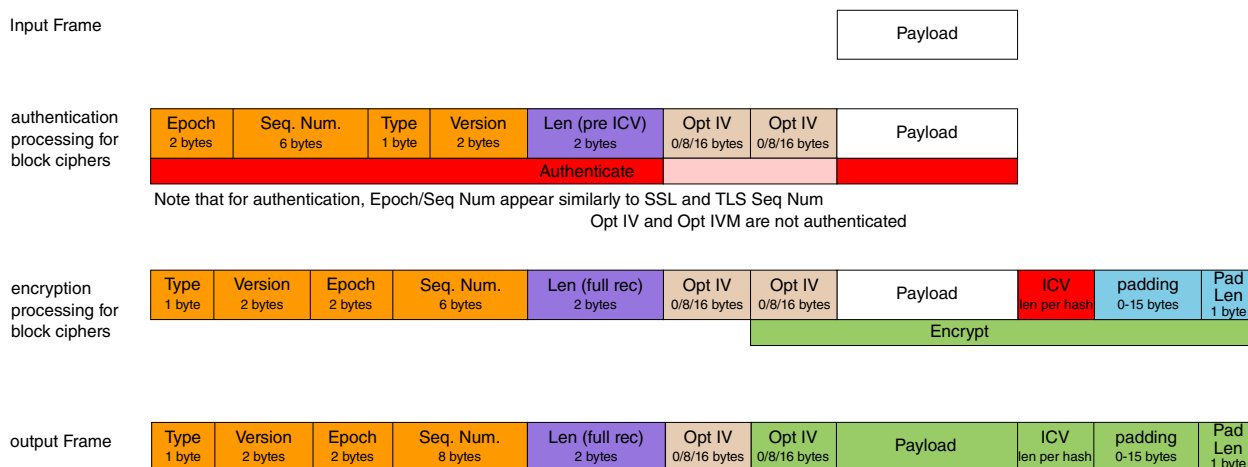


Figure 9-34. DTLS Record Encapsulation when using a Block Cipher

The DTLS block cipher encapsulation procedure is as follows:

1. SEC begins encapsulation when it receives an input frame containing the payload requiring encapsulation.
2. SEC extracts the version, epoch, and sequence number fields from the PDB.
3. SEC checks the most significant bit of the Datapath Override register:
 - If DPOVRD[MS] is cleared, SEC extracts the record type from the PDB
 - If DPOVRD[MS] is set, SEC instead takes the record type field comes from the least significant byte of the DECO Protocol Override Register.
4. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

NOTE

The order of these fields for authentication is different than for transmission.

5. The payload length, which is part of the frame description, is extracted and pushed as a two byte field into the Class 2 CHA for authentication, after adding to it to reflect the length of the record header and the IV. In the diagram, this is called Len (pre ICV)
6. Another record length reflecting the added ICV, padding, and the pad length byte is computed. This version of length is pushed into the output frame, and is transmitted in the clear. Pad Length is computed to be the minimum required for the chosen cipher suite, such that the total length is the smallest multiple of the block size of the

chosen cipher required to be able to encrypt Payload, ICV, padding, and the pad length byte.

7. The sequence number is then incremented and updated to the PDB in memory.
8. PDB Options bits IE and WB are examined to determine how to construct the IV.
 - If w/b is set, then the last block of ciphertext from the previous block was saved in the PDB IV field. This is extracted and XORed with a random number to form the IV
 - If e/i is set, then the the IV is explicitly included prior to the payload. Otherwise, the IV is encrypted as if part of the payload.

NOTE

The proper setting for DTLS 1.2 is w/b=0 and e/i=1.

9. Payload, ICV, padding, and the pad length byte are encrypted use the selected CHA, using CBC mode; the encryption product is pushed into the output frame.

9.2.6.3 Process of DTLS Record Encapsulation when using a Stream Cipher

SEC performs DTLS AES-Counter mode encapsulation very similarly to TLS 1.1 record encapsulation as described in [Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers](#). Like DTLS Block cipher encapsulation, the major difference between TLS AES-Counter and DTLS AES-Counter processing is authentication placement of the explicit sequence number.

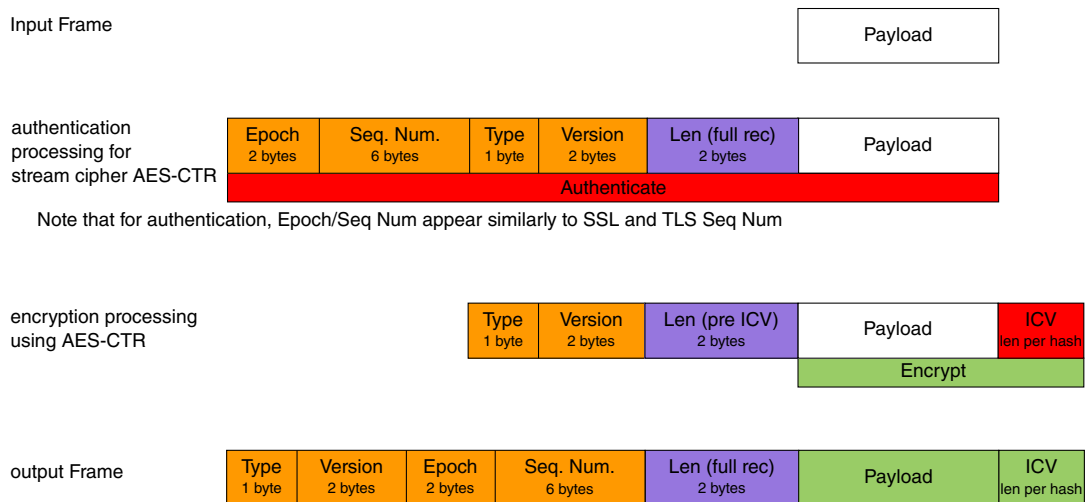


Figure 9-35. DTLS Record Encapsulation when using a Stream Cipher

The DTLS stream-cipher encapsulation procedure is as follows:

1. SEC begins encapsulation when it receives an input frame containing the payload requiring encapsulation.
2. SEC extracts the version, epoch, and sequence number fields from the PDB.
3. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

NOTE

The order of these fields for authentication is different than for transmission.

4. The sequence number and Write_IV are extracted from the PDB and written into Class 1 Context to become the packets' initial counter value for AES-Counter Encryption.
5. The sequence number is then incremented and updated to the PDB in memory.
6. The payload length, which is part of the frame description, is extracted and pushed as a two byte field into the Class 2 CHA for authentication.
7. The ICV length is added to the payload length. This variant of payload length is pushed into the output FIFO and becomes part of the output frame
8. Payload and ICV are encrypted using AES Counter mode; the encryption product is pushed into the output frame.

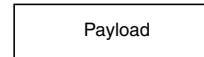
9.2.6.4 DTLS 1.2 Record Encapsulation when using an AEAD Cipher

This version of SEC supports two different AEAD ciphers:

- AES-GCM (using 128, 192, or 256 bit keys)
- AES-CCM (using 128, 192, or 256 bit keys)

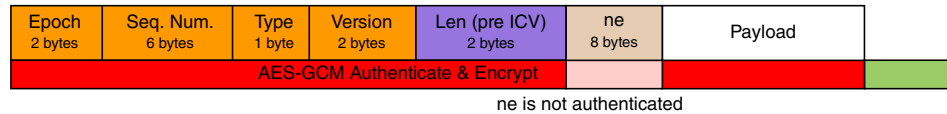
Each AEAD cipher suite operates a little differently. Critical to the cipher is the underlying nonce. In each case, nonce / IV construction is performed identically to how built for TLS 1.2, as described in [Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation](#); particularly as shown in [Figure 9-27](#). RFC 6347 specifies that DTLS 1.2 AEAD cipher suites operate identically the same as with TLS 1.2.

Input Frame

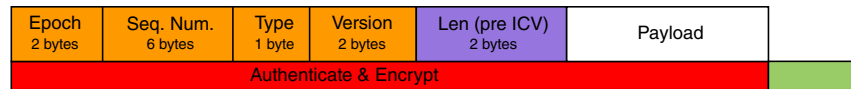


AES-GCM authenticated encryption (AEAD) processing.

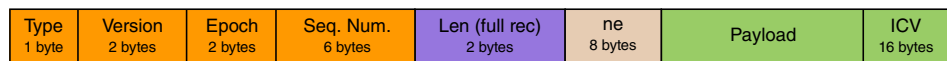
none_explicit (ne) is unique per packet and is combined with SALT to form the AES-GCM IV.



AEAD Processing with AES-CCM B0 and CTR0 are formatted from values stored in the PDB.



Output Frame when using AES-GCM



Output Frame when using AES-CCM



Figure 9-36. DTLS Record Encapsulation when using an AEAD Cipher

For AEAD ciphers, the authenticate-only segment consists of a reordered record header, and is constructed by extracting the epoch, sequence number, type and version from the PDB, and then by using the record length, which reflects the header and payload length, but not the length of the ICV, nor for AES-GCM, of the Nonce Explicit. This construct is passed to the class 1 CHA as type AAD.

The record header is passed to the output frame in proper order, consisting of type, version, epoch, sequence number, and for AES-GCM, nonce_explicit. Nonce Explicit is a random number. Included in the record header is the full record version of the record length, which is the length of the encapsulated record, and includes the length of ICV, and for AES-GCM, the length of nonce_explicit.

9.2.7 Process for DTLS record decapsulation

This version of SEC supports DTLS versions 1.0 and 1.2, and supports several cipher suites, including:

- Triple-DES-CBC with HMAC-SHA-256
- AES-256-CBC with HMAC-SHA-1
- AES-128-CCM-16

Note this list is not complete. For a complete list, please see [Table 7-54](#).

9.2.7.1 Differences between DTLS and TLS

DTLS 1.0 is a variant of TLS 1.1, and DTLS 1.2 is a variant of TLS 1.2. The DTLS procedure for record decapsulation is different from TLS in that:

- DTLS requires the reception of an explicit sequence number.
- DTLS authenticates the header fields in a different order than received. The field order for authentication matches how TLS performs authentication.

The explicit Sequence Number is necessary to support the DTLS-specific requirement of support for out-of-order reception of records. Because TLS uses implicit sequence numbers, it cannot support out-of-order reception.

Because DTLS support out-of-order reception, anti-replay checking is available as part of SEC DTLS decapsulation processing. Please refer to [Anti-Replay built-in checking](#) for more details on how anti-replay works. Anti-Replay operation is controlled by the PDB options byte, and state is maintained in the PDB. Details can be found in [PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation](#).

9.2.7.2 Process of DTLS Record Decapsulation when using a Block Cipher

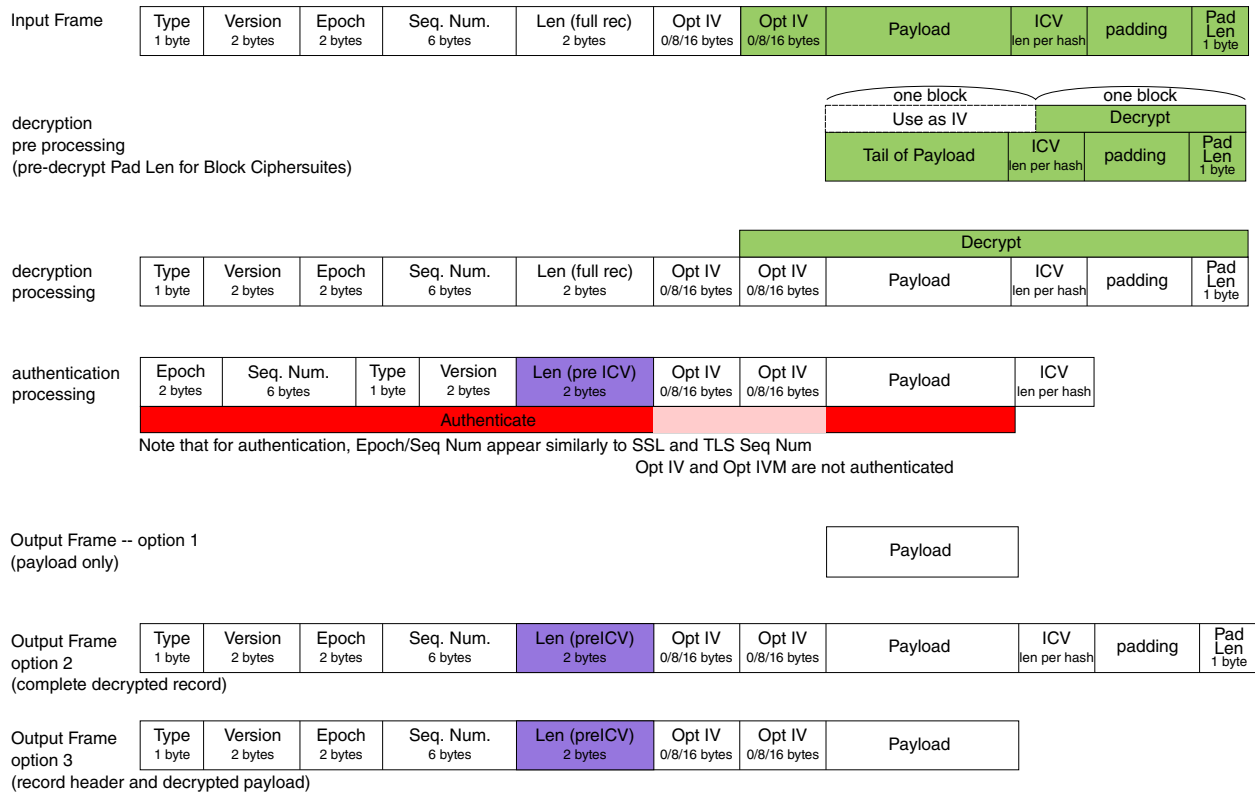


Figure 9-37. DTLS Record Decapsulation when using a Block Cipher

The DTLS block cipher decapsulation procedure is as follows:

1. SEC begins decapsulation when it receives an input frame containing the encapsulated payload.
2. SEC determines the number of bytes of padding by fast-forwarding to the last two cipher blocks of the message, and uses the second to last block as an IV to decrypt the final block, the last byte of which is the pad length byte.
3. SEC rewinds to beginning of the frame, and extracts the type, version, epoch, sequence number, and record length fields from the input frame.
4. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA.

NOTE

The order of these fields for authentication is different than received.

5. SEC takes the received record length field, which reflects the length of the encapsulated frame, and subtracts the length of ICV, padding, and the pad length byte. This computed record length (shown in diagrams as "length (pre ICV)") is passed into the class 2 CHA for authentication.
6. If the PDB Options byte field OutFmt so selects, the record header, including the adjusted record length field, is written to the output frame.
7. If enabled, anti-replay checking is performed, updating state to the PDB.
8. Decryption is performed. The decrypted payload is passed into the class 2 CHA for authentication, and is written to the output frame.
9. Once the HMAC computation is complete, the received ICV is compared to the computed ICV. Any mismatch between the two ICVs generates an error and is signalled back.

9.2.7.3 Process of DTLS Record Decapsulation when using a Stream Cipher

SEC performs DTLS AES-Counter mode decapsulation very similarly to TLS 1.1 record decapsulation as described in [Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used](#). Like DTLS Block cipher decapsulation, the major difference between TLS AES-Counter and DTLS AES-Counter processing is authentication placement of the explicit sequence number.

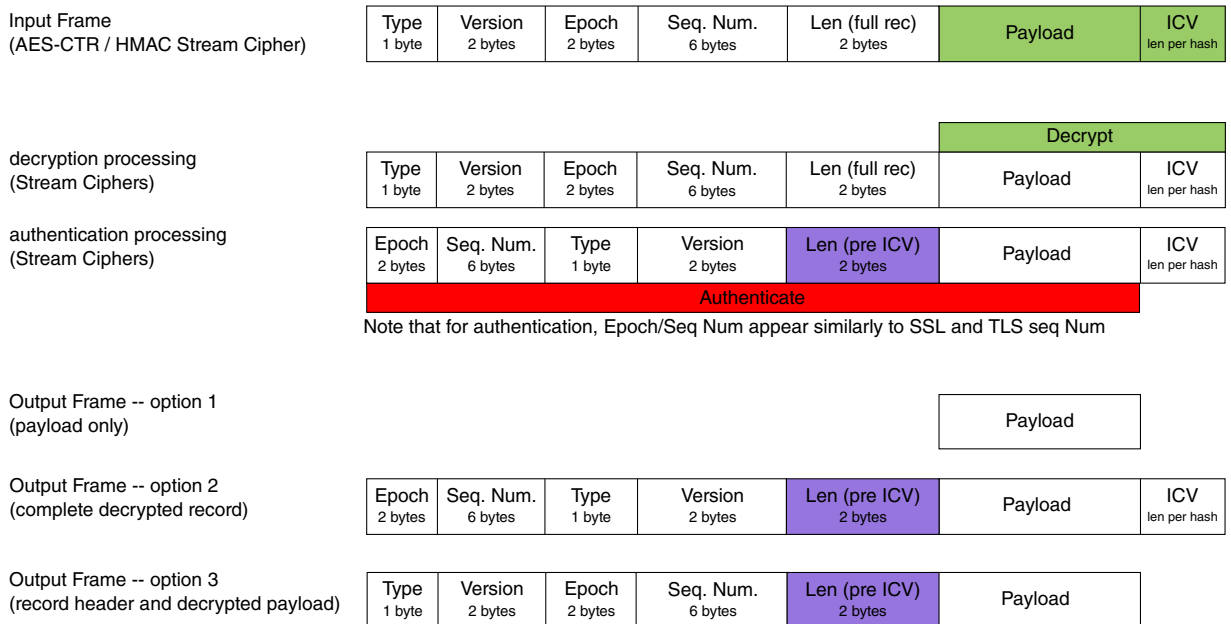


Figure 9-38. DTLS Record Decapsulation when using a Stream Cipher

The DTLS stream-cipher decapsulation procedure is as follows:

1. SEC begins decapsulation when it receives an input frame containing the encapsulated record.
2. SEC extracts the version, epoch, sequence number, and record length fields from the input frame.
3. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

NOTE

The order of these fields for authentication is different than from the input frame.

4. The received record length is adjusted by subtracting the length of ICV from it. The adjusted record length is then pushed into the class 2 CHA for authentication.
5. If so selected by the PDB Options byte field outFMT, the record header, including the adjusted record length, is written to the output frame.
6. The sequence number and Write_IV are extracted from the PDB and written into Class 1 Context to become the packets' initial counter value for AES-Counter Decryption.
7. If enabled, SEC performs anti-replay checking, and updates PDB state as a result.
8. Payload and ICV are decrypted using AES Counter mode; the decrypted payload is pushed into the output frame and into the class 2 CHA for authentication processing.
9. Once the HMAC computation is complete, it is compared to the received HMAC, and an error is signalled if not identical.
10. If enabled by outFMT, then the decrypted ICV is written to the output frame.

9.2.7.4 DTLS 1.2 Record Decapsulation when using an AEAD Cipher

This version of SEC supports two different AEAD ciphers:

- AES-GCM (using 128, 192, or 256 bit keys)
- AES-CCM (using 128, 192, or 256 bit keys)

Each AEAD cipher suite operates a little differently. Critical to the cipher is the underlying nonce. In each case, nonce / IV construction is performed identically to how built for TLS 1.2, as described in [Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation](#); particularly as shown in [Figure 9-27](#). RFC 6347 specifies that DTLS 1.2 AEAD cipher suites operate identically the same as with TLS 1.2.

SSL/TLS/DTLS record encapsulation and decapsulation overview

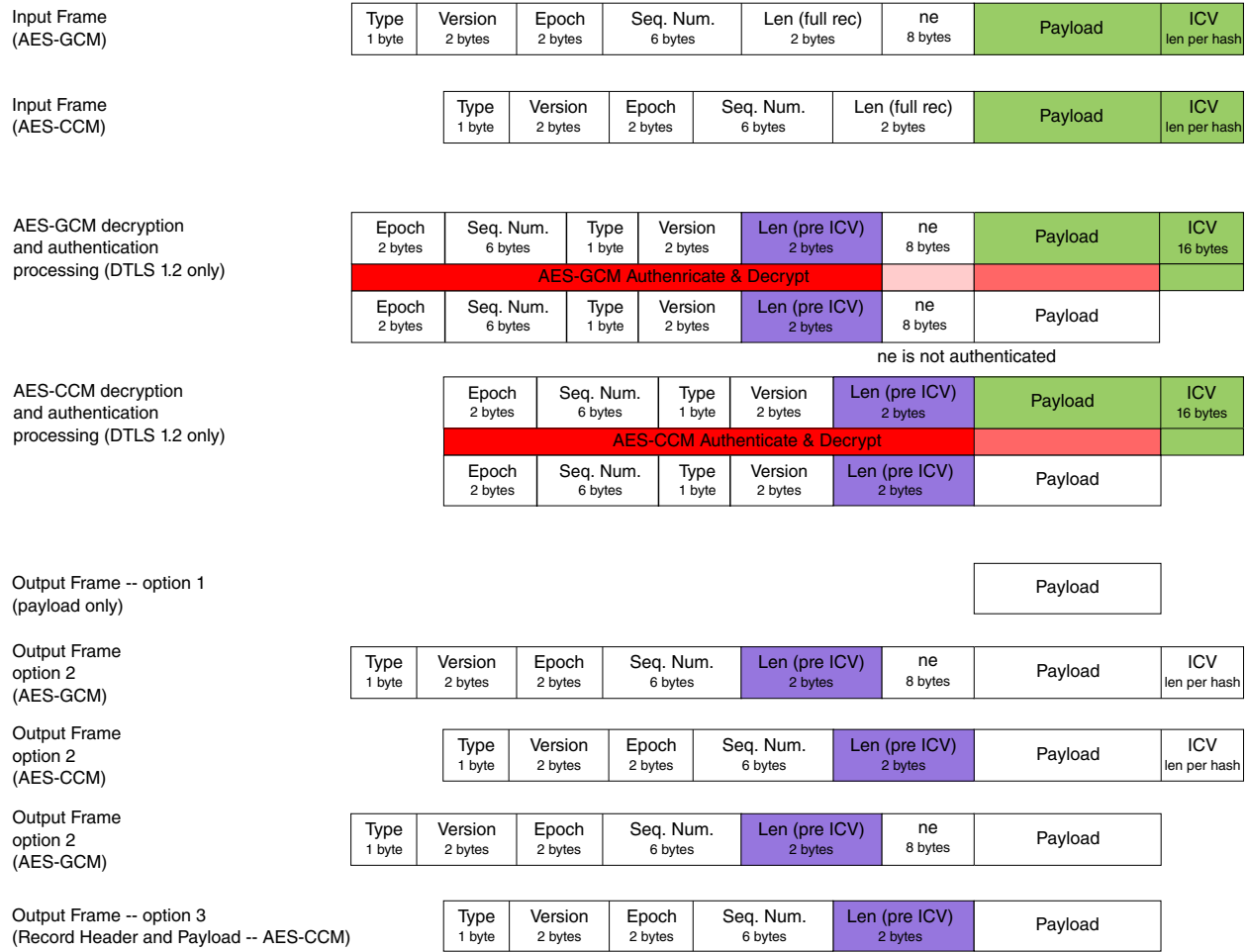


Figure 9-39. DTLS Record Decapsulation when using an AEAD Cipher

The DTLS AEAD-cipher decapsulation procedure is as follows:

1. For AEAD ciphers, the authenticate-only segment consists the received record header, reordered to put epoch and sequence number first and is constructed like for TLS, and with the length field is adjusted to remove the length of the ICV. This modified record header is passed to the class 1 CHA as type AAD.
2. For AES-GCM, the nonce_explicit (ne) field is extracted from the input frame and is used to construct the nonce. Nonce_explicit is not authenticated.
3. If selected by the PDB options byte field outFMT, the record header is passed to the output frame in same order as received in the input frame, consisting of type, version, epoch, sequence number, the *adjusted* record length, and for AES-GCM, nonce_explicit.
4. The encrypted message is passed into the class 1 CHA with type set to message data. The ICV is passed into the class 1 CHA with type set to ICV. Decrypted payload is

written to the output frame. If outFMT is selected to do so, then SEC writes the ICV to the output frame.

9.3 SRTP packet encapsulation and decapsulation

SRTP, which stands for secure real-time transport protocol, is defined in RFC 3711 as a cryptographic encapsulation of RTP, which stands for real time protocol. RFC 3711 defines two ciphers and one authentication function for use in SRTP encapsulation and decapsulation. SEC supports the following:

- AES-Counter Mode for confidentiality
- SHA-1 for authentication.
- AEAD AES-GCM for confidentiality and authentication
- AEAD AES-CCM for confidentiality and authentication

SEC does not support AES-f8, which is the remaining cipher specified by RFC 3711.

SEC's built-in SRTP protocol supports data encapsulation, encryption, and data integrity checking. The PROTINFO field codes enumerated in [Table 7-53](#) define the specific encryption and data integrity algorithms to be used by the protocol, and the hardware handles the remaining details.

Table 9-50. SRTP protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block		Protocol data block
Class 2 key data block		Class 2 key data block
Class 1 key data block		Class 1 key data block
Protocol = SRTP encrypt		Protocol = SRTP decrypt

NOTE

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a [split key](#). Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the [Derived Key Protocol](#) may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

9.3.1 Building the initial counter value (Counter IV)

The first 16 to 80 bytes of an RTP packet consists of a series of fields that are authenticated, but not encrypted, by the encapsulation process. This series of fields is referred to as the SRTP header.

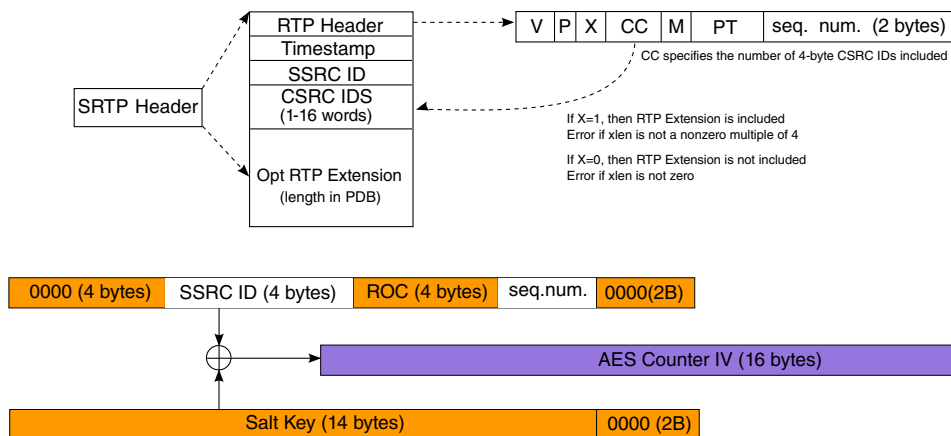


Figure 9-40. SRTP encapsulation and decapsulation AES Counter IV Preparation

AES counter mode requires an initial counter value (Counter IV). SRTP specifies that the initial counter value is obtained by performing a 112-bit bitwise XOR function of the Salt key (14 bytes of extra key material) with a concatenation of the following three fields (found on the input frame): the 4-byte SSRC ID, the 2-byte sequence number, and the 4-byte ROC.

- The SSRC ID is found in bytes 9-13 of the SRTP Header.
- The sequence number is found in bytes 2-3 of the SRTP Header.
- The rollover counter (ROC) is an RTP parameter that is incremented each time the sequence number rolls over; each RTP packet is generated with a monotonically increasing sequence number).

9.3.2 Building the AEAD Nonce

Both AEAD cipher suites supported by SRTP require the construction of a 12-byte Nonce. AES-GCM uses the 12-byte nonce as the 12-byte GCM-IV; effectively becoming the initial counter value for encrypting the packet. AES-CCM requires construction of 16-byte words B_0 and Initial Counter from the 12-byte Nonce.

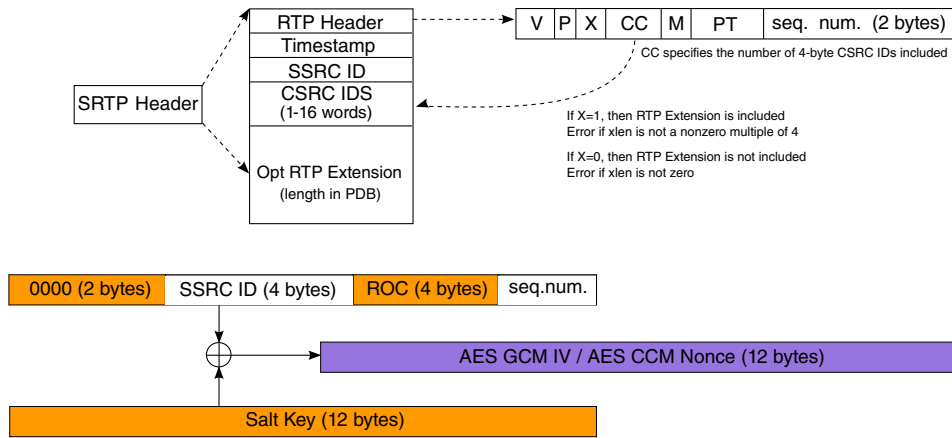


Figure 9-41. SRTP AEAD Nonce Preparation

The Nonce is constructed by taking the 12-byte Salt Key from the PDB, and XORing that value with a 12-byte value constructed from the 4-byte SSRC from the input frame, the 4-byte ROC from the PDB, and the 2-byte Sequence Number from the input frame (the two most significant bytes are treated as zero to construct a 12-byte value).

- The SSRC ID is found in bytes 9-13 of the SRTP Header.
- The sequence number is found in bytes 2-3 of the SRTP Header.
- The rollover counter (ROC) is an RTP parameter that is incremented each time the sequence number rolls over; each RTP packet is generated with a monotonically increasing sequence number).

9.3.3 Constructing the AESA context from the SRTP AEAD Nonce for AES-CCM mode

This figure shows the construction of SRTP AESA context for AES-CCM mode.

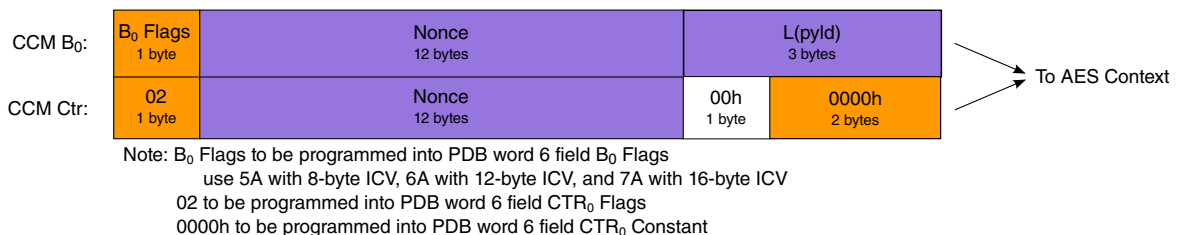


Figure 9-42. SRTP CCM Context construction

SEC uses the nonce to construct both the CCM Initial Counter value and CCM B₀, both of which are written to the Class 1 CHA Context Register.

- The CCM Initial Counter value includes 3 bytes extracted from the protocol data block.
- The CCM B₀ includes one byte extracted from the protocol data block and the length of the payload (as determined by SEC).

9.3.4 SRTP encapsulation

SEC interprets the input frame as:

- A 16-80 byte SRTP Header
- An arbitrary-length RTP payload
- RTP padding that ensures the packet ends on a 4-byte boundary; therefore SEC does not perform padding for SRTP encapsulation.
- A pad length field indicating how many bytes after payload is padding.

Note that the SRTP header contains the following:

- 4-byte RTP header, consisting of a field indicating the number of CSRC IDs included (CC) and an RTP sequence number
- 4-byte timestamp field
- SSRC ID field
- Field from 1 to 16 CSRC IDs, each 4 bytes
- Optional RTP extension header

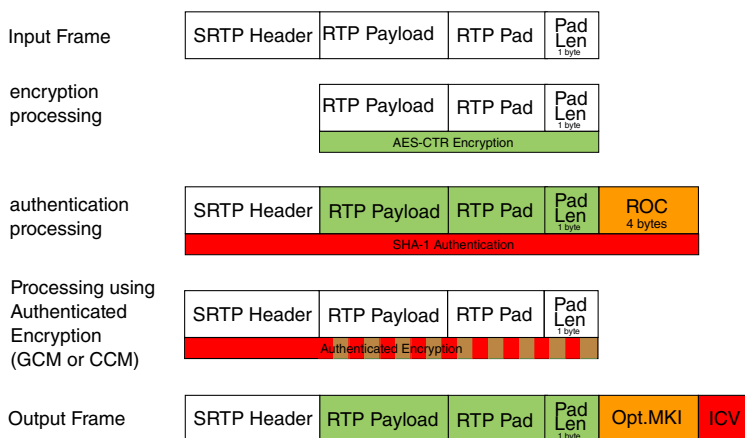


Figure 9-43. SRTP encapsulation process

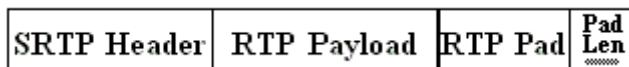


Figure 9-44. SRTP encapsulation input frame

9.3.4.1 Process for SRTP encapsulation

This figure shows SRTP encapsulation with AES Counter encryption and HMAC-SHA-1 authentication.

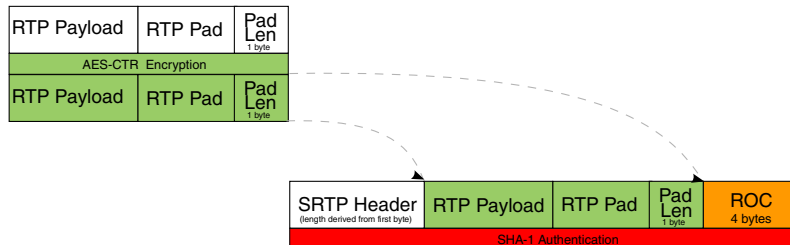


Figure 9-45. SRTP encapsulation encryption and authentication when using AES-Counter and HMAC-SHA-1

1. To begin encapsulation for the AES-Counter / HMAC-SHA-1 cipher suite, SEC builds the Counter IV as described in [Building the initial counter value \(Counter IV\)](#).
2. To begin encapsulation for AEAD cipher suites, SEC builds the Nonce as described in [Building the AEAD Nonce](#).
3. SEC processes the input frame as follows:
 - SEC passes the SRTP header to the Class 2 CHA for HMAC-SHA-1 authentication and then to the output frame.
 - SEC passes the RTP payload, padding, and pad length fields to the Class 1 CHA for AES-Counter encryption.
 - For AEAD Cipher Suites, SEC passes the RTP payload, padding, and pad length fields to the Class 1 CHA for Authenticated Encryption. The SRTP header is processed as AAD, and the RTP Payload, RTP Pad, and Pad Length are processed as plaintext. Note that the ROC is not included as part of AAD for Authenticated Encryption. Instead, the ROC is used in Nonce formation.
 - The encrypted result is passed to the output frame immediately following the SRTP header and then to the Class 2 CHA for authentication.
4. When HMAC-SHA-1 is used, the ROC field is the last item authenticated although it appears first in the PDB; it is not passed to the output frame. As noted above, for AEAD encapsulation the ROC is not treated as data for any authentication computation and instead is part of the IV.
5. ROC is incremented whenever the Seq Num in the SRTP Header rolls over (so after use, ROC should be incremented and written back to PDB if the Seq Num value is FFFF).
6. Once authentication completes, the number of bytes of ICV selected is passed to the output frame. For HMAC-SHA-1, the selection is made by the PDB n_tag field. For AEAD cipher suites, the ICV size is defined by the chosen cipher suite.



Figure 9-46. SRTP encapsulation output frame

9.3.4.2 Handling the optional MKI

If present, the optional MKI is copied from the PDB to the output frame immediately following the encrypted pad length and prior to the ICV. However as MKI is not authenticated, it is not copied to the input-data FIFO. The length of the MKI (in 4-byte words) is also stored in the PDB. Note that the length of MKI supported by SEC is limited by the total size of the descriptor buffer as well as the size of the other contents required to be stored in the descriptor buffer.

9.3.4.3 SRTP encapsulation PDB format descriptions

Table 9-51. SRTP encapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM

PDB word 0	(8 bits) x-len	(8 bits) length of MKI	(8 bits) for AES-CTR: n_tag otherwise: reserved (00h)	(8 bits) options [see table below]
PDB word 1	(16 bits) for AES-CTR: constant=0000h otherwise: constant		(16 bits) for AES-CTR: constant=0000h otherwise: reserved (0000h)	
PDB word 2	(16 bits) reserved (0000h)		(16 bits) for AES-CTR: constant=0000h otherwise: reserved (0000h)	
PDB word 3	salt 1			
PDB word 4	salt 2			
PDB word 5	salt 3			
PDB word 6	(16 bits) for AES-CTR: salt 4 for AES-GCM: reserved (0000h) for AES-CCM:		(16 bits) for AES-CTR: constant=0000h for AES-GCM: reserved (0000h) for AES-CCM: Ctr ₀ constant	
	B ₀ flags (8 bits)	Ctr ₀ flags (8 bits)		
PDB word 7	reserved (00000000h)			
PDB word 8 ¹	ROC			
PDB word 9	optional MKI			

1. Written back to PDB in memory, as needed.

Table 9-52. SRTP encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved				MKI	Reserved		

Table 9-53. SRTP encapsulation PDB, description of the options byte

Field	Description
7-4	Reserved
3 MKI	MKI included in Output Frame 0b - MKI not included in Output Frame. 1b - MKI copied from PDB into Output Frame
2-0	Reserved

9.3.4.4 SRTP encapsulation error conditions

This table lists the conditions under which SRTP encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

Table 9-54. SRTP encapsulation error conditions

Condition	Error status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
OPERATION Command PROT ID selects SRTP Encap, and PROTINFO is not a valid protocol	Protocol Command Error
n_tag in the PDB = 0, or n_tag > 20 and cipher suite is AES-Counter with HMAC-SHA-1	Protocol Command Error
[ROC, SEQNUM] overflows	Protocol Sequence Number Overflow

9.3.5 SRTP decapsulation overview

This figure shows SRTP decapsulation.

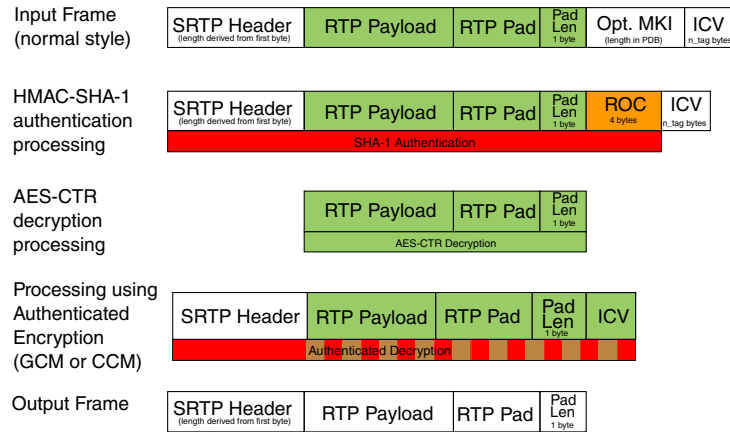


Figure 9-47. SRTP decapsulation overview

To perform decapsulation, SEC receives an input frame interpreted similarly to the encapsulation input frame, with the main differences being the inclusion of an ICV field and an optional MKI field. If present, the optional MKI is located after the padding length and before the ICV.

9.3.5.1 Process for SRTP decapsulation

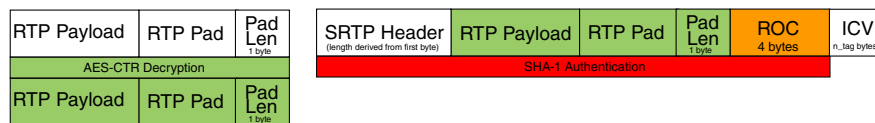


Figure 9-48. SRTP decapsulation decryption and authentication with AES-Counter and HMAC-SHA-1

The SRTP decapsulation procedure for the AES-Counter / HMAC-SHA-1 cipher suite is as follows:

1. SEC grabs the appropriate fields from the SRTP header to generate the Counter IV as described in [Building the initial counter value \(Counter IV\)](#).
2. SEC pushes the SRTP header into the Class 2 CHA for authentication and then onto the output frame.
3. The encrypted payload and padding are passed from the input frame to both CHAs for decryption and authentication.
4. The decrypted results are passed to the output frame.
5. The ROC is pushed into the Class 2 CHA for authentication; if the MKI, is present on the input frame, it is ignored.

6. MDHA completes computation of the ICV across the input frame and compares the result to that received from the input frame. If any difference is detected, a failure is reported.

The SRTP decapsulation procedure for Authenticated Encryption cipher suites is as follows:

1. SEC constructs the AEAD Nonce as described in [Building the AEAD Nonce](#), using SSRC and Sequence Number from the input frame, and ROC and Salt Key from the protocol data block.
2. For AES-GCM cipher suites, the nonce is passed to AESA as IV
3. For AES-CCM cipher suites, the nonce is used to construct B_0 and Ctr_0 , which are then written to Class 1 context register
4. The received SRTP header is passed to AESA as AAD
5. The encrypted part of the received frame is passed to AESA as message data
6. MKI, if present, is stripped
7. ICV is passed to AESA as ICV type, to be compared with the computed ICV

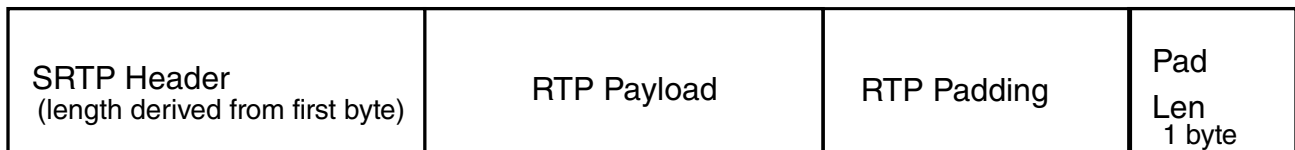


Figure 9-49. SRTP decapsulation output frame

SEC can perform anti-replay checking for SRTP decapsulation, using a window of 64 or 128 packets. A replayed or late PDU is rejected, and the Job Completion Status Word written to the Output Frame Queue is tagged as REPLAY or LATE as appropriate.

SEC does not update the PDB with the anti-replay status until after the ICV check has passed. If the ICV check fails, the PDU is rejected and the anti-replay status in the PDB is not updated. SEC manages a local copy of the sequence number found in the SRTP Header, as well as the ROC. The ROC needs to be incremented whenever the sequence number rolls over; sometimes ROC needs to be adjusted prior to use.

9.3.5.2 SRTP decapsulation PDB format descriptions

Table 9-55. SRTP decapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM

PDB word 0	(8 bits) x-len	(8 bits) length of MKI	(8 bits) for AES-CTR: n_tag otherwise: reserved (00h)	(8 bits) options [see table below]
------------	-------------------	---------------------------	--	--

Table continues on the next page...

Table 9-55. SRTP decapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM (continued)

PDB word 1	(16 bits) for AES-CTR: constant=0000h otherwise: constant	(16 bits) for AES-CTR: constant=0000h otherwise: reserved (0000h)
PDB word 2	(16 bits) reserved (0000h)	(16 bits) for AES-CTR: constant=0000h otherwise: reserved (0000h)
PDB word 3	salt 1	
PDB word 4	salt 2	
PDB word 5	salt 3	
PDB word 6	(16 bits) for AES-CTR: salt 4 for AES-GCM: reserved (0000h) for AES-CCM: B ₀ flags (8 bits)	(16 bits) for AES-CTR: constant=0000h for AES-GCM: reserved (0000h) for AES-CCM: Ctr ₀ constant
	Ctr ₀ flags (8 bits)	
PDB word 7 ¹	(16 bits) reserved	(16 bits) sequence number
PDB word 8	ROC	
PDB word 9	anti-replay scorecard 1 [present if ARS= 01b or 10b]	
PDB word 10	anti-replay scorecard 2 [present if ARS= 01b or 10b]	
PDB word 11	anti-replay scorecard 3 [present if ARS= 10b]	
PDB word 12	anti-replay scorecard 4 [present if ARS= 10b]	

1. Shaded rows are written back to PDB in memory, as needed.

Table 9-56. SRTP decapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
ARS		Reserved		MKI	Reserved		

Table 9-57. SRTP decapsulation PDB, description of the options byte

Field	Description
7-6 ARS	anti-replay checking scorecard 00b - Anti-replay checking disabled 01b - 64-bit anti-replay checking enabled 10b - 128-bit anti-replay checking enabled 11b - Reserved
5-4	Reserved
3 MKI	MKI included in Output Frame 0b - MKI not included in Output Frame. 1b - MKI copied from PDB into Output Frame

Table continues on the next page...

Table 9-57. SRTP decapsulation PDB, description of the options byte (continued)

Field	Description
2-0	Reserved. Must be zero.

9.3.5.3 SRTP decapsulation error conditions

This table lists the conditions under which SRTP decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure can produce an ICV check error.

Table 9-58. SRTP decapsulation error conditions

Condition	Error Status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
OPERATION Command PROT ID selects SRTP Decap, and PROTINFO is not a valid protocol	Protocol Command Error
n_tag in the PDB = 0, or n_tag > 20 for HMAC-SHA-1 cipher suites	Protocol Command Error
[ROC, SEQNUM] overflows	Protocol Sequence Number Overflow
Anti-Replay detects a LATE packet	Protocol LATE error
Anti-Replay detects a REPLAY packet	Protocol REPLAY error

9.4 IEEE 802.1AE MACsec encapsulation and decapsulation overview

SEC supports MACsec encapsulation and decapsulation as described in the IEEE 802.1AE-2006 specification and the IEEE 802.1AEbn-2011 and 802.1AEbw-2013 amendments, using AES-GCM for authentication and confidentiality.

SEC optionally supports the following:

- CRC generation and insertion of the resulting frame check sequence (FCS).
- Insertion of an optional AAD (up to 96 bytes for encapsulation and up to 112 bytes for decapsulation) that is not part of the current IEEE 802.1AE specification.

Both options are controlled by the descriptor's operation command (see [PROTOCOL OPERATION commands](#)).

Table 9-59. IEEE 802.1AE MACsec protocol descriptors

Encapsulation	Decapsulation
Header	Header
Protocol data block	Protocol data block
Class 1 key data block	Class 1 key data block
Protocol = MACsec encrypt	Class 2 key data block
	Protocol = MACsec decrypt

9.4.1 Process for 802.1AE MACsec encapsulation

This figure shows 802.1AE MACsec encapsulation procedure.

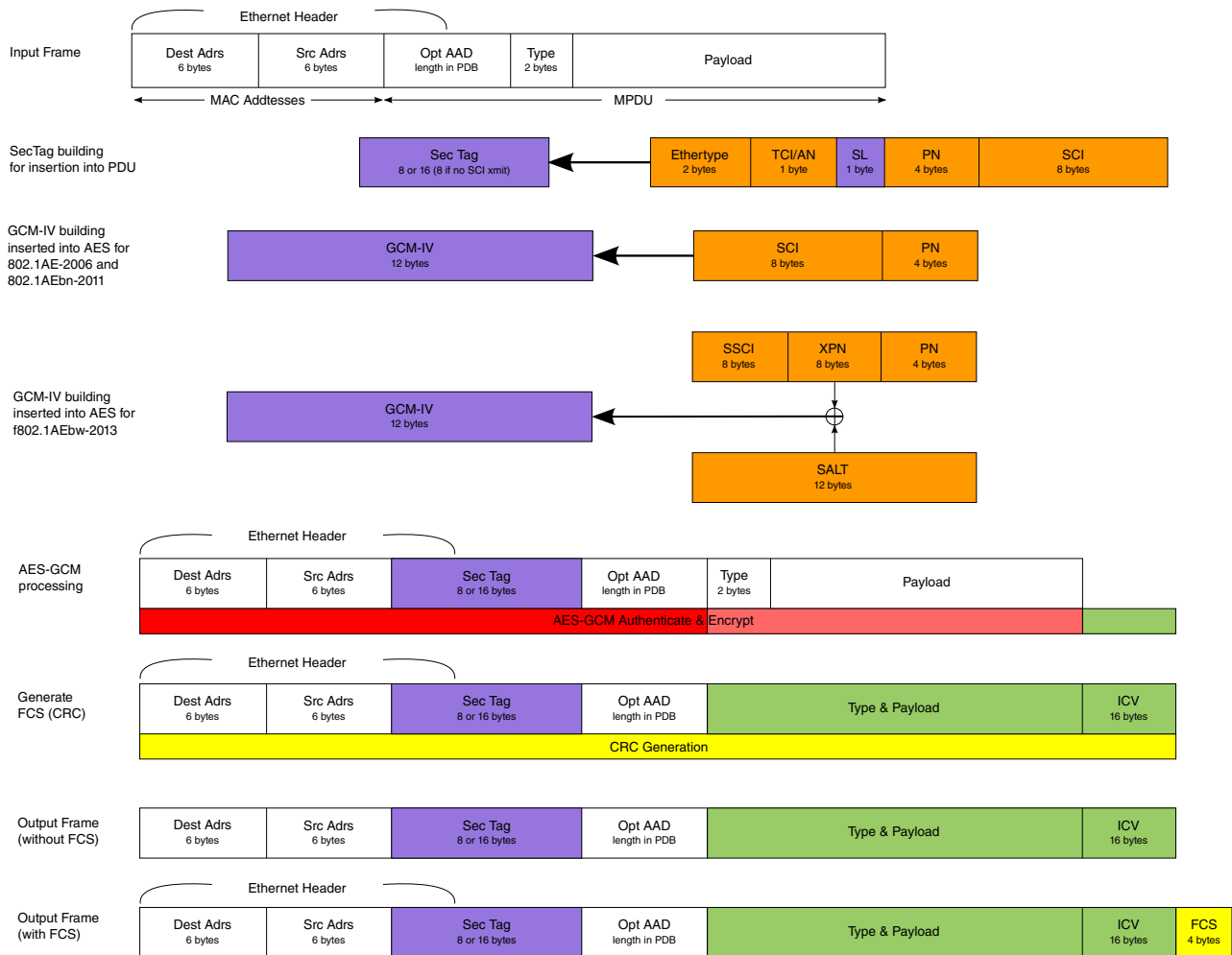


Figure 9-50. 802.1AE MACsec encapsulation

This figure shows the encapsulation input frame.

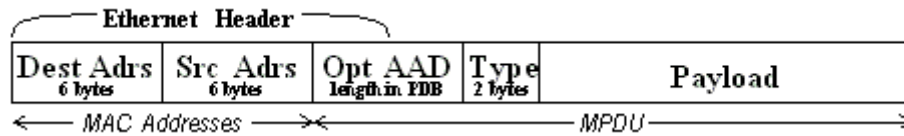


Figure 9-51. 802.1AE MACsec encapsulation input frame

The standard MACsec encapsulation procedure is:

1. SEC receives an Ethernet header from the input frame; the header consists of:
 - A destination address (Dest Adrs)
 - A source address (Src Adrs)
 - Optional AAD (of up to 96 bytes)

NOTE

In [Figure 9-51](#), a second 2-byte ethertype field is assumed to part of the optional AAD because during the encapsulation process a new ethertype field (assumed part of the SecTag) is inserted into the Ethernet header.

- An ethertype field that refers to the type of the frame (Type)
2. SEC receives the payload from the input frame.
 3. The destination address and source address portions of the Ethernet header are copied to both AESA and CRCA, tagged for authentication (AAD) and CRC (if enabled), and also copied to the output frame.
 4. SEC builds the 8 or 16-byte SecTag as shown in the following figure:

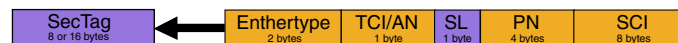


Figure 9-52. Building the MACsec SecTag

- a. SCI transmission is determined by bit 1 of the TCI/AN byte.
 - If set, the 16-byte SecTag includes the SCI.
 - If not set, the 8-byte SecTag omits the SCI.
 - b. The 4-byte PN, or packet number, is incremented and written back to one word of the PDB.
 - c. SEC computes the 1-byte SL as the length of payload if the payload is less than 48 bytes in length and as zero otherwise.
 - d. EtherType and TCI/AN occupy one word of the PDB.
5. (optional) Rollover results in a ROLLOVER status indication in the Job Completion Status Word.
 6. The SecTag is pushed onto output frame and into both CHAs, where it is tagged for authentication and CRC.

- SEC constructs the GCM-IV from the SCI and PN for AES-GCM, or from the SSCI, XPN, PN, and SALT for AES-GCM-XPN; note that SCI is used to construct the GCM-IV even if bit 1 of the TCI/AN byte indicates that SCI is not transmitted.

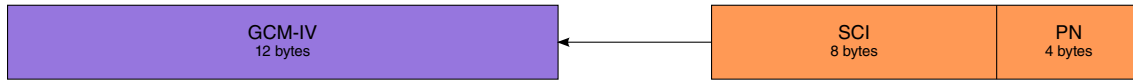


Figure 9-53. Building the GCM initialization vector (GCM IV) for AES-GCM cipher suites

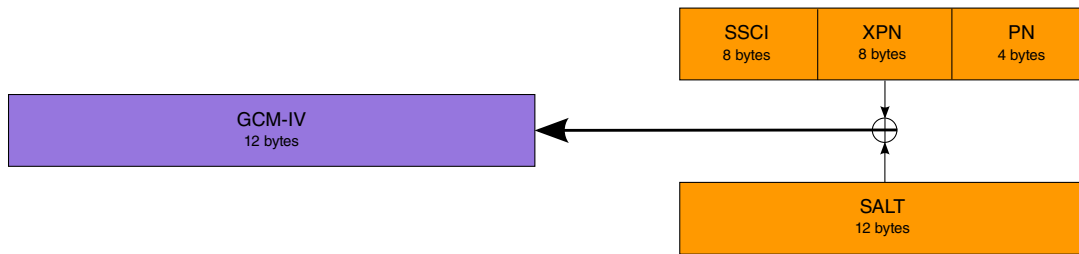


Figure 9-54. Building the GCM initialization vector (GCM IV) for AES-GCM-XPN cipher suites

- SEC pads the GCM IV with trailing zeros and pushes it to AESA (tagged as IV) prior to AAD, ethertype, and payload.
- SEC treats the two byte ethertype field as if it were part of payload and pushes the payload into AESA for encryption and authentication, and into CRCA.
- The resulting encrypted payload (including the encrypted type) is pushed onto the output frame.
- AESA computes the GCM ICV automatically, and that encrypted ICV is also pushed onto the output frame.

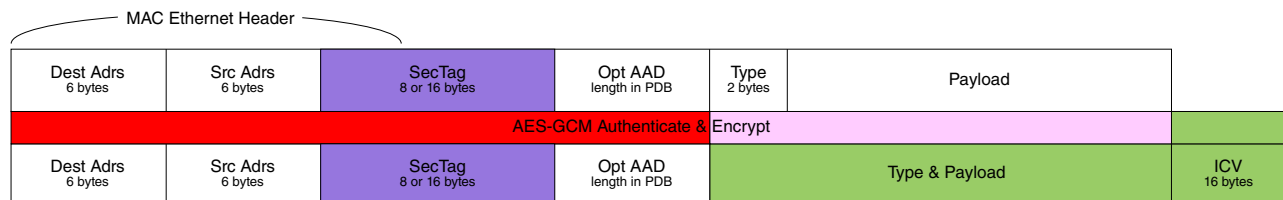


Figure 9-55. 802.1AE MACsec packet encapsulation using AES-GCM

9.4.1.1 Using the frame check sequence (FCS)

SEC's in-built CRCA CHA can be enabled to compute the frame check sequence (FCS). IEEE 802.3 (Ethernet) specifies a minimum frame size of 64 bytes, and the last four bytes of the frame are the FCS.

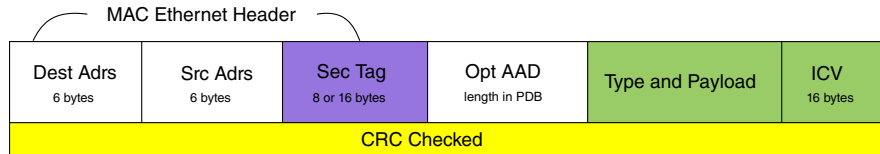


Figure 9-56. Optional 802.1AE MACsec packet encapsulation using CRC

If the FCS is enabled, the CRCA receives all data authenticated. If the output frame is less than 60 bytes, SEC pads the output frame with zeros to bring the frame size to 60 bytes. These additional bytes of zero padding are input to the CRC calculation.

Additionally, the ICV computed by AESA is recirculated back as a final input to Class 2 CHA CRCA. The FCS produced is then a CRC of the entire authenticated and encrypted packet, as shown in [Figure 9-57](#). [Figure 9-58](#) shows the result of appending the FCS to the end of the encapsulated packet.

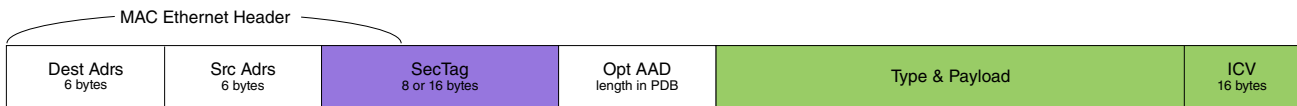


Figure 9-57. 802.1AE MACsec encapsulation output packet option 1 (no FCS)

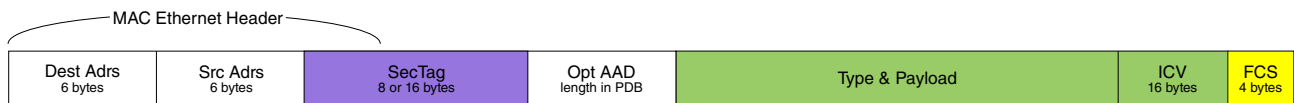


Figure 9-58. 802.1AE MACsec encapsulation output packet option 2 (with FCS)

NOTE

At this point, the Ethernet header consists of the destination address, the source address, and the ethertype that is part of the SecTag.

9.4.1.2 Additional notes for GMAC support

[Process for 802.1AE MACsec encapsulation](#) describes MACsec with GCM. SEC also supports GMAC. MACsec for GMAC is processed like GCM except that the input frame is not encrypted. The ICV is computed in the same fashion.

SEC determines whether to use GCM or GMAC based on the E bit in the TCI field of the SecTag programmed into the Shared Descriptor PDB. The TCI field is found in the next-to-the-rightmost byte of word 4 of the PDB. IEEE standard 802.1AE numbers the TCI octet with msb 8 and lsb 1, and identifies E as bit 4. Therefore, the E bit is programmed into ms bit of the third-from-the-rightmost nibble of word 4 of the PDB.

9.4.2 MACsec encapsulation PDB format descriptions

Table 9-60. MACsec Encapsulation PDB for AES-GCM cipher suites

PDB word 0	AAD Length (16 bits)	reserved (8 bits)	options (8 bits) [see table below]
PDB word 1	SCI 1		
PDB word 2	SCI 2		
PDB word 3	Ethertype (16 bits)	TCI/AN (8 bits)	reserved (8 bits)
PDB word 4	PN		

Table 9-61. MACsec Encapsulation PDB for AES-GCM-XPB cipher suites

PDB word 0	AAD Length (16 bits)	reserved (8 bits)	options (8 bits) [see table below]
PDB word 1	SCI 1		
PDB word 2	SCI 2		
PDB word 3	Ethertype (16 bits)	TCI/AN (8 bits)	reserved (8 bits)
PDB word 4	salt 1		
PDB word 5	salt 2		
PDB word 6	salt 3		
PDB word 7	EPN		
PDB word 8	PN		
PDB word 9	SSCI		

Table 9-62. MACsec encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	SAinSCI	Reserved	Reserved	Reserved	Reserved	FCS

Table 9-63. MACsec Encapsulation PDB - Description of the Options Byte

Field	Description
7-6	Reserved
5	0: SCI used as-is from PDB for encapsulation 1: SCI for encapsulation constructed from SA from input frame, with portID from lower 16 bits of SCI2 in PDB
SAinSCI	
4-1	Reserved
0	FCS included in Output Frame
FCS	0 FCS not included in Output Frame. 1 FCS computed by CRCA and copied into Output Frame

9.4.3 Process for 802.1AE MACSec decapsulation

This figure shows the stages for the 802.1AE MACsec decapsulation process.

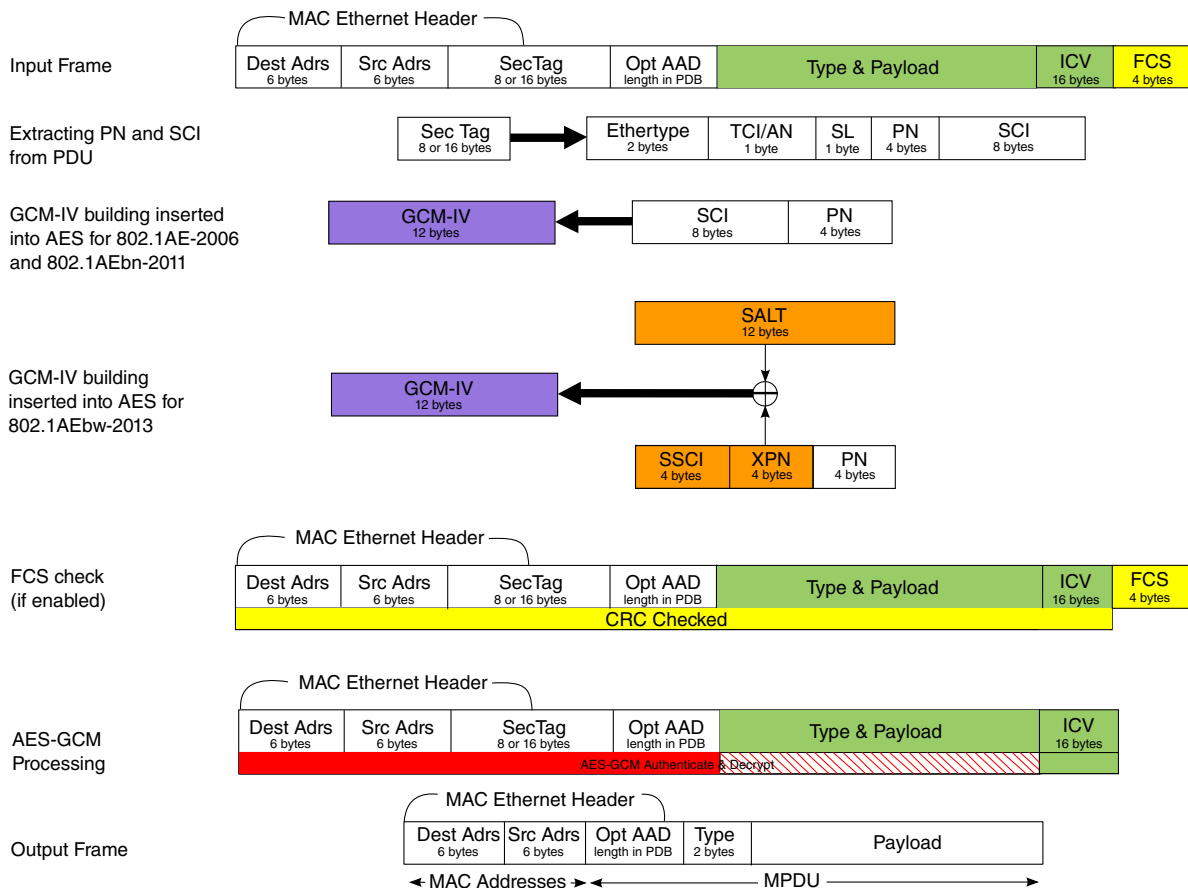


Figure 9-59. 802.1AE MACsec decapsulation process

SEC decapsulates MACsec encapsulated packets as specified by the IEEE 802.1AE specification, with the addition of support for an optional AAD value. This optional AAD is specified to sit in the frame between the SecTag and the encrypted type & payload field (see the following figure).

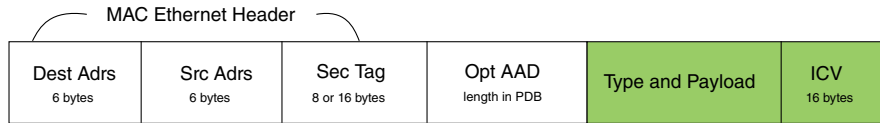


Figure 9-60. 802.1AE MACsec decapsulation input frame

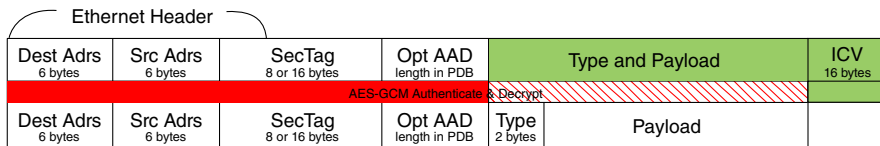


Figure 9-61. 802.1AE MACsec decapsulation

The procedure for MACsec decapsulation is:

1. SEC pulls the MAC Ethernet header and the SecTag from the input frame and pushes them into the AESA tagged for authentication.
2. The destination address and source address fields of the MAC Ethernet header are also copied to the output frame.
3. (optional) If present, the optional AAD is tagged for authentication by AESA.
4. For AES-GCM cipher suites, SCI and PN are pulled out of the SecTag to form the GCM-IV, as shown in the following figures.

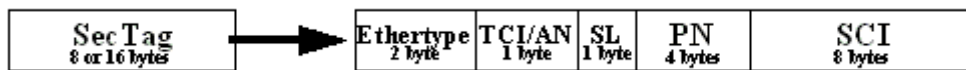


Figure 9-62. 802.1AE MACsec SecTag elements

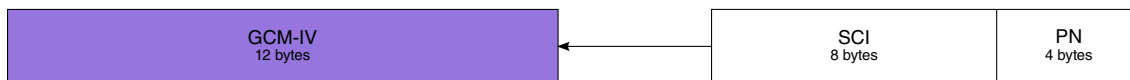


Figure 9-63. 802.1AE MACsec GCM IV building

NOTE

If SCI is not transmitted (this is indicated by bit 1 not set in TCI/AN byte of SecTag), the GCM-IV must be built using a SCI extracted from the PDB.

5. For the GCM-XPB cipher suites, PN is pulled out of the received frame's SecTag and combined with SSCI, XPB and SALT as programmed into the PDB to form the GCM-IV, as shown in the following figures.

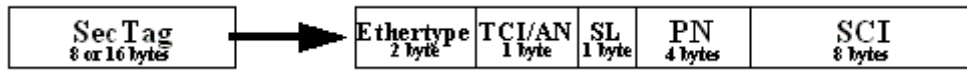


Figure 9-64. 802.1AE MACsec SecTag elements

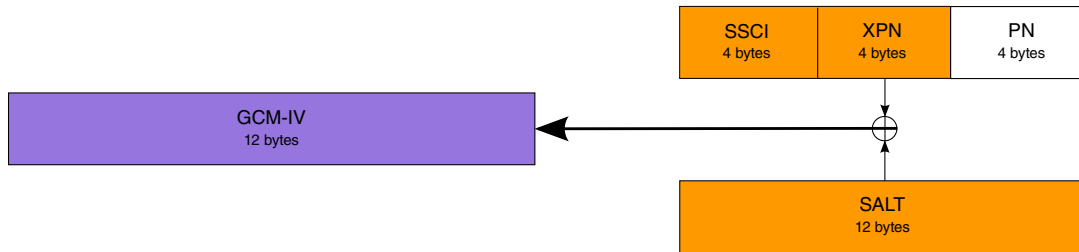


Figure 9-65. 802.1AE MACsec GCM-XPB IV building

6. The GCM-IV is padded with trailing zeros to a 16-byte boundary and pushed to AESA.
7. (optional) If present, the optional AAD is also pushed to the output frame.
8. The encrypted type and payload are pushed into AESA and tagged for authentication and decryption.
9. The decrypted result is pushed onto the output frame.
10. SEC compares the received ICV to the computed ICV; if it detects differences, it indicates ICV CHECK FAIL in the Job Completion Status Word.

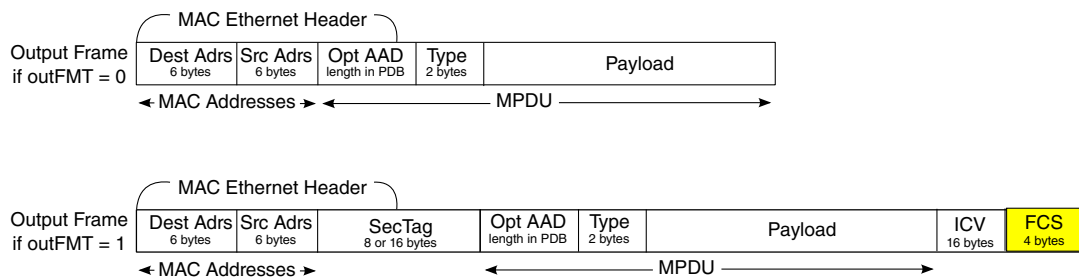


Figure 9-66. 802.1AE MACsec Decapsulation Output Frame

9.4.3.1 Automatically switching between two keys

SEC MACsec decapsulation has the ability to automatically switch between two keys. If AKS (Automatic Key Switching) is set in the PDB, SEC uses the least significant bit of the AN field of the SecTag received in the input frame to select which key is used:

- If the bit is set, the Class 2 Key is used.
- If the bit is cleared, the Class 1 Key is used.

This feature allows reception of frames without software needing to preprocess the frames to determine which key is being used.

9.4.3.2 Additional notes for GMAC support (decapsulation)

[Process for 802.1AE MACSec decapsulation](#) describes MACsec with GCM. SEC also supports GMAC. MACsec for GMAC is processed like GCM, except that the input frame is not encrypted. The same ICV is computed in the same fashion.

During MACsec decapsulation, SEC determines whether to use GCM or GMAC based on the E bit in the TCI field of the SecTag received in the input frame. The TCI field is found in bits 16-23 of word 4 of the PDB. IEEE standard 802.1AE numbers the TCI octet with msb 8 and lsb 1 and identifies E as bit 4.

9.4.4 MACsec decapsulation PDB format descriptions

Table 9-64. MACsec decapsulation PDB

	Descriptor Header (1 or 2 words)			
PDB word 0	AAD Length (16 bits)	reserved (8 bits)	Options (8 bits) [see table below]	
PDB word 1	SCI 1			
PDB word 2	SCI 2			
PDB word 3	reserved		ARLen (8 bits)	
PDB word 4	PN			DECO updates PDB in descriptor buffer and external memory as needed
PDB word 5	Anti Replay Scorecard 1 [present if AR=1]			
PDB word 6	anti-replay scorecard 2 [present if AR=1 and ARLen>32]			
PDB word 7	anti-replay scorecard 3 [present if AR=1 and ARLen>64]			
PDB word 8	anti-replay scorecard 4 [present if AR=1 and ARLen>96]			

Table 9-65. MACsec decapsulation PDB for GCM-XPN cipher suites

	Descriptor Header (1 or 2 words)			
PDB word 0	AAD Length (16 bits)	ARLen (8 bits)	Options (8 bits) [see table below]	

Table continues on the next page...

Table 9-65. MACsec decapsulation PDB for GCM-XPB cipher suites (continued)

PDB word 1	SCI 1	
PDB word 2	SCI 2	
PDB word 3	SSCI	
PDB word 4	salt 1	
PDB word 5	salt 2	
PDB word 6	salt 3	
PDB word 7	EPN	DECO updates PDB in descriptor buffer and external memory as needed
PDB word 8	PN	
PDB word 9	anti-replay scorecard 1 [present if AR=1]	
PDB word 10	anti-replay scorecard 2 [present if AR=1 and ARlen>32]	
PDB word 11	anti-replay scorecard 3 [present if AR=1 and ARlen>64]	
PDB word 12	anti-replay scorecard 4 [present if AR=1 and ARlen>96]	

Table 9-66. MACsec decapsulation PDB, format of the options byte

	7	6	5	4	3	2	1	0
	Reserved	AR	SAinSCI	outFMT	Reserved	Reserved	AKS	FCS

Table 9-67. MACsec decapsulation PDB, description of the options byte

Field	Description
7	Reserved
6 AR	Anti-replay enable 0: Anti-replay checking not enabled. 1: Anti-replay checking enabled
5 SAinSCI	0: use SCI as is from PDB for decapsulation 1: construct SCI using SA from input frame plus lower 16bits of SCI2 from PDB (for portID)
4	Reserved
3 outFMT	0: SECTAG, ICV and FCS stripped from output frame 1: SECTAG, ICV, and FCS left in output frame
2	Reserved
1 AKS	Automatic Key Switching performed 0: Automatic Key Switching not performed -- Class 1 Key is always used 1: Automatic Key Switching performed -- input frame SecTag AN field LSB selects key.
0	FCS included in Output Frame 0: FCS not included in Output Frame. 1: FCS computed by encapsulator and included into Input Frame

9.5 IEEE 802.11 -2012 WPA2 MPDU encapsulation and decapsulation

IEEE 802.11 "WiFi" is a popular standard that provides wireless LAN services. Security has evolved within the WiFi standard; Wired-Equivalent Privacy, or WEP, was the original security service. WEP was replaced by Temporal Key Integrity Protocol (TKIP) in 2002, and by WiFi Protected Access (WPA) in 2003. The 80211i-2004 amendment settled upon WPA2, utilizing CCMP.

SEC supports WPA2 CCMP encapsulation and decapsulation of WiFi MPDUs in support of the IEEE 802.11-2012 standard.

Table 9-68. IEEE 802.11i protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block		Protocol data block
Class 1 key data block		Class 1 key data block
Protocol = WiFi encrypt		Class 2 key data block
		Protocol = WiFi decrypt

9.5.1 Processing Common to WPA2 Encapsulation and Decapsulation

In order for SEC to support WPA2 CCMP encapsulation and decapsulation of WiFi MPDUs, several preprocessing steps have to occur to prepare AESA to receive the MPDU:

- Additional Authentication Data (AAD) requires preparation
- Nonce requires preparation
- AESA Context must be prepared

9.5.1.1 Constructing the AAD for WPA2 encapsulation and decapsulation

The MAC Header is not used as-is as Additional Authentication Data (AAD). Bits in several fields require masking, and the Dur/ID field is not protected at all. In addition, for CCM, AESA requires AAD to be formatted; the 2-byte AAD Length field must be prepended before it is presented to AESA.

This figure shows the construction of formatted Additional Authenticated Data (AAD).

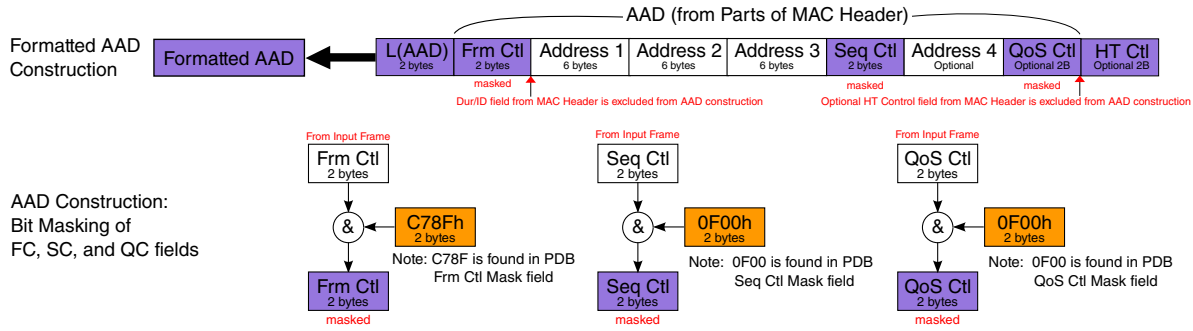


Figure 9-67. 802.11 CCMP construction of formatted AAD

SEC builds the AAD as shown in the figure above, using fields in the MAC Header. The first 2 bytes of the formatted AAD contain the length of the AAD proper.

Note that SEC bit masks the Frame Control and Sequence Control, and QoS Control fields of the MAC header per the IEEE 802.11 spec (as illustrated above), using masks stored in the descriptor's Protocol Data Block.

The AAD is authentication-only data provided to the AES engine. That is, the parts of the MAC header protected as AAD are authenticated, but are not encrypted. AAD is constructed for AESA only; the unmodified MAC header is copied from the input frame to the output frame.

9.5.1.2 Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation

SEC constructs the AES-CCM nonce using:

- A constant priority (Pri) field extracted from the PDB
- The Address2 field found in the MAC header from the input frame
- A packet number (PN) extracted from the PDB (and for decapsulation, matched against the PN found in the input frame).



Figure 9-68. CCMP nonce construction

The PN is incremented following use, and the incremented value is written back to the protocol data block.

9.5.1.3 Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation

For AES-CCM, AESA requires B_0 and CTR_0 to be constructed and written to the Class 1 context register before it can start processing the input frame, per the example formatting function described in Appendix A of NIST Special Publication 800-38C. The 13-byte nonce is the bulk of both 16-byte values, and is prepared first.

To construct B_0 , SEC first computes a 2-byte representation of the length of the payload. The B_0 ; flags byte is extracted from the PDB, and the three are concatenated together as shown below.

To construct CTR_0 , SEC extracts the CTR flags byte and the Counter Init Count field from the PDB, and concatenates the three as shown below.

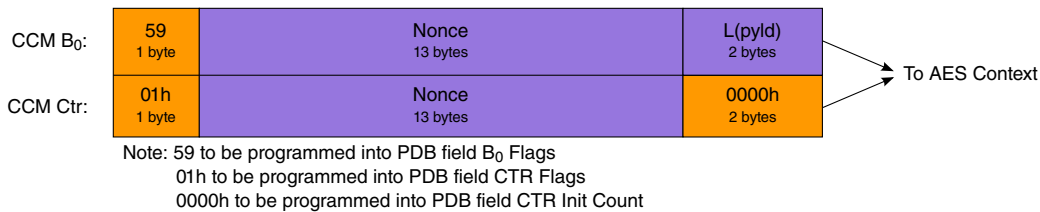


Figure 9-69. WPA2 CCMP context construction

9.5.2 Process for WPA2 encapsulation

This figure shows the WPA2 encapsulation process.

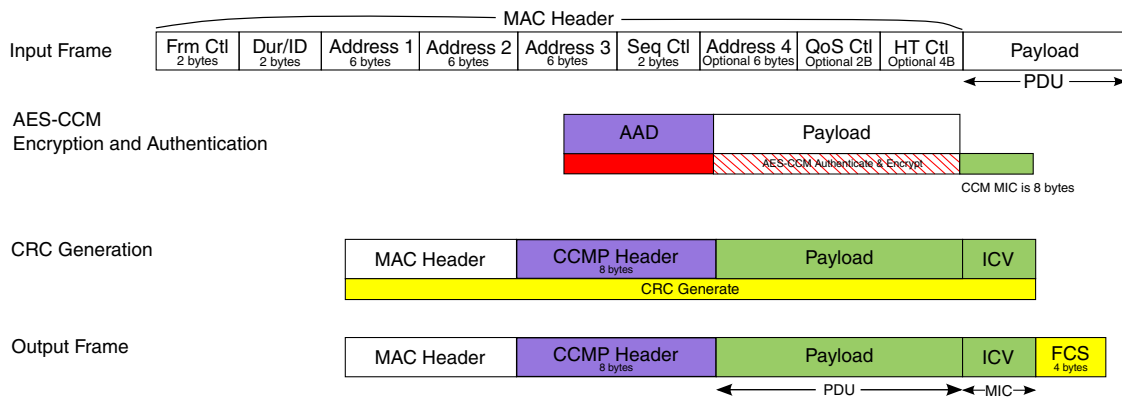


Figure 9-70. WPA2 encapsulation process

For WPA2 encapsulation, SEC requires that the input frame consist of an 802.11 MPDU, including the MAC header and the payload requiring cryptographic protection. The MAC Header Length field in the PDB can be used to determine the contents of the MAC header. If HTE is set in the PDB Options field, then the HT Control field is expected. Alternatively, by setting the DFC bit in the PDB Options field, the MPDU header will be parsed to determine the contents of the header.

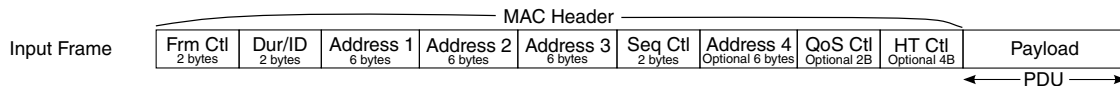


Figure 9-71. WPA2 encapsulation input frame (802.11 MPDU)

SEC performs the encapsulation procedure by doing the following:

1. Receives the input frame.
2. Uses fields from the MAC Header to construct the AAD (see [Constructing the AAD for WPA2 encapsulation and decapsulation](#) for more details).
3. Uses fields from the PDB and MAC header to construct the nonce (see [Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation](#) for more details).
4. Uses the nonce to construct both the CCM Initial Counter value and CCM B₀, both of which are written to the Class 1 CHA Context Register (see [Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation](#) for more details).
5. Constructs a CCMP header and pushes it onto the output frame immediately following the MAC header; note that this header is not cryptographically protected (see [Constructing the CCMP header for WPA2 encapsulation](#) for more details).
6. Extracts the payload from the input frame and pushes it into AESA. (see [WPA2 Payload Encapsulation](#) for more details)
7. AESA automatically produces the appropriate MAC and encrypts it to produce the ICV.
8. Pushes the encrypted payload and ICV onto the output frame.



Figure 9-72. WPA2 CCMP-encapsulated output frame

All data pushed onto the output frame (up through the ICV) may also be pushed into CRCA for CRC computation. If you have CRC computation enabled, see [Computing the FCS for WPA2 encapsulation](#).

9.5.2.1 Constructing the CCMP header for WPA2 encapsulation

SEC constructs a CCMP header, using the original packet number (PN1, PN2) and Key ID found in the protocol data block, plus a constant 00h byte. This construction is pushed onto the output frame immediately following the MAC header, but is not cryptographically protected.

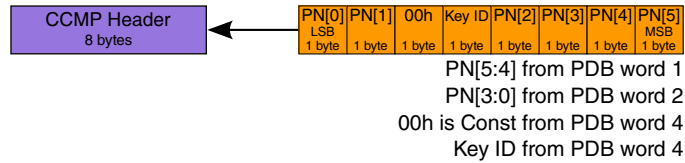


Figure 9-73. WPA2 CCMP header construction

9.5.2.2 WPA2 Payload Encapsulation

The payload is extracted from the input frame and pushed into AESA. As part of the WPA2 authentication and encryption computation, AESA automatically produces the appropriate CBC-MAC and encrypts it to produce the ICV. The encrypted payload and ICV are pushed onto the output frame.

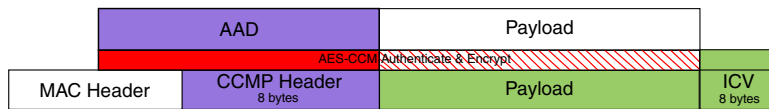


Figure 9-74. WPA2 cryptographic encapsulation

9.5.2.3 Computing the FCS for WPA2 encapsulation

As produced, all data pushed onto the output frame (up through the ICV) may also be pushed into CRCA for CRC computation. If CRC computation is enabled, SEC appends the resulting frame check sequence (FCS) to the output frame.

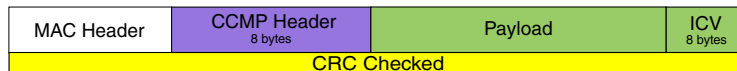


Figure 9-75. WPA2 checksum encapsulation

9.5.2.4 WPA2 encapsulation PDB format descriptions

Table 9-69. 802.11 WPA2 encapsulation PDB

Descriptor Header (1 or 2 words)				
PDB Word 0	MAC Header Length (16 bits)		rsv (8 bits)	Options (8 bits)
PDB Word 1	B ₀ Flags (8 bits)	Pri (8 bits)	PN 1 (16 bits)	
PDB Word 2	PN 2			
PDB Word 3	Frm Ctl Mask (16 bits)		Seq Ctl Mask (16 bits)	
PDB Word 4	QoS Ctl Mask (16 bits)		Const (8 bits)	KeyID (8 bits)
PDB Word 5	CTR Flags (8 bits)	rsv (8 bits)	CTR Init Count (16 bits)	

DECO writes PN back to PDB as needed

Table 9-70. WPA2 encapsulation PDB, format of the options byte

	7	6	5	4	3	2	1	0
	Reserved	Reserved	DFC	Reserved	Reserved	HTE	Reserved	FCS

Table 9-71. WPA2 encapsulation PDB, description of the options byte

Field	Description
7-6	Reserved
5 DFC	Decode Frame Control 0: do not decode Frame Control field; determine format of MAC Header from Mac Header Length and HTE bit. 1: use Frame Control field to determine format of MAC Header (HTE must be programmed to 0; PDB MAC Header Length field is ignored).
4-3	Reserved
2 HTE	HT Enable 0: When DFC=0, HT Control field is not present in MAC Header 1: When DFC=0, HT Control field may be present in MAC Header. SEC will use Frame Control field from input frame to determine if MAC Header includes an HT Control field. If an HT Control field is determined to be present, it is skipped from the authentication processing that most of the MAC header undergoes. <i>Note: HTE must be 0 if DFC=1</i>
1	Reserved
0 FCS	FCS included in Output Frame 0 FCS not included in Output Frame. 1 FCS computed by CRCA and copied into Output Frame

9.5.2.5 WPA2 encapsulation error conditions

This table lists the conditions under which WPA2 encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

Table 9-72. WPA2 encapsulation error conditions

Condition	Error Status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
DFC=1 and HTE=1 in the PDB Options byte	Protocol PDB error
OPERATION Command PROT ID selects IEEE 802.11 WiFi WPA2 Encap, and PROTINFO is not a valid protocol	Protocol Command Error
PN overflows	Protocol Sequence Number Overflow

9.5.3 Process for WPA2 decapsulation

SEC performs WPA2 decapsulation by doing the following:

1. Receives an input frame containing the original MAC header, a CCMP header, an encrypted payload and ICV, and an optional frame check sequence (FCS) (see the following figure).



Figure 9-76. WPA2 decapsulation input frame

2. Constructs the Nonce as described for encapsulation, using the MAC header from the input frame, and various resources programmed in the PDB.
3. Constructs the AAD as described for encapsulation, by masking and removing fields from the MAC header.
4. For AES-CCM mode, constructs the formatted AAD from the constructed AAD, and constructs the AESA context, using the Nonce, and various resources programmed in the PDB.
5. Copies the MAC header from the input frame to the output frame.
 - If PDB Option DFC=0, the MAC Header Length field in the PDB determines the number of bytes to copy.
 - If DFC=1, the Frame Control field of the MAC header is parsed to determine the length and contents of the MAC Header.

- Optional: If enabled, writes the entire input frame into CRCA for CRC computation (see the following figure).

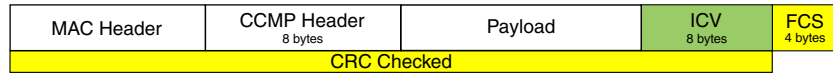


Figure 9-77. WPA2 decapsulation checksum verification

- If the computed CRC does not validate the frame check sequence (FCS) found at the end of the input frame, a CRC fail is indicated in the Job Completion Status Word.
- Optional: If anti-replay checking is enabled, each received PN is compared to the packet number maintained in the PDB, and if the values are different, REPLAY is indicated in the Job Completion Status Word (see [WPA2 Decapsulation Anti-replay checking](#)).
- The AAD is pushed, tagged for authentication only, to AESA.
- Following AAD, the encrypted payload and ICV are pushed into AESA and CRCA; the payload is tagged for authentication and decryption.
- AESA automatically computes an ICV and compares it to the received ICV. If the two ICVs do not match, an ICV CHECK FAIL is indicated in the Job Completion Status Word.

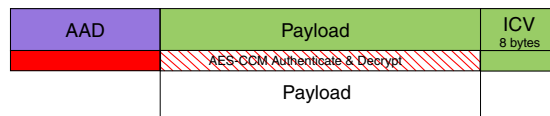


Figure 9-78. WPA2 cryptographic decapsulation

- The decrypted payload is pushed onto the output frame immediately following the MAC header.

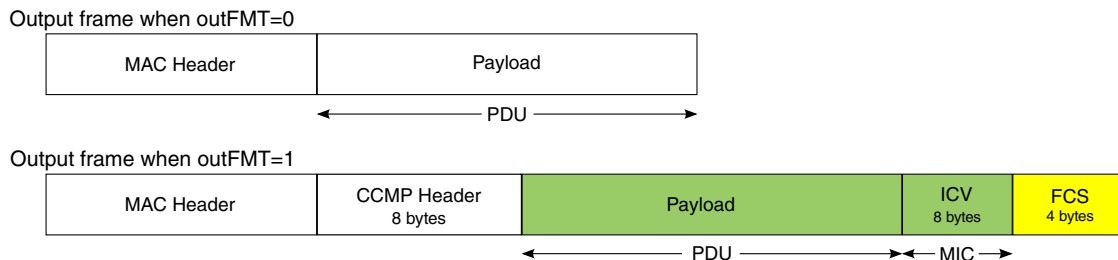


Figure 9-79. WPA2 decapsulation Output Frame (802.11 MPDU)

9.5.3.1 WPA2 Decapsulation Anti-replay checking

If enabled by the options byte AR field, each received PN is compared to the packet number maintained in the PDB. If the value of PN received is not equal to that previously stored, REPLAY is indicated in the Job Completion Status Word. If Anti-Replay is disabled, a packet received with an out-of-sequence PN produces an ICV error and does not decapsulate correctly.

The use of the AR option allows the user to know whether an ICV error was caused by an out-of-sequence PN.

9.5.3.2 Using automatic key-switching

SEC WPA2 decapsulation has the ability to automatically switch between two keys. If Automatic key-switching (AKS) is set in the PDB, SEC uses the least significant bit of the KeyID field of the CCMP Header received in the input frame to select which key is used.

- If the bit is set, the Class 2 Key is used.
- If the bit is cleared, the Class 1 Key is used.

This feature allows reception of frames without software having to preprocess to determine which key is being used.

9.5.3.3 WPA2 decapsulation PDB format descriptions

Table 9-73. 802.11 WPA2 decapsulation PDB

	Descriptor Header (1 or 2 words)			
PDB Word 0	MAC Header Length (16 bits)		rsv (8 bits)	Options (8 bits)
PDB Word 1	B ₀ Flags (8 bits)	Pri (8 bits)	PN 1 (16 bits)	
PDB Word 2	PN 2			DECO writes PN back to PDB as needed
PDB Word 3	Frm Ctl Mask (16 bits)		Seq Ctl Mask (16 bits)	
PDB Word 4	QoS Ctl Mask (16 bits)		rsv (16 bits)	
PDB Word 5	CTR Flags (8 bits)	rsv (8 bits)	CTR Init Count (16 bits)	

Table 9-74. WPA2 decapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	AR	DFC	Reserved	outFMT	HTE	AKS	FCS

Table 9-75. WPA2 decapsulation PDB, description of the options byte

Field	Description
7	Reserved
6 AR	Anti-replay enable 0 Anti-replay checking not enabled. 1 Anti-replay checking enabled
5 DFC	Decode Frame Control field 0: do not decode Frame Control field; determine format of MAC Header from Mac Header Length and HTE bit. 1: use Frame Control field to determine format of MAC Header (HTE must be programmed to 0; PDB MAC Header Length field is ignored).
4	Reserved. Must be zero.
3 outFMT	Output Frame Format 0: CCMP / GCMP Header, ICV, and FCS stripped from output frame 1: CCMP Header, ICV, and FCS included in output frame
2 HTE	HT Field Enable 0: When DFC=0, HT Control field is not present in MAC Header 1: When DFC=0, HT Control field may be present in MAC Header. SEC will use Frame Control field from input frame to determine if MAC Header includes an HT Control field. If an HT Control field is determined to be present, it is skipped from the authentication processing that most of the MAC header undergoes. <i>Note: HTE must be 0 if DFC=1</i>
1 AKS	Automatic Key Switching 0 Automatic Key Switching not performed -- Class 1 Key is always used 1 Automatic Key Switching performed -- input frame CCMP Header KEYID field selects key.
0 FCS	FCS included in input Frame 0 FCS not included in inputFrame. 1 FCS computed by encapsulator and included into Input Frame

9.5.3.4 WPA2 decapsulation error conditions

This table lists the conditions under which WPA2 Decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine; authentication failure can produce an ICV check error.

Table 9-76. WPA2 decapsulation error conditions

Condition	Error status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
DFC=1 and HTE=1	Protocol PDB error
OPERATION Command PROT ID selects WPA2 Decap, and PROTINFO is not a valid protocol	Protocol Command Error
PN overflows	Protocol Sequence Number Overflow
AR = 1, and PN is not received Sequentially (PN PDB doesn't match received PN)	Protocol REPLAY Error

9.6 IEEE 802.16 WiMAX encapsulation and decapsulation overview

The IEEE 802.16 standard describes how the generic MAC header (or GMH) is to be modified during the encapsulation process. SEC assumes the input frame includes the GMH appropriate for transmission with the encapsulated payload. If applying AES-CCM, SEC uses fields from the GMH to build the initial counter CTR_0 and B_0 values required to be written into Class 1 Context for AESA to perform AES-CCM processing, but does not perform any of the modifications specified in IEEE 802.16. Software is required to include the GMH with the value of the EC bit changed, with the length field updated to include the ICV and FCS additions, and with the HCS updated.

The IEEE 802.16 standard specifies the use of AES-CCM for confidentiality and data integrity protection, without any Additional Authentication Data (AAD). The standard does not require confidentiality or data integrity.

The PROTINFO field of the Protocol Operation command specifies whether or not AES-CCM is to be applied to the flow.

- If AES-CCM is applied to encapsulation, the payload is encrypted and an ICV is computed and inserted into the output frame.
- If AES-CCM is not applied to encapsulation, then no encryption, ICV computation, or ICV insertion is performed.
- If AES-CCM is not applied to decapsulation, then decryption is not performed, and no ICV check is performed.

Table 9-77. IEEE 802.16 protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block		Protocol data block

Table continues on the next page...

Table 9-77. IEEE 802.16 protocol descriptors (continued)

Encapsulation		Decapsulation
Class 1 key data block		Class 1 key data block
Protocol = WiMAX encrypt		Class 2 key data block
		Protocol = WiMAX decrypt

9.6.1 Process for IEEE 802.16 WiMAX encapsulation

SEC performs IEEE 802.16 WiMAX encapsulation by doing the following:

1. Receives an input frame including the encapsulation-appropriate GMH and payload



Figure 9-80. 802.16 WiMAX encapsulation input frame

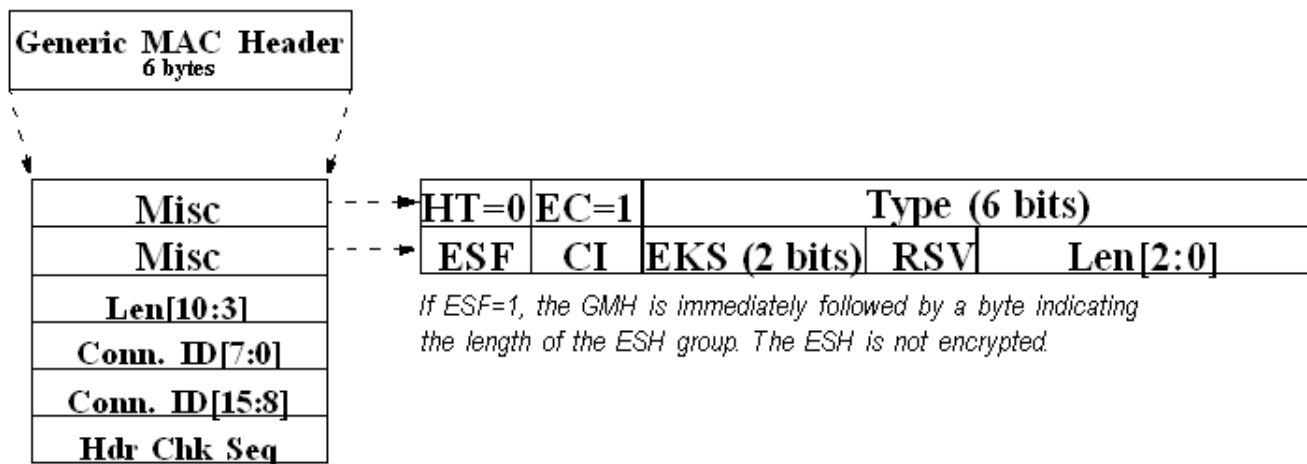


Figure 9-81. 802.16 WiMAX generic MAC header detail

2. Takes this input frame and performs the following actions:
 - Optional: If selected, encrypts and authenticates the payload
 - Inserts a packet number
 - Optional: If enabled, inserts a frame check sequence (FCS) generated using a CRC32 algorithm.
3. Optional: If the ESF bit in the GMH is set, indicating that extended subheaders (ESH) are present in the input frame, the ESH is passed to the output frame and FCS protected (if CRC checking is enabled), but is not passed to the AESA for authentication or confidentiality.

- The first byte of the ESH is the length, in bytes, of the ESH.
 - There must be at least one more byte of ESH after the ESH length or an error is returned.
4. Uses the first 5 bytes of the GMH, along with a PDB-maintained 4-byte constant and 4-byte packet number (PN), and builds a nonce.
 - The packet number is incremented and written back to the PDB after use.
 - (optional) If the most significant bit of PN changes upon incrementing, the Job Completion Status Word contains the ROLLOVER indicator.
 5. Writes the GMH, Optional ESH, and original PN to the output frame and passes them to CRCA

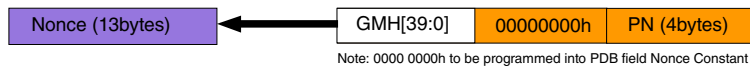


Figure 9-82. 802.16 WiMAX nonce construction

6. Uses the 13-byte nonce to build the initial counter value CTR_0 and the B_0 , which are provided to the Class 1 Context Register for AES to use in performing CCM computations.
 - The constants used in Nonce construction and in CTR_0 and B_0 construction are extracted from the PDB, future proofing against the possibility of a simple change to the WiMAX spec.

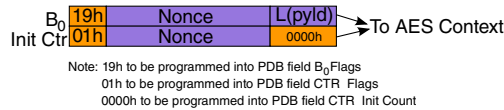


Figure 9-83. 802.16 WiMAX AES-CCM context construction

NOTE

If the Protocol Operation command PROTINFO field indicates AES-CCM is not used, construction of CTR_0 and B_0 is skipped.

7. After loading the key, CTR_0 , and B_0 into the AES engine, uses AES-CCM mode to encrypt the payload and compute an ICV
8. Writes the encrypted payload and ICV the output frame and passes them to CRCA.

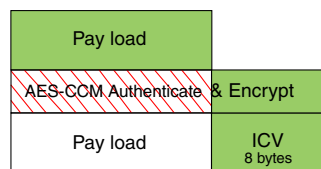


Figure 9-84. 802.16 WiMAX cryptographic encapsulation

9. CRC is computed across the entire output frame, and the FCS is appended to the output frame (see the following figure).

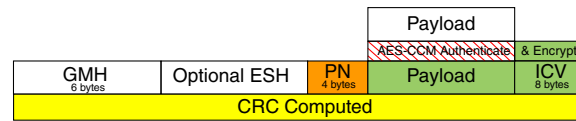


Figure 9-85. 802.16 WiMAX cryptographic and checksum encapsulation

After the FCS has been appended, the output frame resulting from WiMAX encapsulation consists of:

- the software modified GMH (6 bytes)
- an optional ESH (present if the ESF bit is set in the GMH)
- the packet number (4 bytes)
- the encrypted payload
- the AES-CCM produced ICV
- the CRC-produced FCS

NOTE

The WiMAX spec uses two different FCS computation schemes depending on the type of connection. The built-in CRC engine is capable of performing either of the required CRC types.



Figure 9-86. 802.16 WiMAX encapsulation output frame

The processing that SEC performs presumes processor precomputation of the encapsulation- appropriate Generic MAC Header. As shown in [Figure 9-86](#), HT and EC are shown to be set as appropriate for an encapsulated packet, not a decapsulated one. However, [Figure 9-86](#) fails to show that the GMH processor precomputation includes updating the Len field (adding 4 for the PN, 8 for the ICV, and 4 for the FCS) and updating the Header Check Sequence as appropriate for the other changes made to the GMH.

9.6.2 IEEE 802.16 WiMAX encapsulation PDB format descriptions

The IEEE 802.16 WiMAX specification shows two different formats for PN: that labelled PN and that shown as transmitted on the wire. In the PDB, PN is stored (and incremented) in spec PN order, and the order is reversed for building the nonce and for building the output frame.

Table 9-78. WiMAX encapsulation PDB

Descriptor Header (1 or 2 words)				
PDB Word 0	Reserved (24 bits)			
PDB Word 1	Options (8 bits)			
PDB Word 2	Nonce Constant			
PDB Word 3	<table border="1"> <tr> <td>B₀ Flags (8 bits)</td> <td>CTR₀ Flags (8 bits)</td> <td>Counter Initial Count (16 bits)</td> </tr> </table>	B ₀ Flags (8 bits)	CTR ₀ Flags (8 bits)	Counter Initial Count (16 bits)
B ₀ Flags (8 bits)	CTR ₀ Flags (8 bits)	Counter Initial Count (16 bits)		
PDB word 3	PN			
DECO updates PDB as needed				

Table 9-79. WiMAX encapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	FCS

Table 9-80. WiMAX encapsulation PDB, description of the options byte

Field	Description
7-1	Reserved
0	FCS included in Output Frame
FCS	0 FCS not included in Output Frame. 1 FCS computed by encapsulator and included into Input Frame

9.6.3 WiMax encapsulation error conditions

This table lists the conditions under which WiMAX encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

Table 9-81. WiMAX encapsulation error conditions

Condition	Error status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
OPERATION Command PROT ID selects WMAX Encap, and PROTINFO is not a valid protocol	Protocol Command Error
ESH enabled in GMH, and ESH Length byte < 2	Protocol Command Error
PN overflows	Protocol Sequence Number Overflow

9.6.4 Procedure for IEEE 802.16 WiMAX decapsulation

SEC performs decapsulation by doing the following:

1. Receives the input frame without software modification to the GMH (see following figure)

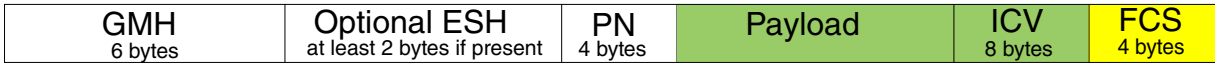


Figure 9-87. 802.16 WiMAX decapsulation input frame

2. Optional: If the frame check sequence has not been verified by software or other external hardware prior to SEC receiving the input frame, verifies the FCS, signalling an error if the received FCS does not match the expected result
 - This FCS check occurs in parallel with other SEC processing of the input frame.
 - The FCS is computed across the GMH, the ESH (present if ESF = 1 in the GMH), the PN, the Payload, and the ICV.
 - If an ESH is present, the ESH must be at least 2 bytes in length or an error is returned.

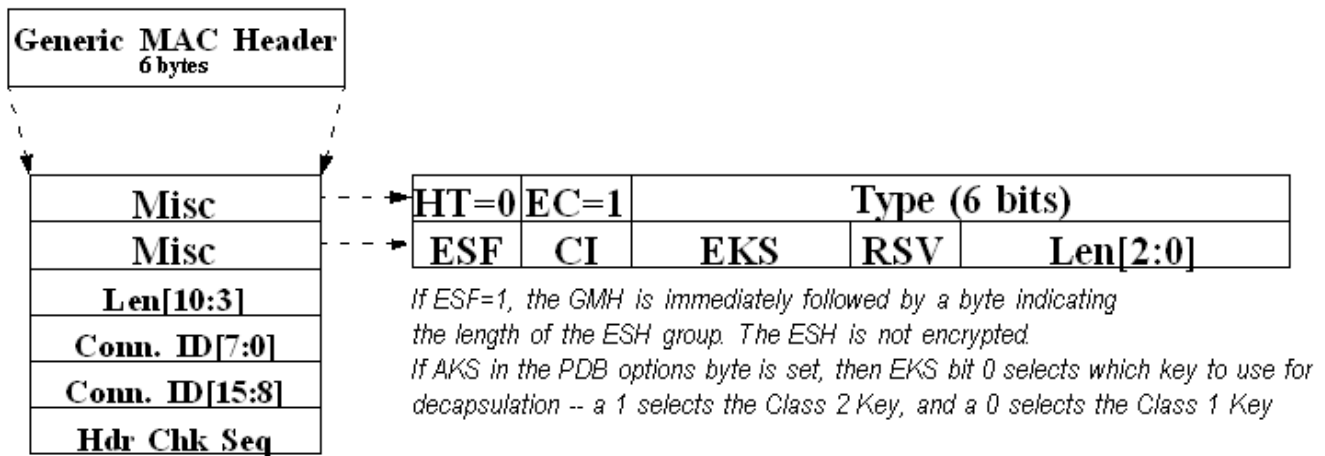


Figure 9-88. 802.16 WiMAX generic MAC header detail

3. Constructs the nonce from the GMH and packet number

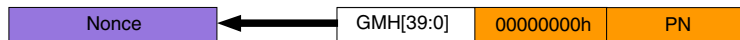


Figure 9-89. 802.16 WiMAX nonce construction

4. Builds the AES-CCM initial counter (CTR₀) and B₀ values using constants extracted from the PDB. Note that if the Protocol Operation command PROTINFO field indicates AES-CCM is not used, then construction of CTR₀ and B₀ is skipped.

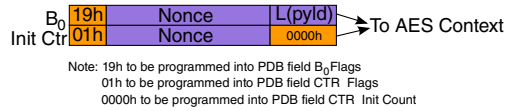


Figure 9-90. 802.16 WiMAX AES-CCM context construction

5. Completes decapsulation of the packet by decrypting the payload and computing the ICV on the received, decrypted payload.
6. Computes and compares the ICV to the ICV value received, and if the two ICVs fail to match, an error is asserted.

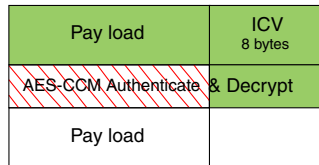


Figure 9-91. 802.16 WiMAX cryptographic decapsulation

7. After the received ICV is checked against the computed ICV, performs anti-replay checking on the Packet Number field.
 - The PDB is configurable for either 32, 64, or 128-packet windows (size determined by the 2-byte Anti-replay length field), and all state information is stored back into the PDB before SEC finishes decapsulating the packet.
 - SEC does not pass the PN onto the output frame.
8. After decryption, pushes the payload to the output frame.

The packet in the output frame consists of the unmodified GMH, the ESH if present in the input frame, and the decrypted payload. Unless option outFMT is set, The ICV, FCS, and PN fields are not included in the Output Frame.

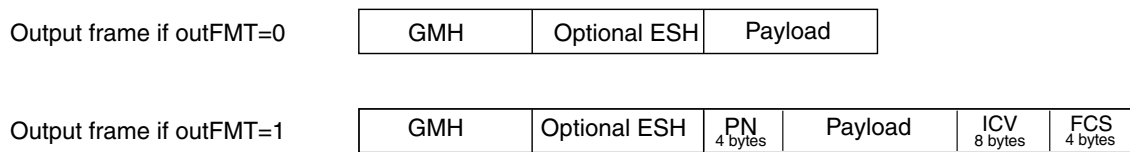


Figure 9-92. 802.16 WiMAX decapsulation output frame

9.6.4.1 Transforming the GMH (WiMAX decapsulation)

The decapsulated output frame still contains the encapsulation GMH. Transformation of the GMH (setting EC to 0, reducing the length as appropriate, and recomputing the header check sequence) requires processor intervention after SEC finishes processing the PDU.

9.6.4.2 Automatic key switching (WiMAX decapsulation)

SEC WiMAX decapsulation has the capability to automatically switch between two keys. If AKS (Automatic Key Switching) is set in the PDB, then SEC uses the least significant bit of the EKS field of the GMH received in the input frame to select which key is used; if the bit is asserted, then the Class 2 Key is used. If the bit is negated, then the Class 1 Key is used. This feature allows reception of frames without software having to preprocess to determine which key is being used.

9.6.5 IEEE 802.16 WiMAX decapsulation PDB format descriptions

Table 9-82. WiMAX decapsulation PDB

Descriptor Header (1 or 2 words)			
PDB Word 0	Reserved (24 bits)	Options (8 bits)	
PDB Word 1	Nonce Constant		
PDB Word 2	B ₀ Flags (8 bits)	CTR ₀ Flags (8 bits)	Counter Initial Count (16 bits)
PDB word 3	PN		
PDB Word 4	reserved	Anti-Replay Length (16 bits)	
PDB word 5	Anti Replay Scorecard 1 [present if AR=1]		
PDB word 6	anti-replay scorecard 2 [present if AR=1 and Anti-Replay Length > 32]		
PDB word 7	anti-replay scorecard 3 [present if AR=1 and Anti-Replay Length > 64]		
PDB word 8	anti-replay scorecard 4 [present if AR=1 and Anti-Replay Length > 96]		

DECO updates PDB in descriptor buffer and external memory as needed

Table 9-83. WiMAX decapsulation PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	AR	Reserved	Reserved	outFMT	Reserved	AKS	FCS

Table 9-84. WiMAX Decapsulation PDB - Description of the Options Byte

Field	Description
7	Reserved
6 AR	Anti-replay enable 0 Anti-replay checking not enabled. 1 Anti-replay checking enabled; length determined by PDB field Anti-Replay Length
5-4	Reserved. Must be zero.

Table continues on the next page...

Table 9-84. WiMAX Decapsulation PDB - Description of the Options Byte (continued)

Field	Description
3	output frame format 0 PN, ICV, and FCS stripped from output frame. 1 PN, ICV, and FCS included in output frame.
2	Reserved. Must be zero.
1 AKS	Automatic Key Switching performed 0 Automatic Key Switching not performed; Class 1 Key is always used. 1 Automatic Key Switching performed; input frame EKS field of GMH selects key.
0 FCS	FCS included in input Frame 0 FCS not included in input Frame. 1 FCS computed by encapsulator and included into Input Frame

9.6.6 WiMAX decapsulation error conditions

This table lists the conditions under which WiMAX decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure can produce an ICV check error.

Table 9-85. WiMAX decapsulation error conditions

Condition	Error Status
Reserved bit set to 1 in the PDB options byte	Protocol PDB error
AR = 1, and Anti-Replay Length > 64	Protocol PDB error
AR = 0, and Anti-Replay Length != 0	Protocol PDB error
OPERATION Command PROT ID selects WiMAX Decap, and PROTINFO is not a valid protocol	Protocol Command Error
ESH enabled in GMH, and ESH Length byte < 2	Protocol Command Error
PN overflows	Protocol Sequence Number Overflow
Anti-Replay detects a LATE packet	Protocol LATE Error
Anti-Replay detects a REPLAY packet	Protocol REPLAY Error

9.7 Anti-Replay built-in checking

Several network protocol decapsulation commands include the ability to perform Anti-Replay checking. This capability is actually built as a separate command that can be invoked separately. As a separate command, Anti-Replay supports a packet number of 16, 32, 48, or 64 bits in length, and an anti-replay window sized anywhere between 1 and 128 entries.

The Anti-Replay operation compares the Packet Number stored in the PDB to the Packet Number stored right-justified in MATH0. It is the responsibility of other descriptor commands to put the Packet Number into MATH0 in the correct form. Four checks can be performed by the Anti-Replay operation:

1. Late Check determines if the Packet Number found in MATH0 is smaller than the Packet Number found in the PDB by at least the size of the Anti Replay Window. That is, MATH0's Packet Number is so old that it cannot be checked for a replay. If determined LATE, then either an error or a warning will be signaled as determined by PDB options bit RLST.
2. Replay Check determines if the Packet Number found in MATH0 is already reflected by appropriate entry in the Anti Replay Window. In particular, the window entry associated with the value in MATH0 is checked. If already set, then it is determined to be REPLAY. If not yet set, then it is not REPLAY, and the entry is set. If determined REPLAY, then either an error or a warning will be signaled as determined by PDB options bit RLST.
3. Packet Number Overflow check (optional; occurs if COF=1) detects if Packet Number in MATH0 has rolled past the maximum Packet Number, as determined by the length of the Packet Number. An Overflow indicates the Packet Number in the PDB is close to $2^{\text{PNLen}} - 1$, but the Packet Number in MATH0 is close to zero. When an overflow is detected, either an error or a warning is signalled, as determined by Option OUST.
4. Packet Number Underflow check (optional; occurs if CUF=1) detects if Packet Number in MATH0 has rolled back past zero. For this to be detected, the Packet Number in the PDB must be close to zero and the Packet Number in MATH0 must be close to $2^{\text{PNLen}} - 1$. When an underflow is detected, either an error or a warning is signalled, as determined by Option OUST.

If the value in MATH0 is not a replay and is newer than the Packet Number in the PDB, then the PDB is updated with the Packet Number value from MATH0, and the Anti Replay Window is shifted such that the newest entry corresponds to the new Packet Number. The Packet Number and Anti Replay window fields are updated back to memory where the descriptor was fetched from.

Upon successful completion of the Anti-Replay operation, the status field of the PDB is updated.

NOTE

If an error is signalled, then descriptor execution is terminated immediately; further tasks will not be completed.

When the Anti-Replay operation is invoked as a stand-alone operation, it requires its own PDB that is of a form different than seen when Anti-Replay is part of a network protocol. ARLen, a field in the first PDB word, control the size of the anti-replay window, and is programmable to any integer between 1 and 128. Options byte field PNLen selects the packet number length. Options bits CUF and COF indicate whether or not to check for packet number underflow (roll-under) or overflow (roll-over). Bit OUST determines whether to signal an error or a warning for any detected packet number underflow or overflow. Bit RLST determines whether to signal an error or a warning if either a LATE or a REPLAY Packet Number is detected.

Table 9-86. Anti-Replay built-in checking PDB

		Descriptor Header (1 or 2 words)			
PDB Word 0	Status (4 bits)	Reserved (12 bits)	ARLen (8 bits)	Options (8 bits)	
PDB Word 1	Upper Packet Number (unused if PNLen is 32 or 16)				DECO writes back to PDB as needed
PDB Word 2	Lower Packet Number				
PDB Word 3	Anti Replay Window entries 31-0 (always present)				
PDB Word 4	Anti Replay Window entries 63-32 (present if ARLen > 32)				
PDB Word 5	Anti Replay Window entries 95-64 (present if ARLen > 64)				
PDB Word 6	Anti Replay Window entries 127-96 (present if ARLen > 96)				

Table 9-87. Anti-Replay built-in checking PDB, format of the status nibble

31	30	29	28
PNUupdate	LATE	REPLAY	OUPD

Table 9-88. Anti-Replay built-in checking PDB, description of the status nibble

Field	Description
31 PNUupdate	Indicates if the Packet Number field in the PDB was updated by the Anti Replay command 0 : was not updated 1 : was updated
30 LATE	LATE was detected but no error was issued 0 : no LATE detected 1 : LATE Packet Number detected
29 REPLAY	REPLAY was detected but no error was issued 0 : no REPLAY detected

Table continues on the next page...

Table 9-88. Anti-Replay built-in checking PDB, description of the status nibble (continued)

Field	Description
	1 : REPLAY Packet Number detected
28	Underflow or Overflow Packet Number detected but no error issued
OUPD	0 : no Underflow or Overflow detected 1 : Underflow or Overflow of Packet Number detected

Table 9-89. Anti-Replay built-in checking PDB, format of the options byte

7-6	5-4	3	2	1	0
Reserved	PNLen	CUF	COF	OUST	RLST

Table 9-90. Anti-Replay built-in checking PDB, description of the options byte

Field	Description
7-6	Reserved
5-4	Selects Packet Number Size
PNLen	00 : use 16 bit Packet Number 01 : use 32 bit Packet Number 10 : use 48 bit Packet Number 11 : use 64 bit Packet Number
3	Check for packet number Underflow.
CUF	0 : do not check for underflow 1 : check for underflow -- if Packet Number rolls back past zero, signal per OUE
2	Check for packet number Overflow.
COF	0 : do not check for overflow 1 : check for overflow -- if Packet Number rolls forward past $2^{\text{PNLen}} - 1$, signal per OUST
1	Overflow / Underflow signal type.
OUST	0 : signal warning upon detection of Packet Number Overflow or Underflow 1 : signal error upon detection of Packet Number Overflow or Underflow
0	Replay / Late signal type.
RLST	0 : signal warning upon detection of late or replay Packet Number 1 : signal error upon detection of late or replay Packet Number

9.8 Process for 3G double-CRC encapsulation and decapsulation

SEC includes a double-CRC encapsulation and decapsulation protocol thread designed for 3G MAC-d protection.

The unencapsulated frame contains a packet consisting of:

- A PDU header
- A PDU payload consisting of 0 or more bytes

SEC computes either a 7-bit or 11-bit CRC of the PDU Header (the length of which is in the PDB) and the 16-bit CRC of the PDU Payload.

- The 7-bit CRC computation uses an irreducible polynomial of $D^7+D^6+D^2+1$.
- The 11-bit CRC uses an irreducible polynomial is $D^{11}+D^9+D^8+D^2+D+1$.
- The 16-bit CRC uses an irreducible polynomial of $D^{16}+D^{15}+D^2+1$.

For all three computations, the CRC engine is configured with DIS, DOS, DOC, and IVZ all set.

9.8.1 3G double-CRC encapsulation process

The encapsulated output frame consists of the following:

- The PDU header, which is copied from the input frame after being modified by insertion of the header CRC
- The PDU payload, which is copied without modification from the input frame
- The sixteen-bit payload CRC

9.8.1.1 Calculating the 7-bit CRC of the PDU header for encapsulation

This figure shows 3G double-CRC encapsulation with 7-bit header CRC.

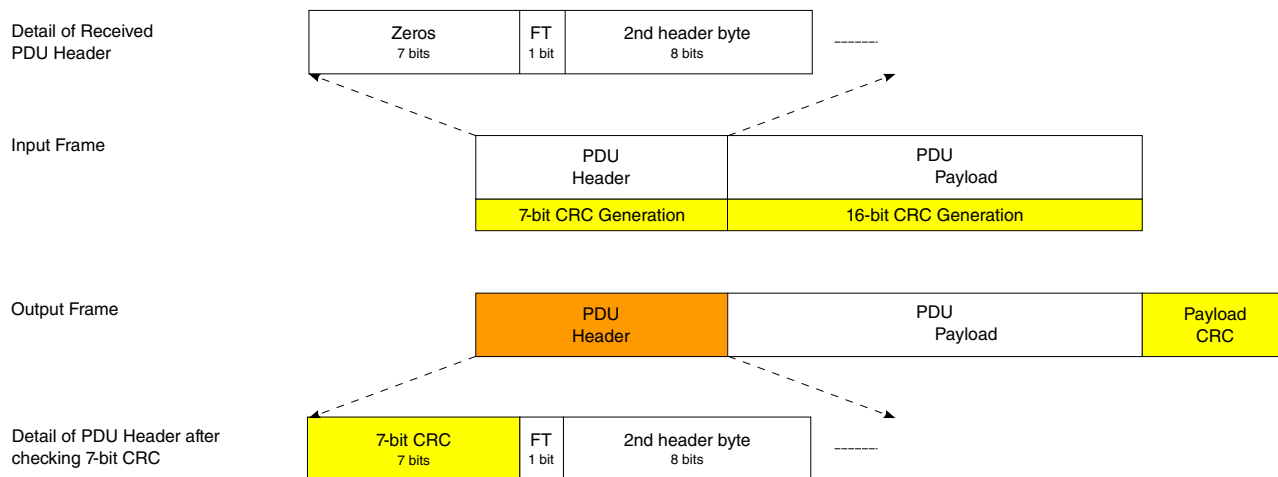


Figure 9-93. 3G double-CRC encapsulation with 7-bit header CRC

If the PROTINFO field of the Operation Command selects a 7-bit CRC, it is computed as follows:

1. SEC takes the number of bytes for the PDU Header from the input frame as defined by the PDU Header Length field in the PDB.
2. SEC zeroes the first 7 bits as the header is passed to the CRC engine.
3. After the 7-bit CRC computation is completed, the computed value is moved to a DECO register, along with the FT bit that completes the byte containing the CRC.

9.8.1.2 Calculating the 11-bit CRC of the PDU header for encapsulation

This figure shows 3G double-CRC encapsulation with 11-bit header CRC.

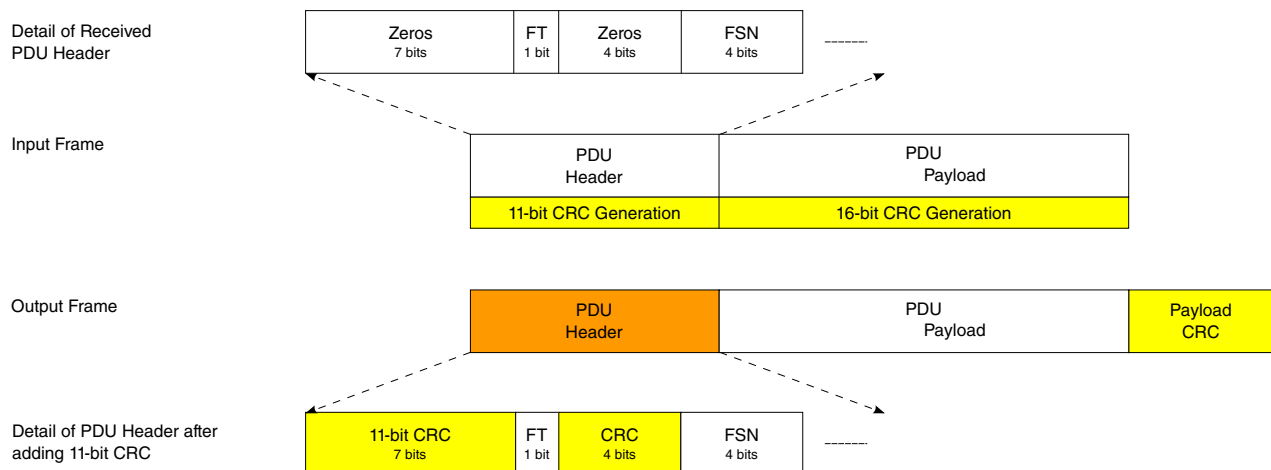


Figure 9-94. 3G double-CRC encapsulation with 11-bit header CRC

If the PROTINFO field of the Operation Command selects an 11-bit CRC, it is computed against a modified PDU Header as follows:

1. SEC takes the number of bytes for the PDU Header from the input frame as defined by the PDU Header Length field in the PDB.
2. The PDU Header bits reserved for the CRC are zeroed.
3. The second byte of the PDU Header is re-ordered before being passed to the CRC engine: the FT bit is moved to be after the entire CRC field.
4. The PDU Header is copied from the input frame to the output frame, unmodified, along with the PDU Payload.
5. The completed 11-bit CRC value is moved to a DECO register, along with the rest of the first two bytes of the PDU Header.

9.8.1.3 Calculating the 16-bit payload CRC for encapsulation

After the header CRC computation has been completed, the CRC CHA is reconfigured for the sixteen-bit payload CRC computation.

1. The PDU Payload is passed unmodified from the input frame to the output frame and is also passed to the CRC CHA.
2. The CRC CHA calculates the payload CRC.
3. The CRC value is appended to the end of the output frame by the CRC engine.
4. The saved first byte or two of the PDU Header with inserted CRC is written out, overwriting the start of the output frame.

9.8.2 3G double-CRC encapsulation PDB format descriptions



Figure 9-95. 3G double-CRC encapsulation PDB

Table 9-91. 3G double-CRC encapsulation PDB, format of the options byte

7..0
Reserved

Table 9-92. 3G double-CRC encapsulation PDB, description of the options byte

Field	Description
0-7	Reserved

9.8.3 3G double-CRC decapsulation process

The decapsulation process is as follows:

1. The PDU Header CRC is checked and passed as-is from the input frame to the output frame, along with the rest of the PDU Header.
2. If the received header CRC does not match the computed header CRC, an error results.

3. The received payload CRC is checked against the computed payload CRC, but the output frame does not include the payload CRC, meaning that the decapsulated output frame is 2 bytes shorter than the input frame.
4. If the received payload CRC does not match the computed payload CRC, an error is returned.

9.8.3.1 Calculating the 7-bit CRC of the PDU header for decapsulation

This figure shows 3G double-CRC decapsulation with 7-bit header.

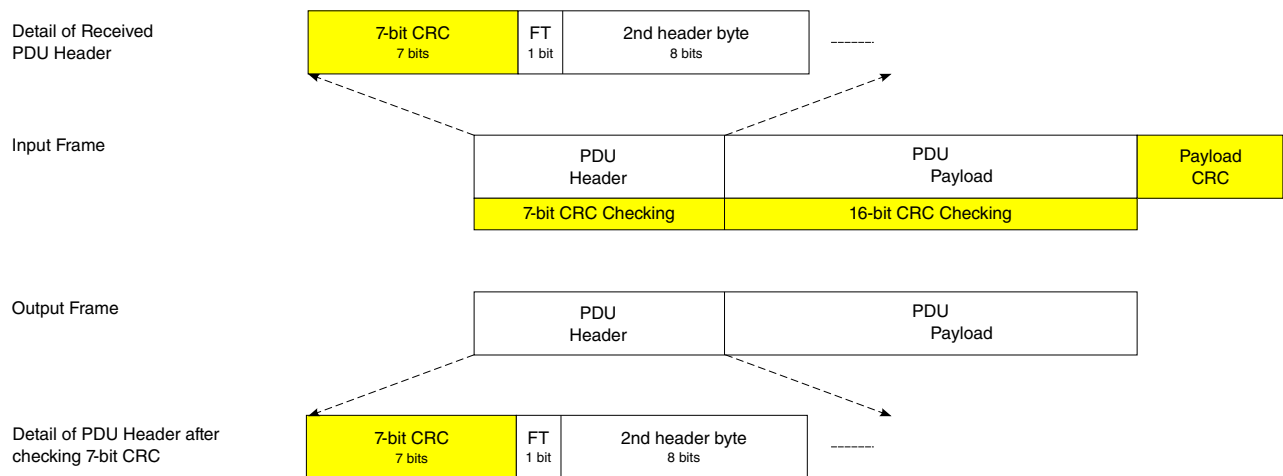


Figure 9-96. 3G double-CRC decapsulation with 7-bit header

If the value of the Operation Command PROTINFO field selects a seven-bit CRC computation, the procedure is:

1. The first 7 bits of the PDU header are checked against a CRC computed across the PDU header; the length of the PDU header is determined by the PDU Header Length field in the PDB.
2. The PDU header is copied verbatim from the input frame to the output frame.
3. For the purposes of the CRC computation, the received seven-bit CRC is zeroed prior to passing it to the CRC engine.
4. The seven-bit CRC received from the input frame is written separately into the CRC engine, and is compared to the computed CRC.
5. If the two CRC values do not match, an error is returned.

9.8.3.2 Calculating the 11-bit CRC of the PDU header for decapsulation

This figure shows 3G double-CRC decapsulation with a 11-bit header.

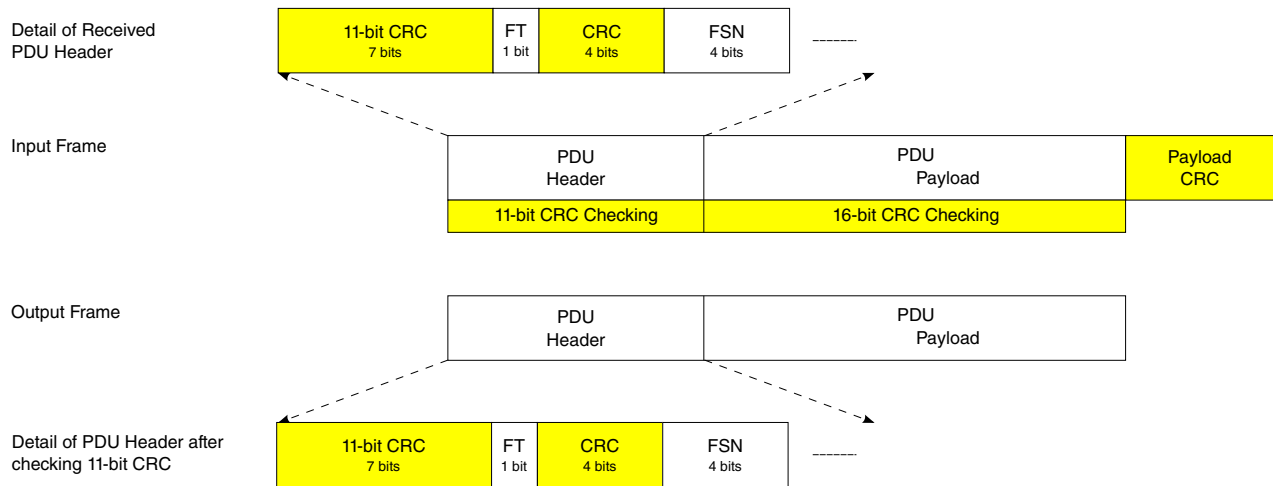


Figure 9-97. 3G double-CRC decapsulation with 11-bit header

If the value of the Operation Command PROTINFO field selects an 11-bit CRC computation, the procedure is the same as for the 7-bit CRC except for how the PDU Header is presented to the CRC engine. Because the FT bit appears in the last bit of the first byte of the PDU Header, the first two bytes of the header are reordered going into the CRC engine, in addition to zeroizing the CRC fields. The PDU header is presented such that the FT bit follows eleven zeroed CRC bits, but precedes all other parts of the PDU header.

9.8.3.3 Calculating the 16-bit payload CRC for decapsulation

After the header CRC has been checked, a 16-bit CRC is computed across the PDU payload, which comprises everything after the PDU Header in the input frame except for the final two bytes. The final two bytes of the input frame comprise the payload CRC, which the CRC engine compares to the computed payload CRC. If the two CRC values do not match, an error is returned.

9.8.4 3G double-CRC decapsulation PDB format descriptions



Figure 9-98. 3G double-CRC decapsulation PDB

Table 9-93. 3G double-CRC decapsulation PDB, format of the options byte

7..0
Reserved

Table 9-94. 3G double-CRC decapsulation PDB, description of the options byte

Field	Description
7-0	Reserved.

9.9 3G RLC PDU Encapsulation and Decapsulation overview

SEC implements encapsulation and decapsulation for 3G RLC PDUs. For RLC PDUs, only confidentiality is provided; authentication is handled at a different protocol layer. SEC supports both Kasumi-f8 (UEA1) and SNOW-3g f8 (UEA2) for NULL confidentiality (UEA0) as well as for confidentiality.

Table 9-95. 3G RLC protocol descriptors

Encapsulation	Decapsulation
Header	Header
Protocol data block, includes HFN	Protocol data block, includes HFN
Class 1 key data block	Class 1 key data block
Protocol = <protocol> encrypt	Protocol = <protocol> decrypt

9.9.1 3G RLC PDU encapsulation overview

The input frame consists of a single 3G RLC layer PDU header and payload.

3G RLC PDU Encapsulation and Decapsulation overview

- The header is either one or two bytes, depending on the length of the sequence number.
- The payload consists of zero or more bytes.

The encapsulation output frame consists of:

- The PDU header (unmodified)
- The encrypted PDU payload
- Optional padding

Encapsulation can occur in either unacknowledged mode or acknowledged mode:

- If the PDB options byte SNS bit = 1, the PDU is interpreted as an unacknowledged mode PDU, with a 7-bit sequence number in a one-byte PDU Header.
- If the PDB options byte SNS bit is not 1, the PDU is interpreted as an acknowledged mode PDU header, with a twelve-bit sequence number in a two-byte PDU header.

This figure shows 3G RLC PDU encapsulation.

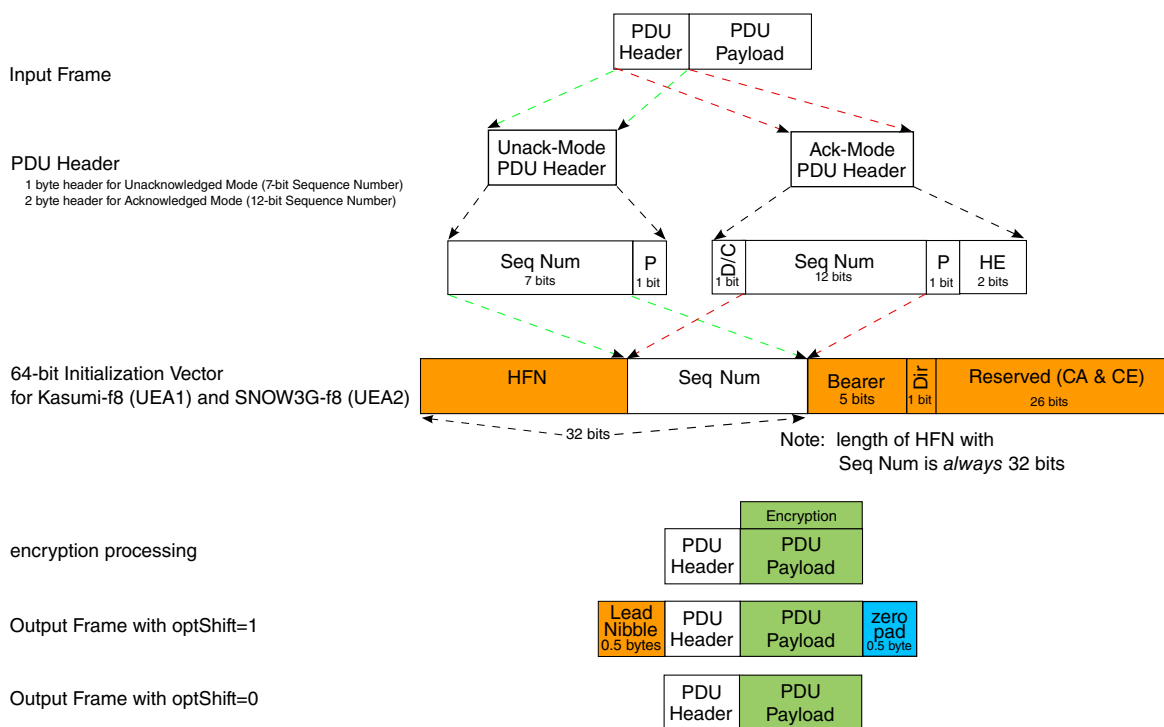


Figure 9-99. 3G RLC PDU encapsulation

9.9.2 Process for 3G RLC PDU encapsulation

The 3G RLC PDU encapsulation procedure is:

1. Prior to starting encryption, the sequence number is extracted and combined with the HyperFrame Number (HFN) field maintained in the PDB.
2. (optional) When the sequence number rolls over from all 1s to zero, the HFN field in the PDB is incremented and written back.
3. Whether incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in f8 IV construction, but is not provided to the selected encryption engine (KFHA or SNOW-3G-f8) for encryption.
6. An f8 initialization vector (IV) is built from the following:
 - HFN field as found in the input PDB
 - The PDB word that includes Bearer and Direction
 - The Sequence Number from the input frame
7. The IV is written to the Class 1 Context Register prior to commencing encryption.
8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting encrypted PDU payload is sent to the output frame.

9.9.3 3G RLC PDU encapsulation PDB format descriptions

Table 9-96. 3G RLC PDU encapsulation PDB

		Descriptor Header (1 or 2 words)				
PDB Word 0	Resvd (4 bits)	Lead Nibble (4 bits)	Reserved (16 bits)	Options (8 bits)		
PDB Word 1	HFN (25 bits)			Reserved (7 bits)	DECO writes back to PDB as needed	
PDB Word 2	Bearer, Dir, Reserved for CA & CE					

The PDB options byte includes a bit enabling an optional shift. If set, the output frame is 1 byte longer than the input frame, and the entire PDU is offset by 4 bits from the start of the frame. The 4 bits added to the output frame at the front are taken from the Lead Nibble field of the PDB and the 4 bits added at the tail are all zeros.

Table 9-97. 3G RLC PDU encapsulation PDB, format of the options byte

7-3	2-1	0
Reserved	SNS	optShift

Table 9-98. 3G RLC PDU encapsulation PDB, description of the options byte

Field	Description
7-3	Reserved
2-1 SNS	Selects Serial Number Size. Ignored for LTE C-Plane. If SNS = 00b : 12-bit Serial Number. This is Acknowledged mode for 3G. If SNS = 01b : 7-bit Serial Number. This is Unacknowledged mode for 3G. If SNS = 10b : 15-bit Serial Number. Not recommended for 3G RLC encapsulation. If SNS = 11b : Reserved.
0 optShift	Enables the optional four-bit Output Frame shift If optShift = 0 : No shift If optShift = 1 : Shift the output frame 4 bits, inserting zeros.

9.9.4 3G RLC PDU decapsulation overview

The encapsulated input frame consists of:

- A single 3G RLC layer PDU Header
- An encrypted PDU Payload consisting of zero or more bytes

NOTE

If the optShift bit in the PDB Options byte is set, the input frame includes an extra byte beyond the PDU: half the byte (four bits) at the front (before the header) and half the byte (four bits) at the end (after the payload). The frame must be unshifted before decryption can take place.

The PDU Header is either one or two bytes, depending on the length of the sequence number.

The decapsulation output frame consists of:

- The PDU header (unmodified)
- The decrypted PDU payload

Decapsulation can occur in either unacknowledged or acknowledged mode.

- If the PDB Options byte SNS bit = 1, the PDU is interpreted as an unacknowledged mode PDU, with a 7-bit sequence number in a 1-byte PDU Header.
- If the PDB options byte SNS bit is not 1, the PDU is interpreted as an acknowledged mode PDU header, with a twelve-bit sequence number in a two-byte PDU header.

This figure shows 3G RLC PDU decapsulation.

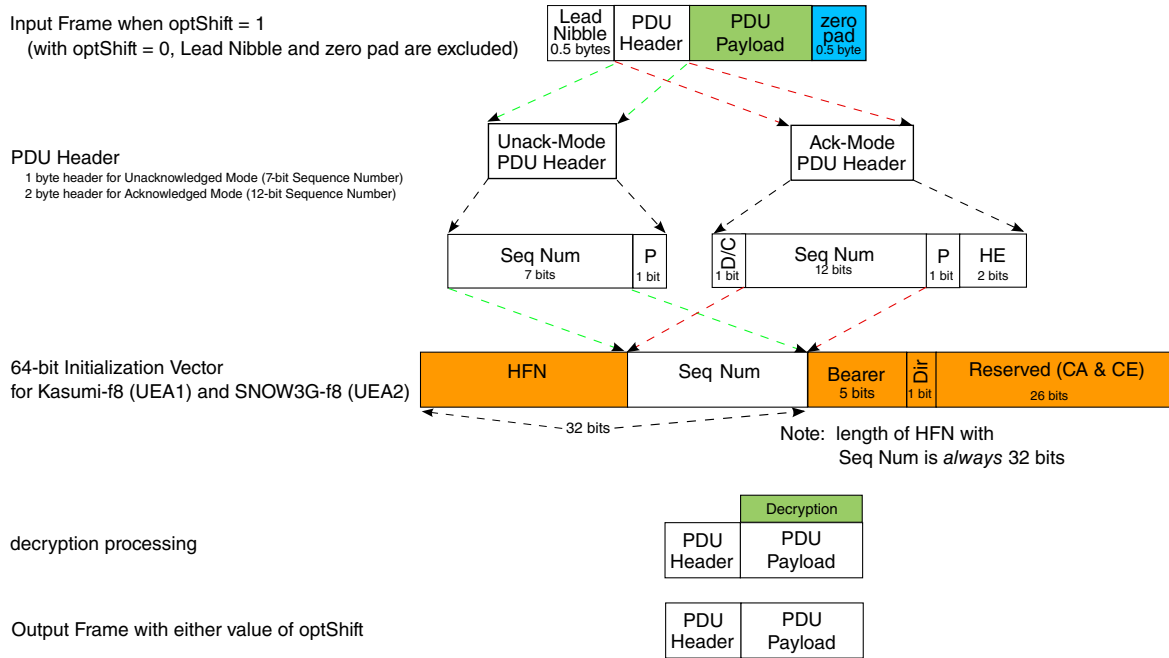


Figure 9-100. 3G RLC PDU decapsulation

9.9.5 Process for 3G RLC PDU decapsulation

The decapsulation procedure is:

1. Prior to starting decryption, the sequence number is extracted and combined with the Hyper Frame Number (HFN) field maintained in the PDB.
2. Whenever the sequence number rolls over from all 1s back to zero, the HFN field in the PDB is incremented and written back; note that there is no provision for rolling back the HFN, so frames must be provided in-order.
3. Whether HFN is incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN field matches or exceeds the Threshold field, a warning is returned when frame processing is complete. This warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame (shifted if optShift = 1). It is used in f8 IV construction, but is not provided to the selected confidentiality engine (KFHA or SNOW-3G-f8) for decryption.
6. An f8 Initialization Vector (IV) is built from:
 - The Hyper Frame Number (HFN) as found in the input PDB

- The PDB word that includes Bearer and Direction
 - The Sequence Number from the input frame
7. The IV is written to the Class 1 Context Register prior to commencing decryption.
 8. The entire PDU Payload is moved from the Input Frame into the input-data FIFO as message data.
 9. The resulting decrypted PDU Payload is sent to the output frame.

9.9.6 3G RLC PDU decapsulation PDB format descriptions

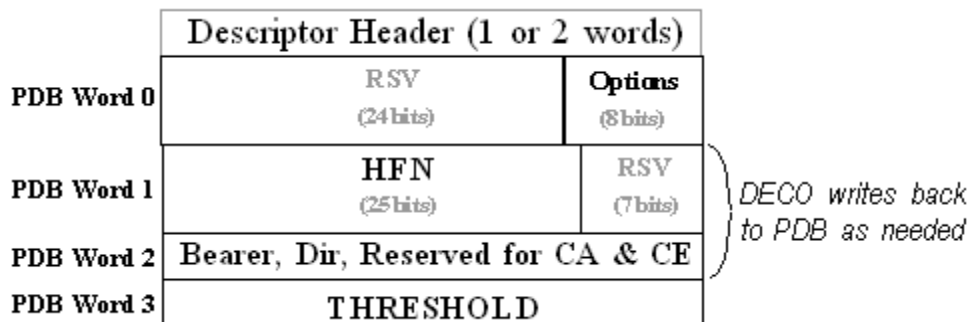


Figure 9-101. 3G RLC PDU decapsulation PDB

Table 9-99. 3G RLC PDU decapsulation PDB, format of the options byte

	7	6	5	4	3	2	1	0
	RSV	RSV	RSV	RSV	RSV	RSV	SNS	optShift

Table 9-100. 3G RLC PDU decapsulation PDB, description of the options byte

Field	Description
	Reserved.
SNS	Selects Serial Number Size. Ignored for LTE C-plane 0: 12-bit Serial Number. This is Acknowledged mode for 3G. 1: 7-bit Serial Number. This is Unacknowledged mode for 3G.
optShift	Selects whether or not to perform a 4-bit shift before decryption. 0: No shift 1: 4-bit shift performed

9.9.7 Overriding the PDB for 3G RLC PDU encapsulation and decapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For 3G RLC PDU encapsulation and decapsulation, the HFN is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired HFN value in the job descriptor. For more information, see [Job Ring interface](#). When using the Queue Manager Interface, QI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field in the FD. For more information, see [Queue Manager Interface \(QI\)](#). When using the Advanced IO Processor interface, AI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field provided in the FD. For more information, see [Advanced IO Processor Interface \(AI\)](#).

Table 9-101. Format of the DPOVRD register when used with the 3G RLC protocol

format when HFN is 20 bits	OVRD	Reserved (12 bits)						HFN (20 bits)																				
format when HFN is 25 bits	OVRD	Reserved (7 bits)			HFN (25 bits)																							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

9.10 LTE PDCP PDU encapsulation and decapsulation overview

For LTE, 3GPP moved confidentiality and integrity to both reside in the PDCP layer. Not all modes use authentication.

- PDCP User Plane uses confidentiality only. The encapsulation and decapsulation process is very much like that for 3G RLC PDUs.
- PDCP User Plane for RN uses confidentiality and integrity. The encapsulation and decapsulation process is very much like that for PDCP Control Plane PDUs.
- PDCP Control Plane uses confidentiality and integrity.

The following algorithms are supported for LTE PDCP:

- 128-EEA0 (null-confidentiality) for confidentiality.
- 128-EEA1 (SNOW-3G) for confidentiality.
- 128-EEA2 (AES-CTR) for confidentiality.

- 128-EEA3 (ZUC) for confidentiality.
- 128-EEA0 (null-integrity) for integrity.
- 128-EIA1 (SNOW-3G) for integrity.
- 128-EIA2 (AES-CMAC) for integrity.
- 128-EIA3 (ZUC) for integrity.

NOTE

Note that whenever processing the integrity function is included, any combination of confidentiality and integrity algorithm may be selected.

Table 9-102. LTE control plane protocol descriptors

Encapsulation		Decapsulation
Header		Header
Protocol data block, includes HFN		Protocol data block, includes HFN
Class 2 key data block		Class 2 key data block
Class 1 key data block		Class 1 key data block
Protocol = <protocol> encrypt		Protocol = <protocol> decrypt

9.10.1 LTE PDCP PDU IV generation

Each of the algorithms supported by SEC require the generation of initialization vectors (IV). In most cases, a confidentiality IV is written into the Class 1 context register, and a different integrity IV is written into the Class 2 context register to affect processing. EIA2 (AES-CMAC) is the exception; CMAC has no provision for an IV, so the value computed is used as AAD (additional authenticated data) instead. The various IV values are constructed using Direction, Bearer, some constants programmed into the PDB (in case of future standards work), and Count. Count itself is created by using HFN as the most significant bits and the header's sequence number for least significant bits. Generation of the forms of IV used are shown in the diagrams following. These IVs are used for both encapsulation and decapsulation.

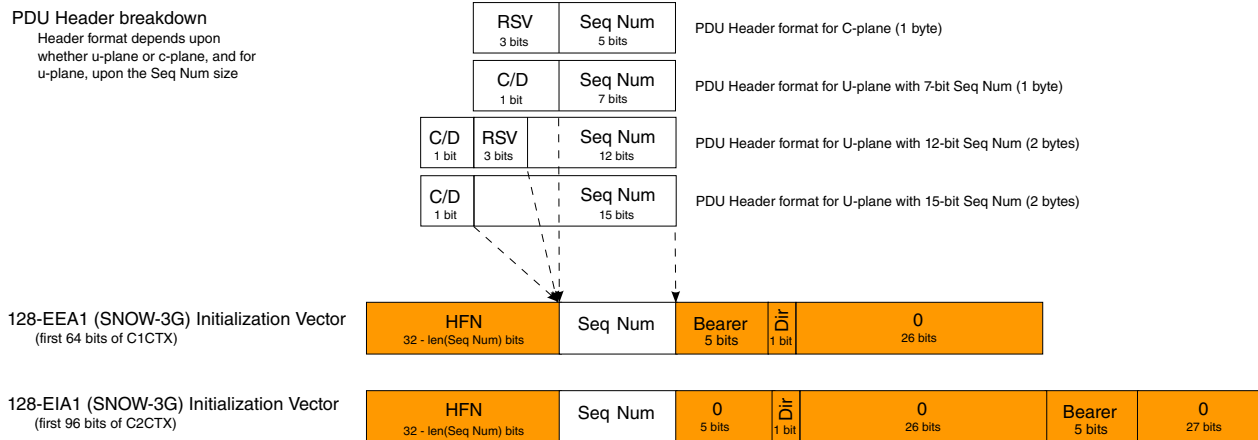


Figure 9-102. IV generation for 128-EEA1 and 128-EIA1 (SNOW3G)

Generation of IVs for SNOW3G occurs as follows:

- COUNT is constructed by taking HFN from the PDB and appending the Sequence Number from the input frame
- For the confidentiality IV, Bearer and Direction are taken from the PDB and appended immediately following COUNT
- The confidentiality IV is completed by appending a 26-bit constant zero value from the PDB (labelled CA & CE).
- For the integrity IV, the Bearer value is moved (and replaced with zeros) from immediately after COUNT to following the end of the confidentiality IV.
- The integrity IV is completed by appending 27 bits of zeros.

NOTE

The integrity IV is only constructed if integrity processing is being performed.

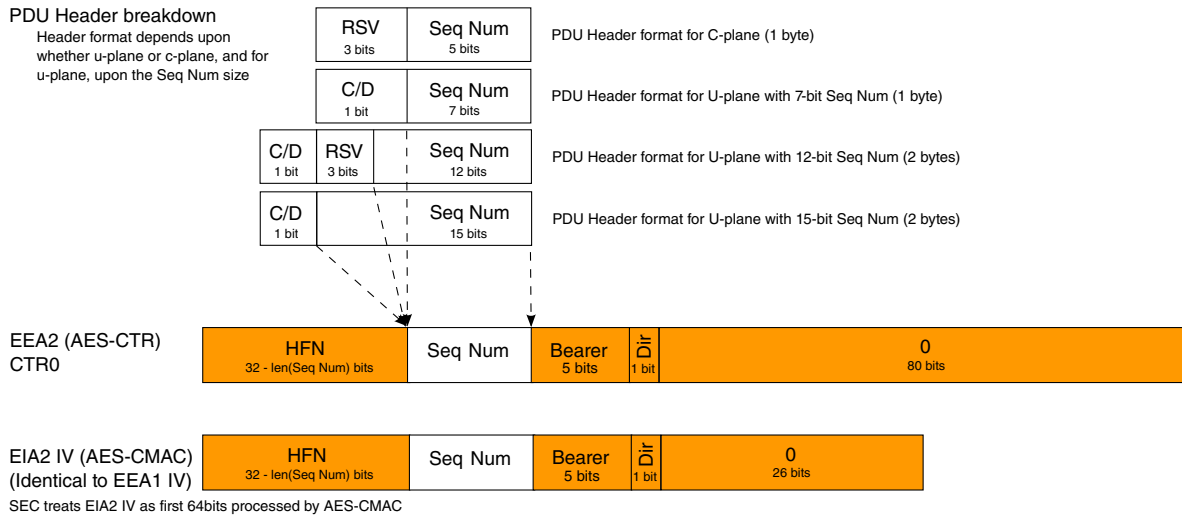


Figure 9-103. IV generation for 128-EEA2 and 128-EIA2 (AES)

Generation of IVs for AES occurs as follows:

- COUNT is constructed by taking HFN from the PDB and appending the Sequence Number from the input frame
- Bearer and Direction are taken from the PDB and appended after COUNT.
- a zero constant is applied to make the confidentiality IV 128 bits, and to make the integrity IV 96 bits.

NOTE

The integrity IV is only constructed if integrity processing is being performed.

NOTE

AES-CMAC has no provision for an IV. As a result, the generated integrity IV is applied as additional authenticated data (AAD) prior to the input frame.

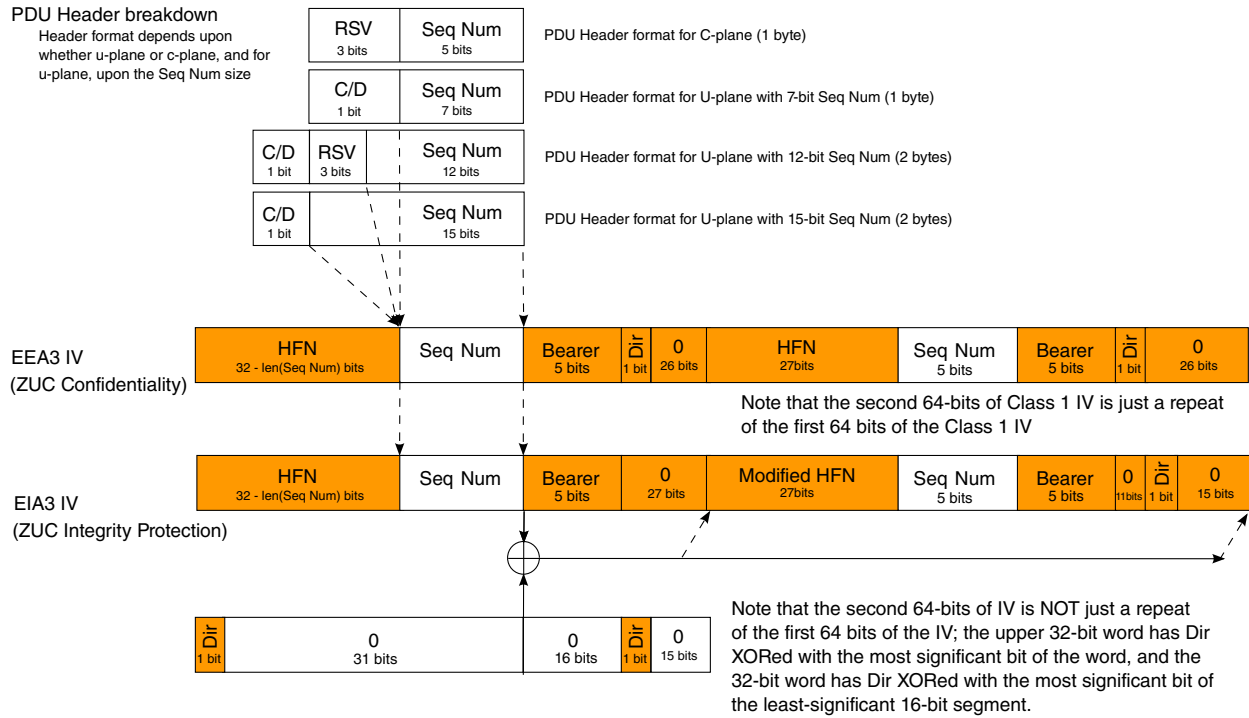


Figure 9-104. IV generation for 128-EEA3 and 128-EIA3 (ZUC)

Generation of IVs for ZUC occurs as follows:

- The ZUC confidentiality IV is just the SNOW3G confidentiality IV repeated twice.
- The integrity IV is created from the confidentiality IV by removing the Direction bits and
 - A direction bit is XORed into the most significant bit of the lower-half HFN.
 - The direction bit from the lower-half of the confidentiality IV is shifted 10 bits to the right.

NOTE

The integrity IV is only constructed if integrity processing is being performed.

9.10.2 LTE PDCP PDU encapsulation process for confidentiality only

LTE PDCP PDUs use only confidentiality and no integrity function for user plane operation, when not for relay nodes (RN). A RN may be configured to use integrity protection; for that case see [LTE PDCP PDU encapsulation for confidentiality and integrity](#). The encapsulation procedure is as follows:

1. Prior to starting encryption, Count is created by extracting the sequence number from the input frame header and combining it with the Hyper Frame Number (HFN) maintained in the PDB.
2. Whenever the sequence number rolls over from all 1s back to zero, the HFN field in the PDB is incremented and written back.
3. The HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in the construction of the IV, but is not provided to the selected encryption engine (AESA or SNOW3Gor ZUC) for encryption.
6. A confidentiality initialization vector (IV) is built as shown in [LTE PDCP PDU IV generation](#) from:
 - The HFN as found in the input PDB
 - The PDB word that includes Bearer and Direction
 - The Sequence Number from the input frame.
7. This IV is written to the Class 1 Context Register prior to commencing encryption.
8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting encrypted PDU Payload is sent to the output frame.

The process is shown in the figure below.

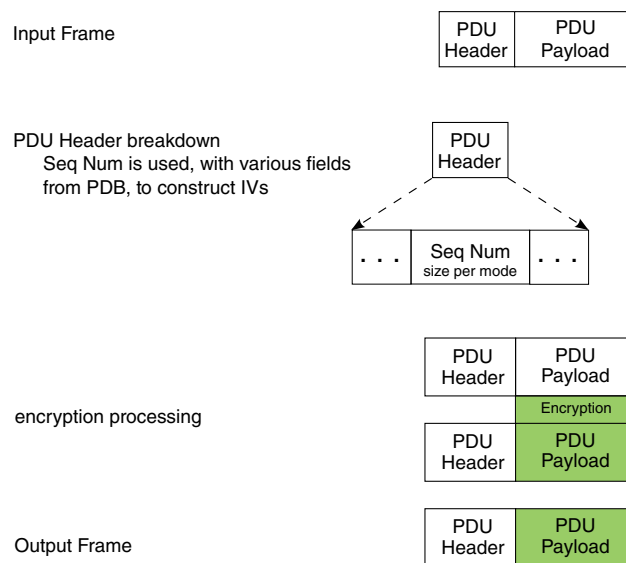


Figure 9-105. LTE PDCP PDU encapsulation for confidentiality only

9.10.3 LTE PDCP PDU encapsulation for confidentiality and integrity

This figure shows LTE PDCP PDU encapsulation for confidentiality and integrity.

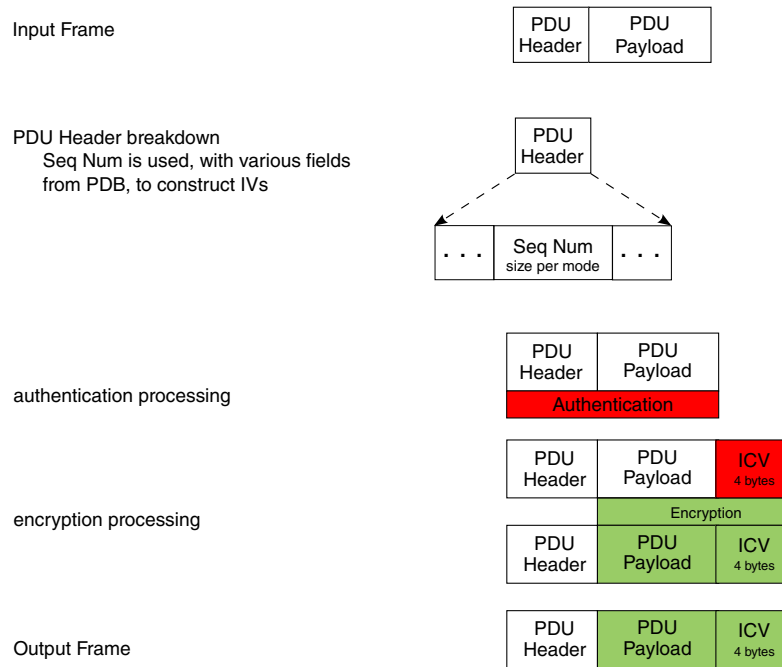


Figure 9-106. LTE PDCP PDU encapsulation for confidentiality and integrity

Control plane PDUs include a 1-byte PDU header with a 5-bit sequence number. User plane PDUs for relay nodes (RN) have 7, 12, or 15 bit sequence numbers. User plane headers are 2-bytes for larger sequence numbers, or 1-byte for 7-bit sequence numbers.

1. SEC receives an input frame containing the PDU header and PDU payload.
2. SEC extracts the sequence number from the PDU header.
3. SEC creates the confidentiality IV by taking the two words that contain the Hyper Frame Number (HFN), Bearer, and Direction fields from the PDB and dropping in the sequence number that was extracted from the PDU Header.
4. After the confidentiality IV has been written to the Class 1 Context Register and the integrity IV has been written to the Class 2 Context Register, SEC begins cryptographic computations.
5. The PDU Header is written to the data FIFO to be written to the output frame and also to be processed by the selected integrity (class 2) CHA.

6. The PDU Payload is written to the data FIFO, to be processed by both the confidentiality (class 1) and integrity CHAs. The encrypted PDU Payload is written to the output frame
7. After computation of the integrity MAC-I (Message Authentication Check) value, it is passed back for encryption by the confidentiality CHA.
8. The four-byte encrypted MAC-I is appended to the end of the output frame.

9.10.4 LTE PDCP PDU decapsulation process for confidentiality only

LTE PDCP PDUs use only confidentiality for user plane operation, when not for relay nodes (RN). A RN may be configured to use integrity protection; for that case see [LTE PDCP PDU decapsulation for confidentiality and integrity](#). The decapsulation process is:

1. Prior to starting decryption, SEC extracts the sequence number and combines it with the Hyper Frame Number (HFN) maintained in the PDB.
2. Whenever the sequence number rolls over from all 1's back to zero, the HFN field in the PDB is incremented and written back; note that because there is no provision for rolling the HFN back, PDUs must be provided in-order.
3. Whether the HFN is incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in the construction of the IV, but is not provided to the selected confidentiality engine (SNOW-3G-f8 or AESA or ZUC) for decryption.
6. A confidentiality Initialization Vector (IV) is built from:
 - The Hyper Frame Number (HFN) as found in the input PDB
 - The PDB word that includes Bearer and Direction
 - The Sequence Number from the input frame.
7. This IV is written to the Class 1 Context Register prior to commencing decryption.

NOTE

Note that the IV is constructed slightly differently, depending on the underlying cryptographic algorithm. This is described in [LTE PDCP PDU IV generation](#).

8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting decrypted PDU Payload is sent to the output frame.

The process is shown in the figure below.

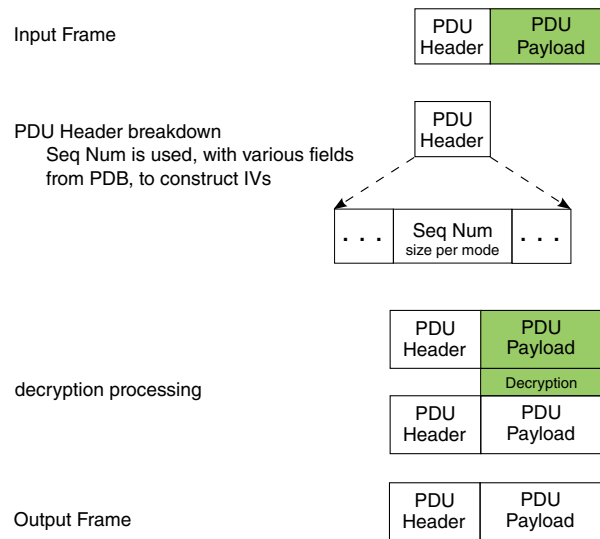


Figure 9-107. LTE PDCP PDU decapsulation for confidentiality only

9.10.5 LTE PDCP PDU decapsulation for confidentiality and integrity

This figure shows LTE PDCP PDU decapsulation when using confidentiality and integrity.

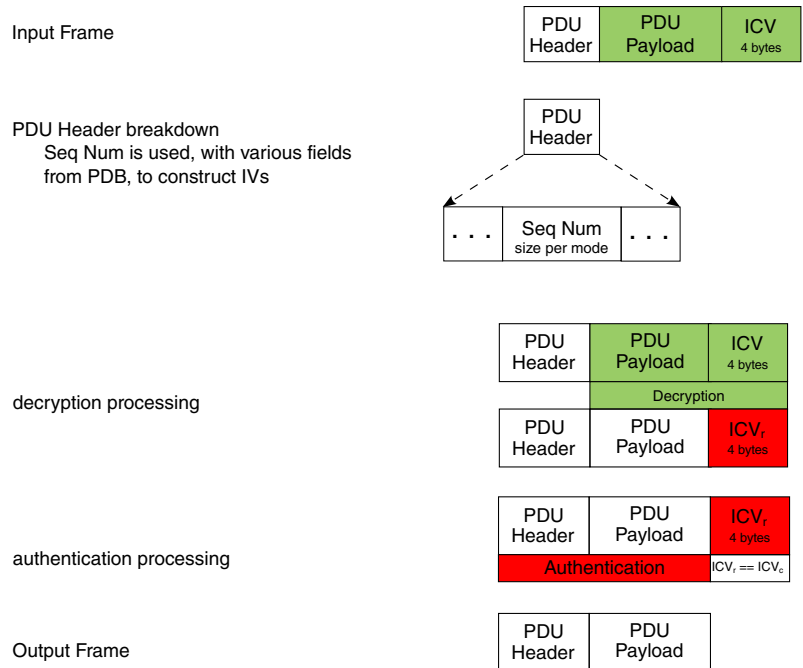


Figure 9-108. LTE PDCP PDU decapsulation for confidentiality and integrity

The decapsulation procedure is:

1. SEC receives an input frame that contains the PDU header, PDU payload, and ICV.
2. SEC extracts the sequence number from the PDU header.
3. SEC creates the confidentiality and integrity IVs for decapsulation as described in [LTE PDCP PDU IV generation](#); taking the two words from the PDB containing the Hyper Frame Number (HFN), Bearer, and Direction fields, and inserting in the sequence number that it extracted from the PDU Header.
4. After the Confidentiality IV has been written to the Class 1 Context Register and the integrity IV has been written to the Class 2 Context Register, cryptographic computations begin.
5. Just like for encapsulation, the PDU Header gets written to the data FIFO, to be processed by the selected Class 2 CHA.
6. The PDU Payload also gets written to the data FIFO, to be decrypted. The decrypted data is then passed to the selected Class 2 CHA for the integrity function.
7. After decryption, the received MAC (XMAC-I) is passed into the Class-2 CHA.
8. After computation of the integrity MAC-I (Message Authentication Check) value, it is compared to the XMAC-I value received from the input frame. If the two values are not identical, an error is returned.

The output frame consists of the unmodified PDU Header and the decrypted PDU payload. The XMAC-I is not part of the output frame.

9.10.6 LTE PDCP shared descriptor PDB format descriptions

Table 9-103. LTE PDCP shared descriptor PDB

Descriptor Header (1 or 2 words)		
PDB Word 0	Reserved (24 bits)	Options (8 bits)
PDB Word 1 (when 5-bit Sequence Number used)	HFN (27 bits)	Reserved (5 bits)
PDB Word 1 (when 7-bit Sequence Number used)	HFN (25 bits)	Reserved (7 bits)
PDB Word 1 (when 12-bit Sequence Number used)	HFN (20 bits)	Reserved (12 bits)
PDB Word 1 (when 15-bit Sequence Number used)	HFN (17 bits)	Reserved (15 bits)
PDB Word 2	Bearer, Dir, Reserved for CA & CE	
PDB Word 3	Threshold	

DECO updates PDB in descriptor buffer and external memory as needed

Table 9-104. LTE PDCP shared descriptor PDB, format of the options byte

7-3	2-1	0
Reserved	SNS	Reserved

Table 9-105. LTE PDCP shared descriptor PDB, description of the options byte

Field	Description
7-3	Reserved
2-1	Selects Serial Number Size. Ignored for LTE C-Plane.
SNS	If SNS = 00b : 12-bit Serial Number. If SNS = 01b : 7-bit Serial Number. If SNS = 10b : 15-bit Serial Number. If SNS = 11b : Reserved.
	Reserved

9.10.7 Overriding the PDB for LTE PDCP encapsulation and decapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For LTE PDCP PDU encapsulation and decapsulation, The HFN is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired HFN value in the job descriptor. For more information, see [Job Ring interface](#). When using the Queue Manager Interface, QI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field in the FD. For more information, see [Queue Manager Interface \(QI\)](#). When using the Advanced IO Processor interface, AI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the FRC field provided in the FD. For more information, see [Advanced IO Processor Interface \(AI\)](#).

Table 9-106. Format of the DPOVRD register when used with the LTE PDCP protocol

format when HFN is 17 bits	OVRD	Reserved (15 bits)								HFN (17 bits)																	
format when HFN is 20 bits	OVRD	Reserved (12 bits)							HFN (20 bits)																		
format when HFN is 25 bits	OVRD	Reserved (7 bits)				HFN (25 bits)																					
format when HFN is 27 bits	OVRD	Resvd (5 bits)			HFN (27 bits)																						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Chapter 10

Key agreement functions

The SEC protocol processing capabilities described in [Protocol acceleration](#) are centered on bulk data encryption and authentication. This section focuses on key agreement, which is another important part of protocol processing.

Certain protocols specify the use of a pseudo-random function (PRF) as a way that two parties can generate an identical pseudo-random byte string for use as a shared secret key. Internet Key Exchange (IKE) specifies a PRF for use in conjunction with IPsec. SSL and TLS have their own unique PRFs.

Several protocols also specify public key methods for generating or exchanging a shared secret key.

SEC implements the following key agreement methods as built-in functions:

- IKEv2 PRF
- SSL 3.0 PRF
- TLS 1.0, TLS 1.1, DTLS 1.0 PRF
- TLS 1.2, DTLS 1.2 PRF

10.1 IKEv2 PRF overview

The Internet Key Exchange v2 pseudo-random function requirements are covered in SEC by two separate but related functions.

- The IKE PRF function can be used to compute SKEYSEED per RFC 5996 section 2.13.
- The IKE PRF+ function can be used to compute IKE Security Association Keying Material per RFC 5996 section 2.14.
- The IKE PRF+ function can also be used to compute Child Security Association Keying Material per RFC 5996 section 2.17.

IKE PRF is a simply a Keyed MAC function -- either one of several selected HMACs based on MD5, SHA-1 or SHA-2; or based on AES-CMAC or AES-XCBC-MAC. As such, it generates a fixed length of material; at least 16 bytes (for MD5, AES-CMAC, and AES-XCBC-MAC); as many as 64 bytes when using SHA-512. IKE PRF+ is a well defined recursive use of PRF. It includes a single byte recursion count such that PRF+ is undefined past the 254th recursion. PRF+ can be used to generate a large pseudo-random byte string, suitable to be used for several keys and / or salting values. The PROTINFO field codes found in [Table 7-55](#) define the cipher suites to be used by the protocol, and SEC's built-in protocol processing sequences handle the remaining details. The protocol permits selected input parameters to be decrypted on the way in, and for results to be encrypted on the way out. By using the IKE PRF function, key material can be generated without allowing any other system resources to have access to unencrypted precursor material.

10.1.1 Using IKE PRF to generate SKEYSEED

IKEv2, as described in RFC 5996, specifies generation of a first master key material seed called SKEYSEED:

- SKEYSEED = prf(Ni | Nr, g^{ir}) for initial setup
- SKEYSEED = prf(SK_d (old), g^{ir} (new) | Ni | Nr) for rekeying

SKEYSEED is the master seed used by IKE PRF+ to generate IKE SA key material. One of the IKE SA keys generated is subsequently used to generate Child SA key material -- for example, the Child SA material may be for IPsec ESP.

In the initial setup computation of SKEYSEED, Ni and Nr are concatenated to form K -- the key string. Before concatenation, if PDB Option KOV is set, then Ni will be decrypted. The Diffie Hellman shared secret, g^{ir}, will be decrypted before use as string S if PDB Option IOV is set.

For the rekeying instance, SK_d will be decrypted if PDB Option KOV is set, and g^{ir} will be decrypted if PDB Option IOV is set. The unencrypted form of g^{ir} is then concatenated with Ni and Nr to form string S.

10.1.2 Using IKE PRF+ to generate keying material for the IKEv2 SA

The IKE PRF+ function can be used as part of IKEv2 to generate IKE key material. The form shown in RFC 5996, is

- $\{SK_d \mid SK_ai \mid SK_ar \mid SK_ei \mid SK_er \mid SK_pi \mid SK_pr\} = PRF+(SKEYSEED, Ni \mid Nr \mid SPIi \mid SPIr)$

In this case, the PRF+ function generates seven separate key blocks. SEC, can generate up to 8 outputs, any of which may be encrypted using the JDKEK. SKEYSEED and Ni may also have been encrypted; SKEYSEED will be decrypted using the JDKEK before use if PDB option KOV is set, and Ni will be decrypted using the JDKEK before use if PDB option IOV is set.

10.1.3 Using IKE PRF+ to generate Child SA key material

The IKE PRF+ function can be used as part of IKEv2 to generate Child SA key material. Several forms of this are shown in RFC 5996, including

- $KEYMAT = prf+(SK_d, Ni \mid Nr)$
- $KEYMAT = prf+(SK_d, g^{ir} \mid Ni \mid Nr)$

In both forms, SK_d may have been protected by SEC, using its JDKEK. PDB Option KOV controls whether or not to decrypt SK_d.

In the form with the Diffie-Hellman exchange result, g^{ir} may have been encrypted using the JDKEK. In the form that excludes the Diffie-Hellman exchange result, Ni might have been encrypted. PDB Option IOV Controls whether or not to decrypt the first segment of string S.

How string KEYMAT is sliced up into keys for child security associations is beyond the scope of RFC 5996. SEC supports splitting this KEYMAT by means of multiple FIFO STORE Commands into up to 8 separate locations as defined by the PDB Output Reference block.

In each case, the resulting PRF output material is passed to another protocol (such as IPsec) for further disposition or is used whole by another element of IKE.

10.1.4 Restrictions on programming control blocks

Note the following restrictions on the programming of the control blocks:

- The total length of the key control blocks may not exceed 2046 bytes.
- When not encrypted, the length of either key control block may not exceed 1023 bytes individually.
- Only the first referenced key may be encrypted, and the length of the first key control block when encrypted may not exceed 128 bytes.

- For input material, only the first reference may be encrypted; if encrypted, it is limited to 512 bytes.
- The first two unencrypted referenced inputs may not exceed a total of 512 bytes.
- The remaining three or four unencrypted references may not exceed a total of 512 bytes.

The length of each output segment is limited by the size of the output FIFO. In current designs this is 128 bytes. The total amount of output cannot exceed 512 bytes.

10.1.5 IKE PRF PDB format descriptions

Both IKE PRF and PRF+ use the exact same form of PDB. The PDB is used to specify each key block (up to 2), each input block (up to 6), and each output block (up to 8). Restrictions as to length and what may be encrypted are described in [Restrictions on programming control blocks](#). More detail is provided below as to how processing occurs. However, the general idea is that the two Key References can be used to supply up to two separately stored key segments. The two segments are concatenated together and used as a key to the underlying function. The key material fetched from the first address is decrypted before use if PDB option KOV is set. The key material from the second address is never decrypted. In RFC 5996, the resulting concatenated data is called "K."

Similarly, material is fetched from the locations specified by the up-to 6 Input References, the first of which will be decrypted if PDB option IOV is set. All of the fetched data is concatenated to form what RFC 5996 refers to as "S." For output, the PRF-generated string is chopped up into up to 8 different memory locations, as specified by the lengths found in the Output Reference Control Block and the addresses found in the Output Reference Block. Each of the 8 output references controls whether the string segment written to the associated location is encrypted using the KEK in AES-ECB, or using the KEK in AES-CCM.

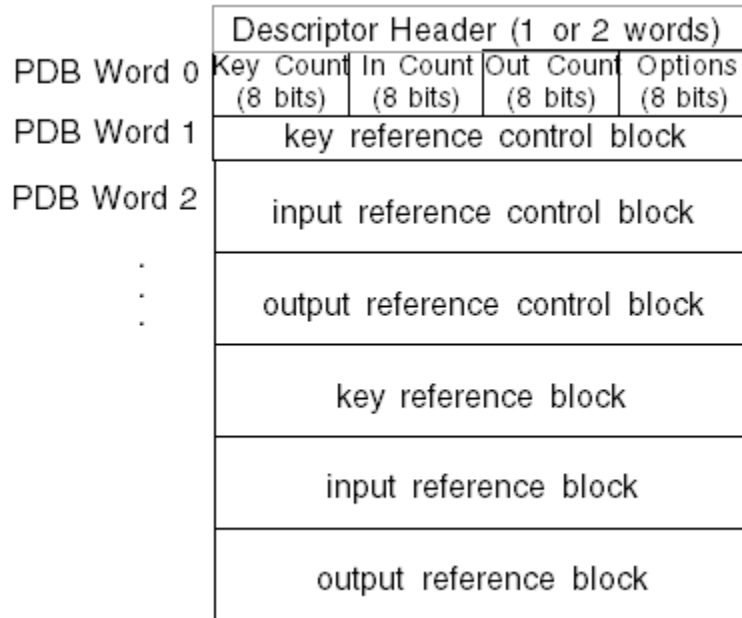


Figure 10-1. IKE PRF PDB

Table 10-1. IKE PRF PDB, description of the PDB

Field	Description
Key Count	Number of key inputs to the PRF function. Valid values are 1 and 2.
In Count	Number of inputs to the PRF function. Valid values are 1 - 6.
Out Count	Number of outputs from the PRF function. Valid values are 1 - 8.
Options	See Table 10-3
Key Reference Control Block	A series of 16-bit fields, called reference controls. There are two Key Reference Controls, one for each Key Reference below. If KOV=1 in the Options field, then the material referenced by the first pointer is considered encrypted, and will be decrypted before use. The ENC, SPLIT and OEKT fields in the Key Reference controls are reserved and must be zero.
Input Reference Control Block	A series of 16-bit fields, called Reference controls. There are 6 input reference controls, one for each Input reference. If IOV=1 in the Options field, then the first Input Reference has been encrypted. The ENC, SPLIT and OEKT fields in the Input Reference controls are reserved and must be zero.
Output Reference Control Block	A series of 16-bit fields, called reference controls. There are 8 output reference controls, one for each Output Reference.
Key Reference Block	Two pointer fields. The width of each field is determined by the PS field of the Master Configuration Register.
Input Reference Block	Six pointer fields. The width of each field is determined by the PS field of the Master Configuration Register.
Output Reference Block	Eight pointer fields. The width of each field is determined by the PS field of the Master Configuration Register.

Table 10-2. IKE PRF PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	IKEKT	IDEKT	KOV	IOV	Reserved

Table 10-3. IKE PRF PDB, description of the options byte

Field	Description
7-5	Reserved
4 IKEKT	Input Key Encryption Key Type. Note: this field is ignored if keys are not encrypted. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
3 IDEKT	Input Data Encryption Key Type. Note: this field is ignored if data is not encrypted. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
2 KOV	Key input override. If KOV=1, the data referenced by the <i>firstKey</i> Reference is treated as if it has been encrypted using the Job Descriptor Key Encryption Key. 0 Key or keys are not encrypted 1 Key Override -- the first referenced key is decrypted before use.
1 IOV	Input override. If 1, then the first reference is to a value that has been encrypted using the Job Descriptor Key Encryption Key. 0 No encrypted inputs 1 Input Override -- one encrypted input is referenced
0	Reserved

Table 10-4. IKE PRF PDB reference controls, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SGT	ENC	SPLIT	OEKT	Reserved		LENGTH									
Reference Control i															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SGT	ENC	SPLIT	OEKT	Reserved		LENGTH									
Reference Control i+1 (if necessary)															

Table 10-5. IKE PRF PDB reference controls - description of the fields

Field	Description
31 SGT	For Reference Control i, SGT specifies whether the pointer is a direct reference to the data or a pointer to a Scatter/Gather Table. 0 Direct reference to data 1 Reference to a Scatter/Gather Table
30 ENC	(output reference only) - For Reference Control i, ENC=1 specifies that the value is encrypted (with the Job Descriptor Key Encryption Key) before it is written to memory.
29 SPLIT	(output reference only) - For Reference Control i, SPLIT=1 specifies that the value is used as an HMAC key and is stored in IPAD/OPAD form. If SPLIT=1, then the HMAC key is encrypted regardless of the ENC value.
28 OEKT	(output reference only) - For Reference Control i, OEKT specifies the type of algorithm used for encrypting the output

Table continues on the next page...

Table 10-5. IKE PRF PDB reference controls - description of the fields (continued)

Field	Description
	0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
27-26	Reserved
25-16 LENGTH	For Reference Control i, LENGTH specifies the number of bytes of data when in plaintext form. Note that if ENC=1, the encrypted form of the data may be longer than the length specified here.
15-0 contain the same fields as bits 31-16, but are used for the next reference. Note that if there is an odd number of input references in the PDB, then the fields in the least-significant half of the PDB word are ignored.	
15 SGT	For Reference Control i+1, SGT specifies whether the pointer is a direct reference to the data or a pointer to a Scatter/Gather Table. 0 Direct reference to data 1 Reference to a Scatter/Gather Table
14 ENC	(output reference only) - For Reference Control i+1, ENC =1 specifies that the value is encrypted (with the Job Descriptor Key Encryption Key) before it is written to memory.
13 SPLIT	(output reference only) - For Reference Control i+1, SPLIT=1 specifies that the particular value is used as an HMAC key and is to be stored in IPAD/OPAD form. If SPLIT =1, then ENC is ignored.
12 OEKT	(output reference only) - For Reference Control i+1, OEKT specifies the type of algorithm used for encrypting the output 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
11-10	Reserved
9-0 LENGTH	For Reference Control i, LENGTH specifies the number of bytes of data when in plaintext form. Note that if ENC=1, the encrypted form of the data may be longer than the length specified here.

10.1.6 Implementation details for IKE PRF function

This figure shows the IKE PRF-material in system memory.

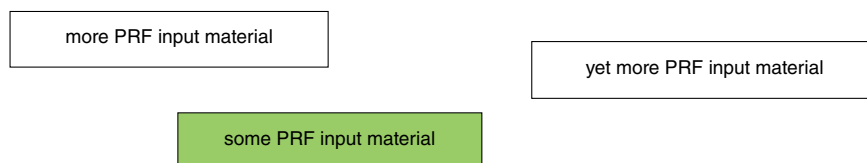


Figure 10-2. IKE PRF-material in system memory

The procedure is as follows:

1. SEC collects the key, as shown in the following figure.

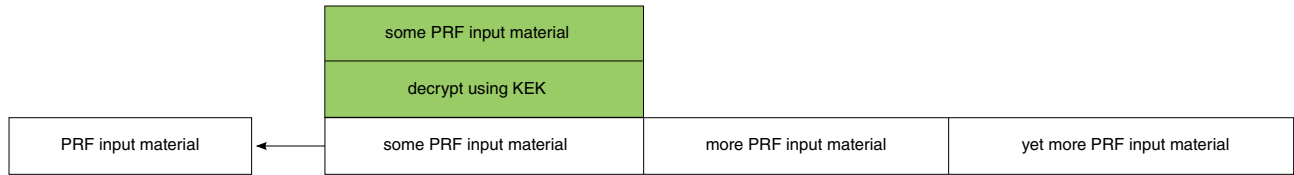


Figure 10-3. Collecting IKE PRF-material in SEC

2. If decryption is required, the decryption procedure is applied to the whole first segment.
3. For Key material, up to two segments may be fetched. RFC 5996 refers to the concatenated result as K.
4. The procedure is repeated for input material. Up to six segments may be fetched. RFC 5996 refers to the concatenated result as S.
5. A MAC of the resultant string is then generated.
 - For IKE, legitimate MAC functions include all HMACs supported by MDHA (except SHA-224), as well as AES-CMAC and AES-XCBC-MAC.

10.1.7 Implementation Details for IKE PRF+ function

The IKE PRF (Pseudo Random Function) Protocol is a method defined by RFC 5996 for generating session keys using cryptographic algorithms to create pseudo-random data. The SEC's built in protocol processing sequences handle most of the details, so the PROTINFO field codes enumerated in [Table 7-55](#) are used to define the ciphersuites to be used by the Protocol.

The IKE PRF+ is similar to the IKE PRF described in [Implementation details for IKE PRF function](#), with the additional step of adding a byte of counter value to the end of the input material for every recursion of the PRF (see the following figure).

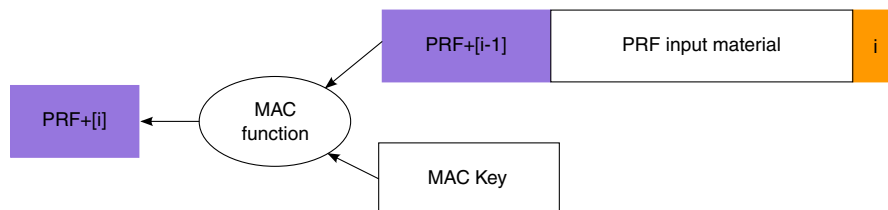


Figure 10-4. Recursive IKE PRF+ material generation

SEC uses PRF input material and the HMAC key in the same way as for the IKE PRF. However, it manages the byte counter *i*. Each PRF+ invocation starts with *i* = 01h; each iteration increments *i* by 1.

Valid IKE PRF+ underlying MAC functions include all HMACs supported by MDHA (except SHA-224), AES-CMAC, and AES-XCBC-MAC.

10.2 SSL/TLS/DTLS pseudo-random functions (PRF)

The TLS revisions have the following differences in how they define the PRFs.

- SSL 3.0 uses simple hash functions, relying on both MD5 and SHA-1.
- TLS 1.0, TLS 1.1, and DTLS use MD5 and SHA-1, but with HMAC enabled instead of a straight hash.
- TLS 1.2 drops the requirement for any particular hash function and does not combine the output of multiple hash functions to produce the PRF result.

Note that all SSL/TLS/DTLS PRFs use only hash functions supported by MDHA.

See [Table 7-54](#) for a description of the PROTINFO field as used with SSL and TLS PRF Commands.

10.2.1 SSL 3.0 PRF overview

This figure shows a functional diagram of SSL 3.0 PRF.

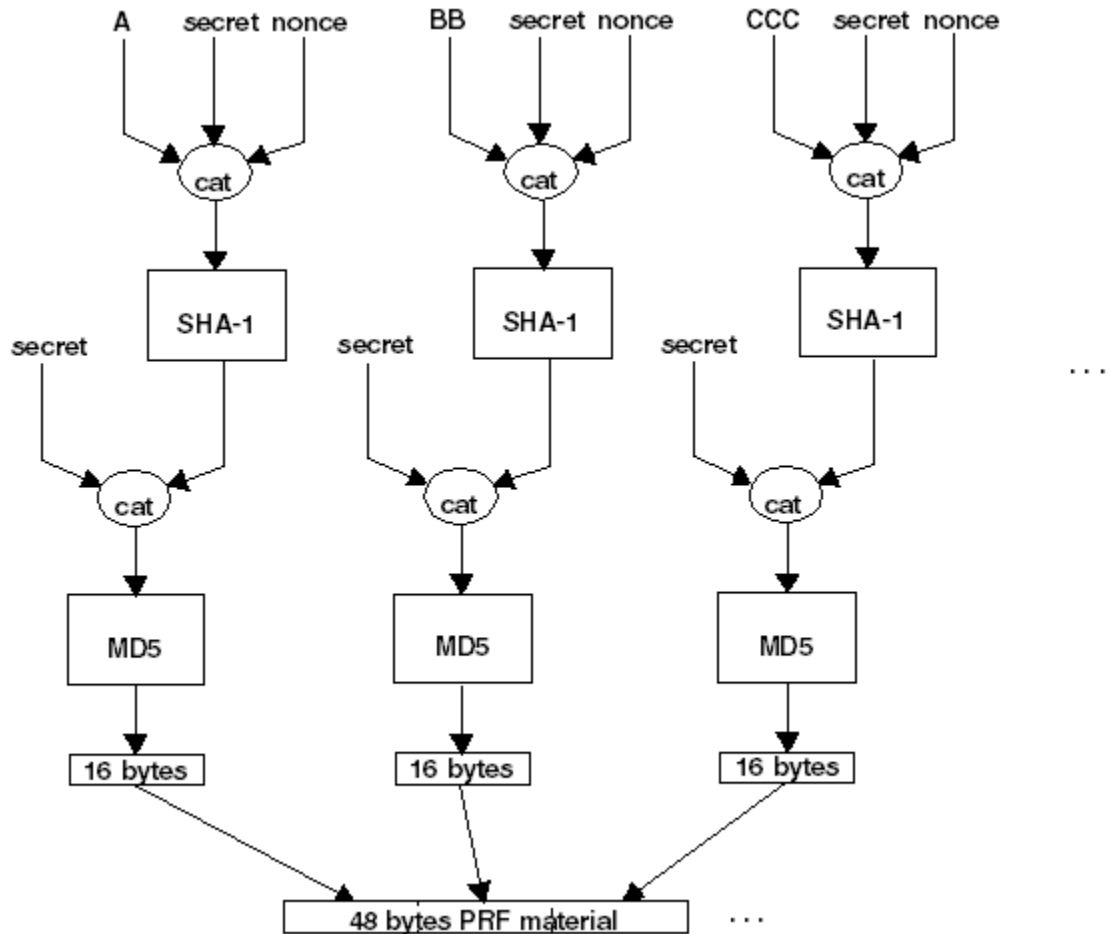


Figure 10-5. SSL 3.0 PRF

The SSL 3.0 PRF is referred to as an ad-hoc PRF because it does not use the typical constructs, such as an HMAC, used by other PRFs. Instead, the SSL 3.0 PRF uses straight MD5 and straight SHA-1 to produce PRF material.

10.2.1.1 SSL 3.0 PRF definitions

The secret's value is typically the master_secret, which is 48 bytes in length. If the secret's value is the premaster_secret, SSL has no explicit bounds on the length of the premaster_secret. However, SEC does not support a premaster_secret greater than 512 bytes in length.

The seed is composed of two values. Per standards, the values typically used are the 32-byte client_random and server_random values.

- When generating the master_secret value, seed1 is clientHello.random, and seed2 is serverHello.random.

- When generating key material, that is reversed: seed1 is serverHello.random and seed2 is clientHello.random.
- For FINISHED message generation, typical use is that seed1 is a 16-byte MD5 hash of all the handshake messages that the FINISHED message is covering, and seed2 is the 20-byte SHA-1 hash of all the same handshake messages the FINISHED message is covering. Note that the SSL 3.0 PRF function does not support the computation of those hashes.

Whatever the type of seed value, the descriptor PDB can either include the immediate values or can reference them in external memory.

10.2.2 Process for SSL 3.0 PRF

For every 16 bytes of PRF material that SEC needs to generate, SEC performs the following actions:

1. Concatenates an iteration constant, a secret, and a nonce
2. Produces a SHA-1 hash
3. Takes the same secret (from the concatenation step) and postpends the SHA-1 hash result to it
4. Produces an MD5 hash of that concatenation.

The descriptor provides the secret to SEC. The secret typically has already been encrypted with the key encryption key. The descriptor also provides the nonce to SEC, but the nonce is not considered sensitive.

SEC iterates to produce as many bytes of PRF material as required. The iteration constant for the first iteration is 41, which is ASCII for "A". Note that with each iteration, both the value and the number of instances of the iteration constant increments; that is the second iteration uses 42, 42 and the seventh iteration uses seven copies of the byte 47.

SEC divides an arbitrary number of bytes of SSL PRF material into up to six distinct memory locations, meaning that the SSL PRF descriptor can have between one and six output pointers for returning PRF material. Note the following:

- Each of these output pointers may be considered sensitive and require encryption of that memory location's PRF material using the key encryption key.

10.2.3 SSL 3.0 PRF PDB format descriptions

PRF for Master Secret or FINISHED message generation

	Descriptor Header (1 or 2 words)	
PDB Word 0	RSV (24 bits)	Options (8 bits)
PDB Word 1	Input Reference Control	
:	Output Reference Control	
:	input secret reference	
:	reserved reference	
:	Input Seed Part 1 reference	
:	Input Seed Part 2 reference	
:	master_secret/verify_data output reference	

PRF for key material generation

	Descriptor Header (1 or 2 words)	
PDB Word 0	RSV (24 bits)	Options (8 bits)
PDB Word 1	Input Reference Control	
:	Output Reference Control	
:	input master_secret reference	
:	reserved reference	
:	Input Seed Part 1 reference	
:	Input Seed Part 2 reference	
:	client-write MAC secret output reference	
:	server-write MAC secret output reference	
:	client-write key output reference	
:	server-write key output reference	
:	client-write IV output reference	
:	server-write IV output reference	

See the tables and figures that follow for a description of the fields in these PDBs.

Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the job descriptor and any shared descriptors. Exceeding that limit yields undesirable results.

Figure 10-6. SSL 3.0 PRF PDB

Table 10-6. SSL 3.0 PRF PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	IEKT	OEKT	IEOV	OEOV

Table 10-7. SSL 3.0 PRF PDB, description of the options byte

Field	Description
7-4	Reserved
3 IEKT	Input Encryption Key Type. Note this field applies only to encrypted inputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
2 OEKT	Output Encryption Key Type. Note this field applies only to encrypted outputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption

Table continues on the next page...

Table 10-7. SSL 3.0 PRF PDB, description of the options byte (continued)

Field	Description
1 IEOV	Input Encryption Override 1 Master-Secret input is not encrypted. 0 Master-Secret input is encrypted.
0 OEOV	Output Override default. Note this field is ignored if PROTINFO != FFFF or FFFE 1 If PROTINFO=FFFF or FFFE, the generated key material is not encrypted. 0 If PROTINFO=FFFF or FFFE, the generated key material is encrypted.

Table 10-8. SSL 3.0 PRF PDB input and output reference, format

Pointer (32-bit or 64-bit) (See Address Pointers)

Table 10-9. SSL 3.0 PRF PDB input and output reference, field descriptions

Field	Description
31-0 or 63-0 Pointer	Pointer to the reference. The size of this field is determined by the PS field of the Master Configuration Register. <ul style="list-style-type: none"> If 32 bit addresses are selected, 4 bytes are reserved for this field. If larger addresses are selected, 8 bytes are reserved for this field.

Table 10-10. SSL 3.0 PRF PDB input reference control, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	Input Secret Length										Reserved				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		Input Seed Part 1 Length							Input Seed Part 2 Length						

Table 10-11. SSL 3.0 PRF PDB input reference control, field descriptions

Field	Description
31	Reserved
30-21 Input Secret Length	Length of the input secret in bytes. NOTE: If the PRF material is split into keys, (if PROTINFO != FFFF or FFFE) the master_secret must be 48 bytes, and this field is ignored.
20-14	Reserved
13-7 Input Seed Part 1 Length	Length of the Input Seed Part 1 in bytes. Per protocol definitions, legitimate values are 16 or 32.
6-0 Input Seed Part 2 Length	Length of the Input Seed Part 2 in bytes. Per protocol definitions, legitimate values are 20 or 32.

Table 10-12. SSL 3.0 PRF PDB output reference control, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SGT	Reserved							LENGTH							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															

Table 10-13. SSL 3.0 PRF PDB output reference control, format

Field	Description
31 SGT	Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table. 0 Direct reference 1 Reference to a scatter/gather table
30-24	Reserved
23-16 LENGTH	Length of the data. Ignored if PROTINFO != FFFF or FFFE. In these cases the derived from the cipher suite determined by the PROTINFO field of the OPERATION Command. If PROTINFO = FFFF or FFFE, this field signals the length, in bytes, of the master_secret/verify_data output.
15-0	Reserved

10.2.4 TLS 1.0/TLS 1.1/DTLS PRF overview

This figure shows a functional diagram of TLS 1.0/TLS 1.1/DTLS PRF.

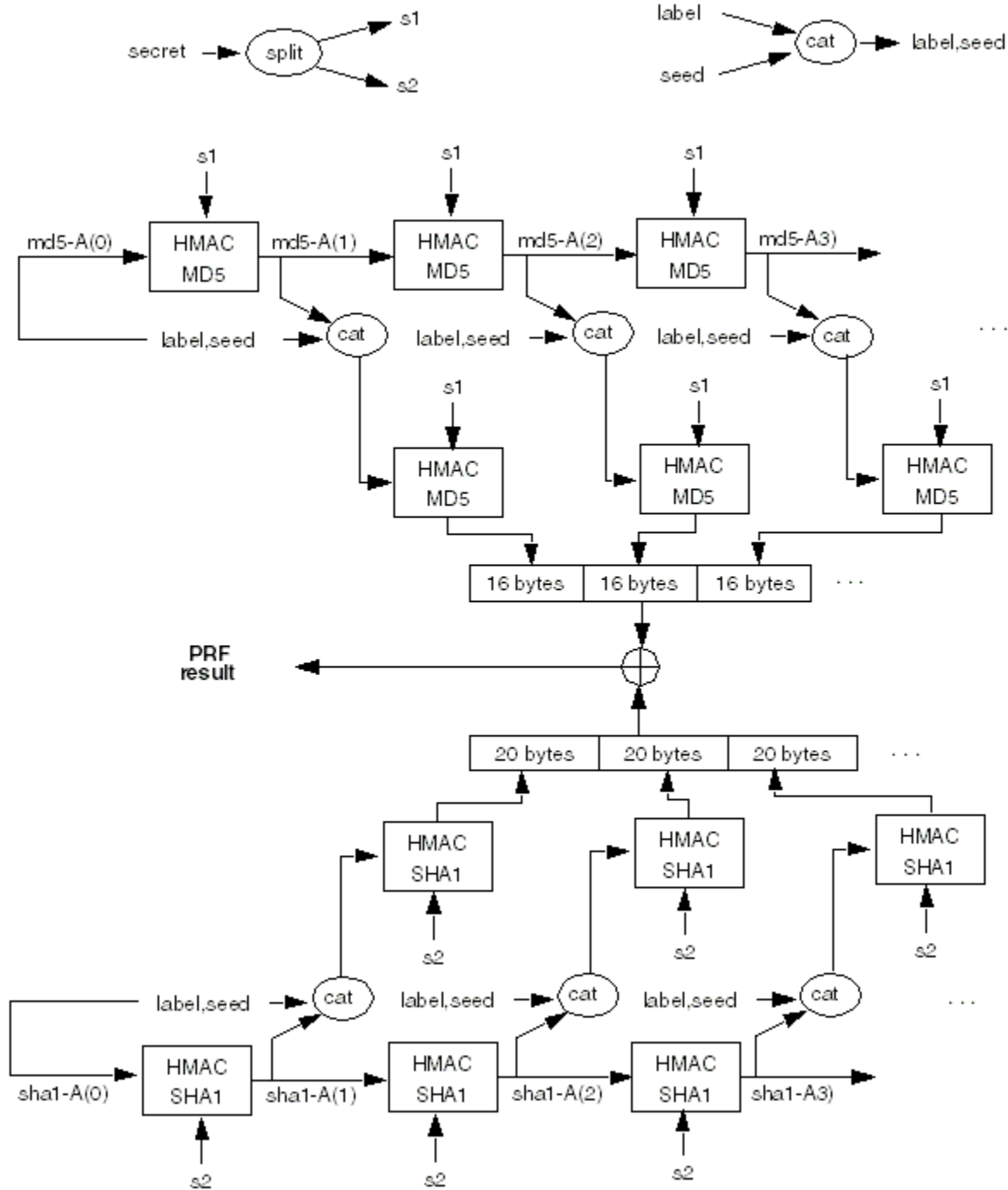


Figure 10-7. TLS 1.0/TLS 1.1/DTLS PRF

The TLS 1.0, TLS 1.1, and DTLS PRF uses both HMAC-MD5 and HMAC-SHA-1. SEC implements this PRF with a protocol descriptor that specifies the following inputs: secret, label, and seed. Any of those inputs may be considered sensitive and therefore decrypted using the key-encryption key prior to use.

The seed may be split across multiple memory segments, each individually referenced in the protocol descriptor. Each seed segment referenced in the descriptor is either entirely considered sensitive or not. Each seed segment may be further split in memory using a scatter/gather table. All material referred to by the table must be either completely sensitive or completely not sensitive.

10.2.4.1 TLS PRF RFC definitions

RFCs 2246 and 4346 define the TLS PRF as follows:

```
PRF(secret, label, seed) = P_MD5(S1, label + seed) XOR P_SHA-1(S2, label + seed);
P_MD5(secret, seed) = HMAC_MD5(secret, A(1) + seed) + HMAC_MD5(secret, A(2) + seed) + ...
P_SHA-1(secret, seed) = HMAC_SHA-1(secret, A(1) + seed) + HMAC_SHA-1(secret, A(2) + seed)
+ ...
```

Where for both P_MD5 and P_SHA-1,
A(0) = label + seed
A(i) = HMAC_hash (secret, A(i-1))

Note the following:

- The keyword seed is multiply defined, meaning that the value for seed in all subsequent lines takes the meaning that label + seed has in the first line.
- The + operator is defined to mean concatenation (see [Figure 10-7](#) for an illustration).
For example:
 - Generating anywhere between 41 and 48 bytes of PRF material requires four iterative executions of P_MD5 and four iterative executions of P_SHA-1, resulting in a total of 14 HMAC computations.
 - Generating between 33 and 40 bytes of PRF material requires four P_MD5 executions, but only three P_SHA-1 executions.
- The secret value is either the master_secret or the premaster_secret.
 - master_secret is typically used and is 48 bytes in length.
 - If the premaster_secret is used, TLS has no explicit bounds on the length, but SEC does not support a premaster_secret greater than 512 bytes in length.
 - If the premaster_secret is longer than 128 bytes, the split secrets s1 and s2 are each larger than the 64-byte maximum permissible HMAC key value, unless preprocessing is performed. SEC performs that preprocessing if required, hashing S1 using MD5 and hashing S2 using SHA-1.
- label is a byte string
 - All valid byte strings identified in standards are less than 16 bytes in length.
 - The descriptor PDB permits the byte string to be either an immediate value in the PDB or to be stored in external memory and be referenced by a pointer.
- The seed is composed of two values. per standards, typical use are the 32-byte client_random and server_random values.

- When generating the master_secret value, seed1 is clientHello.random, and seed2 is serverHello.random.
- When generating key material, that is reversed; seed1 is serverHello.random, and seed2 is clientHello.random.
- For FINISHED message generation, typical use is that seed1 is a 16-byte MD5 hash of all the handshake messages the FINISHED message is covering, and seed2 is the 20-byte SHA-1 hash of all the same handshake messages the FINISHED message is covering. Note that the TLS 1.0/ TLS 1.1/DTLS PRF function does not support the computation of those hashes.
- Whatever the type of seed value, the descriptor PDB can either include the immediate values or can reference them in external memory.

10.2.5 Process for TLS 1.0, TLS 1.1, DTLS PRF

SEC splits the secret into two equal portions, entitled s1 and s2.

- s1 is used as the HMAC key for all HMAC-MD5 computations.
- s2 is used as the HMAC key for all HMAC-SHA-1 computations.

If the secret is not even in length, then s1 and s2 both encompass the middle byte. For example, if the secret is 11 bytes:

- s1 and s2 are each 6 bytes in length.
- The middle byte of secret is the last byte of s1 and the first byte of s2.

SEC divides an arbitrary number of bytes of TLS PRF material into up to six distinct memory locations, meaning that the TLS PRF descriptor can have between one and six output pointers for returning PRF material. Note the following:

- Each of these output pointers may be considered sensitive and thus require encryption of that memory location's PRF material using the key encryption key.
- The PRF material generated for any of those memory locations may be prepared for use as an HMAC key.

10.2.5.1 How TLS uses PRF material

Typical TLS use of PRF material is to split it the following ways:

- Client-write MAC secret
- Server-write MAC secret
- Client-write key
- Server-write key

- Client-write IV
- Server-write IV

The MAC secrets and keys are sensitive, but the IVs are not.

The MAC secrets are encrypted after they have been expanded into HMAC IPAD and OPAD key material.

If the PRF is being used in this way, the PROTINFO field described in [Table 7-54](#) indicates two things:

- How the material is to be split—for example, HMAC-MD5 keys are 16 bytes long
- How the material is to be treated after it is split—for example, HMAC keys are turned into IPAD/OPAD split keys to maximize performance through the use of these keys

Per the PROTINFO field, keys are encrypted using the key encryption key before being stored in memory.

The PRF can also be used to generate a fixed length value. Per standards, that is done either when generating:

- The master_secret from the premaster_secret
- A FINISHED message

The result of generating the master_secret is a 48-byte value that is encrypted with the key encryption key. The result of generating a FINISHED message is a 12-byte value that is not encrypted with the key encryption key.

10.2.5.2 Concatenating input material into one input string (TLS 1.0/1.1/DTLS)

SEC concatenates all input material entitled label and seed into one input string that is persistent throughout the PRF computation.

Using the labels from [Figure 10-7](#) :

- md5-A(0) is the result of an HMAC-MD5 computation over the persistent input string
- sha1-A(0) is the result of an HMAC-SHA-1 computation over the persistent input string.
- md5-A(i) is the result of an HMAC-MD5 computation over md5-A(i-1)
- sha1-A(i) is the result of an HMAC-MD5 computation over sha1-A(i-1).

For every $\text{md5-A}(i)$ computed (except for $i = 0$), $\text{md5-A}(i)$ is also prepended to the persistent input string. An HMAC-MD5 computation is performed over the resultant string. This is performed for enough iterations of i to produce the required number of bytes of output.

A similar prepending of $\text{sha1-A}(i)$ to the persistent input string occurs. An HMAC-SHA-1 computation is performed, and the HMAC-SHA-1 procedure is repeated for enough iterations to produce the required number of bytes of output.

The iterative HMAC-SHA-1 results are concatenated together as are the iterative HMAC-MD5 results. Note that because SHA-1 produces 20-byte results and MD5 produces 16-byte results, there are likely to be more iterations on the MD5 side than on the SHA-1 side.

The PRF material results from performing an XOR of the concatenated MD5 output string with the concatenated SHA-1 output string, truncated to the required length.

10.2.6 TLS 1.0, TLS 1.1, DTLS PRF PDB format descriptions

The figure below illustrates the PDB format for the Master Secret, FINISHED and Key Material Generation forms for the TLS 1.0, TLS 1.1 and DTLS PRF protocols.

PRF for Master Secret or FINISHED Message Generation

Descriptor Header (1 or 2 words)	
PDB Word 0	reserved (24 bits) Options (8 bits)
PDB Word 1	Input Reference Control
PDB Word 2	Output Reference Control
.	Input Secret Reference
.	Input Label Reference
.	Input Seed Part 1 Reference
.	Input Seed Part 2 Reference
	Master_Secret/Verify_Data Output Reference

Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the Job Descriptor and any Shared Descriptor. Exceeding that limit yields undesirable results.

PRF for Key Material Generation

Descriptor Header (1 or 2 words)	
PDB Word 0	reserved (24 bits) Options (8 bits)
PDB Word 1	Input Reference Control
PDB Word 2	Output Reference Control
.	Input Secret Reference
.	Input Label Reference
.	Input Seed Part 1 Reference
.	Input Seed Part 2 Reference
	Client-Write MAC Secret Output Reference
	Server-Write MAC Secret Output Reference
	Client-Write Key Output Reference
	Server-Write Key Output Reference
	Client-Write IV Output Reference
	Server-Write IV Output Reference

See the tables below for a description of the fields in these PDBs.

Figure 10-8. TLS 1.0/TLS 1.1/DTLS PRF PDB
LS2088A Security (SEC) Reference Manual, Rev. 0, 04/2018

Table 10-14. TLS 1.0/TLS 1.1/DTLS PRF PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	IEKT	OEKT	IEOV	OEOV

Table 10-15. TLS 1.0/TLS 1.1/DTLS PRF PDB, description of the options byte

Field	Description
7-4	Reserved
3 IEKT	Input Encryption Key Type. Note this field applies only to encrypted inputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
2 OEKT	Output Encryption Key Type. Note this field applies only to encrypted outputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
1 IEOV	Input Encryption Override. 1 Master-Secret input is not encrypted 0 Master-Secret input is encrypted
0 OEOV	Output Encryption Override default. Note this field is ignored if PROTINFO != FFFF or FFFE 1 If PROTINFO = FFFF or FFFE, the generated key material is not encrypted. 0 If PROTINFO = FFFF or FFFE, the generated key material is encrypted.

Table 10-16. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, format

Pointer (32-bit or 64-bit) (see Address Pointers)

Table 10-17. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, field descriptions

Field	Description
Pointer	Pointer to the reference The size of this field is determined by the PS field of the Master Configuration Register. If 32 bit addresses are selected, 4 bytes are reserved for this field. If larger addresses are selected, 8 bytes are reserved for this field.

Table 10-18. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	Input Secret Length										Input Label Length				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Label Length (cont)		Input Seed Part 1 Length							Input Seed Part 2 Length						

Table 10-19. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, field descriptions

Field	Description
31	Reserved
30-21 Input Secret Length	Length of the Input Secret in bytes. NOTE: if the PRF material is split into keys, (if PROTINFO != FFFF or FFFE), the master_secret must be 48 bytes, and this field is ignored.
20-14 Input Label Length	Length of the Input Label in bytes. Per protocol definitions, legitimate values are between 11 and 15, inclusive.
13-7 Input Seed Part 1 Length	Length of the Input Seed Part 1 in bytes. Per protocol definitions, legitimate values are 16 or 32.
6-0 Input Seed Part 2 Length	Length of the Input Seed Part 2 in bytes. Per protocol definitions, legitimate values are 20 or 32.

Table 10-20. TLS 1.0 & 1.1/DTLS PRF PDB output reference control, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SGT	Reserved							LENGTH							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Reserved							

Table 10-21. TLS 1.0 & 1.1/DTLS PRF PDB output reference control, field descriptions

Field	Description
31 SGT	Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table. 0 Direct reference 1 Reference to a scatter/gather table
30-24	Reserved
23-16 LENGTH	Length of the data <ul style="list-style-type: none"> If PROTINFO = FFFF or FFFE, this field signals the length in bytes of the master_secret/verify_data output. This field is ignored if PROTINFO != FFFF or FFFE. In these cases the length is derived from the cipher suite determined by the PROTINFO field of the OPERATION Command.
15-0	Reserved

10.2.7 TLS 1.2 PRF overview

This figure is a functional diagram of TLS 1.2 PRF.

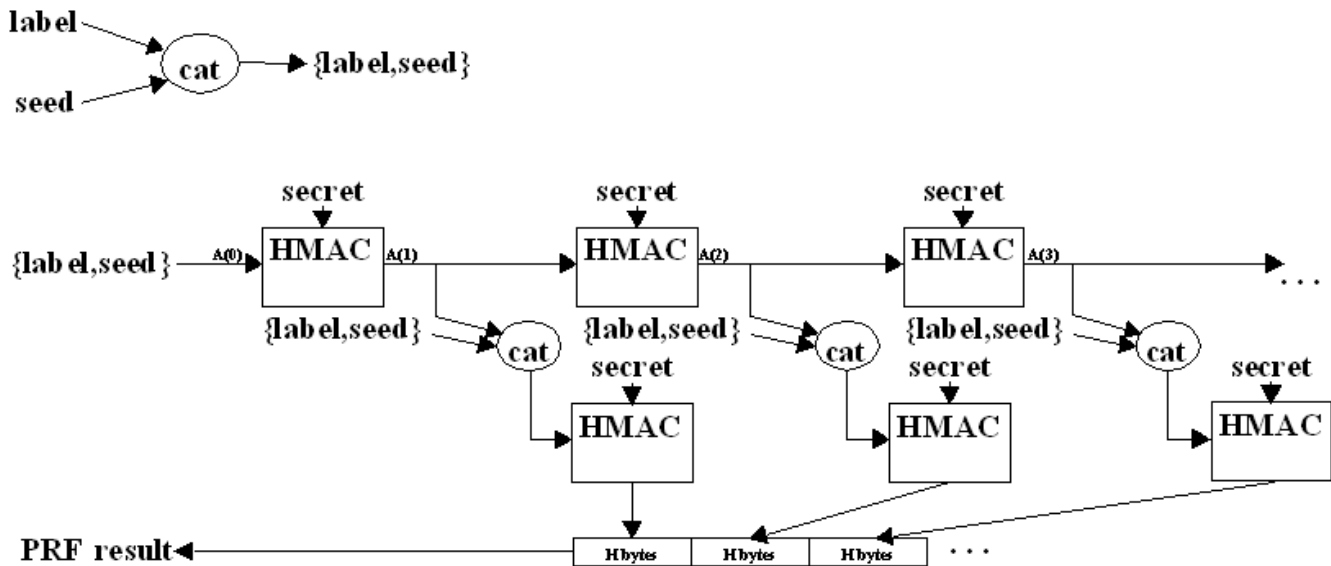


Figure 10-9. TLS 1.2 PRF

The TLS 1.2 PRF uses the same computational structure as the PRF for TLS 1.0, TLS 1.1, and DTLS, but it only uses one underlying hash function for performing the HMAC. The PRF is defined in [RFC 5246](#).

- For all cipher suites supported by TLS 1.1, HMAC-SHA-256 is the underlying hash function used.
- Cipher suites defined under TLS 1.2 are required to explicitly specify a PRF hash function.

Only cipher suites employing HMAC-SHA-256 and HMAC-SHA-384 for PRF are supported for encapsulation and decapsulation.

10.2.8 Process for TLS 1.2 PRF

SEC implements this PRF with a protocol descriptor that specifies the following inputs: secret, label, and seed. Any of those inputs may be considered sensitive and so be input in the form of Black Keys. If so, they are decrypted using the key encryption key prior to use.

The seed may be split across multiple memory segments, each referenced by pointers in the protocol descriptor. Each seed segment referenced in the descriptor is either entirely considered sensitive or not. Each seed segment may be further split in memory using a scatter/gather table. Individual segments in a scatter/gather table may not be considered sensitive: either all material referred to by the table is sensitive or it is all not sensitive.

The secret value can be either:

- The master_secret, which is 48 bytes in length and is the value typically used
- The premaster_secret

Although TLS has no explicit bounds on the length of the premaster_secret, SEC does not support a premaster_secret greater than 512 bytes in length. The maximum permissible HMAC key value is:

- 64 bytes for most HMAC algorithms
- 128 bytes for HMAC-SHA-384 and HMAC-SHA-512

If the premaster_secret is longer than this, preprocessing is performed. SEC performs the preprocessing if required, using the specified algorithm to hash the premaster_secret.

10.2.8.1 Concatenating input material into one input string (TLS 1.2)

SEC concatenates all input material entitled label and seed into one input string that is persistent throughout the PRF computation.

Referring to the labels in [Figure 8-1](#), there are two lines (or rows) of HMAC computations. The top row keeps producing an HMAC of the previous HMAC result, using secret as the key. That is, $A(i)$ is the result of performing an HMAC of $A(i-1)$. For each of these $A(i)$ values produced (except $A(0)$), the next D bytes of PRF is the result of performing an HMAC of the persistent input string with the appropriate $A(i)$ value prepended (where D is the size of the digest).

10.2.8.2 How TLS uses PRF material (TLS 1.2)

The SEC PRF functions are designed to generate an arbitrary number of bytes of PRF material into an arbitrary number of destinations. Each destination may be designated as secure key material, in which case the result is encrypted using the key encryption key.

Typical TLS use of PRF material is to split it the following ways:

- Client-write MAC secret
- Server-write MAC secret
- Client-write key
- Server-write key
- Client-write IV
- Server-write IV

The MAC secrets and keys are sensitive, but the IVs are not. Typically, the MAC secrets are encrypted after they have been expanded into HMAC IPAD and OPAD key material.

Note that AES-GCM does not make use of a MAC secret. As a result, if an AES-GCM cipher suite is selected, SEC skips the client-write and server-write MAC secrets, regardless of the length specified for each in their output references in the PDB.

Because the PRF is used for other purposes, including generation of a MAC across a series of messages, SEC supports splitting PRF material across between 1 and 8 unique destinations, each of which can enable or disable use of the key encryption key and each of which can enable or disable pre-preparation of HMAC material prior to encryption.

10.2.9 TLS 1.2 PRF PDB format descriptions

PRF for Master Secret or FINISHED message generation

	Descriptor Header (1 or 2 words)	
PDB Word 0	RSV (24 bits)	Options (8 bits)
PDB Word 1	Input Reference Control	
PDB Word 2	Output Reference Control	
	input secret reference	
	input label reference	
	Input Seed Part 1 reference	
	Input Seed Part 2 reference	
	master_secret/verify_data output reference	

PRF for key material generation

	Descriptor Header (1 or 2 words)	
PDB Word 0	RSV (24 bits)	Options (8 bits)
PDB Word 1	Input Reference Control	
PDB Word 2	Output Reference Control	
	input master_secret reference	
	input label reference	
	Input Seed Part 1 reference	
	Input Seed Part 2 reference	
	client-write MAC secret output reference	
	server-write MAC secret output reference	
	client-write key output reference	
	server-write key output reference	
	client-write IV output reference	
	server-write IV output reference	

See the tables and figures that follow for a description of the fields in these PDBs.

Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the Job Descriptor and any Shared Descriptor. Exceeding that limit yields undesirable results.

Figure 10-10. TLS 1.2 PRF PDB

Table 10-22. TLS 1.2 PRF PDB, format of the options byte

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	IEKT	OEKT	IEOV	OEOV

Table 10-23. TLS 1.2 PRF PDB, description of the options byte

Bits	Description
7-4	Reserved
3 IEKT	Input Encryption Key Type. Note that this field applies only to encrypted inputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
2 OEKT	Output Encryption Key Type. Note that this field applies only to encrypted outputs. 0 AES-ECB-256 encryption 1 AES-CCM-256 encryption
1 IEOV	Input Encryption Override 1 Master-Secret input is not encrypted 0 Master-Secret input is encrypted
0 OEOV	Output Encryption Override Default. Note this field is ignored if PROTINFO != FFFF or FFFE 1 If PROTINFO=FFFF or FFFE, the generated key material is not encrypted. 0 If PROTINFO=FFFF or FFFE, the generated key material is encrypted.

Table 10-24. TLS 1.2 PRF PDB input and output reference, format

Pointer (32-bit or 64-bit) (see Address Pointers)

Table 10-25. TLS 1.2 PRF PDB input and output reference, field descriptions

Field	Description
Pointer	Pointer to the reference. The size of this field is determined by the PS field of the Master Configuration Register. <ul style="list-style-type: none"> • If 32 bit addresses are selected, 4 bytes are reserved for this field. • If addresses larger than 32 bits are selected, 8 bytes are reserved for this field.

Table 10-26. TLS 1.2 PRF PDB input reference control, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	Input Secret Length										Input Label Length				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Label Length (cont)		Input Seed Part 1 Length							Input Seed Part 2 Length						

Table 10-27. TLS 1.2 PRF PDB input reference control, field descriptions

Field	Description
31	Reserved

Table continues on the next page...

Table 10-27. TLS 1.2 PRF PDB input reference control, field descriptions (continued)

Field	Description
30-21 Input Secret Length	Length of the input secret in bytes. NOTE: if the PRF material is split into keys, (if PROTINFO != FFFF or FFFE) then the master_secret must be 48 bytes, and this field is ignored.
20-14 Input Label Length	Length of the input label in bytes. Per protocol definitions, legitimate values are between 11 and 15, inclusive.
13-7 Input Seed Part 1 Length	Length of the Input Seed Part 1 in bytes. Per protocol definitions, legitimate values are 16 or 32.
6-0 Input Seed Part 2 Length	Length of the Input Seed Part 2 in bytes. Per protocol definitions, legitimate values are 20 or 32.

Table 10-28. TLS 1.2 PRF PDB output reference control, format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SGT	Reserved							LENGTH							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Reserved							

Table 10-29. TLS 1.2 PRF PDB output reference control, field descriptions

Field	Description
31 SGT	Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table. 0: Direct reference 1: Reference to a Scatter/Gather Table
30-24	Reserved
23-16 LENGTH	Length of the data. <ul style="list-style-type: none"> If PROTINFO = FFFF or FFFE, this field signals the length, in bytes, of the master_secret/verify_data output. If PROTINFO != FFFF or FFFE, this field is ignored. The length is derived from the cipher suite determined by the PROTINFO field of the OPERATION Command.
15-0	Reserved

10.3 Implementation of the derived key protocol

This protocol is available to assist with replacing a negotiated key with a derived form of that key. In particular, this protocol can be used for these tasks:

- Compute the IPAD/OPAD split key form of a HMAC key. [see [Using the MDHA Key Register with IPAD/OPAD "split keys"](#) for more information]

The use of the derived form of the key is mandatory for bulk-data protocols such as IPsec, where the use of the derived form provides a significant speed advantage.

The derived key protocol (DKP) is designed to allow a negotiated key to be replaced with the derived form in-place in a shared descriptor. For example, an IPsec descriptor can be written to supply an immediate HMAC key in negotiated form as a parameter to the DKP operation command. The DKP computes the IPAD/OPAD "split key" form, leaving the derived key in the Class 2 Key register, available for the subsequent IPsec command. Further, the DKP updates the descriptor, replacing the DKP operation command with the appropriate KEY command, and replacing the negotiated form of the key with the derived form of the key. It is the responsibility of the descriptor author to ensure the resulting derived key will not overwrite any descriptor commands that need to be kept.

10.3.1 Using DKP with HMAC keys

When used to generate HMAC keys, DKP receives an unprotected negotiated key and generates an unprotected derived key. If an encrypted split key is desired, or if an encrypted negotiated key is provided, see the FIFO STORE command and Output Data Types 16, 17, 26, 27 in [Table 7-30](#).

When generating derived HMAC keys (also known as "Split Keys"), the four-bit I/O control subfield of the PROTINFO field in the DKP Operation command is split in half; the upper 2 bits define the Input Source, and the lower two bits define the Output Destination. Not all combinations are valid.

Input Source - bits 16-17

Table 10-30. DKP input destination field

Setting	Description
00	IMM - negotiated key is in words immediately following the DKP Operation Command. This option can only be used with an Immediate Output Destination (OD=00).
01	SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command. This must be the choice when DKP is used in a trusted descriptor.
10	PTR - the input key is referenced by the address found immediately following the DKP Operation Command.
11	SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command.

Output Destination - bits 18-19

Table 10-31. DKP output destination field

Setting	Description
00	IMM - resulting derived HMAC "split key" will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting derived HMAC key is twice the underlying hash context length. See Table 10-32 Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source.
01	SEQ - the resulting derived HMAC "split key" will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source. This must be the choice when DKP is used in a trusted descriptor.
10	PTR - the resulting derived HMAC "split key" will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF.
11	SGF - the resulting derived HMAC "split key" will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR.

The twelve-bit length field designates the number of bytes the negotiated key takes. The length of the derived "split" key is determined by the underlying hash function chosen, as shown.

Table 10-32. HMAC derived key lengths

Hashing algorithm	Length of derived "split" key
MD5	32 bytes / 8 words
SHA-1	40 bytes / 10 words
SHA-224	64 bytes / 16 words
SHA-256	
SHA-384	128 bytes / 32 words
SHA-512	

10.3.2 Implementation of the Blob Protocol

The blob protocol provides a method for cryptographically protecting the confidentiality and integrity of user data across SoC power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the SoC is powered down. The key used to encrypt the blob is derived from a non-volatile master secret key so the blob can be decrypted when the SoC powers up again. More details on the Blob protocol can be found in section [Blobs](#)

Chapter 11

Cryptographic hardware accelerators (CHAs)

This section describes the functionality of each individual CHA used by the DECOs.

Table 11-1. Summary of cryptographic hardware accelerators (CHAs)

Definition	Abbreviation	What it implements	Cross-reference
Public-key hardware accelerator	PKHA	RSA, Diffie-Hellman, DSA, Elliptic-Curve Diffie-Hellman, Elliptic-Curve DSA	Public-key hardware accelerator (PKHA) functionality
Kasumi f8 and f9 hardware accelerator	KFHA	The Kasumi f8 encryption and Kasumi f9 authentication algorithms	Kasumi f8 and f9 hardware accelerator(KFHA) functionality
Data encryption standard accelerator	DESA	The DES and Triple-DES encryption algorithms	Data encryption standard accelerator (DES) functionality
Cyclic-redundancy check accelerator	CRCA	The coudble-CRC authentication algorithm	Cyclic-redundancy check accelerator (CRCA) functionality
Random number generator	RNG	A true hardware random number generator and a pseudo-random number generator	Random-number generator (RNG) functionality
SNOW 3G f8 accelerator	SNOWf8	The SNOW f8 encryption algorithm	SNOW 3G f8 accelerator functionality
SNOW 3G f9 accelerator	SNOWf9	The SNOW f9 authentication algorithm	SNOW 3G f9 accelerator functionality
Message-digest hardware accelerator	MDHA	The MD-5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512SHA-512/224, SHA-512/256 authentication algorithms	Message digest hardware accelerator (MDHA) functionality
AES accelerator	AESA	The AES encryption algorithm	AES accelerator (AESA) functionality

Table continues on the next page...

Table 11-1. Summary of cryptographic hardware accelerators (CHAs) (continued)

Definition	Abbreviation	What it implements	Cross-reference
ZUC encryption accelerator	ZUCE	The ZUC encryption algorithm	ZUC encryption accelerator (ZUCE) functionality
ZUC authentication accelerator	ZUCA	The ZUC authentication algorithm	ZUC authentication accelerator (ZUCA) functionality

11.1 Public-key hardware accelerator (PKHA) functionality

The PKHA module is capable of performing a number of different operations used in public-key cryptography, including modular arithmetic functions such as addition, subtraction, multiplication, exponentiation, reduction, squaring, cubing, simultaneous exponentiation, and inversion. All of these functions are provided in both integer and polynomial-binary field versions, except modular subtraction, which is the same as addition for binary polynomials. There are also elliptic-curve functions for point addition, point doubling, point validation, and point multiplication the standard prime and binary curves. Most of these functions can be performed timing-equalized to thwart timing-related side-channel attacks. PKHA also includes a Miller-Rabin primality test function for detecting prime numbers.

The PKHA internally performs modular multiply operations using "Montgomery multiplication". For efficiency, many of these functions have a variant which allows either inputs or outputs in Montgomery form. Some have variants to supply the Montgomery conversion factor. These save time over the variations without. Internally, the PKHA operates on digits of these values. Different versions of the PKHA may have a different digit size. This PKHA has a digit size of 128 bits. This has implications for the inputs and outputs of certain functions. See [the discussion on Montgomery arithmetic](#).

Because the numbers used in public-key cryptography are typically quite large and often referenced many times during a function, the inputs to PKHA are loaded into registers. PKHA has four of these labeled A, B, E, and N. A and B are for operands and results. E is for "keys", and N holds the modulus. For ECC functions, A and B are divided up into equal-size quadrants to accommodate the greater number of inputs required.

PKHA also has two other types of functions for manipulating the data in the registers. These are the Clear Memory and Copy Memory functions. The Clear Memory function allows all or any combination of the registers to be overwritten with zeros. The Copy Memory functions can be used to copy data from any of the A, B or N registers or register quadrant to any register A, B, E or N.

PKHA requires that all data for a given function be loaded before the Mode Register is written to invoke a function. This convention indicates to PKHA that all needed data has been loaded, and the function can now be launched. The typical procedure for executing a PKHA function is to use KEY and FIFO LOAD Commands to load the registers (usually N first), followed by an OPERATION Command to write the Mode Register, followed by one or more FIFO STORE Commands to store the result. PKHA functions may also be cascaded, so that the output of one function stays in a PKHA memory to become an input for the next function.

When loading or storing a value, it is important that its associated size register not change during the operation. To help avoid this issue when loading ECC parameters, make sure that all quadrants of a given register have the same size values by left-filling "short" values with zero. If a size register for a FIFOLoad command may change before it is complete, it is necessary to cause the Descriptor to stall until safe to proceed: insert a JUMP Command before offending command: JUMP jsl = 1 type = 0 cond = nifp offset = 1 (instruction 0xA1000401). In the other case, where a FIFOSTORE may still be in progress when a subsequent command will change the value in its size register, insert a SEQ FIFOSTORE Command before the offending command: SEQ FIFOSTORE length=0 (Instruction 0x68000000).

11.1.1 Modular math

Almost all math operations require with a modulus value in the N Memory. Math operations involving multiplication (multiplication, exponentiation, prime test, and ECC functions) are performed internally using Montgomery values.

11.1.2 About Montgomery values

The PKHA contains a Modular Arithmetic Unit. Multiplication is always modular multiplication:

$$A * B \text{ mod } N.$$

The PKHA performs this computation with a Montgomery multiplier. A Montgomery multiplier can be more efficient than a multiply-then-reduce calculation because the modular reduction is done as part of the multiplication and the working product never gets larger than the modulus. In a normal multiplication, the product, before reduction, would be the size of the sum of the factors, so usually twice the size of the modulus. The factors in a Montgomery multiplication each have an R factor, and, as part of the multiplication and modular reduction, one R is removed. Thus, the computation performed is:

$$(AR * BR) / R \bmod N.$$

The equivalent of "division by R " occurs even if one of the inputs does not have an R factor.

A number of PKHA functions accept inputs in Montgomery form instead of normal values. Some instead take $R^2 \bmod N$ as an input. These functions can be faster than their normal-value alternatives if several operations are performed in a row or if these values are known in advance. This is because, before being used, ($R^2 \bmod N$ needs to be computed and) normal values need to be converted internally to Montgomery form.

The Montgomery form of a value is $value * R \bmod N$, referred to here as *value*. The term $R = 2^{SD}$ is the Montgomery factor, where D is the digit size (of a digit in the PKHA arithmetic unit), in bits, and S is the minimum number of digits needed to hold the value in N . R is therefore dependent on N and D .

To use the PKHA to convert a normal value to a Montgomery value, one must first compute (or know) $R^2 \bmod N$, the Montgomery Conversion Factor. The following steps can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

$$R^2 = \text{MOD_R2}(N)$$

$$\underline{A} = \text{MOD_MUL_IM_OM}(A, B=R^2, N)$$

The equivalent F2M function can be used for binary polynomial values.

Eventually, the value needs to be converted out of Montgomery form. This can be done by performing another multiply (R^2 is not needed for this).

$$A = \text{MOD_MUL_IM_OM}(A=A, B=1, N)$$

Another method is to cause the PKHA to perform a multiplication and conversion to normal form. Internally, there are two multiplications: first the two inputs, then the product by one.

$$AB \bmod N = \text{MOD_MUL_IM}(A=A, B=B, N)$$

A third method is to have just one factor (either one) in Montgomery form:

$$AB \bmod N = \text{MOD_MUL_IM_OM}(A=A, B=B, N)$$

The following operations can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

The equivalent F2M functions can be used for binary polynomial values.

It is possible to add and subtract Montgomery values, if $R \bmod N$ is the same. Do not mix and match Montgomery and normal values for addition or subtraction. $5 + 3R \Rightarrow 5/R + 3R$ or $5/R + 3$; neither is likely the desired result.

11.1.3 Non-modular Math

Although addition, subtraction, and multiplication functions require a modulus, it is possible to perform these calculations without any reduction: the modulus must be larger than the expected result.

For addition and subtraction, this is easily done. For multiplication, the MOD_MUL function may be used, but it is not the most efficient, as internally first $R^2 \bmod N$ will be computed, then two multiplications will be performed (first to convert one factor into Montgomery, then to compute the product, not in Montgomery).

For non-modular multiplication, MUL_IM_OM is much more efficient, as only one multiplication will be performed. This can be used if the factors are not in Montgomery form, i.e., if the product to be calculated is $A*B$ instead of $A*B \bmod N$. Since the multiplier always "divides by R ", a special modulus value in Nram is required which will make R have the value 1. This is done by creating a modulus $N = R-1$ so that $R \bmod N$ will have the value one. This way, normal values are the same as Montgomery values; no conversion is necessary and the multiplier will quietly "divide by one" to no effect.

As an example, on a PKHA with a digit size of 32 bits and a product which will be no more than six bytes long, $R = 2^{SD} = 2^{2*32} = 2^{64}$. Therefore the modulus must be 0xFFFFFFFFFFFFFFFF.

For computation with binary polynomials, the equivalent F2M functions may be used.

11.1.4 Elliptic-Curve Math

The PKHA provides point math operations on different types of elliptic curves. These include the ability to add two points (+ operator), double a point, and multiply a point by an integer (scalar) value (x operator).

The input points are assumed to be valid points on the curve. If non-point coordinates or invalid curve parameters are used an input, then a non-point set of coordinates are likely to be returned as output. The "ECC Point Check" functions may be used to verify that a point's (x,y) values constitute a point which satisfies the equation for the curve.

The minimum modulus is 1 byte. The maximum modulus is 1024 bits, 128 in length, or one quadrant.

If ∞ is the Point at Infinity, P and Q are points on the curve, and j and k are integers, then the following identities, as well as others easily derived by taking advantage of associative and commutative properties, hold:

- $P + Q = Q + P$
- $P = \infty + P$
- $\infty = 0 \times P$
- $(j + k)P = (j \times P) + (k \times P)$

There may be times when the negative of a point is necessary:

- When subtracting points $P_A - P_B$
- When multiplying by a negative integer: $-\text{abs}(k) \times P_A$

To subtract, one can negate the second term and perform an addition, i.e.

$$P_C = P_A - P_B = P_A + (-P_B)$$

When multiplying by a negative value, one can either negate the starting point or the ending point. The multiplication value is the absolute value of the scalar, i.e., when k is negative

$$P_C = k \times P_A = -\text{abs}(k) \times P_A = \text{abs}(k) \times (-1P_A) = -(\text{abs}(k) \times P_A)$$

11.1.4.1 ECC_MOD: Point math on a standard curve over a prime field (F_p)

The ECC_MOD family of functions perform Add, Double, and scalar Multiply operations on points on a curve defined by the short Weierstrass equation:

$$E: y^2 = x^3 + ax + b \pmod{p}$$

where p is the a prime integer > 3 . These operations are available in Affine Coordinates (x,y) .

The modulus (value in N memory) for these operations is p , also referred to as q .

The equality for the negative of a point P , in affine coordinates, is $-P = -(x,y) = (x, -y)$

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, depends upon the type of curve and the value of the b term of the curve's equation:

- Where b is equal to 0: (0, 1)
- Where b is not equal to 0: (0, 0)

11.1.4.2 ECC_F2M: Point math on a standard curve over a binary field (F_{2^m})

The ECC_F2M family of functions perform add, double, and scalar multiply operations on points on a curve defined by the short Weierstrass equation:

$$E: y^2 + xy = x^3 + ax^2 + b$$

These operations are available in Affine Coordinates (x,y) . All inputs and output values of polynomial values are in polynomial basis. For example, x^5+x+1 is represented as 23h

The modulus (value in N memory) for these functions is q , the field-defining irreducible polynomial for the curve. Other documents use other symbols, including $p(t)$, $f(t)$, and f .

The equality for the negative of a point, in affine coordinates, is $-P = -(x,y) = (x, x+y)$.

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

Because of the way the point operations are performed over a binary field, these functions require as an input the value c rather than b . The relationship between these two values is:

$$b = c^4 \bmod q$$

and

$$c = b^{2^{m-2}} \bmod q, \text{ where } m \text{ is the degree (the power of its highest-power term) of } q.$$

This c value is referred to as b' in the ECC Public-Key protocols for ECDSA sign, and so on. The calculation of c is expensive, so it is obviously an advantage to calculate it only once or have it precomputed. See [Special values for common ECC domains](#) for these values for common ECC domains.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, is: (0, 0).

11.1.5 PKHA Mode Register

The formats of the PKHA Mode Register are described in detail in [PKHA OPERATION command](#).

The following tables list the valid PKHA_MODE values for all PKHA functions:

PKHA Clear Memory Functions: [Table 11-4](#)

PKHA Modular Arithmetic functions: [Table 7-74](#)

PKHA Elliptic Curve functions: [PKHA OPERATION: Elliptic Curve Functions](#)

PKHA Elliptic Curve functions: [PKHA OPERATION: Elliptic Curve Functions](#)

PKHA Copy Memory functions: [Table 7-79](#)

NOTE

Use of any PKHA_MODE value not listed in these tables results in an invalid mode error.

11.1.6 PKHA functions

The various PKHA functions are described in the following subsections. The following information applies to all PKHA functions.

- Mode Register bits that may be either 1's or 0's for the given function are represented with x .
- For convenience, in all the descriptions below the output is shown as the default B, although the actual output destination can be specified for most functions via the Class 1 Mode Register[OutSel] field to be either the B RAM or the A RAM.
- For each PKHA function, the specified mode bits are in the Class 1 Mode Register[PKHA_MODE_LS] field.
- For all of the PKHA functions, the Class 1 Mode Register[PKHA_MODE_MS] field is set to 8h.
- The descriptions specify the output register(s) and any other registers that might be modified. Note that the default output register is still modified but the output is placed into the specified destination register(s).
- Note that any parameter underlined is in Montgomery form (for example, $\underline{A} = AR \text{ mod } N$).
- Errors reported by PKHA are written to the Job Ring Output Status Register and termination status word ([Job termination status/error codes](#)). They are encoded in the ERRID field.
- Three flags in the CCB Status Register may be set by PKHA: PIZ, PIO, and PRM. These flags can be tested by the [JUMP \(HALT\) command](#). is set to indicate that

PKHA generated a result equal to zero, or, in the case of ECC functions, the point at infinity.

- PIO is set whenever a GCD routine finds that the Greatest Common Denominator of two numbers is the number 1. For other general non-ECC functions, it means that the result is equal to one. This may also be referred to as the GCD flag.
- PRM is set by the PRIME_TEST routine if it finds that a candidate integer is probably prime (that is, passes the Miller-Rabin primality test).
- It is important to note that the PKHA mathematical functions work in terms of "digits"; that is, the arithmetic unit is pipelined to work on a digit of data at a time. For PKHA-32 a digit = 32 bits (4 bytes) of data, for PKHA-64 a digit = 64 bits (8 bytes), and for PKHA-128 a digit = 128 bits (16 bytes). Therefore, the term 'digit' refers to 32, 64, or 128 bits of data in the input and/or output values used by the PKHA arithmetic unit.

11.1.6.1 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)

These functions copy data from a PKHA register (or register quadrant) specified as a source, to another PKHA register (or register quadrant) specified as a destination. COPY_NSZ copies the amount of data specified by the N Size register. COPY_SSZ copies the amount of data specified in the source register's size register. The source and destination are specified in the Mode Value. The source can be A, B or N. The destination can be A, B, E or N, but not the same as the source.

In a quadrant copy, when NSZ/SSZ exceeds the length of a quadrant, the copy will carry on into the next (higher-numbered) quadrant(s).

When the copy operation has completed, the destination register's size register will be updated to contain the number of bytes copied.

Table 11-2. COPY_NSZ and COPY_SSZ function properties

Property	Notes				
Mode value	Bits 19:17	Bits 16,11,10	Bits 9:8	Bits 7:6	Bits 5:0
	Source Register	Destination Register	Source Segment	Destination Segment	Function Code
	000 = A Register	000 = A Register	00 = Segment 0	00 = Segment 0	01_0000 = Copy_NSZ 01_0001 = Copy_SSZ
	001 = B Register	001 = B Register	01 = Segment 1	01 = Segment 1	
	011 = N Register	011 = N Register	10 = Segment 2	10 = Segment 2	
		010 = E Register	11 = Segment 3	11 = Segment 3	
other values reserved	other values reserved				

Table continues on the next page...

Table 11-2. COPY_NSZ and COPY_SSZ function properties (continued)

Property	Notes
	1. If the destination register is E, the source and destination segments must be 00b.
Input	None
Output	None
Requirements	For Copy_NSZ, the N-size Register must contain a valid value. For Copy_SSZ, the source register's size register must contain a valid value.
Side effects	The destination register's size register is updated to the number of bytes copied.
Errors reported	None
Flags set	None

1. If the destination register is E, the source and destination segments must be 00b.

11.1.6.2 Clear Memory (CLEAR_MEMORY) function

This function clears the specified registers or quadrants of registers in the PKHA. This includes the A, B, N and E. All registers or quadrants of registers are written with zeros.

A detailed description may be found in [PKHA OPERATION: clear memory function](#).

Table 11-3. CLEAR_MEMORY function properties

Property	Notes
Mode value	<i>ABEN_0000_00 QQ_QQ 00_0001</i> , with the following restrictions on the combinations of ABEN and QQQQ: At least one of ABEN must be on. If E is on, all Q must be zero. Some example encodings are in the table below.
Input	None
Output	A = 0, B = 0, E = 0, N = 0, or some quadrant(s) thereof, as specified by ABEN and QQQQ. Each Q specifies a quadrant, in order from 3 through 0.
Requirements	The Mode Register specifies which registers or quadrants of registers to clear. If no quadrants are selected, then all quadrants of the specified register(s) are cleared.
Side effects	None
Errors reported	Invalid Mode, if no registers are selected, or E with one or more quadrants is selected
Flags set	None

Table 11-4. Example mode values for PKHA clear memory functions

Function name	Register selects				Quadrant selects				Brief description	Bits 19-0, including PKHA_MODE and reserved bits ¹ (Hex)
	A	B	E	N	3	2	1	0		
CLEAR_MEMORY	1	1	1	1	0	0	0	0	Clear registers A, B, E, N	F0001
	1	1	1	0	0	0	0	0	Clear registers A, B, E	E0001
	1	1	0	1	0	0	0	0	Clear registers A, B, N	D0001
	1	0	1	0	0	0	0	0	Clear registers A, E	A0001
	1	0	0	1	0	0	0	0	Clear registers A, N	90001
	0	1	0	1	0	0	0	0	Clear registers B, N	50001
	0	1	0	0	0	0	0	0	Clear register B	40001
	0	0	1	0	0	0	0	0	Clear register E	20001
	0	0	0	1	0	0	0	0	Clear register N	10001
	1	0	0	0	1	1	0	0	Clear quadrants 2 and 3 of register A	80301
	0	1	0	1	0	0	0	1	Clear quadrant 0 of registers B, N	50041
	1	1	0	0	1	0	0	0	Clear quadrant 3 of registers A, B	C0201

1. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

11.1.6.3 Arithmetic Functions

11.1.6.3.1 Integer Modular Addition (MOD_ADD) function

Table 11-5. MOD_ADD function properties

Property	Notes
Mode value	0000_0000_0000_0000_0010 (output placed in B) 0000_0000_0001_0000_0010 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus and data size, any integer A = first addend, any integer less than N B = second addend, any integer less than N
Output	B (or A, if selected) = (A + B) mod N
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and B are each < N.
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N.

Table continues on the next page...

Table 11-5. MOD_ADD function properties (continued)

Property	Notes
	If $(A + B) \geq N$, N will be subtracted just once from the sum. That is, if $(A + B) \geq 2N$, then the result will not be mod N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.2 Integer Modular Subtraction (MOD_SUB_1) function

Modular subtraction can be described as follows. If $A \geq B$ or $A = B = 0$, then $B = A - B$. Otherwise, if $A < B$, then $B = A + N - B$. The result is always positive and less than N.

Table 11-6. MOD_SUB_1 function properties

Property	Notes
Mode value	0000_0000_0000_0000_0011 (output placed in B) 0000_0000_0001_0000_0011 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any integer A = minuend, any integer less than N B = subtrahend, any integer less than N
Output	B (or A, if selected) = $(A - B) \bmod N$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and B are less than N.
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.3 Integer Modular Subtraction (MOD_SUB_2) function

Table 11-7. MOD_SUB_2 function properties

Property	Notes
Mode value	0000_0000_0000_0000_0100 (output placed in B) 0000_0000_0001_0000_0100 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any integer B = minuend, any integer less than or equal to N A = subtrahend, any integer less than or equal to N
Output	B (or A, if selected) = $(B - A) \bmod N$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte

Table continues on the next page...

Table 11-7. MOD_SUB_2 function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> Maximum modulus size = 512 bytes A and B are < N
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.4 Integer Modular Multiplication (MOD_MUL)

The $(AB) \bmod N$ computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, MOD_MUL_IM or MOD_MUL_IM_OM) are more efficient. This function first computes $R^2 \bmod N$, then multiplies one factor to produce AR , then multiplies $AR*B$ to produce AB .

Table 11-8. MOD_MUL function properties

Property	Notes
Mode value	0000_0000_0000_0000_0101 (output placed in B) 0000_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any odd integer A = multiplicand, any integer less than N B = multiplier, any integer less than N
Output	B (or A, if selected) = $(AxB) \bmod N$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and B are < N.
Side effects	A and E are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N. Divide-By-Zero Error is set if the most significant digit of the modulus is all zeros.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.5 Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM)

This function takes its inputs, integers, in Montgomery form, multiplies them modulo the value in the N register and returns the result as a field value. To do this, it performs two multiplications: $AR*BR \Rightarrow ABR$ and $ABR*1 \Rightarrow AB$.

Table 11-9. MOD_MUL_IM function properties

Property	Notes
Mode value	1000_0000_0000_0000_0101 (output placed in B) 1000_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = multiplicand, a value in Montgomery form • B = multiplier, a value in Montgomery form
Output	B (or A if selected) = $A \times B \bmod N$, the non-Montgomery product of the inputs
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are less than modulus N
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.6 Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function

This function performs the calculation $A*B/R \bmod N$, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is $R^2 \bmod N$, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

Table 11-10. MOD_MUL_IM_OM function properties

Property	Notes
Mode value	1100_0000_0000_0000_0101 (output placed in B)

Table continues on the next page...

Table 11-10. MOD_MUL_IM_OM function properties (continued)

Property	Notes
	1100_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = multiplicand, a value in Montgomery format $0 \leq A < N$ • B = multiplier, a value in Montgomery format $0 \leq B < N$
Output	B (or A, if selected) = (AxB) mod N
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero.

11.1.6.3.7 Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ)

This function is commonly used to perform a single-step RSA operation. It computes $R^2 \text{ mod } N$ and converts A to Montgomery form before beginning the exponentiation. MOD_EXP_TEQ performs the same operation as MOD_EXP but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_TEQ, for a given modulus and size of exponent, is constant. In general MOD_EXP will run faster than MOD_EXP_TEQ, but will never run slower.

Table 11-11. MOD_EXP and MOD_EXP_TEQ function properties

Property	Notes
Mode value for MOD_EXP	0000_0000_0000_0000_0110 (output placed in B) 0000_0000_0001_0000_0110 (output placed in A)
Mode value for MOD_EXP_TEQ	0000_0000_0100_0000_0110 (output placed in B) 0000_0000_0101_0000_0110 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = an integer $0 \leq A < N$ • E = exponent, any integer
Output	B (or A, if selected) = (A ^E) mod N, a an integer $0 \leq A < N$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key (exponent) size = 512> bytes • A < N
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even.

Table continues on the next page...

Table 11-11. MOD_EXP and MOD_EXP_TEQ function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> Key Size Error is set if size of E = 0 or size of E > 512. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero.

11.1.6.3.8 Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function

This function is commonly used to perform a single-step RSA operation. It computes $\underline{A} = AR \pmod{N}$, where R is the Montgomery constant). The input data (base) to be exponentiated must be provided in the Montgomery form. The result will be returned in normal integer (non-Montgomery) representation. MOD_EXP_IM_TEQ performs the same operation as MOD_EXP_IM but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_IM_TEQ is constant for a given modulus and size of exponent. In general MOD_EXP_IM will run faster than MOD_EXP_IM_TEQ, but will never run slower.

Table 11-12. MOD_EXP_IM and MOD_EXP_IM_TEQ function properties

Property	Notes
Mode value for MOD_EXP_IM	1000_0000_0000_0000_0110 (output placed in B) 1000_0000_0001_0000_0110 (output placed in A)
Mode value for MOD_EXP_IM_TEQ	1000_0000_0100_0000_0110 (output placed in B) 1000_0000_0101_0000_0110 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any odd integer A = a value $0 \leq A < N$, in Montgomery form E = exponent, any integer (normal integer representation)
Output	B (or A, if selected) = $(A^E) \pmod{N}$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes Maximum key (exponent) size = 512 bytes $A < N$
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. Key Size Error is set if size of E = 0 or size of E > 512. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.9 Integer Simultaneous Modular Exponentiation (MOD_SML_EXP)

MOD_SML_EXP performs two modular exponentiations and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

Table 11-13. MOD_SML_EXP function properties

Property	Notes
Mode value	0000_0000_0000_0001_0110 (output placed in B) 0000_0000_0001_0001_0110 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A0 = an integer < N, first base • E = an integer, first exponent • A2 = an integer < N, second base • B = an integer, second exponent
Output	B (or A if selected) = $A0^E * A2^B \text{ mod } N$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 1/2 RAM size • Maximum key (exponent) size and B size = 1/4 RAM size • $A0 < N$ • $A2 < N$ • The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N
Side effects	A, B, and E are modified.
Errors reported	<ul style="list-style-type: none"> • N size error is set if N size > half RAM • A size error is set if A size > half RAM • B size error is set if B size > quarter-RAM • E size error is set if E size > quarter-RAM
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.10 Integer Modular Square (MOD_SQR and MOD_SQR_TEQ)

This function may be used to square an integer value. MOD_SQR_TEQ, the timing equalized version, will take the same time to complete for a given modulus. In general the MOD_SQR version will run faster than MOD_SQR_TEQ, but will never run slower.

Table 11-14. MOD_SQR and MOD_SQR_EXP function properties

Property	Notes
Mode value for MOD_SQR	0000_0000_0000_0001_1010 (output placed in B) 0000_0000_0001_0001_1010 (output placed in A)
Mode value for MOD_SQR_TEQ	0000_0000_0100_0001_1010 (output placed in B) 0000_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = a field element
Output	B (or A , if selected) = $(A * A) \text{ mod } N$

Table continues on the next page...

Table 11-14. MOD_SQR and MOD_SQR_EXP function properties (continued)

Property	Notes
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • $A < N$
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.11 Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_TEQ will take the same time to complete for any value of A . In general MOD_SQR_IM will run faster than MOD_SQR_IM_TEQ, and will never run slower.

Table 11-15. MOD_SQR_IM and MOD_SQR_IM_TEQ function properties

Property	Notes
Mode value for MOD_SQR_IM	1000_0000_0000_0001_1010 (output placed in B) 1000_0000_0001_0001_1010 (output placed in A)
Mode value for MOD_SQR_IM_TEQ	1000_0000_0100_0001_1010 (output placed in B) 1000_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = an integer $< N$ in Montgomery form
Output	B (or A , if selected) = $(A * A) \bmod N$, an integer
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • $A < N$
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.12 Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_OM_TEQ will take the same time to complete for any value of A. In general MOD_SQR_IM_OM will run faster than MOD_SQR_IM_OM_TEQ, and will never run slower.

Table 11-16. MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ function properties

Property	Notes
Mode value for MOD_SQR_IM_OM	1100_0000_0000_0001_1010 (output placed in B) 1100_0000_0001_0001_1010 (output placed in A)
Mode value for MOD_SQR_IM_OM_TEQ	1100_0000_0100_0001_1010 (output placed in B) 1100_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = a Montgomery value < N
Output	B (or A, if selected) = (A*A) mod N, in Montgomery form
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = >Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.13 Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ)

This function may be used to cube an integer value. These functions first compute the Montgomery conversion factor, $R^2 \text{ mod } N$ and then convert the value to cube. For a given modulus, MOD_CUBE_TEQ will take the same time to complete for any value of A. In general MOD_CUBE will run faster than MOD_CUBE_TEQ, and will never run slower.

Table 11-17. MOD_CUBE and MOD_CUBE_TEQ function properties

Property	Notes
Mode value for MOD_CUBE	0000_0000_0000_0001_1011 (output placed in B) 0000_0000_0001_0001_1011 (output placed in A)
Mode value for MOD_CUBE_TEQ	0000_0000_0100_0001_1011 (output placed in B) 0000_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = a value < N

Table continues on the next page...

Table 11-17. MOD_CUBE and MOD_CUBE_TEQ function properties (continued)

Property	Notes
Output	B (or A , if selected) = $(A*A*A) \bmod N$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • $A < N$
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.14 Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ)

MOD_CUBE_IM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM will run faster than MOD_CUBE_IM_TEQ, but will never run slower.

Table 11-18. MOD_CUBE_IM and MOD_CUBE_IM_TEQ function properties

Property	Notes
Mode value for MOD_CUBE_IM	1000_0000_0000_0001_1011 (output placed in B) 1000_0000_0001_0001_1011 (output placed in A)
Mode value for MOD_CUBE_IM_TEQ	1000_0000_0100_0001_1011 (output placed in B) 1000_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A = a Montgomery value
Output	B (or A , if selected) = $(A*A*A) \bmod N$, a normal value
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • $A < N$
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.15 Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ)

MOD_CUBE_IM_OM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_OM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM_OM will run faster than MOD_CUBE_IM_OM_TEQ, but will never run slower.

Table 11-19. MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ function properties

Property	Notes
Mode value for MOD_CUBE_IM_OM	1100_0000_0000_0001_1011 (output placed in B) 1100_0000_0001_0001_1011 (output placed in A)
Mode value for MOD_CUBE_IM_OM_TEQ	1100_0000_0100_0001_1011 (output placed in B) 1100_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any odd integer A = a Montgomery value
Output	B (or A, if selected) = $(A*A*A) \bmod N$, a Montgomery value
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes $A < N$
Side effects	
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.16 Integer Modulo Reduction (MOD_AMODN)

MOD_AMODN computes the remainder of A divided by N. A and N can be of any size and it is not required that $A > N$, but N must be non-zero.

Table 11-20. MOD_AMODN function properties

Property	Notes
Mode value	0000_0000_0000_0000_0111 (output placed in B) 0000_0000_0001_0000_0111 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any non-zero integer A = any integer
Output	B (or A, if selected) = $A \bmod N$, A reduced modulo N
Requirements	<ul style="list-style-type: none"> N = non-zero value Minimum modulus size = 1 byte Maximum modulus size = 512 bytes

Table continues on the next page...

Table 11-20. MOD_AMODN function properties (continued)

Property	Notes
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Divide By Zero Error is set if N = 0.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.17 Integer Modular Inversion (MOD_INV)

MOD_INV computes the inverse of A, if an inverse exists. If the modulus, N, is prime, then all values of A, $1 \leq A < N$, are guaranteed to have an inverse mod N. If N is not prime, A may or may not have an inverse. It will have one only if $GCD(A, N) == 1$.

Table 11-21. MOD_INV function properties

Property	Notes
Mode value	0000_0000_0000_0000_1000 (output placed in B) 0000_0000_0001_0000_1000 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any non-zero integer A = any non-zero integer less than N
Output	B (or A, if selected) = $A^{-1} \text{ mod } N$, an integer, the multiplicative inverse of A
Requirements	<ul style="list-style-type: none"> Neither A or N can be zero. A must be less than N. Minimum modulus size = 1 byte Maximum modulus size = 512 bytes
Side effects	A and E are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. A Size Error is set if the size of A is greater than size of N. Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0, or if there is no inverse.
Flags set	None

11.1.6.3.18 Integer Montgomery Factor Computation (MOD_R2)

This function is used to compute a constant to assist in converting operands into the Montgomery residue system representation. The constant $R^2(\text{mod } N)$ is dependent upon the digit size of the PKHA and the value in N.

MUL, EXP, and ECC functions that do not have "IM" (Montgomery inputs) or an R2 input will internally invoke this routine to determine the constant and do the conversions before other operations.

If the modulus N is a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a not-insignificant amount of time to complete.

Table 11-22. MOD_R2 function properties

Property	Notes
Mode value	1000_0000_0000_0000_1100 (output placed in B) 1000_0000_0001_0000_1100 (output placed in A)
Input	N = modulus, any odd integer
Output	B (or A , if selected) = $R2 \bmod N$, where $R = 2^{SD}$ where S is size of a digit in bits and D is the number of digits of N ; in other words, $D = \text{ceiling}[\text{sizeof}(N) \text{ in bits} / S]$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • Modulus Even Error is set if N is even.
Flags set	None

11.1.6.3.19 Integer Greatest Common Divisor (MOD_GCD)

MOD_GCD finds the greatest common divisor of two integers.

Table 11-23. MOD_GCD function properties

Property	Notes
Mode value	0000_0000_0000_0000_1110 (output placed in B) 0000_0000_0001_0000_1110 (output placed in A)
Input	<ul style="list-style-type: none"> • N = any integer. The most-significant digit of N must be non-zero. • A = any integer less than or equal to N
Output	B (or A , if selected) = $\text{GCD}(A,N)$, an integer less than or equal to A that divides both A and N If the output is placed in B , the MOD_INV result is available in A .
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and N may not both be zero
Side effects	A is modified
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 512$. • A Size Error is set if the size of A is greater than size of N. • Divide-By-Zero Error is set if N or $A = 0$, or if the most significant digit of $N = 0$.
Flags set	PIO is set if the result is 1.

11.1.6.3.20 Miller_Rabin Primality Test (PRIME_TEST)**Table 11-24. PRIME_TEST function properties**

Property	Notes
Mode value	0000_0000_0000_0000_1111 (output placed in B) 0000_0000_0001_0000_1111 (output placed in A)
Input	<ul style="list-style-type: none"> • N^1 = Candidate prime integer • A = An initial random seed for the base value of exponentiation; can be any integer $2 < A < N - 2$ • B = "t" parameter, which is the number of trial runs. By default, it is set at 1 or B[7:0], whichever is greater. Only the lowest byte of the supplied value is used.
Output	B (or A, if selected) = 1 if candidate is believed to be prime, otherwise 0
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and N may not both be zero
Side effects	N and A are modified
Errors reported	<ul style="list-style-type: none"> • N Size Error is set if size of N = 0 or size of N > 512. • B Size Error is set if N size > 256. and the least-significant byte of B > 31. • Divide-By-Zero Error is set if no seed can be found that is in the legal range of $2 < A < N-2$. This occurs if N = 1 or N = 3.
Flags set	PRM is set if the candidate is believed to be prime

1. If the most significant digit of N is zero, the result is always composite, the output is the value zero, and the PRM flag is not set, regardless of the primality of the value of N.

11.1.6.3.21 Right Shift A (RIGHT_SHIFT_A) function**Table 11-25. RIGHT_SHIFT_A function properties**

Property	Notes
Mode value	0000_0000_0000_0001_1101 (output placed in B) 0000_0000_0001_0001_1101 (output placed in A)
Input	<ul style="list-style-type: none"> • A = first addend, any integer less than N
Output	B (or A, if selected) = (A + B) mod N
Requirements	<ul style="list-style-type: none"> • Minimum A size = 1 byte • Maximum A size = 512 bytes
Side effects	None
Errors reported	<ul style="list-style-type: none"> • A Size Error is set if the size of A is greater than max size. <p>If (A + B) >= N, N will be subtracted just once from the sum. That is, if (A + B) >= 2N, then the result will not be mod N.</p>
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.22 Compare A B (COMPARE) function

Table 11-26. COMPARE function properties

Property	Notes
Mode value	0000_0000_0000_0001_1110
Input	<ul style="list-style-type: none"> • A = Value to be compared • B = Value to be compared • A Size = the number of least-significant bits that will be compared
Output	None (other than flags)
Requirements	A Size must be \geq B Size.
Side effects	none
Errors reported	If B Size $>$ A Size a "B Size" error will be generated.
Flags set	<ul style="list-style-type: none"> • PKHA_GCD_ONE is set if B $>$ A • PKHA_ZERO is set if B == A • no flag is set if B \leq A

11.1.6.3.23 Evaluate A (EVALUATE) function

Table 11-27. EVALUATE function properties

Property	Notes
Mode value	<p>SB00_0000_0000_0001_1111 (output placed in B)</p> <p>SB00_0000_0001_0001_1111 (do not modify B)</p> <ul style="list-style-type: none"> • If the S bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0sss, where sss is the updated A Size. • If the B bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0bbbb, where bbbb is the updated number of bits in A. • If the S bit is set and the B bit is set, the A-Size DWord will be pushed before the number-of-bits-in-A Dword.
Input	<ul style="list-style-type: none"> • A = Value to be evaluated
Output	<ul style="list-style-type: none"> • A Size is updated with the number of least-significant non-zero bytes, i.e. the position of the most-significant non-zero byte (least-significant byte is byte-position 0). This evaluation considers only the bytes specified by the incoming value of A Size. This allows the incoming value of A Size to be set so that A0 will be evaluated, ignoring the values in A1, A2 and A3, or the incoming value could be set so that (A1,A0) will be evaluated, ignoring A2 and A3, or (A2,A1,A0) could be evaluated, ignoring A3. • If the mode value specifies that the output is to be placed in B, the updated value of A Size will be copied into the least-significant two bytes of B and B Size will be set to 2. If the "do not modify B" option is selected, the updated value of A Size will not be copied into B.
Requirements	none
Side effects	<ul style="list-style-type: none"> • A Size will be modified. • B Size may be modified. • One or two DWords may be pushed to the output FIFO.
Errors reported	none
Flags set	<ul style="list-style-type: none"> • PKHA_GCD_ONE is set if A == 1 • PKHA_ZERO is set if A == 0. A Size (and number of bits in A) will be set to 0. Note that this could cause an A Size error in a subsequent PKHA operation.

11.1.6.3.24 Binary Polynomial (F_{2^m}) Addition (F2M_ADD) function

This function performs binary polynomial modular addition without any modulo reduction, as the value in the N register is ignored. Only its size is used, to determine the size of the result.

This type of addition is the equivalent of a bitwise XOR and this function may be used for that purpose.

This function could as easily be labeled F2M_SUB, as it is mathematically equivalent.

Table 11-28. F2M_ADD function properties

Property	Notes
Mode value	0000_0000_0000_0000_0010 (output placed in B) 0000_0000_0001_0000_0010 (output placed in A)
Input	<ul style="list-style-type: none"> • Size of N (modulus size) • A = first addend, a binary polynomial • B = second addend, a binary polynomial
Output	B (or A, if selected) = A xor B, a binary polynomial
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • The N need not be written, because its contents are ignored, but the size of N must be written. This size is needed because inputs A and B are considered binary polynomials modulo some irreducible polynomial N.
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.25 Binary Polynomial (F_{2^m}) Modular Multiplication (F2M_MUL)

The $(AB) \bmod N$ computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, F2M_MUL_IM or F2M_MUL_IM_OM) are more efficient. This function first computes $R^2 \bmod N$, then multiplies $A \cdot R^2$ to produce AR , then multiplies $AR \cdot B$ to produce AB .

Table 11-29. F2M_MUL function properties

Property	Notes
Mode value	0010_0000_0000_0000_0101 (output placed in B) 0010_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial

Table continues on the next page...

Table 11-29. F2M_MUL function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> A = multiplicand, a field element B = multiplier, a field element
Output	B (or A, if selected) = (AB) mod N, a field element
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and B are field elements modulo N.
Side effects	A and E are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.26 Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function

This function takes its inputs, binary polynomials, in Montgomery form, multiplies them modulo the value in the N register, used as a reduction polynomial, and returns the result as a field value. To do this, it performs two multiplications: $AR * BR \Rightarrow ABR$ and $ABR * 1 \Rightarrow AB$.

Table 11-30. F2M_MUL_IM function properties

Property	Notes
Mode value	1010_0000_0000_0000_0101 (output placed in B) 1010_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an "odd" binary polynomial of order m A = multiplicand, a binary polynomial $< 2^m$, in Montgomery form B = multiplier, a binary polynomial $< 2^m$, in Montgomery form
Output	B (or A, if selected) = (AxB) mod N, a binary polynomial $< 2^m$, non-Montgomery form
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and B are field elements in Montgomery form and must be modulo reduced by irreducible polynomial N.
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N. B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.27 Binary Polynomial (F_{2^m}) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function

This function performs the calculation $A*B/R \bmod N$, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is $R^2 \bmod N$, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

Table 11-31. F2M_MUL_IM_OM function properties

Property	Notes
Mode value	1110_0000_0000_0000_0101 (output placed in B) 1110_0000_0001_0000_0101 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial $0 < 2^m$, in Montgomery form. • B = a binary polynomial $0 < 2^m$, in Montgomery form.
Output	B (or A, if selected) = $(AxB) \bmod N$, in Montgomery form
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A and B are field elements in Montgomery form.
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error is set if the size of B is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.28 Binary Polynomial (F_{2^m}) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)

This function is similar to MOD_EXP but works on binary polynomials. It is provided mainly to assist in the computation of elliptic curve parameter "c", where $c = b^{2^{m-2}} \bmod n$ given an elliptic curve parameter "b" and the field-defining polynomial in N. It computes $R^2 \bmod N$ and converts A to Montgomery form before beginning the exponentiation. F2M_EXP_TEQ performs the same operation as F2M_EXP but with an added timing

equalization security feature. Its exponentiation run-time, for a given modulus and size of exponent, is constant. In general F2M_EXP will run faster than F2M_EXP_TEQ, but will never run slower.

Table 11-32. F2M_EXP function properties

Property	Notes
Mode value for F2M_EXP	0010_0000_0000_0000_0110 (output placed in B) 0010_0000_0001_0000_0110 (output placed in A)
Mode value for F2M_EXP_TEQ	0010_0000_0100_0000_0110 (output placed in B) 0010_0000_0101_0000_0110 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial <math>2^m</math> • E = exponent, any integer
Output	B (or A, if selected) = $(A^E) \bmod N$, a binary polynomial
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • Maximum key (exponent) size = 512 bytes
Side effects	None
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • Key Size Error is set if size of E = 0 or size of E > 512. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.29 Binary Polynomial (F_{2^m}) Simultaneous Modular Exponentiation (F2M_SML_EXP)

F2M_SML_EXP performs two modular exponentiations on binary polynomials, and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

Table 11-33. F2M_SML_EXP function properties

Property	Notes
Mode value	0000_0000_0000_0001_0110 (output placed in B) 0000_0000_0001_0001_0110 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, any odd integer • A0 = a field element, first base • E = an integer, first exponent • A2 = a field element, second base • B = an integer, second exponent
Output	B (or A if selected) = $A0^E * A2^B \bmod N$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 1/2 RAM

Table continues on the next page...

Table 11-33. F2M_SML_EXP function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> Maximum key (exponent) size and B size = 1/4 RAM size A0 < N A2 < N The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N
Side effects	A, B, and E are modified.
Errors reported	<ul style="list-style-type: none"> N size error is set if N size > half RAM A size error is set if A size > half RAM B size error is set if B size > quarter-RAM E size error is set if E size > quarter-RAM
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.30 Binary Polynomial (F_{2^m}) Modular Square (F2M_SQR and F2M_SQR_TEQ)

F2M_SQR may be used to square an binary polynomial value. The timing equalized version, F2M_SQR_TEQ, will also square a binary polynomial value but for a given modulus will always take the same time to complete. F2M_SQR will usually run faster than F2M_SQR_TEQ, but will never run slower

Table 11-34. F2M_SQR and F2M_SQR_TEQ function properties

Property	Notes
Mode value for F2M_SQR	0010_0000_0000_0001_1010 (output placed in B) 0010_0000_0001_0001_1010 (output placed in A)
Mode value for F2M_SQR_TEQ	0010_0000_0100_0001_1010 (output placed in B) 0010_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an "odd" binary polynomial of order m A = a binary polynomial of order < m
Output	B (or A, if selected) = (A*A) mod N
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.31 Binary Polynomial (F_{2^m}) Modular Square, Montgomery Input (F2M_SQR_IM and F2M_SQR_IM_TEQ)

F2M_SQR_IM may be used to square a binary polynomial in Montgomery form. F2M_SQR_IM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM will generally run faster than F2M_SQR_IM_TEQ, but will never run slower.

Table 11-35. F2M_SQR_IM and F2M_SQR_IM_TEQ function properties

Property	Notes
Mode value for F2M_SQR_IM	1010_0000_0000_0001_1010 (output placed in B) 1010_0000_0001_0001_1010 (output placed in A)
Mode value for F2M_SQR_IM_TEQ	1010_0000_0100_0001_1010 (output placed in B) 1010_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m in Montgomery form
Output	B (or A, if selected) = (A*A) mod N
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.32 Binary Polynomial (F_{2^m}) Modular Square, Montgomery Input and Output (F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ)

F2M_SQR_IM_OM may be used to square a binary polynomial in Montgomery form, and will output the result in Montgomery form. F2M_SQR_IM_OM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM_OM will generally run faster than F2M_SQR_IM_OM_TEQ, but will never run slower.

Table 11-36. F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ function properties

Property	Notes
Mode value for F2M_SQR_IM_OM	1110_0000_0000_0001_1010 (output placed in B) 1110_0000_0001_0001_1010 (output placed in A)

Table continues on the next page...

Table 11-36. F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ function properties (continued)

Property	Notes
Mode value for F2M_SQR_IM_OM_TEQ	1110_0000_0100_0001_1010 (output placed in B) 1110_0000_0101_0001_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order < m, possibly in Montgomery form
Output	B (or A, if selected) = (A*A) mod N, in Montgomery form
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.33 Binary Polynomial (F_{2^m}) Modular Cube (F2M_CUBE and F2M_CUBE_TEQ)

F2M_CUBE may be used to cube a binary polynomial value. The function will first compute the Montgomery conversion factor, $R^2 \text{ mod } N$ and convert the value to cube. F2M_CUBE_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE will generally run faster than F2M_CUBE_TEQ, but will never run slower.

Table 11-37. F2M_CUBE and F2M_CUBE_TEQ function properties

Property	Notes
Mode value for F2M_CUBE	0010_0000_0000_0001_1011 (output placed in B) 0010_0000_0001_0001_1011 (output placed in A)
Mode value for F2M_CUBE_TEQ	0010_0000_0100_0001_1011 (output placed in B) 0010_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an "odd" binary polynomial of order m • A = a binary polynomial of order $< m$
Output	B (or A, if selected) = (A*A*A) mod N
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 512 bytes • A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 512. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N.

Table continues on the next page...

Table 11-37. F2M_CUBE and F2M_CUBE_TEQ function properties (continued)

Property	Notes
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.34 Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input (F2M_CUBE_IM and F2M_CUBE_IM_TEQ)

F2M_CUBE_IM may be used to cube a binary polynomial value that is in Montgomery form. F2M_CUBE_IM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM will generally run faster than F2M_CUBE_IM_TEQ, but will never run slower.

Table 11-38. F2M_CUBE_IM and F2M_CUBE_IM_TEQ function properties

Property	Notes
Mode value for F2M_CUBE_IM	1010_0000_0000_0001_1011 (output placed in B) 1010_0000_0001_0001_1011 (output placed in A)
Mode value for F2M_CUBE_IM_TEQ	1010_0000_0100_0001_1011 (output placed in B) 1010_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an "odd" binary polynomial of order m A = a binary polynomial of order < m in Montgomery form
Output	B (or A, if selected) = (A*A*A) mod N, a binary polynomial
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.35 Binary Polynomial (F_{2^m}) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ)

F2M_CUBE_IM_OM may be used to cube a binary polynomial value that is in Montgomery form, and output the result in Montgomery form.

F2M_CUBE_IM_OM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM_OM will generally run faster than F2M_CUBE_IM_OM_TEQ, but will never run slower.

Table 11-39. F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_EXP function properties

Property	Notes
Mode value for F2M_CUBE_IM_OM	1110_0000_0000_0001_1011 (output placed in B) 1110_0000_0001_0001_1011 (output placed in A)
Mode value for F2M_CUBE_IM_OM_TEQ	1110_0000_0100_0001_1011 (output placed in B) 1110_0000_0101_0001_1011 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an "odd" binary polynomial of order m A = a binary polynomial of order < m, possibly in Montgomery form
Output	B (or A, if selected) = (A*A*A) mod N, in Montgomery form
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A < N
Side effects	
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N.
Flags set	PIZ is set if the result is zero. GCD is set if the result is one.

11.1.6.3.36 Binary Polynomial (F_{2^m}) Modulo Reduction (F2M_AMODN)

F2M_AMODN computes the remainder of A divided by N. This is the equivalent of the MOD_AMODN function applied to a binary polynomial. A and N can be of any size and it is not required that $A > N$, but N must be non-zero.

Table 11-40. F2M_AMODN function properties

Property	Notes
Mode value	0010_0000_0000_0000_0111 (output placed in B) 0010_0000_0001_0000_0111 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, any non-zero polynomial A = any polynomial
Output	B (or A, if selected) = A mod N, a polynomial, binary element modulo N
Requirements	<ul style="list-style-type: none"> N = non-zero value Minimum modulus size = 1 byte Maximum modulus size = 512 bytes
Side effects	None
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Divide By Zero Error is set if N = 0.
Flags set	PIZ is set if the result is zero. PIO is set if the result is one.

11.1.6.3.37 Binary Polynomial (F_{2^m}) Modular Inversion (F2M_INV)

F2M_INV computes the multiplicative inverse of a binary polynomial.

Table 11-41. F2M_INV function properties

Property	Notes
Mode value	0010_0000_0000_0000_1000 (output placed in B) 0010_0000_0001_0000_1000 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an irreducible polynomial A = a field element
Output	B (or A, if selected) = $A^{-1} \bmod N$, a field element, the multiplicative inverse of A
Requirements	<ul style="list-style-type: none"> A is an element of the binary polynomial field. Minimum modulus size = 1 byte Maximum modulus size = 512 bytes
Side effects	A and E are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even. A Size Error is set if the size of A is greater than size of N. Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0.
Flags set	None

11.1.6.3.38 Binary Polynomial (F_{2^m}) R^2 Mod N (F2M_R2) Function

This function is used to compute the Montgomery Conversion Factor, which is used to convert operands into the Montgomery residue system representation. The constant $R^2(\bmod N)$ is dependent upon the digit size of the PKHA and the value of N. If this value is not available, then this routine (function) is called to determine the constant before other operations. If N contains a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a considerable amount of time to complete.

Table 11-42. F2M_R2 function properties

Property	Notes
Mode value	0010_0000_0000_0000_1100 (output placed in B) 0010_0000_0001_0000_1100 (output placed in A)
Input	N = modulus, an irreducible polynomial
Output	B (or A, if selected) = $R^2 \bmod N$, where $R = 2^{SD}$ where S is size of a digit in bits and D is the number of digits of an irreducible polynomial, in other words $D = \text{ceiling}[\text{sizeof}(N) \text{ in bits} / S]$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes
Side effects	None

Table continues on the next page...

Table 11-42. F2M_R2 function properties (continued)

Property	Notes
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. Modulus Even Error is set if N is even.
Flags set	None

11.1.6.3.39 Binary Polynomial (F_{2^m}) Greatest Common Divisor (F2M_GCD) Function

MOD_GCD finds the greatest common divisor of two binary polynomials.

Table 11-43. F2M_GCD function properties

Property	Notes
Mode value	0010_0000_0000_0000_1110 (output placed in B) 0010_0000_0001_0000_1110 (output placed in A)
Input	<ul style="list-style-type: none"> N = any polynomial. The most-significant digit of N must be non-zero. A = any polynomial with degree less than or equal to N
Output	B (or A, if selected) = BINARY_GCD(A,N), a polynomial with degree less than or equal to polynomial A that divides both A and N
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 512 bytes A and N may not both be zero
Side effects	A is modified
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 512. A Size Error is set if the size of A is greater than size of N. Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0.
Flags set	PIO is set if the result is 1.

11.1.6.4 Elliptic Curve Functions

11.1.6.4.1 ECC F_p Point Add, Affine Coordinates (ECC_MOD_ADD) Function

ECC_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 11-44. ECC_MOD_ADD function properties

Property	Notes
Mode value	0000_0000_0000_0000_1001 (output placed in B) 0000_0000_0001_0000_1001 (output placed in A)

Table continues on the next page...

Table 11-44. ECC_MOD_ADD function properties (continued)

Property	Notes
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

11.1.6.4.2 ECC F_p Point Add, Affine Coordinates, R^2 Mod N Input (ECC_MOD_ADD_R2) Function

ECC_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECC_MOD_ADD_R2 has $R^2 \bmod N$ as an additional input, this function is more efficient than ECC_MOD_ADD, which first must compute $R^2 \bmod N$ before performing the addition.

Table 11-45. ECC_MOD_ADD_R2 function properties

Property	Notes
Mode value	0001_0000_0000_0000_1001 (output placed in B) 0001_0000_0001_0000_1001 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero • [A0, A1] = first addend point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend point in affine coordinates (x,y) • B3 = R2 ($R^2 \bmod N$) input
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.

Table continues on the next page...

Table 11-45. ECC_MOD_ADD_R2 function properties (continued)

Property	Notes
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

11.1.6.4.3 ECC F_p Point Double, Affine Coordinates (ECC_MOD_DBL) Function

ECC_MOD_DBL computes the double ($B + B$) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 11-46. ECC_MOD_DBL function properties

Property	Notes
Mode value	0000_0000_0000_0000_1010 (output placed in B) 0000_0000_0001_0000_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1, A2] = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = input point in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [B1, B2] + [B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates (x, y).
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

11.1.6.4.4 ECC F_p Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function

ECC_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECC_MOD_MUL in general will run faster than ECC_MOD_MUL_TEQ, but will never run slower.

Table 11-47. ECC_MOD_MUL and ECC_MOD_MUL_TEQ function properties

Property	Notes
Mode value for ECC_MOD_MUL	0000_0000_0000_0000_1011 (output placed in B) 0000_0000_0001_0000_1011 (output placed in A)
Mode value for ECC_MOD_MUL_TEQ	0000_0000_0100_0000_1011 (output placed in B) 0000_0000_0101_0000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero • E = scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0).
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $k = 0$, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For $k < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = \text{abs}(-k)$. After the computation is

complete, the formula $-P = (x, -y)$ can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

11.1.6.4.5 ECC F_p Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function

ECC_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECC_MOD_MUL_R2 has $R^2 \bmod N$ as an additional input, this function is more efficient than ECC_MOD_MUL, which first must compute $R^2 \bmod N$ before performing the multiplication. ECC_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve $(N, A3, B0)$, constant for a given size of E. ECC_MOD_MUL_R2 in general will run faster than ECC_MOD_MUL_R2_TEQ, but will never run slower.

Table 11-48. ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECC_MOD_MUL_R2	0001_0000_0000_0000_1011 (output placed in B) 0001_0000_0001_0000_1011 (output placed in A)
Mode value for ECC_MOD_MUL_R2_TEQ	0001_0000_0100_0000_1011 (output placed in B) 0001_0000_0101_0000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = $R^2 \bmod N$, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $k = 0$, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For $k < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = \text{abs}(-k)$. After the computation is complete, the formula $-P = (x,-y)$ can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

11.1.6.4.6 ECC F_p Check Point (ECC_MOD_CHECK_POINT) Function

ECC_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^2 = x^3 + ax + b$.

[ECC_MOD_CHECK_POINT_R2](#) has $R^2 \bmod N$ as an additional input, so it is more efficient than this function, which first must compute $R^2 \bmod N$ before performing the operation.

Table 11-49. ECC_MOD_CHECK_POINT function properties

Property	Notes
Mode value	0000_0000_0000_0001_1100
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • B1 = $R^2 \bmod N$, as described in MOD_R2MODN
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

ECC_MOD_CHECK_POINT checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 = x^3 + ax + b \pmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

11.1.6.4.7 ECC F_p Check Point, $R^2 \pmod N$ Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function

ECC_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^2 = x^3 + ax + b$. ECC_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 \pmod N$ and $x^3 + ax + b \pmod N$. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. Since this function takes $R^2 \pmod N$ as an additional input, ECC_MOD_CHECK_POINT_R2 is more efficient than ECC_MOD_CHECK_POINT, which first must compute $R^2 \pmod N$ before performing the point check.

Table 11-50. ECC_MOD_CHECK_POINT_R2 function properties

Property	Notes
Mode value	0001_0000_0000_0001_1100
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = $R^2 \pmod N$, pre-computed as described in MOD_R2MODN (0Eh) • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	<p>PIZ is set if the input is the point at infinity.</p> <p>GCD is set if the point is on the curve (but not point at infinity).</p>

ECC_MOD_CHECK_POINT_R2 checks whether x and y are $< N$. If not, the routine exits with no flags set. If the input is O , the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 = x^3 + ax + b \pmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

11.1.6.4.8 ECC F_{2^m} Point Add, Affine Coordinates (ECC_F2M_ADD) Function

ECC_F2M_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 11-51. ECC_F2M_ADD function properties

Property	Notes
Mode value	0010_0000_0000_0000_1001 (output placed in B) 0010_0000_0001_0000_1001 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • $[A0, A1]$ = first addend in affine coordinates • $A2$ = ignored • $A3$ = elliptic curve "a" parameter • $B0$ = elliptic curve "c" parameter, where $c = b^{2^{m-2}} \pmod n$ • $B1, B2]$ = second addend in affine coordinates • $B3$ = ignored
Output	$P[B1, B2]$ (or $P[A0, A1]$, if A output selected) = $P[A0, A1] + P[B1, B2]$, where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates $A0, A1, B1,$ and $B2$ and elliptic curve parameters $A3$ and $B0$ are elements of the binary polynomial field N.
Side effects	$A0, A1, A2, A3$ and $B3$ are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of $N = 0$ or size of $N > 128$. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide By Zero Error is set if the most significant digit of $N = 0$. • C is Zero Error if $B3$ is zero.
Flags set	None

11.1.6.4.9 ECC F_{2^m} Point Add, Affine Coordinates, $R^2 \pmod N$ Input (ECC_F2M_ADD_R2) Function

ECC_F2M_ADD_R2 performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates. Since this function takes $R^2 \pmod N$ as an additional input, ECC_F2M_ADD_R2 is more efficient than ECC_F2M_ADD, which first must compute $R^2 \pmod N$ before performing the addition.

Table 11-52. ECC_F2M_ADD_R2 function properties

Property	Notes
Mode value	0011_0000_0000_0000_1001 (output placed in B) 0011_0000_0001_0000_1001 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter, where $c = b^{2^{m-2}} \bmod n$. Must not be zero. • [B1, B2] = second addend in affine coordinates • B3 = R2 input
Output	P[B1, B2] (or P[A0, A1], if A output selected) = P[A0, A1] + P[B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide By Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	None

11.1.6.4.10 ECC F_{2m} Point Double - Affine Coordinates (ECC_F2M_DBL) Function

ECC_F2M_DBL computes the double (B + B) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 11-53. ECC_F2M_DBL function properties

Property	Notes
Mode value	0010_0000_0000_0000_1010 (output placed in B) 0010_0000_0001_0000_1010 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • A0, A1, A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter where $c = b^{2^{m-2}} \bmod n$ • [B1, B2] = input point in affine coordinates • B3 = ignored
Output	P[B1, B2] (or P[A0, A1], if A output selected) = P[B1, B2] + P[B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte

Table continues on the next page...

Table 11-53. ECC_F2M_DBL function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> Maximum modulus size = 128 bytes Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field formed by N.
Side effects	A0, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 128. Modulus Even Error is set if N is even. A Size Error is set if size of A is greater than size of N. B Size Error is set if size of B is greater than size of N. Divide-by-Zero Error is set if the most significant digit of N = 0. C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

11.1.6.4.11 ECC F_{2^m} Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function

ECC_F2M_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_F2M_MUL_TEQ performs the same operation as ECC_F2M_MUL but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL will run faster than ECC_F2M_MUL_TEQ, but will never run slower.

Table 11-54. ECC_F2M_MUL and ECC_F2M_MUL_TEQ function properties

Property	Notes
Mode value for ECC_F2M_MUL	0010_0000_0000_0000_1011 (output placed in B) 0010_0000_0001_0000_1011 (output placed in A)
Mode value for ECC_F2M_MUL_TEQ	0010_0000_0110_0000_1011 (output placed in B) 0010_0000_0111_0000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. E = key, scalar multiplier (k), any integer [A0, A1] = multiplicand, an input point in affine coordinates (x,y) A2 = ignored A3 = elliptic curve "a" parameter B0 = elliptic curve "c" parameter where $c = b^{2^{m-2}} \text{ mod } n$ B1 = ignored B2 = ignored B3 = ignored
Output	<ul style="list-style-type: none"> P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). B0 = undefined B3 = undefined
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte

Table continues on the next page...

Table 11-54. ECC_F2M_MUL and ECC_F2M_MUL_TEQ function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023) Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field.
Side effects	A0, A1 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 128. Key Size Error is set if size of E = 0 or size of E > 512. Modulus Even Error is set if N is even. A Size Error is set if size of A is greater than size of N. B Size Error is set if size of B is greater than size of N. Divide-by-Zero Error is set if the most significant digit of N = 0. C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $E = 0$, this function returns a point at infinity (0,0).
- For $E < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = -E$). After the multiplication is complete, the formula $-P = (x, x+y)$ can be used to compute the y coordinate of the effective final result; the x coordinate stays the same.

11.1.6.4.12 ECC F_{2^m} Point Multiply, R^2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function

ECC_F2M_MUL_R2 computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. Since this function takes $R^2 \bmod N$ as an additional input, ECC_F2M_MUL_R2 is more efficient than ECC_F2M_MUL, which first must compute $R^2 \bmod N$ before performing the multiplication.

ECC_F2M_MUL_R2_TEQ performs the same operation as ECC_F2M_MUL_R2 but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL_R2 will run faster than ECC_F2M_MUL_R2_TEQ, but will never run slower.

Table 11-55. ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECC_F2M_MUL_R2	0011_0000_0000_0000_1011 (output placed in B) 0011_0000_0001_0000_1011 (output placed in A)
Mode value for ECC_F2M_MUL_R2_TEQ	0011_0000_0100_0000_1011 (output placed in B) 0011_0000_0101_0000_1011 (output placed in A)

Table continues on the next page...

Table 11-55. ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ function properties (continued)

Property	Notes
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter where $c = b^{2^{m-2}} \bmod n$ and m = degree of polynomial M • B1 = (f2m) R2 mod N, pre-computed as described in F2M_R2MODN (0Eh) • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equally size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023) • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field.
Side effects	A0, A1, A2, and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0. • C is Zero Error if B3 is zero.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $k = 0$, this function returns a point at infinity (0,0).
- For $k < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = \text{abs}(-k)$. After the computation is complete, the formula $-P = (x, x+y)$ can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

11.1.6.4.13 ECC F₂^m Check Point (ECC_F2M_CHECK_POINT) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^2 + xy = x^3 + ax^2 + b$.

[ECC_F2M_CHECK_POINT_R2](#) has $R^2 \bmod N$ as an additional input, so it is more efficient than this function, which first must compute $R^2 \bmod N$ before performing the operation.

Table 11-56. ECC_F2M_CHECK_POINT function properties

Property	Notes
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "c" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • B1 = $R^2 \bmod N$, as described in F2M_R2MODN (0Eh) • B2 = curve "b" parameter = $c^4 \bmod N$
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

This function checks whether x and y are $< N$. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2 + xy = x^3 + ax^2 + b \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. The outputs will not be present if PIZ is set.

11.1.6.4.14 ECC F_{2^m} Check Point, R^2 (ECC_F2M_CHECK_POINT_R2) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^2 + xy = x^3 + ax^2 + b$.

Since this function has $R^2 \bmod N$ as an additional input, it is more efficient than ECC_F2M_CHECK_POINT, which first must compute $R^2 \bmod N$ before performing the operation.

Table 11-57. ECC_F2M_CHECK_POINT_R2 function properties

Property	Notes
Input	<ul style="list-style-type: none"> • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored

Table continues on the next page...

**Table 11-57. ECC_F2M_CHECK_POINT_R2 function properties
(continued)**

Property	Notes
	<ul style="list-style-type: none"> A3 = elliptic curve "a" parameter B0 = elliptic curve "c" parameter B1 = $R^2 \bmod N$, pre-computed as described in F2M_R2MODN (0Eh) B2 = ignored B3 = ignored A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> B2 = curve "b" parameter = $c^4 \bmod N$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 128. Modulus Even Error is set if N is even. A Size Error is set if size of A is greater than size of N. B Size Error is set if size of B is greater than size of N. Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the point at infinity. GCD is set if the point is on the curve (but not point at infinity).

This function checks whether x and y are $< N$. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2 + xy = x^3 + ax^2 + b \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. The output will not be present if PIZ is set.

11.1.6.4.15 ECM Modular Multiplication (ECM_MOD_MUL_X and ECM_MOD_MUL_X_TEQ) Function

ECM_MOD_MUL_X computes the scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. ECM_MOD_MUL_X_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve $(N, A3)$, constant for a given size of E . ECM_MOD_MUL_X in general will run faster than ECM_MOD_MUL_X_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, $P2 = P3 + P1$, where $P1$ is the input and $P3$ is the result. Though this computes just the x coordinate, there is enough information, with $P1(y)$, to compute $P3(y)$.

Table 11-58. ECM_MOD_MUL_X and ECM_MOD_MUL_X_TEQ function properties

Property	Notes
Mode value for ECM_MOD_MUL_X	0000_1000_0000_0100_1011 (output placed in B) 0000_1000_0000_0100_1011 (output placed in A)
Mode value for ECM_MOD_MUL_X_TEQ	0000_1000_0100_0100_1011 (output placed in B) 0000_1000_0101_0100_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. • E = scalar multiplier (k), any integer • [A0] = multiplicand, an input point's affine x coordinate • A2 = ignored • A3 = elliptic curve a24 parameter, that is, (A+2)/4 • B0 = ignored • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate. • N1 = R2 • A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R • A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor • A3 = a24R, the a24 input, with the Montgomery factor • B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R • B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor • B3 = X1R, the x input, with the Montgomery factor
Requirements	<ul style="list-style-type: none"> • Maximum modulus size = 128 bytes • The x in (A0) should be on the elliptic curve formed by (N, A3 and "B").
Side effects	All quadrants of A, B, and N are modified except N0.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N.
Flags set	PIZ is set if the result is the point at infinity.

11.1.6.4.16 ECM F_p Point Multiply, R² Mod N Input, Affine Coordinates (ECM_MOD_MUL_X_R2 and ECM_MOD_MUL_X_R2_TEQ) Function

ECM_MOD_MUL_X_R2 computes a scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. Since ECM_MOD_MUL_X_R2 has R² mod N as an additional input, this function is more efficient than ECM_MOD_MUL_X, which first must compute R² mod N before performing the multiplication. ECM_MOD_MUL_X_R2_TEQ computes the same

function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3), constant for a given size of E. ECM_MOD_MUL_X_R2 in general will run faster than ECM_MOD_MUL_X_R2_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, $P2 = P3 + P1$, where P1 is the input and P3 is the result. Though this computes just the x coordinate, there is enough information, with P1(y), to compute P3(y).

Table 11-59. ECM_MOD_MUL_X_R2 and ECM_MOD_MUL_X_R2_TEQ function properties

Property	Notes
Mode value for ECM_MOD_MUL_X_R2	0001_1000_0000_1000_1011 (output placed in B) 0001_1000_0001_1000_1011 (output placed in A)
Mode value for ECM_MOD_MUL_X_R2_TEQ	0001_1000_0100_1000_1011 (output placed in B) 0001_1000_0101_1000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. • E = scalar multiplier (k), any integer • [A0] = multiplicand, an input point's affine x coordinate • A2 = ignored • A3 = elliptic curve a24 parameter, that is, (A+2)/4 • B0 = ignored • B1 = R2 mod N, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate. • N1 = R2 • A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R • A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor • A3 = a24R, the a24 input, with the Montgomery factor • B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R • B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor • B3 = X1R, the x input, with the Montgomery factor
Requirements	<ul style="list-style-type: none"> • Maximum modulus size = 128 bytes • The x in (A0) should be on the elliptic curve formed by (N, A3 and "B").
Side effects	All quadrants of A, B, and N are modified except N0.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N.
Flags set	PIZ is set if the result is the point at infinity.

11.1.6.4.17 ECT Modular Multiplication (ECT_MOD_MUL and ECT_MOD_MUL_TEQ) Function

ECT_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECT_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL in general will run faster than ECT_MOD_MUL_TEQ, but will never run slower.

Table 11-60. ECT_MOD_MUL and ECT_MOD_MUL_TEQ function properties

Property	Notes
Mode value for ECT_MOD_MUL	0000_0000_0000>_1000_1011 (output placed in B) 0000_0000_0000_1000_1011 (output placed in A)
Mode value for ECT_MOD_MUL_TEQ	0000_0000_0100_1000_1011 (output placed in B) 0000_0000_0101_1000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero • E = scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" (or should this be "d") parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0).
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $k = 0$, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For $k < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = \text{abs}(-k)$. After the computation is

complete, the formula $-P = (x, -y)$ can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

11.1.6.4.18 ECT F_p Point Multiply, R² Mod N Input, Affine Coordinates (ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ) Function

ECT_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECT_MOD_MUL_R2 has R² mod N as an additional input, this function is more efficient than ECT_MOD_MUL, which first must compute R² mod N before performing the multiplication. ECT_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL_R2 in general will run faster than ECT_MOD_MUL_R2_TEQ, but will never run slower.

Table 11-61. ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ function properties

Property	Notes
Mode value for ECT_MOD_MUL_R2	0001_0000_0000_1000_1011 (output placed in B) 0001_0000_0001_1000_1011 (output placed in A)
Mode value for ECT_MOD_MUL_R2_TEQ	0001_0000_0100_1000_1011 (output placed in B) 0001_0000_0101_1000_1011 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • E = key, scalar multiplier (k), any integer • [A0, A1] = multiplicand, an input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • B1 = R2 mod N, pre-computed as described in MOD_R2 • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y). • B0 = undefined • B3 = undefined
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Key Size Error is set if size of E = 0 or size of E > 512. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the result is the point at infinity.

The following special cases should be noted:

- For $k = 0$, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For $k < 0$, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, $k = \text{abs}(-k)$. After the computation is complete, the formula $-P = (x,-y)$ can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

11.1.6.4.19 ECT F_p Point Add, Affine Coordinates (ECT_MOD_ADD) Function

ECT_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

Table 11-62. ECT_MOD_ADD function properties

Property	Notes
Mode value	0000_0000_0000_1000_1001 (output placed in B) 0000_0000_0001_1000_1001 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero • [A0, A1] = first addend in affine coordinates • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend in affine coordinates • B3 = ignored
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

11.1.6.4.20 ECT F_p Point Add, Affine Coordinates, R^2 Mod N Input (ECT_MOD_ADD_R2) Function

ECT_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECT_MOD_ADD_R2 has $R^2 \bmod N$ as an additional input, this function is more efficient than ECT_MOD_ADD, which first must compute $R^2 \bmod N$ before performing the addition.

Table 11-63. ECT_MOD_ADD_R2 function properties

Property	Notes
Mode value	0001_0000_0000_1000_1001 (output placed in B) 0001_0000_0001_1000_1001 (output placed in A)
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero • [A0, A1] = first addend point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "b" parameter • [B1, B2] = second addend point in affine coordinates (x,y) • B3 = R2 ($R^2 \bmod N$) input
Output	[B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates.
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes • Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N.
Side effects	A0, A1, A2, A3 and B3 are modified.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if the size of A is greater than size of N. • B Size Error will be set if size of B is greater than size of N. • Divide-By-Zero Error is set if the most-significant digit of N = 0.
Flags set	None

11.1.6.4.21 ECT F_p Check Point (ECT_MOD_CHECK_POINT) Function

ECT_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $ax^2 + y^2 = 1 + dx^2y^2$.

ECT_MOD_CHECK_POINT_R2 has $R^2 \bmod N$ as an additional input, so it is more efficient than this function, which first must compute $R^2 \bmod N$ before performing the operation.

Table 11-64. ECT_MOD_CHECK_POINT function properties

Property	Notes
Mode value	0000_0000_0000_1001_1100
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "d" parameter • B1 = ignored • B2 = ignored • B3 = ignored • A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> • B1 = $R^2 \bmod N$, as described in MOD_R2MODN (0Eh)
Requirements	<ul style="list-style-type: none"> • Minimum modulus size = 1 byte • Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> • Data Size Error is set if size of N = 0 or size of N > 128. • Modulus Even Error is set if N is even. • A Size Error is set if size of A is greater than size of N. • B Size Error is set if size of B is greater than size of N. • Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	PIZ is set if the input is the "neutral point" (0, 1). GCD is set if the point is on the curve (but not the neutral point).

ECT_MOD_CHECK_POINT checks whether x and y are $< N$. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $ax^2 + y^2 = 1 + dx^2y^2 \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

11.1.6.4.22 ECT F_p Check Point, R^2 (ECT_MOD_CHECK_POINT_R2) Function

ECT_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. x and y satisfy the equation $ax^2 + y^2 = 1 + dx^2y^2$.

Table 11-65. ECT_MOD_CHECK_POINT_R2 function properties

Property	Notes
Mode value	0001_0000_0000_1001_1100
Input	<ul style="list-style-type: none"> • N = modulus, a prime number. The most significant digit of N must be non-zero. • [A0, A1] = a possible input point in affine coordinates (x,y) • A2 = ignored • A3 = elliptic curve "a" parameter • B0 = elliptic curve "d" parameter • B1 = ignored • B2 = $R^2 \bmod N$, pre-computed as described in MOD_R2MODN (0Eh)

Table continues on the next page...

Table 11-65. ECT_MOD_CHECK_POINT_R2 function properties (continued)

Property	Notes
	<ul style="list-style-type: none"> B3 = ignored A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations.
Output	<ul style="list-style-type: none"> $B2 = R^2 \bmod N$
Requirements	<ul style="list-style-type: none"> Minimum modulus size = 1 byte Maximum modulus size = 128 bytes
Side effects	Various quadrants are modified, but inputs are unchanged.
Errors reported	<ul style="list-style-type: none"> Data Size Error is set if size of N = 0 or size of N > 128. Modulus Even Error is set if N is even. A Size Error is set if size of A is greater than size of N. B Size Error is set if size of B is greater than size of N. Divide-by-Zero Error is set if the most significant digit of N = 0.
Flags set	<p>PIZ is set if the input is the "neutral point" (0, 1)..</p> <p>GCD is set if the point is on the curve (but not the neutral point).</p>

ECT_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $ax^2 + y^2 = 1 + dx^2y^2 \bmod N$. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. Since ECT_MOD_CHECK_POINT_R2 has $R^2 \bmod N$ as an additional input, this function is more efficient than ECT_MOD_CHECK_POINT, which first must compute $R^2 \bmod N$ before performing the operation.

11.1.6.4.23 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)

These functions copy data from a PKHA register (or register quadrant) specified as a source, to another PKHA register (or register quadrant) specified as a destination. COPY_NSZ copies the amount of data specified by the N Size register. COPY_SSZ copies the amount of data specified in the source register's size register. The source and destination are specified in the Mode Value. The source can be A, B or N. The destination can be A, B, E or N, but not the same as the source.

In a quadrant copy, when NSZ/SSZ exceeds the length of a quadrant, the copy will carry on into the next (higher-numbered) quadrant(s).

When the copy operation has completed, the destination register's size register will be updated to contain the number of bytes copied.

Table 11-66. COPY_NSZ and COPY_SSZ function properties

Property	Notes				
Mode value	Bits 19:17	Bits 16,11,10	Bits 9:8	Bits 7:6	Bits 5:0
	Source Register	Destination Register	Source Segment	Destination Segment	Function Code
	000 = A Register	000 = A Register	00 = Segment 0	00 = Segment 0	01_0000 = Copy_NSZ 01_0001 = Copy_SSZ
	001 = B Register	001 = B Register	01 = Segment 1	01 = Segment 1	
	011 = N Register	011 = N Register	10 = Segment 2	10 = Segment 2	
		010 = E Register	11 = Segment 3	11 = Segment 3	
other values reserved	other values reserved				
	1. If the destination register is E, the source and destination segments must be 00b.				
Input	None				
Output	None				
Requirements	For Copy_NSZ, the N-size Register must contain a valid value. For Copy_SSZ, the source register's size register must contain a valid value.				
Side effects	The destination register's size register is updated to the number of bytes copied.				
Errors reported	None				
Flags set	None				

1. If the destination register is E, the source and destination segments must be 00b.

11.1.6.4.24 Right Shift A (R_SHIFT) function

Table 11-67. R_SHIFT function properties

Property	Notes
Mode value	0000_0000_0000_0001_1101 (output placed in B) 0000_0000_0001_0001_1101 (output placed in A)
Input	<ul style="list-style-type: none"> A = Input value to be shifted. Bytes above A Size will be assumed to be zero, regardless of the contents of the PKHA A RAM. B = Number of bit positions that the A RAM will be shifted (Only the least-significant two bytes are used. The upper bytes are ignored.)
Output	B (or A, if selected) = the contents of PKHA A RAM (with zeros substituted for bytes above A Size), right-shifted by the number of bit positions specified in the least-significant two bytes of PKHA B RAM, and zero-filled on the left.
Requirements	none
Side effects	B (or A) is modified.
Errors reported	none
Flags set	none

11.1.6.4.25 Compare A B (COMPARE) function

Table 11-68. COMPARE function properties

Property	Notes
Mode value	0000_0000_0000_0001_1110
Input	<ul style="list-style-type: none"> • A = Value to be compared • B = Value to be compared • A Size = the number of least-significant bits that will be compared
Output	None (other than flags)
Requirements	A Size must be \geq B Size.
Side effects	none
Errors reported	If B Size $>$ A Size a "B Size" error will be generated.
Flags set	<ul style="list-style-type: none"> • PKHA_GCD_ONE is set if B $>$ A • PKHA_ZERO is set if B == A • no flag is set if B \leq A

11.1.6.4.26 Evaluate A (EVALUATE) function

Table 11-69. EVALUATE function properties

Property	Notes
Mode value	<p>SB00_0000_0000_0001_1111 (output placed in B)</p> <p>SB00_0000_0001_0001_1111 (do not modify B)</p> <ul style="list-style-type: none"> • If the S bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0sss, where sss is the updated A Size. • If the B bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_bbbb, where bbbb is the updated number of bits in A. • If the S bit is set and the B bit is set, the A-Size DWord will be pushed before the number-of-bits-in-A Dword.
Input	<ul style="list-style-type: none"> • A = Value to be evaluated
Output	<ul style="list-style-type: none"> • A Size is updated with the number of least-significant non-zero bytes, i.e. the position of the most-significant non-zero byte (least-significant byte is byte-position 0). This evaluation considers only the bytes specified by the incoming value of A Size. This allows the incoming value of A Size to be set so that A0 will be evaluated, ignoring the values in A1, A2 and A3, or the incoming value could be set so that (A1,A0) will be evaluated, ignoring A2 and A3, or (A2,A1,A0) could be evaluated, ignoring A3. • If the mode value specifies that the output is to be placed in B, the updated value of A Size will be copied into the least-significant two bytes of B and B Size will be set to 2. If the "do not modify B" option is selected, the updated value of A Size will not be copied into B.
Requirements	none
Side effects	<ul style="list-style-type: none"> • A Size will be modified. • B Size may be modified. • One or two DWords may be pushed to the output FIFO.
Errors reported	none
Flags set	<ul style="list-style-type: none"> • PKHA_GCD_ONE is set if A == 1 • PKHA_ZERO is set if A == 0. A Size (and number of bits in A) will be set to 0. Note that this could cause an A Size error in a subsequent PKHA operation.

11.1.6.5 Special values for common ECC domains

Software can sometimes use the PKHA more effectively if the Montgomery Conversion Factor ($R^2 \bmod N$) is either provided or previously used to convert other inputs into [Montgomery form](#). For convenience, the conversion factors for common ECC domains have been computed and published here. Some of the other domain values are provided to aid in definite identification of the domain, in the case that the name is not found or is not an exact match.

The following tables give these values for the q and r modulus values found in ECC domains. These associated Montgomery values are dependent upon the PKHA digit size (16, 32, 64, 128). These tables are for a PKHA with a 128-bit digit.

ECC F_{2^m} requires a c (also called b') parameter for the elliptic curve in place of the b value. [Table 11-71](#) provides these values in addition to the Montgomery values. The b' values are universal and do not change with PKHA digit size.

The following variable definitions apply to both tables. Variable names (q , r , b , c) follow the conventions of IEEE Std 1363.

Name

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org
- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

R

R is the Montgomery factor. Its value is 2^{SD} , where D is the PKHA digit size in bits, and S is the minimum number of digits needed to hold the modulus. As an example, for a modulus of nine bytes (72 bits), R would be

- 2^{80} for a PKHA with digit size of 16 bits
- 2^{96} for a PKHA with digit size of 32 bits
- 2^{128} for a PKHA with digit size of 64 or 128 bits

q

This is the *field-defining* value for the elliptic curve. For F_p curves, it is the prime number used as the modulus for all point arithmetic; it is named p in some other publications. For F_{2^m} curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

L

This is the number of bytes needed to hold q and each of its associated values: $R2modq$, a,b,c , the point coordinates x and y , the result of an ECDH key agreement, etc.

R2modq

This is $R^2 \bmod q$, the Montgomery Conversion Factor when q is the modulus.

r

This is the (usually prime) number which is the order of G , the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named n in some other publications.

N

This is the number of bytes needed to hold r and each of its associated values: $R2modr$, private keys, each of the two components of an ECDSA signature, etc.

R2modr

This is $R^2 \bmod r$, the Montgomery Conversion Factor when r is the modulus.

b / c (b')

b is the coefficient for the x^0 (ones) term in an F_{2^m} elliptic curve equation. Its relationship with c is $b = c^4$. c is sometimes referred to as b' in NXP documentation.

A / a24

$a24$ is the special value derived from the A coefficient for the y^2 term in a Montgomery-form elliptic curve equation. Its relationship with A is $a24 = (A+2)/4$.

The domains in the table are ordered by size.

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits

Name	L	N	
	var		Value (hex, decimal, sums of powers)
secp112r1	14	14	
wtls6	q		0xDB7C2ABF62E35E668076BEAD208B

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			4451685225093714772084598273548427
		R2modq	0x00000000000000000000000000000009
		r	0xDB7C2ABF62E35E7628DFAC6561C5 4451685225093714776491891542548933
		R2modr	0xDA4A43AD7F34245D42B9C948C559
secp112r2	14	14	
		q	0xDB7C2ABF62E35E668076BEAD208B 4451685225093714772084598273548427
		R2modq	0x00000000000000000000000000000009
		r	0x36DF0AAFD8B8D7597CA10520D04B 1112921306273428674967732714786891
		R2modr	0x2049C67E5F79E8C06B7825955374
wtls8	14	15	
		q	0xFFFFFFFFFFFFFFFFFFFFFFFFFDE7 5192296858534827628530496329219559 $2^{112} - 2^9 - 2^4 - 2^3 - 1$
		R2modq	0x00000000000000004667100000000
		r	0x0100000000000001ECEA551AD837E9 5192296858534827767273836114360297
		R2modr	0x00E074FD104C86569DB6C204A52932
secp128r1	16	16	
		q	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF 340282366762482138434845932244680310783 $2^{128} - 2^{97} - 1$
		R2modq	0x00000024000000040000000800000011
		r	0xFFFFFFFFE000000075A30D1B9038A115 340282366762482138443322565580356624661
		R2modr	0x71875047CDD8151626BC6448FADE9BED
secp128r2	16	16	
		q	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF 340282366762482138434845932244680310783 $2^{128} - 2^{97} - 1$
		R2modq	0x00000024000000040000000800000011
		r	0x3FFFFFFFF7FFFFFFFFBE0024720613B5A3 85070591690620534603955721926813660579
		R2modr	0x0EFCA409C09D126A99CD2E9404A3B434
secp160k1	20	21	

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
ansix9p160k1	q		0xFFFAC73 1461501637330902918203684832716283019651637554291 $2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1$
		R2modq	0x000000010000FAA751CE3BE14DB3271500000000
		r	0x0100000000000000000000001B8FA16DFAB9ACA16B6B3 1461501637330902918203686915170869725397159163571
		R2modr	0x00FEFCCD132FD0CDC714AB01D8C89631C2705ACEF9
secp160r1	20	21	
ansix9p160r1 wtls7	q		0xFF7FFFFFFF 1461501637330902918203684832716283019653785059327 $2^{160} - 2^{31} - 1$
		R2modq	0x0000000020000000C00000018000000100000000
		r	0x0100000000000000000000001F4C8F927AED3CA752257 1461501637330902918203687197606826779884643492439
		R2modr	0x0039DE6382E9DD6F6CD1E8F73FDBB68B25BD025FA7
secp160r2	20	21	
ansix9p160r2	q		0xFFFAC73 1461501637330902918203684832716283019651637554291 $2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1$
		R2modq	0x000000010000FAA751CE3BE14DB3271500000000
		r	0x010000000000000000000000351EE786A818F3A1A16B 1461501637330902918203685083571792140653176136043
		R2modr	0x0011F4417D39F295A59EB754BCD6937441214B0C57
wtls9	20	21	
	q		0xFFC808F 1461501637330902918203684832716283019655932313743 $2^{160} - 2^{17} - 2^{16} - 2^{14} - 2^{13} - 2^{12} - 2^{11} - 2^{10} - 2^9 - 2^8 - 2^6 - 2^5 - 2^4 - 1$
		R2modq	0x00000000000000000000002ACB7C068DE15100000000
		r	0x0100000000000000000000001CDC98AE0E2DE574ABF33 1461501637330902918203687013445034429194588307251
		R2modr	0x00E9C98CDCA32E69F8B04BE9F7626715062C0274A3
brainpoolP160r1	20	20	
	q		0xE95E4A5F737059DC60DFC7AD95B3D8139515620F 1332297598440044874827085558802491743757193798159
		R2modq	0xB39451364CC30F3BB4BA7FB8C57E4353532B7BEB
		r	0xE95E4A5F737059DC60DF5991D45029409E60FC09 1332297598440044874827085038830181364212942568457

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
brainpoolP160t1			R2modr 0x9B44D4F65C5494B10E7C2B8DE00DFA539ADFB54B
	20	20	
	q		0xE95E4A5F737059DC60DFC7AD95B3D8139515620F 1332297598440044874827085558802491743757193798159
	R2modq		0xB39451364CC30F3BB4BA7FB8C57E4353532B7BEB
	r		0xE95E4A5F737059DC60DF5991D45029409E60FC09 1332297598440044874827085038830181364212942568457
			R2modr 0x9B44D4F65C5494B10E7C2B8DE00DFA539ADFB54B
P-192	24	24	
secp192r1	q		0xFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
ansix9p192r1			
prime192v1			
ECPRGF192Random	R2modq		0x000000000000000020000000000000003000000000000002
	r		0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831 6277101735386680763835789423176059013767194773182842284081
	R2modr		0x126792C4CEF5D8C5D69C6961CAAF687F01D1770A83134C27
secp192k1	24	24	
ansix9p192k1	q		0xFFFEE37 6277101735386680763835789423207666416102355444459739541047 $2^{192} - 2^{32} - 2^{12} - 2^8 - 2^7 - 2^6 - 2^3 - 1$
	R2modq		0x00002392013C4FD10000000000000000000001000011C9
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D 6277101735386680763835789423061264271957123915200845512077
	R2modr		0x93FB81A6BB194E8A17BBAD83150F5CF8BACF043487AE967C
prime192v2	24	24	
	q		0xFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
	R2modq		0x000000000000000020000000000000003000000000000002
	r		0xFFFFFFFFFFFFFFFFFFFFFFFFE5FB1A724DC80418648D8DD31 6277101735386680763835789423078825936192100537584385056049
	R2modr		0x42397E48235F926256E9DF9FC33E25C1F9B4D79C5BB9D85D
prime192v3	24	24	
	q		0xFF 6277101735386680763835789423207666416083908700390324961279 $2^{192} - 2^{64} - 1$
	R2modq		0x000000000000000020000000000000003000000000000002

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	r		0xFFFFFFFFFFFFFFFFFFFFFFFF7A62D031C83F4294F640EC13 6277101735386680763835789423166314882687165660350679936019
	R2modr		0x3F29B91806AC07FA45CA7C35941AB31CF572723BA8232EEC
brainpoolP192r1	24	24	
	q		0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297 4781668983906166242955001894344923773259119655253013193367
	R2modq		0xB0AFEB32760DC786AE9CFFA62E2228F01E662B2F1375215
	r		0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1 4781668983906166242955001894269038308119863659119834868929
	R2modr		0xB2C3A70B85FD1D24CFB527419F23B9076ECF9194E7FCB8CE
brainpoolP192t1	24	24	
	q		0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297 4781668983906166242955001894344923773259119655253013193367
	R2modq		0xB0AFEB32760DC786AE9CFFA62E2228F01E662B2F1375215
	r		0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1 4781668983906166242955001894269038308119863659119834868929
	R2modr		0xB2C3A70B85FD1D24CFB527419F23B9076ECF9194E7FCB8CE
P-224 secp224r1 ansix9p224r1 wtls12 ECPRGF224Random	28	28	
	q		0xFF000000000000000000000001 26959946667150639794667015087019630673557916260026308143510066298881
	R2modq		0xFFFFFFFFFFFFFFFFE0000000FFFFFFFF00000000FFFFFFFF00000001
	r		0xFF16A2E0B8F03E13DD29455C5C2A3D 26959946667150639794667015087019625940457807714424391721682722368061
	R2modr		0xB1E979616AD15F7CD9714856ABC8FF5931D63F4B29947A695F517D15
secp224k1 ansix9p224k1	28	29	
	q		0xFFE56D 26959946667150639794667015087019630673637144422540572481099315275117 $2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^9 - 2^7 - 2^4 - 2^1 - 1$
	R2modq		0x00
	r		0x010001DCE8D2EC6184CAF0A971769FB1F7 26959946667150639794667015087019640346510327083120074548994958668279
	R2modr		0x00993FF72BB882BD88BBFF32E48BE0320816F60AF534CE24FBEC9FEAA0
brainpoolP224r1	28	28	
	q		0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF 22721622932454352787552537995910928073340732145944992304435472941311
	R2modq		0x2B3D40DD7EA5577C77E3D7E476F9EE46F691D1053BFC9BF6B3D58FF
	r		0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var	Value (hex, decimal, sums of powers)	
brainpoolP224t1		22721622932454352787552537995910923612567546342330757191396560966559	
	R2modr	0x5234FE17B4581327A35371E1F0F40A07BAB96B21F3D67605E0D86B49	
	28	28	
	q	0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF 22721622932454352787552537995910928073340732145944992304435472941311	
	R2modq	0x2B3D40DD7EA5577C77E3D7E476F9EE46F691D1053BFBC9BF6B3D58FF	
prime239v1	r	0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F 22721622932454352787552537995910923612567546342330757191396560966559	
	R2modr	0x5234FE17B4581327A35371E1F0F40A07BAB96B21F3D67605E0D86B49	
	30	30	
	q	0x7FFF80000000000007FFFFFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699 839	
	R2modq	0x0000000000000000500000000005FFFFFFFFFFFC0000000000800000000	
prime239v2	r	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFF9E5E9A9F5D9071FBD1522688909D0B 883423532389192164791648750360308884807550341691627752275345424702807 307	
	R2modr	0x2BE4B1BE15BDEB5DF3096A7BE4944FA0CB87DC9852A129052EC789ED615B	
	30	30	
	q	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFF800000000007FFFFFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699 839	
	R2modq	0x0000000000000000500000000005FFFFFFFFFFFC0000000000800000000	
prime239v3	r	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF800000CFA7E8594377D414C03821BC582063 883423532389192164791648750360308886392687657546993855147765732451295 331	
	R2modr	0x76CF025EBF73DDE8A5D15F0C7C29FF23EED0AE5C096A0D32ABC4B16B765	
	30	30	
	q	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFF800000000007FFFFFFFFFFFFF 883423532389192164791648750360308885314476597252960362792450860609699 839	
	R2modq	0x0000000000000000500000000005FFFFFFFFFFFC0000000000800000000	
P-256 secp256r1 ansix9p256r1	r	0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFF975DEB41B3A6057C3C432146526551 883423532389192164791648750360308884771190369765922550517967171058034 001	
	R2modr	0x11500EB94E46F16737BEB7A266592D93C18845A5EB3F814C07B00EA6ACF5	
	32	32	
	q	0xFFFFFFFF000000010000000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF	

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	r		0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856 A7 768849563970453442208097466290016490927375317844145295387555190630635 36359079
	R2modr		0x0B25F1B9C32367629B7F25E76C815CB0F35D176A1134E4A0E1D8D8DE3312FC A6
GOSTR3410- CryptoPro-A	32	32	
	q		0xFF D97 115792089237316195423570985008687907853269984665640564039457584007913 129639319 $2^{256} - 2^9 - 2^6 - 2^5 - 2^3 - 1$
	R2modq		0x0005CF11
	r		0xFF6C611070995AD10045841B09B761B89 3 115792089237316195423570985008687907853073762908499243225378155805079 068850323
	R2modr		0x551FE9CB451179DBF74885D08A3714C6FB07F8222E76DD529AC2D7858E79A46 9
GOSTR3410- CryptoPro-B	32	32	
	q		0x8000C99 578960446186580977117854925043439539266349923328202820197287920039565 64823193
	R2modq		0x0027ACDC4
	r		0x8000000000000000000000000000000015F700CFFF1A624E5E497161BCC8A198F 578960446186580977117854925043439539271021331602558268200688444960877 32066703
	R2modr		0x09D1D2C4E50824664A2E7E2F6882CF102A3104A7EA43E85529B721F4E6CD782 3
GOSTR3410- CryptoPro-C	32	32	
	q		0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759 B 703900853520833051995477180190184378410795166300451804712843468437056 33502619
	R2modq		0x807A394EDE097652186304212849C07B1017BB39C2D346C5409973B4C427FCE A
	r		0x9B9F605F5A858107AB1EC85E6B41C8AA582CA3511EDDFB74F02F3A6598980B B9 703900853520833051995477180190184378409208826471640810353226014583522 98396601
	R2modr		0x7AA61B49A49D4759C67E5D0EE96E8ED304FDA8694AFDA24BE94FAAB66ABA1 80E

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	var	Value (hex, decimal, sums of powers)
brainpoolP320r1	40	40		
		q		0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC 28FCD412B1F1B32E27 176359332223916635416190984244601952088951277271951519277296041528864 0868802149818095501499903527
		R2modq		0x403A056E97A2C09CDBC21A817727A6D4C53AA9675C4000F410260B24EF82144 8B4E04050AD8B400D
		r		0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658 E98691555B44C59311 176359332223916635416190984244601952088951277271768606376068612401678 4784845843468355685258203921
		R2modr		0xA70D3DB6D2668A953B5396FEB00B951407AD1666458BCC15DD860295E4A3F1 A0B9B6AC1D66163F17
brainpoolP320t1	40	40		
		q		0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC 28FCD412B1F1B32E27 176359332223916635416190984244601952088951277271951519277296041528864 0868802149818095501499903527
		R2modq		0x403A056E97A2C09CDBC21A817727A6D4C53AA9675C4000F410260B24EF82144 8B4E04050AD8B400D
		r		0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658 E98691555B44C59311 176359332223916635416190984244601952088951277271768606376068612401678 4784845843468355685258203921
		R2modr		0xA70D3DB6D2668A953B5396FEB00B951407AD1666458BCC15DD860295E4A3F1 A0B9B6AC1D66163F17
P-384	48	48		
secp384r1 ansix9p384r1 ECDSA-384		q		0xFF FFFFFFFFFFFFFFFF00 394020061963944792122790401001436138050797392704654466679482934042457 21771496870329047266088258938001861606973112319
ecp_group_20 ECPRGF384Random		R2modq		0x0010000002000000000FFFFFFE000000000 00000200000000FFFFFFFE00000001
		r		0xFFC7634D81F4372 DDF581A0DB248B0A77AECEC196ACCC52973 394020061963944792122790401001436138050797392704654466679469052796276 59399113263569398956308152294913554433653942643
		R2modr		0x0C84EE012B39BF213FB05B7A28266895D40D49174AAB1CC5BC3E483AFCB829 47FF3D81E5DF1AA4192D319B2419B409A9
brainpoolP384r1	48	48		
		q		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB7112 3ACD3A729901D1A71874700133107EC53

Table continues on the next page...

Table 11-70. Special Values for common ECC F_p domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			216592707701193161730692368423326049797961163870176486000816185038210 89934025961822236561982844534088440708417973331
	R2modq		0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC994089 9535283343D7FD965087CEFFF40B64BDE
	r		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A 7CF3AB6AF6B7FC3103B883202E9046565 216592707701193161730692368423326049797961163870176486000756452748216 1150135851553796269511736890325229601718723941
	R2modr		0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688 A37264E202F2B6B6EAC4ED3A2DE771C8E
brainpoolP384t1	48	48	
	q		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB7112 3ACD3A729901D1A71874700133107EC53 216592707701193161730692368423326049797961163870176486000816185038210 89934025961822236561982844534088440708417973331
	R2modq		0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC994089 9535283343D7FD965087CEFFF40B64BDE
	r		0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A 7CF3AB6AF6B7FC3103B883202E9046565 216592707701193161730692368423326049797961163870176486000756452748216 1150135851553796269511736890325229601718723941
	R2modr		0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688 A37264E202F2B6B6EAC4ED3A2DE771C8E
brainpoolP512r1	64	64	
	q		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA7033 08717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A 48F3 894896220765023255165660281515915342216260964409835451134459718720005 701041355243991793430419195694276544653038642734593796389430992392853 6070534607816947
	R2modq		0x3C4C9D05A9FF6450202E19402056EECCA16DAA5FD42BFF8319486FD8D58980 57E0C19A7783514A2553B7F9BC905AFFD3793FB1302715790549AD144A6158F205
	r		0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA7033 0870553E5C414CA92619418661197FAC10471DB1D381085DDADD58796829CA90 069 894896220765023255165660281515915342216260964409835451134459718720005 701041341852837898173064352495985745139837002928058309421561388204397 3354392115544169
	R2modr		0xA794586A718407B095DF1B4C194B2E56723C37A22F16BBDFD7F9CC263B790D E3A6F230C72F0207E83EC64BD033B7627F0886B75895283DDDD2A3681ECDA816 71
brainpoolP512t1	64	64	

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
wtls4			$x^{113} + x^9 + 1$
	R2modq		0x0000000000000000000000001000040000000
	b		0xE8BEE4D3E2260744188BE0E9C723
	c		0x0173E834AF28EC76CB83BD8DFEB2D5
	r		0x0100000000000000D9CCEC8A39E56F 5192296858534827689835882578830703
	R2modr		0x002D02609ABE76F866BDCE5B3F9BCC
sect113r2	15	15	
	q		0x02000000000000000000000000000201 $x^{113} + x^9 + 1$
	R2modq		0x0000000000000000000000001000040000000
	b		0x95E9A9EC9B297BD4BF36E059184F
	c		0x0054D9F03957174A32329167D7FE71
	r		0x01000000000000000108789B2496AF93 5192296858534827702972497909952403
R2modr		0x00471CB662E29CB41ABC888E16FF49	
wtls1	15	14	
	q		0x02000000000000000000000000000201 $x^{113} + x^9 + 1$
	R2modq		0x0000000000000000000000001000040000000
	b		0x01
	c		0x00000000000000000000000000000001
	r		0xFFFFFFFFFFFFFFFFDBF91AF6DEA73 5192296858534827627896703833467507
R2modr		0x9A1AB7E0A60C212FBD48A8239130	
sect131r1	17	17	
	q		0x08000000000000000000000000000010D $x^{131} + x^8 + x^3 + x^2 + 1$
	R2modq		0x014E8000000000000000000000000100029
	b		0x0217C05610884B63B9C6C7291678F9D341
	c		0x03DB89B405E491160E3B2F07B0CE20B37E
	r		0x040000000000000000000023123953A9464B54D 1361129467683753853893932755685365560653
R2modr		0x0083F5BAB162853801B911D62ACCA3D2FE	
sect131r2	17	17	
	q		0x08000000000000000000000000000010D $x^{131} + x^8 + x^3 + x^2 + 1$

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
	R2modq		0x014E8000000000000000000000000100029
	b		0x04B8266A46C55657AC734CE38F018F2192
	c		0x07CBB9920D71A48E099C38D71DA6490EB1
	r		0x0400000000000000016954A233049BA98F 1361129467683753853879535043412812867983
	R2modr		0x026A119CE9028D45E37D085A777D57E9EC
Oakley 3	20	-	
	q		0x08000000000000000000000040000000000001 $x^{155} + x^{62} + 1$
	R2modq		0x0000100000002000000000000000800000010000
	b		0x07338F
B-163 ansix9t163r2 sect163r2 EC2NGF163Random	c		0x00311000000223A000C4474000088E8000111D1D
	21	21	
	q		0x080000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
	R2modq		0x00000000000000000000000000000001F7144800000
	b		0x020A601907B8C953CA1481EB10512F78744A3205FD
K-163 ansix9t163k1 sect163k1 EC2NGF163Koblitz wtls3	c		0x072C4E1EF7CB2F3A035D33104294159609138BB404
	r		0x040000000000000000000000292FE77E70C12A4234C33 5846006549323611672814742442876390689256843201587
	R2modr		0x0074E4DA7AC0AB18383B17A6264E073392CB87C99F
	21	21	
	q		0x080000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
sect163r1 ansix9t163r1	R2modq		0x00000000000000000000000000000001F7144800000
	b		0x01
	c		0x00000000000000000000000000000000000001
	r		0x04000000000000000000000020108A2E0CC0D99F8A5EF 5846006549323611672814741753598448348329118574063
	R2modr		0x0015BD98691E2D7635612A20F570BF080EE3891FED
	21	21	
	q		0x080000000000000000000000000000000000C9 $x^{163} + x^7 + x^6 + x^3 + 1$
	R2modq		0x00000000000000000000000000000001F7144800000
	b		0x0713612DCDDCB40AAB946BDA29CA91F73AF958AFD9
	c		0x05ED403ED58EB45B1CCECA0F4F61655549861BE052
	r		0x03FFFFFFFFFFFFFFFFFFFFFFFF48AAB689C29CA710279B

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
			5846006549323611672814738465098798981304420411291
	R2modr		0x00D6B757B1F380185CE0F1003784C3BBDAFF28FE67
wtls5	21	21	
	q		0x0800107 $x^{163} + x^8 + x^2 + x^1 + 1$
	R2modq		0x000838AB5800000
	b		0xC9517D06D5240D3CFF38C74B20B6CD4D6F9DD4D9
	c		0x0453E1E4B7291F5C2D53CE18483F007081E7EA26EC
	r		0x0400E60FC8821CC74DAEAF1 5846006549323611672814741626226392056573832638401
	R2modr		0x0066A2D87A8D9BBFA06D197538A1A76438A08CA52
Oakley 4	24	-	
	q		0x0200200000000000000001 $x^{185} + x^{69} + 1$
	R2modq		0x000000100000400000000000080000000000000000000000000000000000
	b		0x1EE9
	c		0x000000000000300000018000C00C000000638030001C0009
sect193r1 ansix9t193r1	25	25	
	q		0x02008001 $x^{193} + x^{15} + 1$
	R2modq		0x00000000001000000400
	b		0xFDFB49BFE6C3A89FACADAA7A1E5BBC7CC1C2E5D831478814
	c		0x0167B35EB4313F263D0F7A3D5036F0A0A3C980D40E5A053ED2
	r		0x0100C7F34A778F443ACC920EBA49 6277101735386680763835789423269548053691575186051040197193
	R2modr		0x00578029E7EA1A1F0E8E70080E018BE464F5A2A1DE3134D957
sect193r2 ansix9t193r2	25	25	
	q		0x02008001 $x^{193} + x^{15} + 1$
	R2modq		0x00000000001000000400
	b		0xC9BB9E8927D4D64C377E2AB2856A5B16E3EFB7F61D4316AE
	c		0x006989FE6BFE30EDDC3244269F3AAD18D66CF3DB3E3302FAA8
	r		0x010015AAB561B005413CCD4EE99D5 627710173538668076383578942331495536243729822279840143829
	R2modr		0x006D96955E0EBA3C05CF6334F4D5D81F3BC161BAC38F9F4014
B-233 sect233r1	30	30	
	q		0x0200400000000000000001

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	
			var
ansix9t233r1 EC2NGF233Random wtls11			$x^{233} + x^{74} + 1$
	R2modq		0x000000000000040000000000000000000000000000000040000000000000
	b		0x66647EDE6C332C7F8C0923BB58213B333B20E9CE4281FE115F7D8F90AD
	c		0x0007D5EF4389DFF11ECDBA39C30970D3CE35CEBBA58473F64B4DC0F2686C
	r		0x01000000000000000000000000000013E974E72F8A6922031D2603CFE0D7 690174634679056378743475586227702555583981273734501355537938363448546 3
R2modr		0x006AB044AA57CDD6D0CC9138B004578CD5EFE7E89545CDAA1BA1C26DD4D1	
K-233	30	29	
sect233k1 ansix9t233k1 EC2NGF233Koblitz wtls10	q		0x0200040000000000000001 $x^{233} + x^{74} + 1$
	R2modq		0x000000000004000000000000000000000000000000000040000000000000
	b		0x01
	c		0x0001
	r		0x80000000000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF 345087317339528189371737793113851276057094098886225212632808702474134 3
R2modr		0x59BEBED80293C813EEB5B58A0AF7E3EB91DB9A5B861710AC1009468BB6	
sect239k1 ansix9t239k1	30	30	
	q		0x800000000000000000040001 $x^{239} + x^{158} + 1$
	R2modq		0x00000000000100000000000000000008000000000000000000440000000
	b		0x01
	c		0x0001
	r		0x200000000000000000000000000005A79FEC67CB6E91F1C1DA800E478A5 220855883097298041197912187592864814948216561321709848887480219215362 213
R2modr		0x183E8C975E5EA68E203395FBEC1187B0F40DFECA2CE64F17F77925590A73	
B-283	36	36	
sect283r1 ansix9t283r1 EC2NGF283Random	q		0x08000 010A1 $x^{283} + x^{12} + x^7 + x^5 + 1$
	R2modq		0x0000000000004011004000 00000
	b c		0x027B680AC8B8596DA5A4AF8A19A0303FCA97FD7645309FA2A581485AF6263E3 13B79A2F5 0x03D8C93D3B0EA81D9294034D7EE3135D0AC5FC8D9CB0276F7211F880F0D81C A4C6E87B38

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	
	var		Value (hex, decimal, sums of powers)
		r	0x03FFF90399660FC938A90165B042A7 CEFADB307 777067556890291628367784762729407562656962592437690488910919652677004 4277787378692871
		R2modr	0x03DC043FDEC225C26FE29C27FE358A967FF29805F6072C3A685782B384E2C99 70CC05EC9
K-283	36	36	
sect283k1 ansix9t283k1 EC2NGF283Koblitz		q	0x0800 010A1 $x^{283} + x^{12} + x^7 + x^5 + 1$
		R2modq	0x00000000000000004011004000 00000
		b	0x01
		c	0x00 00001
		r	0x01FFE9AE2ED07577265DFF7F94451E 061E163C61 388533778445145814183892381364703781328481173379306132429587499752981 5829704422603873
		R2modr	0x0003A439439C2C4DC806D328383A44329425C1F7C8C2490D2C9BB804ED84B95 BC349DC9B
B-409	52	52	
sect409r1 ansix9t409r1 EC2NGF409Random		q	0x0200 000000000000080000000000000000001 $x^{409} + x^{87} + 1$
		R2modq	0x000000001000 000000000000000000000000000000000
		b	0x21A5C2C8EE9FEB5C4B9A753B7B476B7FD6422EF1F3DD674761FA99D6AC27C 8A9A197B272822F6CD57A55AA4F50AE317B13545F
		c	0x0149B8B7BEBD9B63653EF1CD8C6A5DD105A2AAAC36FE2EAE43CF28CE1CB7 C830C1ECDBFA413AB07FE35A57811AE4F88D30AC63FB
		r	0x0100 3307BE5FA47C3C9E052F838164CD37D9A21173 661055968790248598951915308032771039828404682964281219284648798304157 774827374805208143723762179110965979867288366567526771
		R2modr	0x00A152B06BE1B84F42C5B0030C7628C0F64C76A9D2367C03FBD9AFFA0603C9 32F7AA85C4C7F7547FDB7E134A10755B94E091D1C4
K-409	52	51	
sect409k1 ansix9t409k1 EC2NGF409Koblitz		q	0x0200 000000000000080000000000000000001 $x^{409} + x^{87} + 1$

Table continues on the next page...

Table 11-71. Special Values for common ECC F_{2^m} domains when PKHA digit size is 128 bits (continued)

Name	L	N	Value (hex, decimal, sums of powers)
			var
			R2modq
			b
			c
			r
			R2modr
B-571	72	72	
sect571r1		q	
ansix9t571r1			
EC2NGF571Random		R2modq	
		b	
		c	
		r	
		R2modr	
K-571	72	72	
sect571k1		q	
ansix9t571k1			
EC2NGF571Koblitz		R2modq	
		b	
		c	

Table continues on the next page...

Table 11-73. Special Values for common ECT MOD (Edwards curves) domains when PKHA digit size is 128 bits

Name	L	N	
		var	Value (hex, decimal, sums of powers)
edwards25519	32	32	
		q	0x7FFF FED $2^{255}-19$
		R2modq	0x0005A4
		r	0x10014DEF9DEA2F79CD65812631A5CF5D3ED 723700557733226221397318656304299424085711635937990760600195093828545 4250989
	R2modr	0x0399411B7C309A3DCEEC73D217F5BE65D00E1BA768859347A40611E3449C0F0 1	

11.2 Kasumi f8 and f9 hardware accelerator(KFHA) functionality

Kasumi is a radio-interface, cryptographic algorithm set for LTE (two other sets are SNOW and ZUC). Kasumi 3G f8 is the encryption algorithm, and Kasumi 3G f9 is the authentication/integrity algorithm within this cryptographic set. The KFHA (Kasumi f8/9 hardware accelerator) implements both the f8 and f9 modes of the Kasumi cryptography algorithm. The KFHA CHA is controlled via the class 1 CHA registers. Because it is not possible to own two Class 1 CHAs simultaneously, it is not possible to "snoop" between two KFHAs or between a KFHA and an AESA.

11.2.1 KFHA use of the Mode Register

The KFHA uses the Mode Register as follows:

- The Encrypt/Decrypt field of the Mode Register is not used by KFHA, because there is no difference between encrypt and decrypt in Kasumi. However, this bit should be set to indicate whether encryption or decryption is being performed so that the Performance Counter registers properly count events (see the description of the PM registers in the register descriptions section of this document).
- The ICV check field of the Mode Register is used to enable/disable ICV checking.
- The Algorithm (ALG) field of the Mode Register must be set to "Kasumi" to activate the KFHA.

- The Additional Algorithm Information (AAI) field must be set to either "f8" or "f9". In addition, the bits for either GSM or EDGE (but not both) may be OR'ed into the values for f8 or f9 to enable GSM or EDGE.
- The Algorithm State (AS) field of the Mode Register is used to select between the operations described in this table.

Table 11-74. Mode Register[AS] operation selections in KFHA

Mode	Indication
INIT	KFHA starts with an initialization process. This puts the KFHA into an initial state needed at the start of an operation
INIT/FINALIZE	KFHA starts with an initialization process, and at the end of the message process, any selected padding will occur
FINALIZE	KFHA starts with no initialization process, and at the end of the message process, any selected padding will occur
UPDATE	KFHA starts with no initialization process

11.2.2 KFHA use of the Context Register

The KFHA uses the Context Register as follows:

- The 512-bit Context Register is used as the working registers for KFHA.
- The count, bearer, direction, ca, ce, FRESH, and ICV_in fields within the Context Register should be loaded by software before INIT or INIT/FINALIZE operations.
- The Context Register is continually updated during processing, so at the end of an operation, it reflects the current state of the KFHA. The bits are assigned as shown in these tables.

Table 11-75. Context usage in Kasumi 3G modes

Mode	DWord number	Initial input definition	Context switching definition	Final result definition
Kasumi 3G f8 mode	0	count, bearer, direction, ca, ce	count, bearer, direction ca, ce	-
	1	-	-	-
	2	-	-	-
	3	-	-	-
	4	-	f8 C register	-
	5	-	f8 B register	-
	6	-	f8 A register	-
	7	-	core data out	-
Kasumi 3G f9 mode	0	count, bearer, direction, ca, ce	count, bearer, direction, ca, ce	count, bearer, direction, ca, ce
	1	FRESH, ICV_in	FRESH, ICV_in	FRESH, ICV_in
	2	-	f9 C register	ICV_out

Table continues on the next page...

Table 11-75. Context usage in Kasumi 3G modes (continued)

Mode	DWord number	Initial input definition	Context switching definition	Final result definition
	3	-	f9 A register	-

Table 11-76. Format of context DWord 0 in Kasumi 3G f8 and f9 modes

Bits 0-31	Bits 32-36	Bit 37	Bits 38-47	Bits 48-63
count	bearer	direction	ca	ce

Table 11-77. Format of context DWords 1 and 2 in Kasumi 3G f9 mode

DWord number	Bits 0-31	Bits 32-63
1	FRESH	ICV_in
2 (output only)	ICV_out	-

11.2.3 KFHA use of the Key Register

The KFHA uses the Key Register as follows:

- The Key Register contains the 16-byte key used by the KFHA, which is placed in bits [127:0]. (The first key byte is in Key Register bits [127:120], the second key byte is in key [119:112], and so on).
- The Key Size Register must be programmed to a value of 16. Any other value produces a KFHA key size error.

11.2.4 KFHA use of the Data Size Register

The KFHA does not need to know the total size of the message being processed. It only needs to know how many bits will be processed out of the last 64-bit DWord of the message. Therefore, it uses only the 3 LSB of the Data Size Register (PDB 2:0]) and the NUMBITS field (bits 63:61 of the Data Size Register) during message processing.

11.2.5 KFHA error conditions

Errors that can occur while operating the KFHA CHA include the following:

Table 11-78. KFHA error conditions and results

Error condition	Result
After a FINALIZE or INIT/FINALIZE operation completes and the Mode Register is cleared, if the mode is re-written to UPDATE or FINALIZE (with no DECO reset in between)	Mode error NOTE: The assumption is that any "Final" packet requires an initialization or a reset for a following packet.
After an INIT or UPDATE operation (that is, operations that are not "Final") completes and the Mode Register is cleared, if the mode is re-written to INIT or INIT/FINALIZE (with no DECO reset in between)	Mode error NOTE: The assumption is that any "Initial" packet would only follow a "Final" packet or a reset.
If both GSM and EDGE bits are set in the AAI field of the mode register while KFHA is operating	Mode error
When operating KFHA, neither F8 or F9 are selected in the mode register AAI field	Mode error
If the KFHA is operated with a key size of any value other than 16 bytes	Key size error
If any input data is received by KFHA that is not type "Message"	Data sequence error
If KFHA is operating in F9 mode with ICV checking enabled in the Mode Register	ICV mismatch produces an ICV check error

11.3 Data encryption standard accelerator (DES) functionality

DES performs encryption and decryption on 64-bit values using the algorithm found in FIPS46-3. The DES module in SEC supports both single- and triple-DES functionality and ECB, CBC, CFB, and OFB modes as well as key parity checking in compliance with the DES specification. DES is controlled from the class 1 CHA registers.

11.3.1 DESA use of the Mode Register

The DESA uses the Mode Register as follows:

- The encryption field (ENC) controls whether DESA is encrypting or decrypting data.
- The Algorithm State (AS) field is not used to affect DESA functionality and should be set to zero at all times.
- The Additional Algorithm Information field (AAI) specifies the mode DESA runs. The supported modes are electronic code book (ECB), cipher block chaining (CBC), cipher feedback (CFB-8), and output feedback (OFB), described as follows:
 - ECB (0x20h) mode is a confidentiality mode that features, for a given key, the assignment of a fixed ciphertext block to each plaintext block (analogous to the assignment of code words in a codebook).

- CBC (0x10h) mode is a confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. CBC mode requires an IV to combine with the first plaintext block. The IV does not need to be secret, but it must be unpredictable.
- CFB (0x30h) mode is a confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input-block.
- OFB (0x40h) mode is a confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- Key parity checking for DESA that checks for odd parity within each byte of the key is enabled with a value of (0x80h) in the AAI field.
- The algorithm field (ALG) must be programmed to DES (0x20h) or 3DES (0x21h).

11.3.2 DESA use of the Key Register

The DESA uses the Key Register as follows:

- The Key Register contains the 8-, 16-, or 24-byte key that is used during permutation in all DES modes.
- The DES specification defines the key as having odd parity in each byte.
- Key parity can be verified using the correct mode setting.

11.3.3 DESA use of the Key Size Register

DESA uses the Key Size Register as follows:

- Key size can be either 8, 16, or 24 bytes.
- A key size of 8 is valid only in single-DES mode.
- Values of 16 and 24 bytes can be used only in triple-DES mode.
- An illegal key size error is generated when in single-DES mode with a key size other than 8 or when in triple-DES mode with a key size other than 16 or 24.

11.3.4 DESA use of the Data Size Register

The DESA uses the Data Size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- All DES modes except OFB expect to process data that is a multiple of 8 bytes and generates an error if the data size written is not an 8-byte multiple.
- This register must be written to start data processing.
- Because writing to the Data Size Register causes the written value to be added to the previous value in the register, the register may be written multiple times while data is being processed in order to increase the amount of input data that will be processed.

11.3.5 DESA Context Register

The DESA uses the Context Register as follows:

- For CBC, OFB, and CFB modes, the initialization vector is written to and read from the DESA Context Register.
- The value of this register changes as a result of the encryption process and reflects the context of DESA.
- DESA uses the first eight bytes of the Context Register to hold the beginning and final IV value for the CBC, OFB, and CFB modes. The bits are assigned as follows:
Context DWord0: IV = `desa_context[63:0]`

11.3.6 Save and store operations in DESA context data

DESA is able to process data in chunks by saving the intermediate IV from the Context Register after each chunk of data and restoring the IV and key to the correct registers before processing any subsequent chunks of data.

11.4 Cyclic-redundancy check accelerator (CRCA) functionality

CRCA performs 1- to 32-bit cyclic redundancy code computation. Cyclic redundancy checks are a common algorithm for providing additional integrity check bits that are included with data in danger of corruption. A typical usage is to include a CRC for each packet transmitted over a network and then recompute this CRC at the destination to verify that the received packet is uncorrupted. CRC is controlled via the class 2 CHA registers

11.4.1 CRCA modes of operation

The CRCA module supports the following primary modes of operation:

- IEEE 802 CRC32 protocol mode
- iSCSI (IETF 3385) CRC32c protocol mode
- Dynamic polynomial custom mode

The protocol-specific modes automatically use the correct polynomial. The custom mode takes the desired polynomial from the Key Register. All modes support either raw (DOS and DOC equal one), default processing, or any combination of the use of DIS, DOS, DOC, and IVZ control bits. Default processing involves bit-swapping, byte-swapping, complementing the CRC result, and setting initial value to FFFFFFFFh for specifications compliance. Raw mode (with DOS and DOC set) does not modify the CRC result in any way and is commonly used for partial CRC calculations. ICV checking can be performed in all modes.

11.4.2 CRCA use of the Mode Register

The CRCA uses the Mode Register as follows:

- The Mode Register is used to program the function of the CRCA.
- The ICV field selects whether a comparison between the computed CRC and the provided CRC should be performed. When enabled, ICV check compares a CRC calculated across the message against the received ICV. The comparison is performed after the bit manipulations controlled by the DIS, DOS, DOC, IVZ bits are performed.
- The Algorithm State (AS) field controls two features in the CRC module, namely the ability to load context and inhibit mangling output data. The LSB of the AS field selects whether to load the context value that was written into the class 2 Context Register. The MSB of the AS field is used to inhibit both the output swapping and the 1's complement of data when the current mode of operation does not result in a final computation of the CRC. Programming the initialize or update modes overrides the functionality of the DOS and DOC bits. This table is a synopsis of the 2-bit AS field and its functionality.

Table 11-79. 2-bit AS functionality and synopsis

Value	Phase	Actions
00	Update	<ul style="list-style-type: none"> • Loads context • Does not mangle output data

Table continues on the next page...

Table 11-79. 2-bit AS functionality and synopsis (continued)

Value	Phase	Actions
01	Initialize	<ul style="list-style-type: none"> Does not load context Does not mangle output data
10	Finalize	<ul style="list-style-type: none"> Loads context Mangles output data
11	Init/Finalize	<ul style="list-style-type: none"> Does not load context Mangles output data

- The Algorithm field (ALG) must be set to CRC.
- The Additional Algorithm Information (AAI) field controls the modes and functionality of the CRC module. The lower 4 bits select one of the three modes that the CRC operates in while the upper 5 bits allow individual control over how the CRC engine manipulates its input and output data. The lower 4-bit definitions for the mode values are described in this table.

Table 11-80. Lower 4 bits AAI functionality

Value	Mode	Actions
0001	IEEE 802 mode	The CRC32 algorithm is performed using the polynomial 04C11DB7h.
0010	iSCSI mode (also called IETF 3385)	The CRC32c algorithm is performed using the polynomial 1EDC6F41h.
0100	Dynamic custom mode	<ul style="list-style-type: none"> The CRC remainder is computed using the polynomial programmed into the Key Register. The polynomial can be 1-32 bits. The polynomial must be left justified.

- In compliance with the IEEE 802 and iSCSI CRC implementations, the CRCA result is bit-swapped, byte-swapped, and complemented before it is output. The upper 5 bits of the AAI field allow the user to individually control the input bit-swapping (DIS), the output bit-swapping (DOS), complementing of output data (DOC), and initial value zero (IVZ).

Table 11-81. Upper 5 bits AAI functionality

Value	Name	Description	Encodings
00001	DIS	Don't Input Swap	<ul style="list-style-type: none"> 0: Input data is bit-swapped 1: Turns off swapping of the input data bits
00010	DOS	Don't Output Swap	<ul style="list-style-type: none"> 0: Output data is bit and byte-swapped 1: Turns off bit-/byte-swapping of the output data
00100	DOC	Don't Output Complement	<ul style="list-style-type: none"> 0: Output data is complemented 1: Turns off complementing the CRC output data
01000	IVZ	Initial Value Zero	<ul style="list-style-type: none"> 0: Initial CRC value is FFFFFFFFh 1: Initial CRC value is 00000000h

11.4.3 CRCA Key Register

The CRCA uses the Key Register as follows:

- The Class 2 Key Register stores the polynomial for custom mode.
- Any size polynomial may be used, up to 32 bits.
- The key must be left-justified in the Key Register.

11.4.4 CRCA Key Size Register

The CRCA uses the Key Size Register as follows:

- For IEEE 802 CRC32 protocol mode and iSCSI (IETF 3385) CRC32c protocol mode, it is not necessary to write to this register, because the polynomial size is clearly fixed by the algorithm.
- For dynamic custom polynomial mode, write a 4-byte value into the Key Size register to specify the size of the polynomial.

11.4.5 CRCA Data Size Register

The CRCA uses the Data Size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- This register must be written to start data processing.

11.4.6 CRCA Context Register

The CRCA uses the Context Register as follows:

- The 32-bit Context Register is used to load partial CRCs. This register can be written with an intermediate CRC result prior to processing any data. Once processing is complete, the CRC result is available from this register. The reset state of this register is all ones, because this allows the CRC32 algorithm to detect bit errors in the leading zeros of a message.
- CRC uses the first word of the Context Register to load a CRC value for the engine to start from a previous, partial calculation. This register also holds the intermediate or final CRC when calculations are done. The bits are assigned as follows:
- Context DWord0: CRC register = {00000000h, crca_context[0:31]}

11.4.7 Save and restore operations in CRCA context data

To run CRC for multiple, related groups of data, it is necessary to program CRC in the correct initial mode (either initialize or update) and then to save the intermediate CRC value from the Context Register to be used for all subsequent data. When it is necessary to continue from the previous point, the intermediate CRC value must be loaded into the Context Register before the Data Size Register is written.

CRC must be programmed in either update or finalize modes (AS field) to continue from the previous partial calculation. For CRC to compute its final value, the module would need to be programmed in the final mode (AS field). This causes CRC to both swap and complement the output and provide the calculated CRC in the Context Register. If a custom polynomial was used for the above calculations, it would need to be loaded into the Key Register each time calculations were continued.

11.5 Random-number generator (RNG) functionality

The RNG generates cryptographically-strong, random data. SEC's RNG utilizes a true random-number generator (TRNG) as well as a deterministic random-bit generator (DRBG) to achieve both true randomness and cryptographic strength.

The random numbers generated by the RNG are intended for direct use as secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms. Note that before data can be obtained from the RNG, it must be instantiated in a particular mode by executing the appropriate descriptor. Also, a descriptor must be executed to load the JDKEK, TDKEK and TDSK registers with data from the RNG.

11.5.1 RNG features summary

The RNG module includes these distinctive features:

- Complete implementation of DRBG_Hash (SP800-90A) using SHA-256.
- Support for two state handles.
- Built-in entropy source conforming to SP800-90B and BSI AIS/31.
- Integrated entropy source for instantiating and re-seeding the DRBG.
- The RNG may be accessed through a register interface for test purposes.
- IP global interface. The RNG system clock and resets are controlled through the IP global interface.

- Interface to supply SEC with random data as required for descriptors, and to initialize key encryption key and Trusted Descriptor Signing Key Registers.
- Random-number stream obtainable using job descriptor.
- Random numbers automatically supplied as padding when needed by protocols.
- External interface to furnish random data to other modules such as SecMon.

11.5.2 RNG functional description

While the RNG consists of several, functional sub-modules, its overall functionality can be easily described from the top level in terms of a few functional operations. These operations are seed generation and random number generation. Each of these operations require coordination of the RNG's true random-number generator (TRNG) and deterministic random-bit generator (DRBG). TRNG creates real entropy (seed generation) and DRBG generates cryptographically strong data using this entropy (random-number generation).

11.5.2.1 RNG state handles

The RNG in SEC implements two state handles. Each state handle is:

- A completely independent virtual RNG.
- Instantiated independently in deterministic or nondeterministic mode, and with or without prediction resistance
- Seeded (or reseeded) independently with independent entropy

Thereafter, each state handle maintains an independent context for the RNG's deterministic random bit generator (DRBG).

Note that the JDKEK, TDKEK, TDSK and ZMK (if the ZMK is set for hardware programming) and the random data to seed the AES Differential Power Analysis protection (if implemented in the AESA) are initialized by data drawn from State Handle 0, and that any random padding required by SEC's built-in protocols is also drawn from State Handle 0. Because this data may be confidential, there are special security features to ensure that State Handle 0 is not inadvertently or maliciously instantiated in deterministic mode when it should have been instantiated in nondeterministic mode. See the discussion of the RNGSH0 and RANDDPAR fields in the Security Configuration Register.

11.5.2.2 RNG NIST certification

SEC's RNG is designed to be NIST-certifiable. One of the requirements of that certification is the ability to test the RNG prior to normal operation. This requires instantiating the RNG in test (deterministic) mode rather than normal (nondeterministic) operational mode, and then having software run various tests on the RNG. To allow an opportunity for this testing, SEC does not automatically instantiate the RNG in operational mode or automatically load the JDKEK, TDKEK and TDSK registers, and does not automatically respond to a request for random data from the SecMon. After the tests have completed, or if the tests are going to be skipped, the RNG must be instantiated in operational mode and the JDKEK, TDKEK and TDSK registers must be loaded. These steps are accomplished by executing descriptors as described in this table. The execution of these descriptors must be initiated by software, typically via the Job Ring interface.

Table 11-82. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK

Descriptor	Value	Execution
Descriptor to instantiate RNG State Handle 0 in deterministic (test) mode NOTE: This descriptor would be executed prior to running tests on the RNG.	B080 0004h	HEADER command indicating a descriptor with a length of four 32-bit words
	1281 0004h	LOAD Command with 4 bytes of immediate data; destination is Class 1 Key Size register
	0000 0000h	4 bytes of immediate data (entropy input is null)
	8250 0005h	OPERATION command, Class 1, RNG, Instantiate, Test Mode
Descriptor to uninstantiate RNG State Handle 0 NOTE: This descriptor would be executed after running tests on the RNG.	B080 0002h	HEADER command, indicating a descriptor with a length of two 32-bit words
	8250 000Dh	OPERATION command, Class 1, RNG, Uninstantiate, Test Mode
Descriptor to instantiate RNG State Handle 0 in normal mode and load the JDKEK, TDKEK, and TDSK registers NOTE: This descriptor would be executed to start normal operation of the RNG and also initialize JDKEK, TDKEK and TDSK.	B080 0006h	HEADER command, indicating a descriptor with a length of six 32-bit words
	8250 0004h	OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode
	A200 0001h	JUMP command, wait until Class 1 (RNG) done then local jump to next command
	1088 0004h	LOAD command, 4 bytes of Immediate data, destination Clear Written Register
	0000 0001h	4 bytes of immediate data (clear the Class 1 Mode Register. This resets the done interrupt and returns the RNG to idle.)
	8250 1000h	OPERATION command, Class 1, RNG, Secure Key, Generate
The Generate command for secure keys allows for an optional "additional_input" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Generate command.		

Table continues on the next page...

Table 11-82. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK (continued)

Descriptor	Value	Execution
Descriptor to instantiate RNG State Handle 0 in nondeterministic (normal) mode NOTE: This descriptor would be executed to start normal operation of the RNG, but without initializing JDKEK, TDKEK and TDSK.	B080 0002h	HEADER command, indicating a descriptor with a length of two 32-bit words
	8250 0004h	OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode
The Instantiate command allows for an optional "personalization_string" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Instantiate command.		

11.5.3 RNG operations

RNG operations are performed by appropriately setting the Algorithm State (AS) field of the OPERATION command.

Table 11-83. RNG Operations

Value of AS	Name	Function
00	State-handle generate operation	Causes the RNG to generate random data from the selected state handle and push that data to the output FIFO. The amount of data generated is based on the value in the Class 1 Data Size register. The descriptor can also provide 256 bits of additional input via the Class 2 Key Register, which is used as additional entropy when generating the requested data. The TST bit value must match the deterministic/nondeterministic mode of the selected state handle, else a test error is generated. ¹ A test error is also generated if a Generate command is issued to a state handle that is not instantiated.
01	State-handle instantiation operation	Causes the RNG to set up the initial context for the specified state handle. The state handle remains instantiated in the specified mode (deterministic or nondeterministic) until it is uninstantiated or SEC is reset. A test error is generated if an attempt is made to instantiate a state handle that is already instantiated. <ul style="list-style-type: none"> • TST bit = 0. Nondeterministic instantiation. When instantiating a state handle in nondeterministic (normal) mode, the state handle is seeded with 512 bits of high-grade random entropy from the TRNG and an optional 256-bit personalization string supplied by the descriptor via the Class 1 Context Register. • TST bit = 1. Deterministic instantiation. When instantiating a state handle in deterministic (test) mode, the state handle is seeded with 256 bits of user-specified entropy supplied via the Class 1 Key register and an additional 256 bits of nonce supplied via the Class 2 Key register. Seeding the state handle with known entropy and nonce values allows for deterministic testing. Note that once the RENGSH0 bit in the Security Configuration register has been set to 1, State Handle 0 can no longer be instantiated in deterministic mode. State Handle 0

Table continues on the next page...

Table 11-83. RNG Operations (continued)

Value of AS	Name	Function
		produces the random numbers used for nonces and padding within the built-in protocols, so this special protection can be used to prevent accidentally or maliciously substituting a test instantiation in place of a nondeterministic instantiation.
10	State-handle reseed operation	Causes the RNG to reseed an already instantiated state handle; that is, the current state associated with the selected state handle is replaced with new state information. A test error is generated if an attempt is made to reseed a state handle that is not instantiated. <ul style="list-style-type: none"> For a state handle in nondeterministic mode, the DRNG is seeded with 512 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register. For a state handle in deterministic mode, 256 bits of user-specified entropy is taken from the Class 1 Key Register. Nonce is not used for reseeding.
11	State-handle unstantiate operation	Causes the RNG to unstantiate the specified state handle, which prevents the state handle from being used to generate data. The state handle can later be instantiated again. A test error is generated if an attempt is made to unstantiate a state handle that is not instantiated.

- There is one exception to this rule. A test error is not generated if State Handle 0 is in Test mode but a generate operation requests nondeterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

11.5.4 RNG use of the Key Registers

RNG uses the key registers as follows:

- RNG uses the Class 1 Key Register only when instantiating or reseeding a state handle in deterministic (test) mode.
- RNG uses the Class 2 Key Register only when instantiating a state handle in deterministic (test) mode. In these cases, the descriptor has the TST bit set during the OPERATION command and has loaded known values into the following registers:
 - 256-bit entropy input in the Class 1 Key Register (for instantiate and reseed operations)
 - 256-bit nonce in the Class 2 Key register (only for instantiate operations)
- When instantiating or reseeding a state handle in nondeterministic mode, the key registers are ignored and entropy is instead obtained from the TRNG.

11.5.5 RNG use of the Context Register

The Class 1 Context Register is used to supply an optional 256-bit personalization string when instantiating a state handle, or to supply an optional 256 bits of additional input when reseeding a state handle or generating random data.

11.5.6 RNG use of the Data Size Register

The RNG uses the Data Size Register as follows:

- When an RNG generate command is executed, the value in the Data Size Register specifies the number of bytes of random data that should be generated and pushed onto the Output FIFO.
- When an RNG instantiate command is executed, the value in the Data Size Register specifies a reseed interval, measured in number of *generate* requests.
- The RNG uses a default reseed value of 10,000,000 requests. This means that 10,000,000 *generate* requests are processed before an automatic reseed operation occurs. For a system with the clock speed between 133MHz - 400MHz, the reseed happens between 3-20 seconds if RNG operations are being processed at the maximum rate.
- The Data Size Register holds 32 bits so the user can specify a larger or smaller value. If the user does not specify a reseed interval, the default value is used.

11.6 SNOW 3G f8 accelerator functionality

SNOW is a radio interface cryptographic algorithm set for LTE (two other sets are Kasumi and ZUC). The f8 mode confidentiality algorithm is defined as a word-oriented stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialization value. It can be used to encrypt or decrypt blocks of data between 1 and 20000 bits in length. Some of the features of the SNOW f8 accelerator include the following:

- Message encryption and decryption in f8 (UEA2) mode
- Throughput of up to 4 bytes per cycle
- Support for multiple session message processing through context switching
- Support for descriptor sharing
- Total message size of up to $2^{32}-1$ bits (processed in chunks of no more than $2^{17}-1$ bytes per session)
- Support for any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

11.6.1 Differences between SNOW 3G f8 and SNOW 3G f9

Some of the key differences between SNOW 3G f8 and f9 are as follows:

- SNOW 3G f8 is the encryption algorithm, and SNOW 3G f9 is the authentication algorithm within this cryptographic set.
- The SNOWf8 hardware accelerator implements the f8 encryption mode of operation of the SNOW algorithm, whereas the SNOWf9 hardware accelerator implements the f9 integrity authentication mode of the SNOW algorithm.
- The SNOW f8 CHA is programmed using Class 1 CCB registers, whereas the SNOW f9 CHA is programmed using Class 2 CCB registers. Note that it is possible to encrypt or decrypt data using SNOW f8 and also hash the same data using SNOW f9 authentication via "snooping"; that is, passing the same data simultaneously to both CHAs ("in snooping"), or passing the output of one CHA directly to the input of the other CHA ("out snooping"). However, in those versions of SEC that implement more than one DECO but only one SNOW f8 CHA and one SNOW f9 CHA, the descriptor must select the SNOW f9 CHA first. Selecting the SNOW f8 CHA first and then selecting the SNOW f9 CHA within the same descriptor results in an error indication.

11.6.2 SNOW 3G f8 use of the Mode Register

The SNOW 3G f8 uses the Mode Register as follows:

- The SNOW 3G f8 accelerator is enabled by setting the Algorithm (ALG) field of the Class 1 Mode Register to 60h.
- The f8 mode is enabled by setting the Additional Algorithm Information (AAI) field to C0h.
- The Algorithm State (AS) field should be set to "Initialize" state when a new message is to be processed. The SNOW 3G f8 accelerator initializes the core engine (keystream generator) based on the key and initialization parameters COUNT-C, BEARER and DIRECTION in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register. The AS field should be reset (or set to "Update" state) after context switch, assuming that Key/Context Registers are restored, when continuing message processing. In this case, the state of the keystream generator necessary for continuation of message processing is in the Key/Context Registers and initialization is not needed.
- Other fields in the Mode Register have no effect on f8 mode.
- If the AAI field is set to a value that does not correspond to f8 mode, an illegal-mode error is generated. The Mode, Key Size and Data Size Registers can be written in any order. The operation will begin after all of these have been written.

11.6.3 SNOW 3G f8 use of the Context Register

The SNOW 3G f8 uses the Context Register as follows:

- SNOW 3G f8 uses the Class 1 Key and Context registers.
- The usage of the Key and Context registers in the f8 mode is described in this table.

Table 11-84. Key/Context Register usage in SNOW 3G f8 mode

Register	DWord number	Initialize input definition	Update input definition ¹
Key Register	0	Key[0:63]	s0, s1
	1	Key[64:127]	s2, s3
	2	-	s4, s5
	3	-	s6, s7
Context Register	0	Count-C Bearer Direction 0	s8, s9
	1	-	-
	2	-	-
	3	-	r1
	4	-	r2, r3
	5	-	s10, s11
	6	-	s12, s13
	7	-	s14, s15

1. The symbols in this column represent values written back by SNOW 3G f8. These values comprise the state of the keystream generator that must be restored after context switch for the message processing to continue.

- In the f8 mode, the Context Register is treated as an extension of the Key Register; that is, it is automatically encrypted when saved and decrypted when restored. The IV value must be written to the Context Register when starting a new Job in the f8 mode. This value consists of SNOW 3G f8 initialization parameters in the order shown in this table.

Table 11-85. IV that must be written to Class 1 Context Register in SNOW 3G f8 mode

0-31	32-36	37	38-63
Count-C	Bearer	Direction	0

11.6.4 SNOW 3G f8 use of the Data Size Register

The SNOW 3G f8 uses the Data Size Register as follows:

- SNOW 3G f8 uses the 17 lsbs of the Class 1 Data Size Register to indicate the number of bytes of input data and uses bits 63-61 to indicate the number of valid bits in the last byte.
- SNOW 3G f8 internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size register and the AS field of the Mode Register is set to "Initialize", SNOW 3G f8 keystream generator is initialized and Key and Context Registers contain this initialized state.

11.6.5 SNOW 3G f8 use of the Key Register

The SNOW 3G f8 uses the Key Register as follows:

- A 128-bit key must be written to the Class 1 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize".
- The key is necessary for the initialization of the keystream generator but it is not needed when the AS field of the Mode Register is set to "Update"; that is, when a message processing is continued after context switch.
- The Key Register is used to implement internal state of the keystream generator as depicted in [Table 11-84](#).

11.6.6 SNOW 3G f8 use of the Key Size Register

Writing to this register is not required by SNOW 3G f8, because the SNOW 3G f8 key is always 16 bytes long. Writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

11.7 SNOW 3G f9 accelerator functionality

SNOW 3G f9 is a keyed word-oriented stream integrity/authentication algorithm that generates a 32-bit message digest under the control of a 128-bit key and a 128-bit initialization value. The message may be between 1 and 20000 bits in length. The algorithm is based on the same stream cipher (SNOW 3G f9) as is used by the confidentiality algorithm f8. See [Differences between SNOW 3G f8 and SNOW 3G f9](#) for more on the differences between the SNOW f8 and the SNOW f9 CHAs. Some of the features of the SNOW f9 CHA are as follows:

- Message authentication in f9 (UIA2) mode
- Automatic comparison of the received and computed MAC values (ICV check)

- Throughput of up to 2 bytes per cycle
- Support for multiple session message processing through context switching
- Support for descriptor sharing
- Total message size of up to 2^{32} bits (processed in chunks of no more than $2^{17}-1$ bytes per session)
- Supports any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

11.7.1 SNOW 3G f9 use of the Mode Register

The SNOW 3G f9 uses the Mode Register as follows:

- The SNOW 3G f9 accelerator is enabled by setting the Algorithm (ALG) field of the class 2 Mode Register to A0h.
- The f9 mode is enabled by setting the Additional Algorithm Information (AAI) field to C8.
- The Algorithm State (AS) field must be set to "Initialize" state when the first session of message processing is to be performed. This assumes that message processing is split into multiple sessions; that is, that the first one is not also the final session. The SNOW 3G f9 accelerator initializes the core engine (keystream generator) based on the key and an IV built from initialization parameters COUNT-I, DIRECTION, BEARER and FRESH in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register.
- If the AS mode field is set to "Initialize/Finalize" (11b), but there is no message to be processed (0 written to the Data Size Register), the computed MAC will be identical to the keystream word Z5 as defined in the SNOW specification. If the CICV mode bit is also set, the ICV/MAC expected on the input-data FIFO is checked against Z5.
- If the data size is 0, and CICV is 1, AS set to "Update" (00b) means that Check ICV job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the input-data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
- The AS field must be set to "Finalize" state when the last session of message processing is to be performed. This enables computation of the MAC.
- The AS field must be set to "Initialize/Finalize" state when the whole message is processed in one session.
- The ICV bit of the Mode Register must be set for the f9 mode to compare computed MAC/ICV with the received ICV. The received ICV must be provided through the input data FIFO following the message data at which time the FIFO data type must

be set to ICV. If this bit is reset, the f9 mode does not expect ICV to be put on Input Data FIFO.

- The illegal-mode error is generated if ICV bit is set, but the AS field is not set to either "Initialize/Finalize" or "Finalize" state, except for CICV-only jobs; that is, the data size is non-zero and valid (there are no more writes to the Data Size Register).
- If the AAI field is set to a value that does not correspond to f9 mode, the illegal-mode error is generated. The Mode, Key Size and Data Size Registers can be written in any order. The operation will begin after all of these have been written.
- When SEC descriptor sharing mode is used with SNOW 3G f9, clear mode, followed by clear-done interrupt command, must be issued between SNOW 3G f9 jobs.

11.7.2 SNOW 3G f9 use of the Context Register

The SNOW 3G f9 uses the Context Register as follows:

- This table shows context usage in the f9 mode that is relevant for its programming.

Table 11-86. Context usage in SNOW 3G f9 mode

Register	DWord number	Initialization input definition	Update state (for context switching)	Finalize output definition
Key Register	0	Key[0:63]	-	-
	1	Key[64:127]	-	-
	2	-	-	-
	3	-	-	-
	4	-	IV	-
	5	-	z1, z2	-
	6	-	z3, z4	-
	7	-	z5, bit length	-
Context Register	0	Count-C 0 Direction 0	-	{MAC,32'h0}
	1	{FRESH, 32'h0} (3G) {Bearer, 59'h0} (LTE)	-	-

- For 3G, the IV value is built as shown in this table.

Table 11-87. IV in Class 2 Context for 3G in SNOW 3G f9 mode

0-31	32-36	37	38-63	64-95
Count-C	0	Direction	0	FRESH

- LTE systems do not include a FRESH value in the f9 IV value. It is instead built as shown in this table.

Table 11-88. IV in Class 2 Context for LTE in SNOW 3G f9 mode

0-31	32-36	37	38-63	64-68	69-95
Count-C	0	Direction	0	Bearer	0

- At the end of processing, SNOW 3G f9 overwrites IV in the context word 0 with the MAC/ICV. Because the MAC is a 32-bit value, it is written left-justified and the remaining bits are cleared.
- Values z1-z5 are the keystream words computed during initialization of the f9 mode by the keystream generator. After initialization stage is complete, keystream generator is not active any more in the f9 mode. The processing is based on the Galois Field (GF) multiplier implemented as part of the f9 mode logic. The bit length is a value copied from the data size register to be used to compute the final MAC. In case of multi-session message processing, this value represents the total message length as each session's data size is accumulated.
- To read only the final MAC value, the "finalize" option must be present in the AS mode setting. When saving context, the starting address must be the address of the first double word of the Key Registers.

11.7.3 SNOW 3G f9 use of the Data Size Register

The SNOW 3G f9 uses the Data Size Register as follows:

- SNOW 3G f9 uses the 17 lsbs of the Class 2 Data Size register to indicate the number of bytes of input data, and the NUMBITS field to indicate the number of valid bits in the last byte.
- SNOW 3G f9 internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size register and the AS field of the Mode Register is set to "Initialize", SNOW 3G f9 keystream generator is initialized and the context contains this initialized state.
- In the f9 mode, the data size must be divisible by 64 except when the AS field of the Mode Register is set to "Finalize" or "Initialize/Finalize". In other words, the message can be split for multi-session processing only on a 64-bit boundary. If this rule is violated, the illegal data size error is generated.

11.7.4 SNOW 3G f9 use of the Key Register

A 128-bit key must be written to the Class 2 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize" or "Initialize/Finalize". The key is necessary for the initialization of the keystream generator.

11.7.5 SNOW 3G f9 use of the Key Size Register

The SNOW 3G f9 uses the Key Size Register as follows:

- Writing to this register is not required by SNOW 3G f9, because the SNOW 3G f9 key is always 16 bytes long.
- Writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

11.7.6 SNOW 3G f9 use of ICV check

The SNOW 3G f9 uses ICV check as follows:

- The f9 mode can automatically compare received ICV with the computed ICV at the end of processing if the ICV bit of the Mode Register is set and the AS field is set to "Finalize" or "Initialize/Finalize".
- The received ICV must be supplied after message data through the Input Data FIFO.
- The FIFO data type for it must be set to ICV.
- The SNOW 3G f9 mode ICV/MAC is always a 32-bit value.
- If the ICV mode bit is set but the AS field is set to "Initialize" or "Update", an illegal-mode error is generated, except for CICV-only jobs where no processing is done and only ICV check is performed as indicated by data size being 0.
- SNOW 3G f9 generates ICV error if received and computed ICVs do not match.
- It is allowed to create jobs where there is no data to be processed, and only ICV is being checked. For this, the AS mode field should be reset.

11.8 Message digest hardware accelerator (MDHA) functionality

The MDHA performs hashing and authentication operations using the hashing algorithms defined in FIPS 180-3 (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512) and MD5. The MDHA also supports SMAC with MD5 and SHA-1 that is used with SSL 3.0 and HMAC that is used by TLS and other protocols. MDHA is controlled by the Class 2 registers.

11.8.1 MDHA use of the Mode Register

The MDHA uses the Mode Register as follows:

- The Encryption field (ENC) is not used by the MDHA, and the Authenticate/Protect (AP) field is used only for selecting the appropriate Performance Counter register.
- The Algorithm field (ALG) must be programmed to MD5, SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
- The ICV field enables ICV checking for MDHA. Starting at the MSB, MDHA verifies the number of bytes in the digest that are defined in the Class 2 ICV Size Register.
- The Algorithm State (AS) field is defined as follows:

Table 11-89. Mode Register[AS] operation selections in MDHA

Operation	Description
INIT	The hashing algorithm is initialized with the chaining variables and then hashing begins. Input data must be a non-zero multiple of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256, and 128-byte blocks for SHA-384 and SHA-512.
INIT/FINALIZE	The hashing algorithm is initialized with the chaining variables, and padding is automatically put on the final block of data. Any size of data is supported.
UPDATE	The hashing algorithm begins hashing with an intermediate context and running message length. Input data must be a multiple of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256, and 128- byte blocks for SHA-384 and SHA-512.
FINALIZE	The hashing algorithms begin hashing with an intermediate context and running message length. Padding is performed on the final block of data. Any size of data is supported.

- The Additional Algorithm Information field (AAI) field is defined as follows:
 - The Additional Algorithm Information field (AAI) specifies whether Authentication is performed on the data with the specified algorithm. The optional authentication modes are HMAC, SMAC, and HMAC with precomputed IPAD/OPAD.

- The HMAC mode is defined by FIPS 198-1. This can be performed with any of the hashing algorithms.
- The SMAC mode is defined by the SSL 3.0 specification. This can be performed with MD5 or SHA-1 hashing algorithm only.
- The HMAC with precomputed IPAD/OPAD performs the HMAC algorithm but allows the IPAD and OPAD step to be preloaded and started from instead of the KEY.

NOTE

For HMAC and SMAC, the MD5 Key cannot be shared between DECOs until the donor MDHA is done. As a result, if using an MD5 key in a shared descriptor, sharing must be set to NEVER, WAIT or SERIAL, and sharing cannot be permitted to proceed until MDHA is done. For more information on sharing, please refer to [Table 7-1](#).

11.8.2 MDHA use of the Key Register

The MDHA uses the Key Register as follows:

- The Key Register is only used when one of the AAI field bits are specified.
- These registers either hold the key or the precomputed IPAD/OPAD split key.
- The size of the IPAD and OPAD are each the size of the digest that is defined by the specified algorithm, except for SHA-224, which is 32 bytes, and SHA-384, which is 64 bytes.

11.8.2.1 Using the MDHA Key Register with normal keys

When loading the Key Register with the Key Command (See [KEY commands](#)), a KDEST value of 0h results in the key source being loaded, with offset zero, into the Key Register. If the ENC bit = 1 in the Key Command, then the key is decrypted into this register.

11.8.2.2 Using the MDHA Key Register with IPAD/OPAD "split keys"

The HMAC function uses an HMAC key per the following equation:

$$\text{HMAC}(\text{Key}, \text{Message}) = \text{Hash}[(\text{Key} \oplus \text{OPAD}) \parallel \text{HASH}((\text{Key} \oplus \text{IPAD}) \parallel \text{Message})],$$

where "IPAD" and "OPAD" are constants, "Key" is the HMAC Key, and "HASH" is the chosen hashing function (for example, SHA-256).

11.8.2.2.1 Definition and function of IPAD/OPAD split keys

To improve performance, SEC permits the use of pre-computed IPAD/OPAD "split keys". Computing the values $\text{Hash}(\text{Key} \oplus \text{IPAD})$ and $\text{Hash}(\text{Key} \oplus \text{OPAD})$ each require MDHA to perform one block of hashing computation. By performing this computation once, subsequent HMAC computations can save two blocks of a HMAC computation. As a result, the Key Command has an option to signal MDHA that the key being loaded is a precomputed split key.

11.8.2.2.2 Process flow of using the Key Register with split keys

When MDHA runs, it turns a key into an IPAD/OPAD pair. MDHA writes this pair back to the Class 2 Key Register. Because the IPAD/OPAD pair is required every time, it saves time to create it once and then reuse it rather than starting with the key again. However, the IPAD/OPAD pair does not appear in the Key Register contiguously. To make the hardware much simpler, the IPAD appears at the start of the Key Register and the OPAD starts at the midpoint. So, between a 16-byte IPAD and the 16-byte OPAD, there can be 48 bytes of null data. Rather than having to load or store the null data, use the split key type.

11.8.2.2.3 Using padding with the split key type to align with storage

When doing a FIFO STORE of the split key type, the user provides a length equivalent to the sum of the bytes in the IPAD and OPAD. That is, in the example above, a total length of 32 bytes. DECO knows to get 16 bytes from the start and another 16 bytes from the midpoint. This saves in encryption time and bandwidth. When loading 20-byte IPAD and OPAD (size would be 40), there must be "padding" of 4 bytes following each. That is: {20 bytes of IPAD, 4 bytes of PAD, 20 bytes of OPAD, 4 bytes of PAD}. The padding can be anything, because SEC discards it. The reason for this is to make it align with how the encrypted split key is stored, where the extra padding is used to pad each of IPAD and OPAD to an 8-byte boundary so that they can be handled separately.

11.8.2.2.4 Length of a split key

Because the split key consists of two blocks of material processed by the selected hash algorithm, the length of a split key is twice the length of the hash algorithm's running digest (note exception below). Storage of the split Key in the Class 2 Key Register, however, is such that the value $\text{Hash}(\text{Key} \oplus \text{IPAD})$ is at offset zero, and the value $\text{Hash}(\text{Key} \oplus \text{OPAD})$ is at offset 64.

11.8.2.2.5 Loading/storing a split key with a key command

A split key may be loaded either in encrypted or in unencrypted form. The DECO command to load a split key is the key command with the KDEST field set to 3h. A split key loaded in this way must be stored contiguously in external memory, and is twice the length of the hash algorithm's running digest. For example, the running digest for SHA-224 is 32 bytes, so the length of a SHA-224 split key in external memory is 64 bytes. In addition, the length for a key command associated with a SHA-224 split key must be 64. A split key that SEC has generated has also been encrypted, and for SHA-1, has been padded with 8 additional bytes.

11.8.2.2.6 Loading/storing a split key with a FIFO STORE command

The DECO command to store a split key is the FIFO STORE command (see [FIFO STORE command](#)), with output-data type set to 16 or 26 (17 or 27 to encrypt using the TDKEK). Generating a split key in this fashion results in the key being encrypted with the JDKEK or TDKEK. Note that the length of an encrypted split key is longer if the FIFO STORE command output-data type selects AES-CCM (16 or 17) for the encrypted key type. Even if AES-ECB is selected (26 or 27), a SHA-1 encrypted split key is always longer, because SEC must add 8 bytes of padding before the pre-encrypted 40 bytes of actual split key can be encrypted.

11.8.2.2.7 Sizes of split keys

This table describes the different sizes of split keys depending on how they were generated.

Table 11-90. Sizes of split keys

Hash algorithm	Final digest size	Running digest size	Software-generated split-key Size	AES-ECB encrypted split-key size	AES-CCM encrypted split-key size
MD5	16 bytes	16 bytes	32 bytes	32 bytes	44 bytes
SHA-1	20 bytes	20 bytes	40 bytes	48 bytes	52 bytes
SHA-224	28 bytes	32 bytes	64 bytes	64 bytes	76 bytes
SHA-256	32 bytes	32 bytes	64 bytes	64 bytes	76 bytes
SHA-384	48 bytes	64 bytes	128 bytes	96 bytes	140 bytes
SHA-512	64 bytes	64 bytes	128 bytes	128 bytes	140 bytes

11.8.2.2.8 Constructing an HMAC-SHA-1 split key in memory

This figure is an example of how software would construct an HMAC-SHA-1 split key in memory.

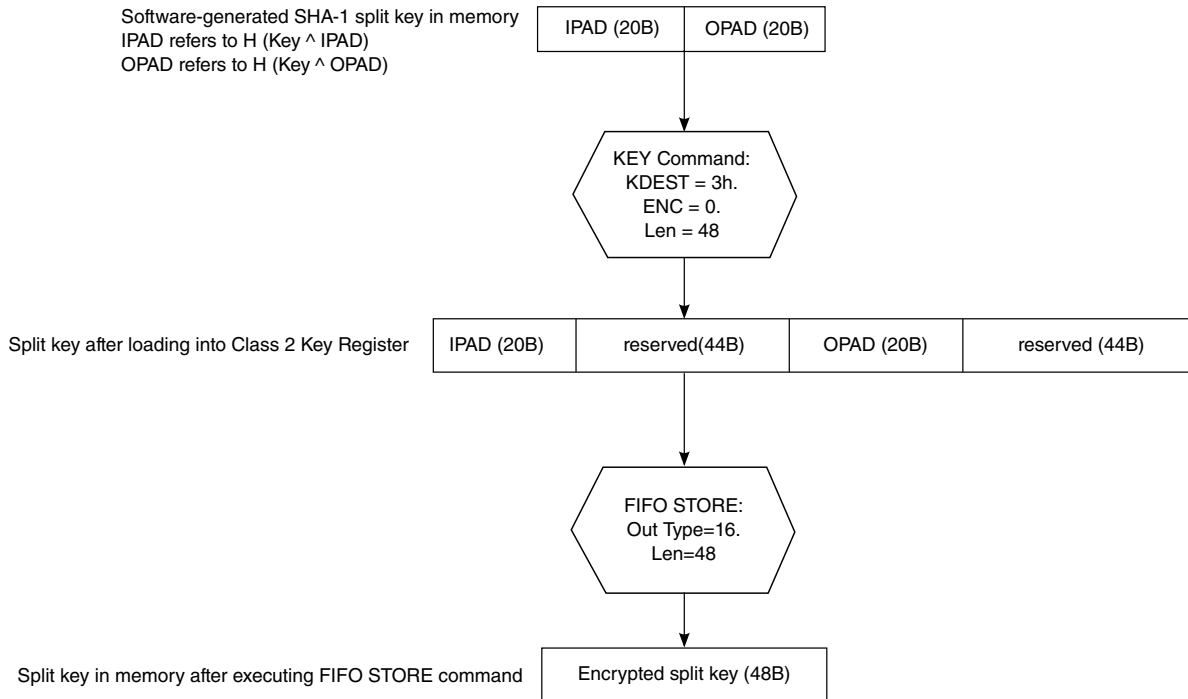


Figure 11-1. Split keys in memory and in the Class 2 Key Register

Use the KEY Command to load it into the Class 2 Key Register, and then use the FIFO STORE Command to write it back out in encrypted form.

11.8.2.3 MDHA use of the Key Size Register

The Key Size Register is defined to be the number of bytes of key that is loaded into the Key Registers. Key Size ranges are defined as followed:

- MD5: 0 → 64 bytes
- SHA-1: 0 → 64 bytes
- SHA-224: 0 → 64 bytes
- SHA-256: 0 → 64 bytes
- SHA-384: 0 → 128 bytes
- SHA-512: 0 → 128 bytes

11.8.3 MDHA use of the Data Size Register

The MDHA uses the Data size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.

- This register must be written to start data processing.
- This register may be written multiple times while data processing is in progress in order to add the amount written to the register to the previous value in the register.

11.8.4 MDHA use of the Context Register

The Context Register stores the current digest and running message length. The running message length will be 8 bytes immediately following the active digest. The digest size is defined as follows:

- MD5: 16 bytes
- SHA-1: 20 bytes
- SHA-224: 28 bytes final digest; 32 bytes running digest
- SHA-256: 32 bytes
- SHA-384: 48 bytes final digest; 64 bytes running digest
- SHA-512: 64 bytes

11.8.5 Save and restore operations in MDHA context data

MDHA is able to process data in chunks by saving the intermediate context and running message length from the Context Register after each chunk of data and restoring the context and running message length to the Context Registers before processing any subsequent chunks of data.

11.9 AES accelerator (AESA) functionality

The advanced encryption standard accelerator (AESA) module is a hardware co-processor capable of accelerating the advanced encryption standard (AES) cryptographic algorithm.

11.9.1 Differences between the AES encrypt and decrypt keys

AES is a block cipher that processes data in 128-bit blocks. It is a symmetric key algorithm, that is, the "same"¹ key is used for both encryption and decryption, although the key appears in a different form for decryption than it does for encryption. The decrypt form of the key is different from the encrypt form of the key because AES successively

1. The two forms are considered the same key because one can be derived from the other.

modifies the cryptographic key during the steps of the cryptographic operation. The decryption operation yields the correct result only if the modified form of the key (the decrypt key) is used at the beginning of the decryption operation. Unless told otherwise (via the DK bit in the OPERATION command), SEC assumes that a key loaded from memory is the encrypt key, that is, the form appropriate for encryption. If a decryption operation is specified and DK = 0, SEC first goes through the steps required to derive the decrypt key from the encrypt key, and then performs the decryption operation. If a decryption operation is specified and DK = 1 (indicating that a decrypt key has been loaded), the steps required to derive the decrypt key are skipped and the decryption operation is performed immediately, significantly improving performance for small data blocks.

Note that the difference between the encrypt key and the decrypt key must be taken into account when sharing keys between jobs. When an AES decryption job loads a key from memory, it is probably an encrypt key, so the DK bit in the OPERATION command should be set to 0 so that SEC derives the decrypt key from the encrypt key before beginning the decryption operation. But when a subsequent AES decryption job shares the key from a previous decryption job, the key that is shared is a decrypt key. In that case, the DK bit should be set to 1, which tells SEC to skip the key derivation steps. If DK were set to 0 in this case, the decrypt key would be modified as if it were an encrypt key, and consequently, the wrong key value would be used in the decryption operation. Note that a JUMP command with TEST CONDITION set to SHRD (see [Table 7-89](#)) can be used to determine whether the OPERATION command should be executed with DK = 0 or DK = 1.

11.9.2 AESA as both Class 1 and Class 2 CHA

AESA can be programmed as either a Class 1 or a Class 2 CHA. When used as a Class 1 CHA, all of the modes of operation that it implements are available. Thus, it can perform both confidentiality and authentication tasks. In this case, AESA is programmed via the Class 1 CCB interface and all of the descriptor commands referring to AESA, its data, context or keys must also use Class 1 designator where appropriate.

In order to support processing modes where an AES algorithm is used to perform authentication while another Class 1 CHA is being used for encryption or decryption, AESA can be programmed as a Class 2 CHA, i.e. using the Class 2 CCB interface. In this case only authentication algorithms are available, specifically XCBC-MAC and CMAC. An attempt to program AESA to perform any other mode algorithm will cause an illegal-mode error. When used as a Class 2 CHA, AESA and its input/output data, context data

or keys must be referred to as Class 2 in any descriptor command that contains the Class field. The designator for AESA in the ALG field of a mode register remains the same for both Class 1 and 2 mode registers.

The ability to function either as a Class 1 or a Class 2 CHA is further leveraged to allow AESA to operate simultaneously as both a Class 1 and a Class 2 CHA. This allows the user to perform encryption/decryption using any of the supported AES confidentiality modes (except the XTS) i.e. ECB, CBC, CTR, OFB, CFB128, while simultaneously performing authentication using either XCBC-MAC or CMAC modes. An attempt to program AESA via Class 1 and 2 mode registers with any other combination of modes will cause an illegal-mode error.

When operating simultaneously as both a Class 1 and a Class 2 CHA, AESA will switch between processing Class 1 or 2 data blocks of 16 bytes after each processed block as long as both Class 1 and 2 blocks are fetched from the corresponding FIFO interface and are ready for processing. It will continue with processing blocks belonging to one Class as long as the data belonging to the other Class is not available. AESA can prefetch data from both Class 1 and Class 2 FIFO interfaces simultaneously. If an error is generated for either Class 1 or Class 2 jobs, the processing is terminated for both classes. The reported error status will contain the appropriate class designator.

AESA can be used either with an in-snooping or an out-snooping data flow. If the Class 2 job needs to process the same data as the Class 1 job, CCB should be programmed to utilize in-snooping. Alternatively, if the Class 2 job performs authentication on the Class 1 job result, out-snooping should be used. If an authentication job using XCBC-MAC or CMAC is to be performed on data in memory, AESA can be programmed either as Class 1 or Class 2.

The descriptor sharing and context switching between different jobs can be utilized with descriptors programming AESA either as Class 1 or Class 2 or both.

When programmed to process Class 1 and Class 2 jobs simultaneously, AESA is considered busy until both jobs complete. For example, after the Class 1 job completes, it is not possible to start another Class 1 job while the initial Class 2 job is still being processed, or vice versa. When clearing an interrupt or issuing a software reset, the corresponding Class Mode Register should also be cleared. The internal registers will not be cleared if AESA is still selected by the other Class Mode Register.

11.9.3 AESA modes of operation

The following modes are supported by AESA:

- Electronic codebook (ECB)

- Cipher block chaining (CBC)
- Output feedback (OFB)
- 128-bit cipher feedback (CFB128)
- Counter (CTR)
- XTS tweakable block cipher
- Extended cipher block chaining message authentication code (XCBC-MAC)
- Cipher-based MAC (CMAC)
- CTR and CBC-MAC (CCM)
- Galois/Counter mode (GCM)
- Combined CBC and XCBC (CBC-XCBC)
- Combined CTR and XCBC (CTR-XCBC)
- Combined CBC and CMAC (CBC-CMAC)
- Combined CTR and CMAC (CTR-CMAC)

AES modes can be classified into these categories:

- Confidentiality (ECB, CBC, CTR, OFB, CFB128, XTS)
- Authenticated Confidentiality (CCM, GCM, CBC-XCBC, CTR-XCBC)
- Authentication (XCBC-MAC, CMAC)

CBC Mode can also be viewed as an authentication mode when used to encrypt data, because it provides CBC-MAC in the context registers.

11.9.4 AESA use of registers

Note the following regarding the AESA's use of registers:

- AESA is controlled by either the Class 1 or Class 2 registers.
- For all modes, if AES is selected and the mode code written to the Mode Register does not correspond to any of the implemented AES modes, the illegal-mode error is generated.
- KEY SIZE, MODE and DATA SIZE can be written in any order. The operation will begin after all of these have been written. As a Class 2 CHA, only XBC-MAC and CMAC authentication modes are available. Writing the Class 2 Mode Register to request any other modes will cause an illegal-mode error. Also, for all AES modes, the bit offset in the Data Size Register must be zero when the last write to that register is completed. Failure to comply with these requirements will generate an error in the CCB Status Register.
- When sharing context between consecutive AES jobs, software reset is not issued. To prepare AES for the next job, the Data Size Register and Mode Register must be

cleared, as well as the Done Interrupt. The order of these should be such that the Done Interrupt is not cleared first.

- If ICV-only jobs are created (no data to be processed, only ICV to be checked) in modes that support ICV check, the AS mode field should be reset.

11.9.5 AESA use of the parity bit

AESA incorporates fault-detection logic based on parity. The parity bit is computed for every byte of input data and key. These parity bits are then fed to the fault detection logic that computes expected parity of every byte for both key and data based on the AES transformations implemented in the main data-path. The expected parity is compared with the parity of the actual key and data bytes and the hardware error is generated if there is a mismatch.

11.9.6 AES ECB mode

The electronic codebook (ECB) mode is a confidentiality mode that features, for a given key, the assignment of a fixed, ciphertext block to each plaintext block, analogous to the assignment of code words in a codebook. In ECB encryption, the forward cipher function is applied directly and independently to each block of the plaintext. The resulting sequence of output blocks is the ciphertext. In ECB decryption, the inverse cipher function is applied directly and independently to each block of the ciphertext. The resulting sequence of output blocks is the plaintext.

11.9.6.1 AES ECB mode use of the Mode Register

AES ECB mode uses the Mode Register as follows:

- The Encrypt (ENC) field should be 1 for ECB encryption and 0 for ECB decryption.
- The ICV/TEST bit is used in ECB mode to activate the fault detection test logic. This logic verifies that the fault detection logic is operational by injecting bit-level errors into input data and key bytes. Because ECB mode does not normally use the Context Registers, the first 128 bits of the context are used in the ECB TEST mode to define which byte of the input data and the key has a bit error injected.
- The Algorithm State (AS) field is not used in ECB mode.
- The Additional Algorithm Information (AAI) field must be set with value 20h that activates ECB mode. Setting the MSB in the AAI field (interpreted as the Decrypt Key or DK bit for AES operations) specifies that the key loaded to the Class 1 Key Register is the decryption form of the key, rather than the encryption form of the key.

If $DK = 0$, when a decryption operation is requested SEC processes the content of the Class 1 Key Register to yield the decryption form of the key. If $DK = 1$, SEC skips this processing. The illegal-mode error is generated if $DK = 1$ and $ENC=1$.

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.6.2 AES ECB mode use of the Context Register

ECB does not use Context Registers except when fault-detection test is activated. In this case, the first 128 bits of the context are reserved for the error code. The error code:

- Defines which byte of the input data and the key will have a bit error injected.
- Can have 32, 40, or 48 active bits depending on the key size (16, 24 or 32 bytes in ECB mode).
- Is right justified within first 128 bits of the context such that bit 0 of Context DWord 1 injects error into the MSB of the input data, while bit 16 of Context DWord 1 injects error into the MSB of the key.

If all bits of the error code are 0, no error is injected and fault detection logic does not activate the hardware error. If the ICV/TEST bit of the Class 1 Mode Register is 0 in the ECB mode, the Context Registers have no effect on ECB processing.

Table 11-91. Context usage in ECB mode

Context DWord	Definition	
	ECB	ECB with ICV/TEST = 1
0	-	ERROR CODE
1	-	

11.9.6.3 AES ECB Mode use of the Data Size Register

The length of the message to be processed in bytes must be written to the Data Size register. If this value is not divisible by 16, the Data Size error is generated.

11.9.6.4 AES ECB Mode use of the Key Register

ECB keys must be written to the Class 1 Key Register and can have 16, 24, or 32 bytes.

11.9.6.5 AES ECB Mode use of the Key Size Register

The number of bytes in the ECB key must be written to the Key Size register. The KEY SIZE, MODE and DATA SIZE can be written in any order. Processing starts after all of them have been written. Any value other than 16, 24, or 32 causes the key-size error to be generated.

11.9.7 AES CBC, OFB, CFB128 modes

The CBC, OFB, CFB128 modes are considered together because of their similarities and are described in this table.

Table 11-92. AES CBC, OFB, CFB128 modes

Name	Abbreviation	Function
Cipher-block chaining mode	CBC	Confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. The CBC mode requires an IV (Initialization Vector) to combine with the first plaintext block NOTE: CBC mode uses both forward and inverse AES cipher. OFB and CFB use only forward AES cipher.
Cipher feedback mode	CFB	Confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input block. AESA implements 128-bit CFB mode where every ciphertext/ plaintext block must have 128 bits
Output feedback mode	OFB	Confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires IV. The last block of OFB input data can have fewer than 16 bytes

11.9.7.1 AES CBC, OFB, and CFB128 modes use of the Mode Register

The AES CBC, OFB, and CFB128 modes use the Mode Register as follows:

- The Encrypt (ENC) field should be 1 for encryption and 0 for decryption, except for OFB mode in which this bit is not used.
- The ICV/TEST bit is not used in these modes.
- The Algorithm State (AS) field is used only in CBC mode to prevent IV update in the context for the last data block when set to "Finalize" (2h).
- The Additional Algorithm Information (AAI) field defines which mode is used for processing. For CBC, OFB, and CFB, these values are 10h, 40h, and 30h,

respectively. The Decrypt Key [DK] (AAI field MSB) bit affects CBC mode and specifies that the key loaded to the Class 1 Key Register is the decrypt key. The illegal mode error is generated if DK=1 and ENC=1. If the DK bit is set in OFB or CFB128 modes the illegal-mode error is also generated, because these modes do not use inverse AES cipher.

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.7.2 AES CBC, OFB, and CFB128 modes use of the Context Register

The AES CBC, OFB, and CFB128 modes use the Context Register as follows:

- AES CBC, OFB, and CFB all use the Context Registers to provide IV, which is updated with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the IV must be saved and later restored for the next chunk to be processed correctly. At the end of CBC processing, IV is also the MAC of the message.
- If the AS field of the Mode Register is set to "Finalize" (2h) in the CBC mode, the last IV update is not written to the context. This enables CBC encryption to effectively perform ECB encryption transformation of a single-block message located in the context in place of IV, and with an all-zero block provided as input data through the FIFO without overwriting the context.

Table 11-93. Context usage in CBC, OFB, CFB modes

Context DWord	Definition
0	IV [127:64]
1	IV [63:0]

11.9.7.3 AES CBC, OFB, and CFB128 modes use of the Data Size Register

The AES CBC, OFB, and CFB128 modes use the Data Size Register as follows:

- The byte length of the message to be processed must be written to the Data Size Register.
- The first write to this register initiates processing. This register can also be written during processing, in which case the value written is accumulated to the current state of the register.

- After the Data Size Register is written for the last time, its value must be divisible by 16 in CBC and CFB modes, otherwise the data-size error is generated.
- Only OFB decrements the value in this register with every processed block.

11.9.7.4 AES CBC, OFB, and CFB128 modes use of the Key Register

The AES CBC, OFB, and CFB128 modes uses the Key Register as follows:

- A CBC, OFB, or CFB key must be written to the Class 1 Key Register.
- Keys can have 16, 24, or 32 bytes.

11.9.7.5 AES CBC, OFB, and CFB128 modes use of the Key Size Register

The AES CBC, OFB, and CFB128 modes use the Key Size Register as follows:

- The number of bytes in a key must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

11.9.8 AES CTR mode

The counter (CTR) mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value will not overflow and wrap around.

NOTE

It is the user's responsibility to ensure that the same key value is not used again following a reset.

11.9.8.1 AES CTR mode use of the Mode Register

The AES CTR mode uses the Mode Register as follows:

- The Additional Algorithm Information (AAI) field should be set to 00h to activate CTR mode. If the Decrypt Key [DK] (AAI field MSB) bit is set, the illegal-mode

error is generated, because CTR uses only forward AES cipher requiring encryption rather than decryption keys.

- The Algorithm State (AS) field when set to "Finalize" (2h) prevents counter update in the context for the last data block.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.8.2 AES CTR mode use of the Context Register

The AES CTR mode uses the Context Register as follows:

- CTR uses context dwords 2 and 3 to provide initial counter value (CTR0). This value is incremented with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the CTR0 field of context has to be saved and later restored for the next chunk to be processed correctly.
- If the AS field of the Mode Register is set to Finalize (2h) in the CTR mode, the last counter update is not written to the context. This enables CTR encryption to effectively perform ECB encryption transformation of a single-block message located in the context dwords 2 and 3 in place of CTR0 and with all-zero block provided as input data through the FIFO without overwriting the context.

Table 11-94. Context usage in CTR mode

Context dword	Initial-input definition	Context-switching definition
0	-	-
1	-	-
2	CTR0 [127:64]	CTR _i [127:64]
3	CTR0 [63:0]	CTR _i [63:0]

11.9.8.3 AES CTR mode use of the Data Size Register

The byte-length of the message to be processed must be written to the Data Size register. The first write to this register initiates processing. It can also be written during processing in which case the value written will be accumulated to the current state of the register. After the Data Size register is written for the last time, the value of this register may not be divisible by 16. CTR decrements the value in this register with every processed block.

11.9.8.4 AES CTR mode use of the Key Register

- CTR key must be written to the Class 1 Key Register.
- The Key Register can have 16, 24 or 32 bytes.

11.9.8.5 AES CTR mode use of the Key Size Register

The number of bytes in a key must be written to the Class 1 Key Size register. Any value other than 16, 24, or 32 will cause Key Size error to be generated.

11.9.9 AES XTS mode

XTS is a tweakable block-cipher that acts on data units (sectors) of 128 bits or more and uses the AES block-cipher as a subroutine. The key material for XTS-AES consists of a data encryption key (used by the AES block cipher) as well as a "tweak key" that is used to incorporate the logical position of the data block into the encryption.

11.9.9.1 AES XTS mode use of the Mode Register

AES XTS uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV/TEST bit is ignored in this mode.
- The Algorithm State (AS) field is ignored in this mode.
- The Additional Algorithm Information (AAI) field's lower 8 bits must be set to 50h for XTS to be activated.
- The Decrypt Key [DK] (AAI field MSB) bit should be set to 1 only if the AES key, written to the Class 1 Key Register with offset 0, is the decryption form of the key. Otherwise, SEC assumes that the key that was loaded is the encryption form of the key, and before beginning a decryption operation SEC first processes the content of the Class 1 Key Register to derive the decryption form of the key.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.9.2 AES XTS mode use of the Context Register

AES XTS uses the Context Register as follows:

- Because XTS uses two keys (see [AES XTS mode use of the Key Register](#)), Key1 (AES key) and Key2 ("tweak" key), and each can be 32 bytes long, both keys cannot always fit in the Key Register. In that case, Key2 spills into first 32 bytes of the Context Register.
- When these keys are 16 bytes long each, XTS does not use the first 32 bytes of the Context Register.

- Context Register dword 4 is used to provide the sector index (I).
- The sector size in bytes must be provided in the low 16 bits of Context Register dword 5.
- When a message is processed in chunks, all of the key and context data (in the first 6 context dwords) except the Block Index (j) must be saved at the end of the last session and restored before starting a new XTS session on the next chunk of the message. The message split must be done on a sector boundary and sectors have integral number of 16-byte blocks, except the last one, whose size can be any number of bytes higher or equal to 16.

Table 11-95. Context usage in XTS mode

Context dword	Initial-input definition	Context-switching definition
0	Key2 [0:63] (only for 64-byte keys)	Key2 [0:63] (only for 64-byte keys)
1	Key2 [64:127] (only for 64-byte keys)	Key2 [64:127] (only for 64-byte keys)
2	Key2 [128:191] (only for 64-byte keys)	Key2 [128:191] (only for 64-byte keys)
3	Key2 [192:255] (only for 64-byte keys)	Key2 [192:255] (only for 64-byte keys)
4	Sector Index (I)	Sector Index (I)
5	Sector Size	Block Index (j), Sector Size

11.9.9.3 AES XTS mode use of the Data Size Register

AES XTS uses the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- Processing starts when mode, key size, and data size are all written in any order. This register can also be written during processing, in which case the value written will be accumulated to the current state of the register. XTS decrements the value in this register with every processed block.
- The message size does not have to be a multiple of sector size. However, the size of data in the last sector must be at least 16 bytes-otherwise, cipher text stealing method, employed for processing messages whose last block has fewer than 16 bytes, would be done across sector boundary, which would produce incorrect result. When detected, this situation generates the Data Size error. This error is also generated if sector size is 0 or is not a multiple of 16 bytes, or if the total message size is less than 16 bytes.

11.9.9.4 AES XTS mode use of the Key Register

AES XTS uses the Key Register as follows:

- The IEEE 1619-2007 standard defining XTS mode refers to a single XTS-AES key of either 256 or 512 bits, but the key is parsed as a concatenation of two fields of equal size called Key1 and Key2 such that: $\text{Key} = \text{Key1} \parallel \text{Key2}$.
 - For a 256-bit key, Key1 must be written to the Class 1 Key Register with offset 0, and Key2 with offset 16.
 - For a 512-bit key, Key2 is written to the first 32 bytes of the context. The AES key(Key1) can be either an encrypt key or a decrypt key.
- If the decrypt key is written to the Key Register, the DK bit (MSB of the AAI field in the Class 1 Mode Register) must be set to 1.

11.9.9.5 AES XTS mode use of the Key Size Register

AES XTS uses the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register. The KEY SIZE, MODE, and DATA SIZE can be written in any order. Processing starts after all of them have been written.
- Any value other than 32, or 64 will cause Key Size error to be generated.

11.9.10 AES XCBC-MAC and CMAC modes

The AES XCBC-MAC and CMAC modes are described together because of their similarities. They are extensions of the AES CBC mode that produces a key-dependent, one-way hash (or message authentication code (MAC)) in a secure fashion across messages of varying lengths. They also provide data-integrity and data-origin authentication regarding the original message source.

11.9.10.1 AES XCBC-MAC and CMAC modes use of the Mode Register

The AES XCBC-MAC and CMAC modes use the Mode Register as follows:

- The Encrypt (ENC) bit is ignored.
- The ICV_TEST bit must be set for computed MAC to be compared with the received MAC. The received MAC must be written to the Input Data FIFO after message data

and the FIFO data type must be set to ICV. If this bit is not set, XCBC-MAC and CMAC do not expect received ICV to be supplied after message data.

- The Algorithm State (AS) field is defined for XCBC-MAC as shown in this table.

Table 11-96. Mode Register[AS] operation selections in AES XCBC-MAC

Operation	Description
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, derived keys K3 and K2 that are XOR-ed with the last message block are computed and stored in the context to be used in the last processing session. The derived key K1 used as an AES key is computed and written back to the Key Register over the original key
INITIALIZE/FINALIZE	Message is processed in a single XCBC session and the final MAC is computed
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
FINALIZE	Message is processed in multiple sessions and the current session is the last one. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. The final MAC is computed

- The Algorithm State (AS) field is defined for CMAC as shown in this table.

Table 11-97. Mode Register[AS] operation selections in CMAC

Operation	Function
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, the constant $L = E(K, 0)$ is computed as encrypted block of zeros using key K and stored in the context to be used in the last processing session for derivation of keys K1 and K2. One of these keys will be XOR-ed with the last message block.
INITIALIZE/FINALIZE	Message is processed in a single session and the final MAC is computed
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. The constant L used for key derivation is provided in the context. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST:w bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session
FINALIZE	Message is processed in multiple sessions and the current session is the last one. The constant L used for key derivation is provided in the context. The final MAC is computed

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV_TEST bit is set to 1, the illegal-mode error is generated, except for CICV-only jobs.
- The Additional Algorithm Information (AAI) field must be set to 70h for XCBC and 60h for CMAC to be activated. Setting the DK bit (AAI field MSB) will cause the Illegal Mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.10.2 AES XCBC-MAC and CMAC Modes use of the Context Register

The AES XCBC-MAC and CMAC modes use the Context Register as follows:

- No data needs to be provided in the context when starting a new XCBC or CMAC session.
- The computed MAC and the derived keys K2 and K3 are written back to the context by XCBC.
- The computed MAC and the constant $L = E(K,0)$, computed as encrypted block of zeros using key K, are written back to the context by CMAC.
- When a message is split into chunks and processed in multiple sessions, these values need to be saved before context switch and restored before the next chunk of a message is to be processed. At the end of message processing the first 2 dwords of the context contain the MAC value.

Table 11-98. Context usage in XCBC-MAC and CMAC modes

Mode	Context dword	Context-switching definition	Final-result definition
XCBC-MAC	0	MAC[127:64]	MAC[127:64]
	1	MAC[63:0]	MAC[63:0]
	2	K3[127:64]	-
	3	K3[63:0]	-
	4	K2[127:64]	-
	5	K2[63:0]	-
CMAC	0	MAC[127:64]	MAC[127:64]
	1	MAC[63:0]	MAC[63:0]
	2	L[127:64]	-
	3	L[63:0]	-

11.9.10.3 AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register

The AES XCBC-MAC and CMAC modes use the ICV Size Register as follows:

- This register is used to provide received ICV/MAC byte-size when it is other than 16 bytes.
- The computed ICV/MAC written to the context in the XCBC mode is always 16 bytes.

- In CMAC mode, this register determines also the computed MAC size-the remaining bytes are cleared.
- Supported values for ICV size are 4 to 16 bytes. If this register is 0, the size of ICV is 16 bytes.

11.9.10.4 AES XCBC-MAC and CMAC modes use of the Data Size Register

The AES XCBC-MAC and CMAC modes use the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written is accumulated to the current state of the register.
- XCBC-MAC and CMAC decrement the value in this register with every processed block.

11.9.10.5 AES XCBC-MAC and CMAC modes use of the Key Register

The AES XCBC-MAC and CMAC modes use the Key Register as follows:

- The key must be written to this register.
- For XCBC-MAC, if the AS mode field is set to either "Initialize" or "Initialize/Finalize", it is the original XCBC key (K) that must be written here. Otherwise, the derived key (K1) must be restored to this register. CMAC only uses original key K as an AES key.

11.9.10.6 AES XCBC-MAC and CMAC modes use of the Key Size Register

The AES XCBC-MAC and CMAC modes use the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register.
- For XCBC-MAC, any value other than 16 causes a key-size error to be generated. For CMAC, this error is generated only if any value other than 16, 24, or 32 is written.

11.9.10.7 ICV checking in AES XCBC-MAC and CMAC modes

Automatic ICV checking is enabled by setting the ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the illegal-mode error is generated, except for CICV-only (Check-ICV-only) jobs.

The received ICV must be provided on the FIFO after the message data. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is provided in the Class 1 ICV Size register.

If the ICV check detects a mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

11.9.11 AESA CCM mode

CCM consists of two related processes: generation encryption and decryption verification, which combine two cryptographic primitives: counter mode encryption (CTR) and cipher-block chaining based authentication (CBC-MAC). Only the forward cipher function of the block cipher algorithm is used within these primitives. Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value does not overflow and wrap around.

NOTE

It is the user's responsibility to ensure that the same key value is not used again following a reset.

11.9.11.1 Generation encryption

A cipher-block chaining is applied to the payload, the associated data (AAD), and the nonce to generate a message authentication code (MAC); then counter mode encryption is applied to the MAC and the payload to transform them into an unreadable form, called the ciphertext. Thus, CCM generation encryption expands the size of the payload by the size of the MAC.

11.9.11.2 Decryption verification

Counter-mode decryption is applied to the purported ciphertext to recover the MAC and the corresponding payload; then cipher block chaining is applied to the payload, the received associated data, and the received nonce to verify the correctness of the MAC.

In CCM mode, the FIFO data type must be set to message type for message data, while for AAD, either AAD or message type can be used.

11.9.11.3 AES CCM mode use of the Mode Register

The AES CCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV_TEST bit must be set for CCM to compare computed MAC with the received MAC when decryption is requested.
- The received MAC must be written to the input-data FIFO after message data and the FIFO data type must be set to ICV.
- Setting the ICV_TEST bit causes the received MAC to be decrypted and compared with the computed MAC.
- The number of MSBs to be compared is defined by the MAC size in the CCM IV (B_0) as described in the CCM specification.
- If the AS field is set to FINALIZE, but $ICV = 0$, AESA does not expect received ICV to be put on the input-data FIFO. In that case, MAC is computed and truncated to the specified size for decryption.
- For encryption, the computed MAC is encrypted and truncated to size. The illegal-mode error is generated if $ICV = 1$ and $ENC = 1$.
- If $ICV = 1$ and the decrypted received MAC do not match computed MAC, the ICV error is generated.
- The Algorithm State (AS) field is defined for CCM as follows:

Table 11-99. Mode Register[AS] operation selections in AES CCM

Operation	Description
INITIALIZE	Message is processed in multiple sessions and the current session is the first one. During initialization, the initial counter CTR0 is encrypted in the CTR mode and the B_0 is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B_0 and written to the context. This AS setting must be used whenever the first part (or whole) AAD is being processed
INITIALIZE/FINALIZE	Message is processed in a single CCM session and the final MAC is computed and encrypted. The initial counter CTR0 and B_0 must be provided in the context
UPDATE	Message is processed in multiple sessions and the current session is neither the first nor the last. All context data is restored from the previous session and the key is written to the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS=UPDATE means that a CICV-only job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, decrypts it and compares it to the computed MAC that is restored with the rest of the context from the previous session
FINALIZE	Message is processed in multiple sessions and the current session is the last one. All context data is restored from the previous session and the key is written to the Key Register. The final MAC is computed and encrypted

- Whenever AS is set to Initialize or Initialize/Finalize, context registers must be zero.
- If the AS field is not set to either Initialize/Finalize or Finalize and the ICV_TEST bit is set to 1, the illegal-mode error is generated. This does not apply in case when only ICV check is requested as described for AS = UPDATE.
- The Additional Algorithm Information (AAI) field must be set to 80h for CCM to be activated. The C2K bit is used to select a key register. If C2K = 0, CCM uses the key in the Class 1 Key Register. If C2K = 1, CCM uses the key in the Class 2 Key Register. Setting the DK bit causes the illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.11.4 AES CCM mode use of the Context Register

The AES CCM mode uses the Context Register as follows:

- B0 and the initial counter CTR0 must be provided in the context before the first chunk of the message is to be processed. During initialization, the initial counter CTR0 is encrypted in the CTR mode and B0 (which functions like a CBC-MAC IV in CCM) is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B0 and written to the lower 32 bits of the context dword 6.
- If there is AAD, the first block of it defines its size, and that value is decoded and written to the upper 32 bits of context dword 6. All of the context data must be restored before the next chunk of the message is to be processed in multi-session processing.
- For CCM encryption, the ICV (encrypted final MAC) is written to context words 4 and 5. For CCM decryption, the ICV (received MAC), which is always encrypted, is decrypted to dwords 4 and 5. The final computed MAC is written (in clear) to dwords 0 and 1.

Table 11-100. Context usage in CCM mode encryption

Context DWord	Initial-input definition	Intermediate definition	Final-output definition
0	B0[127:64]	intermediate MAC state	MAC[127:64]
1	B0[63:0]	intermediate MAC state	MAC[63:0]
2	CTR0[127:64]	CTR[127:64]	-
3	CTR0[63:0]	CTR[63:0]	-
4	-	E(CTR0)[127:64]. ¹	E(MAC)[127:64]
5	-	E(CTR0)[63:0] ¹	E(MAC)[63:0]
6	-	AAD size, MAC size; see Table 11-102	-

1. E(x) means encrypted x

Table 11-101. Context usage in CCM mode decryption

Context DWord	Initial-input definition	Context-switching Definition	Final-result definition
0	B0[127:64]	intermediate MAC state	MAC[127:64]
1	B0[63:0]	intermediate MAC state	MAC[63:0]
2	CTR0[127:64]	CTR[127:64]	-
3	CTR0[63:0]	CTR[63:0]	-
4	-	E(CTR0)[127:64]	Decrypted Received MAC[127:64]
5	-	E(CTR0)[63:0]	Decrypted Received MAC[63:0]
6	-	AAD size, MAC size	-

Table 11-102. Format of Context DWord 6 in AES-CCM mode

Bit 63	Bits 62-48	Bits 47-32	Bits 31-3	Bits 2-0
AAD Presence Flag	0	AAD Size	0	Encoded MAC Size

11.9.11.5 AES CCM mode use of the Data Size Register

The AES CCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written will be added to the current state of the register.
- CCM decrements the value in this register with every processed block.
- The content of the Data Size register must be divisible by 16 after the last write to it if the AS mode field is set to either "Update" or "Initialize". Otherwise, the data-size error is generated. In other words, message splitting can be done only on a 16-byte boundary.

11.9.11.6 AES CCM mode use of the Key Register

CCM key must be written to this register; it is always an encryption key.

11.9.11.7 AES CCM mode use of the Key Size Register

The AES CCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

11.9.11.8 AES CCM mode use of the ICV check

The AES CCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting the ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"-otherwise the illegal-mode error is generated, unless data size is 0 indicating ICV check is only requested. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input data FIFO after the message data. In CCM, received ICV is always encrypted. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is for CCM encoded in the B0.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

11.9.12 AES GCM mode

AES GCM mode provides the following:

- Data confidentiality using counter mode (CTR). Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value does not overflow and "wrap around", but it is the user's responsibility to ensure that the same key value is not used again following a reset.
- Authentication (assurance of integrity) of the confidential data using a universal hash function (GHASH) that is defined over a binary Galois (that is, finite) field. GCM can also provide authentication assurance for additional data (AAD) that is not encrypted.
- Stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both of the following:
 - Accidental modifications of the data
 - Intentional, unauthorized modifications

11.9.12.1 GMAC

If the GCM input is restricted to data that is not encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data.

11.9.12.2 GCM data types

In the GCM mode, the FIFO data type must be set to the message data type for textdata (payload), AAD type for additional data, IV type for IV data and ICV type for the received ICV. These data types must always be provided in the following order:

1. IV
2. AAD
3. Message data

Any of these may be missing.

11.9.12.3 IV processing

IV is processed using GHASH function if the size of IV is not 12 bytes. The result of IV processing is the initial counter (Y0) value used for encryption/decryption. GHASH function is also performed on AAD and textdata before the MAC can be computed.

11.9.12.4 GCM initialization

GCM initialization is completed when all of the IV data is processed and the initial counter value (Y0) is computed as a result. For that to happen, IV data needs to be supplied through the Input Data FIFO and the FIFO data type must be set to IV.

11.9.12.5 AES GCM mode use of the Mode Register

The AES GCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption. Even though operations performed in either case are identical, the authentication is done of the cipher text in parallel with decryption when ENC = 0, and after encryption of each block when ENC = 1.

- The ICV_TEST bit must be set for GCM to compare computed MAC with the received MAC. The received MAC must be written to the input-data FIFO after message data and the FIFO data type must be set to ICV. If this bit is not set, GCM does not expect received ICV to be supplied after textdata. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- The Algorithm State (AS) field is defined for GCM as shown in this table:

Table 11-103. Mode Register[AS] operation selections in AES GCM

Operation	Value	Description
INITIALIZE	1h	Message is processed in multiple sessions and the current session processes final part of IV or textdata; do the final GHASH step, but do not compute MAC. NOTE: This AS state does not indicate initialization in GCM; instead, it means that the final step of the GHASH function is to be performed. In general, whenever the final GHASH iteration needs to be computed (either for GHASH(IV) or GHASH(AAD, ciphertext)), and the current message size provided in the Data Size Register is not equal to the total size for either IV, AAD, or textdata, AS should be set to INITIALIZE (1h). Consequently, an AS = 1h also indicates that the Context Registers 6-7 need to provide the total length of IV, AAD, or textdata for this to be accomplished.
INITIALIZE/ FINALIZE	3h	Message is processed in multiple sessions and the current session is the last. The final MAC is computed.
UPDATE	0h	Message is processed in multiple sessions (descriptors) and the current session is not the last. The descriptor contains a non-final part of IV, AAD, textdata (IV, AAD or textdata split between descriptors). If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session
FINALIZE	2h	Message is processed in a single session. MAC is computed.

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV_TEST bit is set to 1, the Illegal Mode error will be generated except for CICV-only jobs.

Proper AS field settings

Assume that a message has IV, AAD, and textdata and each of these types is split into two sessions (descriptors). The first IV descriptor should have AS set to "Update", the second IV Descriptor should have AS set to "Initialize", both AAD Descriptors and the first textdata descriptor should have AS field set to "Update", and the final Descriptor sets AS to "Initialize/Finalize".

- The Additional Algorithm Information (AAI) field must be set to 90h for GCM to be activated. The C2K bit is used to select a Key Register. If C2K = 0, GCM uses the key in the Class 1 Key Register. If C2K = 1, GCM uses the key in the Class 2 Key Register. Setting the DK bit causes an illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.12.6 AES GCM mode use of the Context Register

The AES GCM mode uses the Context Register as follows:

- New message processing does not need any data provided in the context. All of the context data is written back by the GCM mode and needs to be restored before the next data chunk is to be processed in the multi-session processing. The final MAC is written in the context dwords 0-1.
- The initial counter value required for encryption/decryption is derived from IV and written to dwords 4-5. It is also required for the MAC computation.
- The incremented counter is placed in dwords 2-3 and is updated with every encrypted/decrypted block.
- Bit sizes of IV, AAD and textdata are required for GHASH computation and are accumulated in dwords 6-7 when multi-session processing is used.

Table 11-104. Context usage in GCM mode

Context DWord	Context-switching definition	Final-result definition
0	MAC[0:63]	MAC[0:63]
1	MAC[64:127]	MAC[64:127]
2	Yi[0:63]	-
3	Yi[64:127]	-
4	Y0[0:63]	-
5	Y0[64:127]	-
6	IV bit size (during GHASH of IV), AAD bit size (during message processing)	-
7	textdata bit size	-

11.9.12.7 AES GCM Mode use of the Data Size Register

The AES GCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed (including IV, AAD and textdata) must be written to the Data Size register (IV and AAD sizes must include padding to the 16 byte boundary).
- The first write to this register initiates processing. It can also be written during processing in which case the value written will be accumulated to the current state of the register.
- GCM decrements the value in this register with every processed block.
- Message splitting must be done only on a 16-byte boundary.

11.9.12.8 AES GCM mode use of the Class 1 IV Size Register

The Class 1 IV Size register is written with the number of bytes in the last IV block. If the total IV size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the IV used in the GHASH computation. To do this, GCM also uses the fact that IV size padded to a 16-byte boundary is written to the Data Size register.

11.9.12.9 AES GCM mode use of the AAD Size Register

The AAD Size register is written with the number of bytes in the last AAD block. If the total AAD size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the AAD used in the GHASH computation. To do this, GCM also uses the fact that AAD size padded to a 16-byte boundary is written to the Data Size register.

11.9.12.10 AES GCM mode use of the Class 1 ICV Size Register

The AES GCM mode uses the Class 1 ICV Size Register as follows:

- This Class 1 register is used to provide ICV/MAC byte-size when it is other than 16 bytes. In that case, the remaining bytes of the ICV/MAC written to the context is zero.
- If the ICV mode bit is set, the Class 1 ICV Size register also determines the number of bytes in the received ICV. Supported values for ICV size are 4 to 16 bytes. If this register is 0, ICV size will be 16 bytes.

11.9.12.11 AES GCM mode use of the Key Register

GCM key must be written to this register; it is always an encryption key.

11.9.12.12 AES GCM mode use of the Key Size Register

The AES GCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Key Size register.
- Any value other than 16, 24, or 32 causes key-size error to be generated.

11.9.12.13 AES GCM mode use of the ICV check

The AES GCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the Illegal Mode error is generated except for CICV-only jobs. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input-data FIFO after the message data. The FIFO data-type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is for GCM written to the Class 1 ICV Size register.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

11.9.13 AESA optimization modes

The AESA optimization modes are as follows:

- CBC-XCBC
- CTR-XCBC
- CBC-CMAC
- CTR-CMAC
- CTR-CMAC-LTE

These modes are described together because of their similarities. Unlike CCM and GCM, these optimization modes are not actual AES modes, but instead are hardware modes that AESA implements to perform two block cipher modes of AES on the same data in the same hardware. These modes are typically used to support specific networking protocols.

CBC-XCBC, CTR-XCBC, CBC-CMAC, and CTR-CMAC modes combine a confidentiality mode with an authentication mode in a way suitable for IPsec. In particular, the encrypted data is processed with XCBC-MAC or CMAC mode.

11.9.13.1 CTR-XCBC and CTR-CMAC modes data format

The CTR-XCBC and CTR-CMAC modes data format is a 24-byte header processed with XCBC, followed by 1 or more 16-byte blocks of data processed with CBC and then XCBC, followed by 0 or 1 XCBC-only words of 4 bytes.

11.9.13.2 CTR-XCBC and CTR-CMAC modes message format

The CTR-XCBC and CTR-CMAC modes message format is a 16-byte header processed with XCBC, followed by 1 or more 4-byte words of data processed with CTR and then XCBC, followed by 0 or 1 XCBC-only words of 4 bytes.

11.9.13.3 CTR-CMAC-LTE for LTE PDCP control-plane processing

CTR-CMAC-LTE is designed for LTE PDCP control-plane processing. In particular, unencrypted data is processed with CMAC mode. The data format for CTR-CMAC-LTE is that of a 9-byte segment that is only authenticated, followed by any number of bytes that is both authenticated and encrypted.

11.9.13.4 Authentication-only data

Authentication- or MAC-only data requires special authentication data (SAD) type to be used as FIFO data type. The CBC or CTR data uses message data type, while ICV is using ICV data type.

11.9.13.5 AES optimization modes use of the Mode Register

The AES optimization modes use the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV bit must be set for computed MAC to be compared with the received MAC. The received MAC must be written to the Input Data FIFO after message data and the FIFO data type must be set to ICV. If this bit is not set, AESA does not expect received ICV to be supplied after textdata. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- The Algorithm State (AS) field is defined for all these optimization modes as described in this table.

Table 11-105. Mode Register[AS] operation selections in AES optimization modes

Operation	Value	Description
INITIALIZE	1h	<p>Message is processed in multiple sessions and the current session is the first one.</p> <ul style="list-style-type: none"> • During CBC-XCBC initialization, derived key K3 for XCBC-MAC that is XOR-ed with the last message block is computed and stored in the context to be used in the last processing session. Derived key K1 used as the AES key for XCBC-MAC processing is computed and replaces the original XCBC-MAC key in the Class 2 Key register. XCBC-MAC derived key K2 is never computed, because the message length cannot be divisible by 16 bytes.

Table continues on the next page...

Table 11-105. Mode Register[AS] operation selections in AES optimization modes (continued)

Operation	Value	Description
		<ul style="list-style-type: none"> During CTR-XCBC initialization both K2 and K3 XCBC-MAC keys are computed. Derived key K1, used as the AES key for XCBC-MAC processing, is computed and replaces the original XCBC-MAC key in the Class 2 Key register. During CMAC-based mode initialization, the key-derived value L is computed and written to Context words 4 and 5. Derived keys K1 and K2, which are derived from L, are computed as needed but not saved into Context. CTR-CMAC, CTR-XCBC and CTR-CMAC-LTE will generate the Data Size Error if the data size value is not divisible by 16.
INITIALIZE/ FINALIZE	-	Message is processed in a single session and the final MAC is computed. For CBC-based modes, only CBC IV must be written to the context. For CTR-based modes, only initial CTR must be written to the context.
UPDATE	-	Message is processed in multiple sessions and the current session is neither the first nor the last. For XCBC-MAC-based modes, keys K2 and K3 are provided in the context and the derived key K1 is provided in the Class 2 Key Register. For CMAC-based modes, Key-derived value L is provided in the Context Register, and is used to compute K1 and K2. If decryption is requested, and data size is written to 0, and ICV bit is 1 - AS=UPDATE means that Check ICV (CICV) job is requested. Data size must be written, even if written with 0. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session. CTR-CMAC, CTR-XCBC and CTR-CMAC-LTE will generate the Data Size Error if the data size value is not divisible by 16. Note that a data size of 0 is not supported for CTR-CMAC-LTE
FINALIZE	-	Message is processed in multiple sessions and the current session is the last one. For XCBC-MAC based modes keys K2 and K3 are provided in the context and the derived key K1 is provided in the Class 2 Key Register. For CMAC-based modes key-derived value L is provided in the context. The final MAC is computed using either K1 or K2, derived from L as needed

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV bit is set to 1, the Illegal Mode error will be generated, except for CICV-only jobs.
- The Additional Algorithm Information (AAI) field's lower 8 bits must be set as shown in [Table 11-106](#) for the desired Optimization Mode to be activated.
- The Decrypt Key bit (DK) can be used in CBC-based Optimization Modes for decryption to avoid the time required for key expansion by providing already expanded key in the Class 1 Key Register. This must be used in multi-session processing if the expanded key is saved and later restored to the Key Register. For any CTR-based Optimization Mode, setting the DK (Decrypt Key) bit causes the illegal-mode error.

Table 11-106. Optimization modes

AAI value (mode [19:27])	Optimization mode name	Confidentiality mode	Authentication mode
0A0h	CBC-XCBC	CBC	XCBC-MAC
0B0h	CTR-XCBC	CTR	XCBC-MAC
0C0h	CBC-CMAC	CBC	CMAC
0D0h	CBC-CMAC-LTE	CBC	CMAC
0E0h	CTR-CMAC	CTR	CMAC

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

11.9.13.6 AES optimization modes use of the Context Register

The AES optimization modes use the Context Register as follows:

- A new message processing session needs CBC IV to be provided in the context for CBC-based Optimization Modes, and the initial counter (CTR0) for CTR-based Optimization Modes.
- The final MAC is in all cases written to context words 0-1.
- The XCBC derived keys K3 and K2 are written back to the context by CTR-XCBC and only K3 by CBC-XCBC. Because the data format for the CBC-XCBC guarantees that the message size is never divisible by 16, only K3 XCBC key is needed. For the same reason, splitting of the CBC processed data on a 16 byte boundary means that XCBC/CMAC processed data is not split on the 16 byte boundary. Hence, the context DWord 6 is used by CBC-XCBC and CBC-CMAC to save the least significant word (8 bytes) of the last CBC encrypted data block from the previous session because it cannot be processed until the next 8 bytes are known. Similarly, for CTR-CMAC-LTE, context DWords 6-7 are used to save 9 bytes of previous data/header for the next processing session.
- This mode's 9-byte authenticate-only header cannot be processed until CTR message data is available. Hence, if only header is provided in the current session, it will be saved to the context for processing in the next session. The same applies for message data in later sessions.
- The data continuation flag in context DWord 7 is used only for CTR-CMAC-LTE encryption to signal the presence of data from the previous session and is set/reset by the mode logic automatically in case of context switching.
- For decryption, the context DWords 6-7 are always used to store the remainder of the previous block of data, because the current block on the input needs to be decrypted before a new AES block can be formed by concatenating 9 bytes from the previous block with 7 bytes from the current input block.
- In CTR-XCBC, CTR-CMAC, CTR-CMAC-LTE modes, message splitting must be done on a 16-byte boundary for multi-session message processing.
- The CBC-XCBC and CBC-CMAC modes require that the first session processes the whole authenticate-only header and at least 16 bytes of the CBC data. From that point on, message splitting is supported on a 16-byte boundary of CBC data.

Table 11-107. Context usage in CBC-XCBC mode

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
0	-	MAC[127:64]	MAC[127:64]
1	-	MAC[63:0]	MAC[63:0]
2	CBC-IV[127:64]	CBC-IVi[127:64]	-
3	CBC-IV[63:0]	CBC-IVi[63:0]	-
4	-	K3[127:64]	-
5	-	K3[63:0]	-
6	-	LS word of CBC encrypted data from previous session	-

Table 11-108. Context usage in CTR-XCBC mode

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
0	-	MAC[127:64]	MAC[127:64]
1	-	MAC[63:0]	MAC[63:0]
2	CTR0[127:64]	CTRi[127:64]	-
3	CTR0[63:0]	CTRi[63:0]	-
4	-	K3[127:64]	-
5	-	K3[63:0]	-
6	-	K2[127:64]	-
7	-	K2[63:0]	-

Table 11-109. Context usage in CBC-CMAC mode

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
0	-	MAC[127:64]	MAC[127:64]
1	-	MAC[63:0]	MAC[63:0]
2	CBC-IV[127:64]	CBC-IVi[127:64]	-
3	CBC-IV[63:0]	CBC-IVi[63:0]	-
4	-	L[127:64]	-
5	-	L[63:0]	-
6	-	LS word of CBC encrypted data from previous session	-

Table 11-110. Context usage in CTR-CMAC mode

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
0	-	MAC[127:64]	MAC[127:64]
1	-	MAC[63:0]	MAC[63:0]
2	CTR0[127:64]	CTRi[127:64]	-
3	CTR0[63:0]	CTRi[63:0]	-

Table continues on the next page...

Table 11-110. Context usage in CTR-CMAC mode (continued)

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
4	-	L[127:64]	-
5	-	L[63:0]	-

Table 11-111. Context usage in CTR-CMAC-LTE mode

Context DWord	Initial-input definition	Context-switching definition	Final-result definition
0	-	MAC[127:64]	MAC[127:64]
1	-	MAC[63:0]	MAC[63:0]
2	CTR0[127:64]	CTRi[127:64]	-
3	CTR0[63:0]	CTRi[63:0]	-
4	-	L[127:64]	-
5	-	L[63:0]	-
6	-	9 LS bytes of previous session's data and data continuation flag	Encrypted MAC[0:31]
7	-		-

This table summarizes all the ways in which a message can be split for processing in multiple sessions and what settings should be used for each.

Table 11-112. Context DWord 6-7 for CTR-CMAC-LTE mode

Bits 127-56	Bits 55-8	Bit 7	Bits 6-0
9 LSBs of previous session's data	0	Data continuation flag	0

NOTE

The check-ICV-only session assumes that a complete message has been processed and final ICV computed in one or more previous session. When combining sessions from this table to process a message, note that only session 2.1 can be repeated multiple times. Also, if there is a session with AS = 1, then there can only be one such session and there must be one and only one with AS = 2 but none with AS = 3. If there is a session with AS = 3, then there cannot be any other sessions except check-ICV-only.

Table 11-113. Multi-session processing options for AES optimization modes

Session number	Session description	Mode supported	Mode AS field	Mode ICV field	AAD size	Data size
1.1	Initialization-only	CTR-XCBC CTR-CMAC	1	0	0	0

Table continues on the next page...

Table 11-113. Multi-session processing options for AES optimization modes (continued)

	(Computing L for CMAC or K2/K3 for XCBC)	CTR-CMAC-LTE				
1.2	Header-only	CTR-XCBC CTR-CMAC CTR-CMAC-LTE	1 or 0	0	0 9	16
1.3	Header and partial message	CTR-XCBC CTR-CMAC CTR-CMAC-LTE CBC-XCBC CBC-CMAC	1 or 0 1	0	0 9 8	16 + 16 * (Number of MDATA blocks) 24 + 16 * (Number of MDATA blocks)
2.1	Message-only	All	0	0	0	16 * (Number of MDATA blocks)
2.2	Final message	CTR-XCBC CTR-CMAC CTR-CMAC-LTE CBC-XCBC CBC-CMAC	2	1 (ENC=0) or 0	0	4 * (Number of MDATA nibbles) > 0 16 * (Number of MDATA blocks)
2.3	Final message and ESN	CTR-XCBC CTR-CMAC CBC-XCBC CBC-CMAC	2	1 (ENC=0) or 0	4	4 + 4 * (Number of MDATA nibbles) 4 + 16 * (Number of MDATA blocks)
3.1	ESN-only	CTR-XCBC CTR-CMAC CBC-XCBC CBC-CMAC	2	1 (ENC=0) or 0	4	4
4.1	Header and final message	CTR-CMAC-LTE CTR-XCBC CTR-CMAC CBC-XCBC CBC-CMAC	3 or 2 3 or 2 1 3 1	1 (ENC=0) or 0 0	9 0 8 0	> 16 16 + 4 * (Number of MDATA nibbles) 24 + 16 * (Number of MDATA blocks)
4.2	Header, final message and ESN	CTR-XCBC CTR-CMAC CBC-XCBC CBC-CMAC	2	1 (ENC=0) or 0	4 12	20 + 4 * (Number of MDATA nibbles) 28 + 16 * (Number of MDATA blocks)
5	Check-ICV-only (For ENC = 0)	All	0	1	0	0

11.9.13.7 AES optimization modes use of the Data Size Register

The AES optimization modes use the Data Size Register as follows:

- The byte-length of the message to be authenticated must be written to the Data Size Register. In case of CTR-CMAC-LTE, 16 must be written to this register for the 9-byte, authenticate-only header as the header is provided in 2 FIFO DWords to AES.
- The first write to this register initiates processing. It can also be written during processing, in which case the value written is accumulated to the current state of the register.
- An additional restriction applies to CTR-XCBC and CTR-CMAC modes: the final value of the Data Size Register must be a multiple of 4-byte words, otherwise the data-size error is generated.
- For all CMAC-based optimization modes, the data-size error is generated if the data size value is not divisible by 16 when AS mode field is set to INITIALIZE or UPDATE. CBC-based modes generate the same error if the data size value is 0 and the AS field has INITIALIZE bit set. This is to enforce that the header must be processed with INITIALIZE bit set.

11.9.13.8 AES optimization modes use of the AAD Size Register

The AES optimization modes use the AAD Size Register as follows:

- The AAD Size Register is written with the number of bytes in the Authentication-only data. Only the low 4 bits will be registered.
- For CTR-XCBC and CTR-CMAC, because authenticate-only header is always 16 bytes long, the value of the AAD Size Register reflects the presence or absence of the optional trailing 4 bytes.
- For CBC-XCBC and CBC-CMAC, the content of this register is either 8 or 12 based on the fixed size of the 24-byte header and the optional 4 trailing bytes.
- For CTR-CMAC-LTE, 9 must be written to this register for any session that processes the 9-byte header.
- The AAD Size Register must be written before the last write to the Data Size Register.
- Writes to the AAD Size Register are cumulative.

11.9.13.9 AES optimization modes use of the Class 1 ICV Size Register

The AES optimization modes use the Class 1 ICV Size Register as follows:

- For optimization modes other than CTR-CMAC-LTE, this Class 1 register is used to provide received ICV/MAC byte-size when it is other than 16 bytes. The Class 1 ICV Size register also determines the number of bytes in the computed ICV/MAC that is provided in context DWords 0-1.
- The computed MAC written to the context is always 16 bytes long.
- Supported values for the ICV size are 4 to 16 bytes. If this register is 0, the ICV size is 16 bytes.
- For CTR-CMAC-LTE, the ICV length is always 4 bytes.
- As long as any bytes trailing the received ICV are zero, there is no need to write this register.
- The encrypted MAC provided in context DWord 6 is also always 4 bytes long.

11.9.13.10 AES optimization modes use of the Class 1 Key Register

The AES optimization modes use the Class 1 Key Register as follows:

- The confidentiality key must be written to this register.
- For CTR-based Optimization Modes, it is always an encryption key.
- For CBC-based Optimization Modes, this key must be a decryption key when DK mode bit (AAI field MSB) is set.

11.9.13.11 AES optimization modes use of the Class 2 Key Register

The AES optimization modes use the Class 2 Key Register as follows:

- The Class 2 Key Register is used to provide a 16 byte Authentication key.
- It is overwritten by the derived key K1 during XCBC-MAC initialization.
- It is not overwritten for CMAC-based Optimization Modes.

11.9.13.12 AES optimization modes use of the Class 1 Key Size Register

The AES optimization modes use the Class 1 Key Size Register as follows:

- The total number of confidentiality-key bytes must be written to the Class 1 Key Size register by the time that Mode Register and Data Size Register have been written.
- Any value other than 16, 24, or 32 causes key-size error to be generated.
- The out-of-sequence error is generated if this Key Size Register is not written by the time both Mode Register and Data Size Register are written.

11.9.13.13 AES optimization modes use of the Class 2 Key Size Register

The AES optimization modes use the Class 2 Key Size Register as follows:

- This value is the length of the written authentication key.
- For XCBC-MAC-based modes, they must be 16 bytes long, otherwise the key-size error is generated.
- For CMAC-based modes, 16-, 24-, and 32-byte keys are supported.

11.9.13.14 AES optimization modes use of the ICV check

The AES optimization modes use ICV checking as follows:

- Automatic ICV checking is enabled by setting ICV bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the illegal-mode error is generated, except for CICV-only jobs. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input-data FIFO after the message data.
- The FIFO data type must be set to ICV when it is put on the FIFO.
- The size of the received ICV is written to the Class 1 ICV Size register.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

11.9.13.15 AES optimization modes error conditions

This table shows all the error checks implemented for the AES optimization modes.

Table 11-114. AES optimization modes error conditions

Description	Modes affected	Mode DK bit	Mode AS field	Mode ICV bit	Mode ENC bit	Data size	Key size	Error type generated
ICV check can be requested only when decrypting.	All	-	-	1	1	-	-	Mode error
Must compute final ICV/MAC before checking ICV except for check-ICV-only jobs that have no data to be processed (AS = UPDATE, DS = 0).	All	-	1 or 0	1	-	> 0	-	

Table continues on the next page...

Table 11-114. AES optimization modes error conditions (continued)

Description	Modes affected	Mode DK bit	Mode AS field	Mode ICV bit	Mode ENC bit	Data size	Key size	Error type generated
CTR mode does not use the decrypt key.	CTR-XCBC CTR-CMAC CTR-CMAC-LTE	1	-	-	-	-	-	
Using the decrypt key for encryption would produce wrong result.	CBC-XCBC CBC-CMAC	1	-	-	1	-	-	
Only a messages with integral number of bytes allowed.	All	-	-	-	-	Bit-size not divisible by 8	-	Data-size error
CTR message data must consist of 1 or more 4-byte chunks, that is, ESN cannot be split between blocks.	CTR-CMAC CTR-XCBC	-	-	-	-	Last block of CTR message data does not have 4, 8, 12 or 16 bytes	-	
Finalize-only jobs are not supported.	All	-	3 or 2	-	-	0	-	
Header must be processed in an initialize session.	CBC-XCBC CBC-CMAC	-	1	-	-	0	-	
Must switch the context on a block boundary.	CTR-XCBC CTR-CMAC CTR-CMAC-LTE	-	1 or 0	-	-	Not divisible by 16	-	
Wrong Class 1 key size	All	-	-	-	-	-	Class 1 key size not 16, 24, or 32	
Wrong Class 2 key size	CBC-XCBC CTR-XCBC	-	-	-	-	-	Class 2 key size not 16	
	CBC-CMAC CTR-CMAC CTR-CMAC-LTE	-	-	-	-	-	Class 2 key size not 16, 24, or 32	
Key sizes are not written before both mode and data size are written.	All	-	-	-	-	-	-	Out-of-sequence error
The header is loaded for processing in a non-initialize session.	CBC-XCBC CBC-CMAC	-	2 or 0	-	-	-	-	
The computed ICV does not match received ICV.	All	-	3 or 2	1	0	> 0	-	ICV error

Table 11-114. AES optimization modes error conditions

Description	Modes affected	Mode DK bit	Mode AS field	Mode ICV bit	Mode ENC bit	Data size	Key size	Error type generated
			(or 0 for DS=0)			(or 0 for AS=0)		

11.10 ZUC encryption accelerator (ZUCE) functionality

The ZUCE hardware accelerator implements the encryption mode of operation of the ZUC algorithm. The encryption mode confidentiality algorithm is defined as a stream cipher that encrypts/decrypts blocks of data between 1 and 20000 bits in length. The f9 authentication mode of the ZUC algorithm is implemented in the ZUCE CHA. See [Differences between ZUCE and ZUCA](#) for more information. The features of the ZUCE accelerator include the following:

- Message encryption and decryption in encryption (UEA2) mode
- Throughput of up to 4 bytes per cycle
- Support for multiple session message processing through context switching
- Support for Descriptor sharing
- Total message size of up to 2^{32} bits (processed in chunks of no more than $2^{17}-1$ bytes (per session))
- Support for any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

11.10.1 Differences between ZUCE and ZUCA

ZUC is a proposed third radio interface cryptographic algorithm set for LTE (in addition to Kasumi and SNOW), which was submitted as a candidate for standardization by a 3GPP member company. ZUC forms the basis of the f8 encryption algorithm and f9 integrity/authentication algorithm. ZUC encryption (ZUCE) is a word-oriented stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialization value. ZUCE is programmed using Class 1 CCB registers, whereas ZUCA is programmed using Class 2 CCB registers. Note that it is possible to encrypt or decrypt data using ZUC and also hash the same data using ZUC authentication via "snooping", that is, passing the same data simultaneously to both CHAs ("in-snooping"), or passing the output of one CHA directly to the input of the other CHA ("out-snooping"). But in those versions of SEC that implement more than one DECO but only

one ZUCE CHA and one ZUCA CHA, the descriptor must select the ZUCA CHA first. Selecting the ZUCE CHA and then selecting the ZUCA CHA within the same descriptor results in an error indication.

11.10.2 ZUCE use of the Mode Register

ZUCE uses the Mode Register as follows:

- The ZUC encryption accelerator (ZUCE) is enabled by setting the Algorithm (ALG) field of the Class 1 Mode Register to B0.
- The encryption mode is enabled by setting the Additional Algorithm Information (AAI) field to C0.
- The Algorithm State (AS) field should be set to "Initialize" state when a new message is to be processed. The ZUCE accelerator initializes the core engine (keystream generator) based on the key and initialization parameters COUNT-C, BEARER and DIRECTION in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register. The AS field should be reset (or set to "Update" state) after context switch, assuming that Context Registers are restored, when continuing message processing. In this case, the state of the keystream generator necessary for continuation of message processing is in the Context Registers and initialization is not needed.
- Other fields in the Mode Register have no effect on encryption mode.
- If the AAI field is set to a value that does not correspond to encryption mode, the illegal-mode error is generated. The message processing is initiated by writing a message size to the Data Size Register.

11.10.3 ZUCE use of the Context Register

ZUCE uses the Class 1 Key and Context registers. The usage of the Key and Context registers in the encryption mode is shown in [Table 11-116](#). The symbols in the Update Input Definition column represent values written back by ZUCE. These values comprise the state of the keystream generator that must be restored after context switch for the message processing to continue. In encryption mode, the Context Register is treated as an extension of the Key Register, that is, it is automatically encrypted when saved and decrypted when restored. The symbol IV represents a value that must be written to the Context Register when starting a new Job in the encryption mode. This value consists of ZUCE initialization parameters in the following order:

Table 11-115. Context 1 usage in ZUCE mode

0-31	32-36	37	38-63
Count-C	Bearer	Direction	0

Table 11-116. Key/Context Register usage in ZUCE mode

Register	DWord Number	Initialize-input definition	Update-input definition
Key Register	0	Key[0:63]	s0, s1
	1	Key[64:127]	s2, s3
	2	-	s4, s5
	3	-	s6, s7
Context Register	0	Count-C Bearer Direction 0	s8, s9
	1	-	-
	2	-	-
	3	-	r1
	4	-	r2, r3
	5	-	s10, s11
	6	-	s12, s13
	7	-	s14, s15

11.10.4 ZUCE use of the Data Size Register

ZUCE uses the Data Size Register as follows:

- ZUCE uses the 16 lsbs of the Class 1 Data Size register to indicate the number of bytes of input data, and the 3 msbs to indicate the number of valid bits in the last byte.
- Writing to the LSB of the Class 1 Data Size register initiates processing. ZUCE internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero.
- If 0 is written to the Data Size Register and the AS field of the Mode Register is set to "Initialize", ZUCE keystream generator is initialized and the Context Register contains this initialized state.

11.10.5 ZUCE use of the Key Register

A 128-bit key must be written to the Class 1 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize". The key is necessary for the initialization of the keystream generator but it is not needed when the AS field of the Mode Register is set to

"Update", that is, when a message processing is continued after context switch. However, the Key Register is used to implement internal state of the keystream generator as depicted in [Table 11-116](#).

11.10.6 ZUCE use of the Key Size Register

Writing to this register is not required by ZUCE, because the ZUC encryption key is always 16 bytes long. However, writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

11.11 ZUC authentication accelerator (ZUCA) functionality

The ZUCA hardware accelerator implements the f9 authentication mode of the ZUC algorithm. The ZUCA algorithm is based on the same stream cipher (ZUC) as is used by the encryption algorithm (ZUCE). The f8 encryption mode of the ZUC algorithm is implemented in the ZUCE CHA. See [Differences between ZUCE and ZUCA](#) for more information. ZUC Authentication is a word-oriented stream cipher that generates a 32-bit Message Authentication Code (MAC) under the control of a 128-bit key and a 128-bit initialization value. The message may be between 1 and 20000 bits in length.

The features of ZUCA include the following:

- Computation of 32-bit MAC in ZUC authentication (UIA2) mode
- Automatic comparison of the received and computed MAC values (ICV check)
- Throughput of up to 2 bytes per cycle
- Support for multiple session message processing through context switching
- Support for Descriptor sharing
- Total message size of up to 2^{32} bits (processed in chunks of no more than 2^{17} -8 bytes per session)
- Supports any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

11.11.1 ZUCA use of the Mode Register

The ZUCA uses the Mode Register as follows:

- The ZUC authentication accelerator (ZUCA) is enabled by setting the Algorithm (ALG) field of the Class 2 Mode Register to C0.

- The authentication mode is enabled by setting the Additional Algorithm Information (AAI) field to C8.
- The Algorithm State (AS) field must be set to "Initialize" state when the first session of message processing is to be performed. This assumes that message processing is split into multiple sessions, that is, that the first one is not also the final session. The ZUCA accelerator initializes the core engine (keystream generator) based on the key and an IV built from initialization parameters COUNT-I, DIRECTION, BEARER and FRESH in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register.
- If an initialization-only job is executed (data size is 0 and AS set for initialization), "Finalize" in the AS setting is ignored, as well as the ICV mode bit.
- If decryption is requested, and data size is not written or is set to 0, and ICV bit is 1 - AS = "Update" means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
- The AS field must be set to "Finalize" state when the last session of message processing is to be performed. This enables computation of the MAC.
- The AS field must be set to "Initialize/Finalize" state when the whole message is processed in one session.
- The ICV bit of the Mode Register must be set for the authentication mode to compare computed MAC/ICV with the received ICV. The received ICV must be provided through the Input Data FIFO following the message data at which time the FIFO data type must be set to ICV. If this bit is reset, the authentication mode does not expect ICV to be put on Input Data FIFO.
- The illegal-mode error is generated if ICV bit is set but AS field is not set to either "Initialize/Finalize" or "Finalize" state, except for CICV-only jobs, that is, the data size is non-zero and valid (there will be no more writes to the Data Size Register).
- If the AAI field is set to a value that does not correspond to authentication mode, the illegal-mode error is generated. The message processing is initiated by writing a message size to the Data Size Register.
- When SEC descriptor sharing mode is used with ZUCA, clear mode, followed by clear done interrupt command, must be issued between ZUCA jobs.

11.11.2 ZUCA use of the Context Register

ZUCA uses the Context Register as follows:

- This table shows context usage in the authentication mode that is relevant for its programming.

Table 11-117. Context usage in ZUCA mode

Register	Word number	Initialization-input definition	Update state (for context switching)	Finalize-output definition
Key Register	0	Key[0:63]	-	-
	1	Key[64:127]	-	-
	2	-	-	-
	3	-	-	-
	4	-	IV	-
	5	-	z1, z2	-
	6	-	z3, z4	-
	7	-	z5, bit length	-
Context Register	0	Count-C 0 Direction 0	-	{MAC,32'h0}
	1	{FRESH, 32'h0} (3G) {Bearer, 59'h0} (LTE)	-	-

- For 3G, the IV value is built as shown in this table.

Table 11-118. Usage of Context 2 for 3G in ZUCA mode

0-31	32-36	37	38-63	64-95
Count-C	0	Direction	0	FRESH

- LTE systems do not include a FRESH value in the authentication IV value. It is instead built as shown in this table.

Table 11-119. Usage of Context 2 for LTE in ZUCA mode

0-31	32-36	37	38-63	64-68	69-95
Count-C	0	Direction	0	Bearer	0

- At the end of processing, ZUCA overwrites IV in the context word 0 with the MAC/ICV. Because MAC is a 32-bit value, it is written to low-order bit locations (right-justified) and the remaining bits are cleared.
- Values z1-z5 are the keystream words computed during initialization of the authentication mode by the keystream generator. After the initialization stage is complete, the keystream generator is not active any more in authentication mode. The processing is based on the Galois Field (GF) multiplier implemented as part of the authentication mode logic. The bit length is a value copied from the Data Size Register to be used to compute the final MAC. In case of multi-session message

processing, this value represents the total message length as each session's data size is accumulated.

- To read only the final MAC value, the "finalize" option must be present in the AS mode setting. When saving context, the starting address must be the address of the first double word of the Key Registers.

11.11.3 ZUCA use of the Data Size Register

ZUCA uses the Data Size Register as follows:

- ZUCA uses the 16 lsbs of the Class 2 Data Size Register to indicate the number of bytes of input data, and bits 63-61 to indicate the number of valid bits in the last byte.
- Writing to the LSB of the Class 2 Data Size Register initiates processing.
- The number of input data bits that ZUCA is to process must be written into the lower 20 bits of the Class 2 Data Size register.
- ZUCA internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size Register and the AS field of the Mode Register is set to "Initialize", ZUCA keystream generator is initialized and the context contains this initialized state.
- In authentication mode, the data size must be divisible by 64 except when the AS field of the Mode Register is set to "Finalize" or "Initialize/Finalize". In other words, the message can be split for multi-session processing only on a 64-bit boundary. If this rule is violated, the illegal data-size error is generated.

11.11.4 ZUCA use of the Key Register

A 128-bit key must be written to the Class 2 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize" or "Initialize/Finalize". The key is necessary for the initialization of the keystream generator.

11.11.5 ZUCA use of the Key Size Register

Writing to this register is not required by ZUCA, because the ZUC authentication key is always 16 bytes long. However, writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

11.11.6 ZUCA use of ICV checking

ZUCA uses ICV checking as follows:

- Authentication mode can automatically compare received ICV with the computed ICV at the end of processing if the ICV bit of the Mode Register is set and the AS field is set to "Finalize" or "Initialize/Finalize".
- The received ICV must be supplied after message data through the Input Data FIFO. The FIFO data type for it must be set to ICV. The ZUCA mode ICV/MAC is always a 32-bit value.
- If the ICV mode bit is set but the AS field is set to "Initialize" or "Update", the illegal-mode error is generated, except for CICV-only jobs where no processing is done and only ICV check is performed as indicated by data size being 0. ZUCA generates ICV error if received and computed ICVs do not match. It is allowed to create jobs where there is no data to be processed, and only ICV is being checked. For this, the AS mode field should be reset.

Chapter 12

Trust Architecture modules

The SEC Trust Architecture functions are performed in the run-time integrity checker (RTIC), and the secure key module.

12.1 Run-time integrity checker (RTIC)

The run-time integrity checker (RTIC) is a component of SEC that is used to ensure the integrity of the peripheral memory contents and assist with boot authentication. The RTIC has the ability to verify the memory contents during system boot and during run-time execution. If the memory contents at runtime fail to match a reference hash signature, then a security violation is asserted. This security violation should then be captured by a monitoring device on the platform.

12.1.1 RTIC modes of operation

The RTIC modes of operation are described in this table.

Table 12-1. RTIC modes of operation

Mode	Description
One-time hash mode	<ul style="list-style-type: none">• Used during high assurance boot for code authentication or one time integrity checking• Stores a reference hash result internally and signals an interrupt to the processor
Continuous hash mode	<ul style="list-style-type: none">• Used at run time to continuously verify the integrity of memory contents• Checks a re-generated hash against an internally stored reference value and interrupts the processor only if an error occurs

12.1.2 RTIC initialization and operation

RTIC supports integrity checking of up to four, independent memory blocks. RTIC's Hash Register File stores a reference hash for each memory block. During the boot stage integrity checking, each independent memory block's content is hashed and the result is stored in the hash register file. At boot-time, the memory contents are read and hashed (authenticated) as quickly as possible (RTIC Throttle Register should be set to 00h) to minimize the performance impact at startup. The reference hash result for each memory block is stored in RTIC for the processor to compare against the signed code hash value. Once RTIC has finished hashing the boot image, RTIC interrupts software, which then can check the generated hash value against digitally-signed hash value(s) stored with the code. Software policy determines what actions to take in the event of a hash mismatch at boot time. Note that in chips supporting high-assurance boot, RTIC's boot-image hashing may take place after secure-boot software validates the first code to execute. This means that any unauthorized code modification would either be caught by the secure boot software before RTIC runs, or trusted software would detect the hash mismatch after RTIC had integrity-checked the boot image.

After the trusted boot software has verified the boot image, the software can put RTIC into run-time mode to ensure that the boot image remains uncorrupted. In run-time mode, RTIC periodically reads a small section of memory, waits for a specified period of time, and then reads another small section of memory. During this process, RTIC computes a hash of the software image. When RTIC has eventually read the entire software image, it compares the newly computed hash with the reference hash that was validated at boot time. If the RTIC hardware detects a hash mismatch, RTIC generates an interrupt and signals a security violation to the chip's security monitor (SecMon) hardware. If the hash matches, RTIC starts over and re-validates the software image. This process repeats until the chip is powered down or RTIC checking is turned off.

12.1.3 RTIC use of the Throttle Register

The RTIC scan rate is controlled using the Throttle Register. This allows the user to trade off the software image revalidation rate against memory bandwidth utilization.

Depending on the settings, the software image might be revalidated every few seconds or every few days. RTIC also implements a watchdog timer that can be used to ensure that an attacker isn't able to block RTIC's access to memory for an extended period of time. If a DMA read error, illegal address/length error, RTIC Watchdog time-out, or hash mismatch occurs, the RTIC enters an error state and signals a security violation. A hardware reset is required to resume operation.

12.1.4 RTIC use of command, configuration, and status registers

The RTIC controller holds the command/configuration registers, which are programmed through SEC's register interface. RTIC uses SEC's DMA interface only to read memory contents. The command/control registers are used to:

- Set the DMA burst and throttle level
- Specify which memory blocks to hash (one time and continuously)
- Enable/Disable/Clear interrupts
- Enable one-time or run-time hashing, software reset, and clear interrupts

A status register in the RTIC indicates the current state of the controller, which includes:

- Interrupt status
- Processing status
- Error status

The controller also contains a comparator to check the generated hash value against the reference hash value.

12.1.5 Initializing RTIC

At boot time, RTIC can be used to accelerate software-image verification. This is accomplished by first selecting the hash algorithm (SHA-256 or SHA-512) and one or more RTIC memory blocks by writing to the RTIC Control Register, specifying the areas of memory to be hashed by writing one or more pairs of RTIC Memory Block Address and Length registers and, if necessary, altering the endianness settings via the RTIC Control Register and then writing to the RTIC Command Register to initiate the hashing operations.

At chip-initialization time, RTIC is configured for run-time mode operation by writing to the RTIC Control Register, the RTIC Throttle Register the RTIC Watchdog Register and the RTIC Memory Block Address and Length registers and, if necessary, writing to the RTIC Control Register to change the endianness settings, and then writing to the RTIC Command Register to put RTIC into run-time mode.

12.1.6 RTIC Memory Block Address/Length Registers

Up to four independent memory blocks can be hashed by the RTIC, each with its own message digest (reference hash value). The RTIC scans through the memory blocks in the order they are defined in the RTIC Memory Block Address registers. Each of the four

Memory Blocks can be divided into two separate segments, with separate starting addresses and segment lengths. RTIC computes the hash over each Memory Block that is enabled by first reading segment 0 of the Memory Block and then appending segment 1.

Since there can be two segments per Memory Block, each memory block (A, B, C, D) is defined by two address/length register pairs (RTIC Memory Block Address 0 Register / RTIC Memory Block Length 0 Register and RTIC Memory Block Address 1 Register / RTIC Memory Block Length 1 Register). For each memory block, starting at the address indicated in the RTIC Memory Block Address 0 Register RTIC reads the number of memory bytes specified in the RTIC Memory Block Length 0 Register. When that is complete, RTIC starts at the address indicated in the RTIC Memory Block Address 1 Register and reads the number of memory bytes specified in the RTIC Memory Block Length 1 Register. If a Length Register is set to zero, RTIC skips over that memory segment. Once the specified number of bytes are read from both segments within the memory block, the data is hashed and stored (Hash-Once mode) or compared to the reference value (Run-Time mode). Information on additional registers used for RTIC configuration can be found in the sections describing the RTIC registers in SEC register page 6.

12.2 SEC virtualization and security domain identifiers (SDIDs)

This section describes the SEC features that are intended to support virtualization of the SEC hardware; that is, the ability to share the SEC functionality among multiple software entities.

12.2.1 Virtualization

SEC has been designed so that it can be "virtualized", that is, it can be shared among multiple software entities while still maintaining individual security protections for each of these entities. These software entities might include guest operating systems running under a hypervisor that allocates the chip's hardware resources among the guest OSs.

12.2.2 Security domain identifiers (SDIDs)

SEC implements 4096 security domain identifier (SDID) values that system control software (for example: hypervisor, kernel, operating system) can associate with different software entities. SDIDs are used to provide separation between software entities, and are used with black keys, blobs, and trusted descriptors. An SDID is a static value that must

be maintained across power cycles as it is used to provide separation even across power cycles. A unique SDID value can be associated with each software entity, or the same SDID value can be associated with multiple software entities, or multiple SDID values can be associated with a single software entity. The SDID values must be assigned at boot configuration time, and the same SDID must always be assigned to the same software entity. In the case of a guest OS running under a hypervisor, the guest OS may assign some of its different SDID values to processes under the guest OS's control. Note that the hypervisor is itself one of the software entities that can utilize SEC's functionality, and can assign itself as many SDID values as it wishes. SEC imposes no restrictions on how these SDID values are assigned, but simply uses the SDID values to control how data is shared among SDID assignees, or kept private to each assignee. SDID values are assigned to RTIC and to Job Rings by writing to registers in page 0 of SEC's register address space. SDID values are assigned to jobs submitted via the queue manager interface and the AIOP interface via fields in the Dequeue response Frame Queue Descriptor Context (FQD_CTX) or in the Flow Context (FLC). The sections below describe how SEC uses the SDID values to "virtualize" black keys, trusted descriptors and blobs.

12.2.3 TrustZone SecureWorld

SEC recognizes TrustZone SecureWorld as a unique software entity with special privileges, and identifies SecureWorld using a special "TZ" security identifier. All SEC registers that are used to hold 12-bit SDID values also have a separate TZ bit. Hardware signals ensure that only TrustZone SecureWorld can write to TZ bits. TrustZone SecureWorld indicates that specific SEC resources belong to SecureWorld by setting the TZ bit to 1. This allows SecureWorld to generate black keys that cannot be encrypted or decrypted by non-SecureWorld, to encapsulate and decapsulate blobs that cannot be encapsulated or decapsulated by non-SecureWorld, and to claim SEC Job Rings and Secure Memory partitions for its exclusive use. The sections below describe how SEC enforces these SecureWorld privileges.

12.3 Special-purpose cryptographic keys

SEC provides protection of session keys by means of black keys (see [Black keys](#)), integrity protection of SEC descriptors by means of trusted descriptors (see [Trusted descriptors](#)), and protection of long-term secrets by means of blobs (see [Blobs](#)). All of these protection mechanisms make use of special-purpose cryptographic keys that are managed by SEC.

12.3.1 Initializing and clearing black and trusted descriptor keys

The SEC hardware implements special black key key encryption keys (see [Black keys](#)) and trusted descriptor signing keys (see [Trusted descriptors](#)). These must be initialized to random values each time that SEC powers up. These keys are cleared when SecMon enters a FAIL state. The hardware implementation ensures that only SEC itself can use these keys. The values cannot be read or extracted from the chip by any means. However, test values can be written or read by software for debug purposes when SEC is in non-secure mode, (see [Keys available in non-secure mode](#)).

12.3.2 Black keys and JDKEK/TDKEK

One special cryptographic key used with the black key mechanism is the 256-bit job descriptor key encryption key (JDKEK) (see [Black keys](#)). When a job descriptor instructs SEC to store a Key Register into memory, the hardware first encrypts the content of the Key Register using the JDKEK and then stores the resulting black key into memory. When a job descriptor later references that key, the descriptor identifies the key as a black key, causing the hardware to decrypt the key using the JDKEK before loading the key into a Key Register. Trusted descriptors can also use the JDKEK, but they are permitted to choose the 256-bit trusted descriptor key encryption key (TDKEK) instead of the JDKEK. Using the TDKEK ensures that only trusted descriptors can use particularly sensitive keys, such as keys that are used to derive session keys. If a TDKEK-encrypted key is embedded as immediate data within a trusted descriptor, this ensures that no other key could be substituted for that particular key.

12.3.3 Trusted descriptors and TDSK

The SEC hardware controls use of the 256-bit trusted descriptor signing key (TDSK) that is used to compute the signature (keyed hash) over trusted descriptors (see [Trusted descriptors](#)). The TDSK is used for verifying the signature whenever a trusted descriptor is executed. The TDSK is used to sign a descriptor only if the descriptor is executed in a specially privileged Job Ring, or if a trusted descriptor modifies itself during execution.

12.3.4 Master key and blobs

The special cryptographic key used for blobs is the 256-bit master key that SEC receives from SecMon. The SEC hardware uses this master key to derive keys that are used for blob encryption and decryption when SEC is in secure mode or trusted mode, but uses a known test key for key derivation when SEC is in non-secure mode or fail mode.

12.4 Black keys

SEC's black key mechanism is intended for protection of user keys against bus snooping while the keys are being written to or read from memory external to the SoC. The black key mechanism automatically encapsulates and decapsulates cryptographic keys on-the-fly in an encrypted data structure called a black key. Before a value is copied from a key register to memory, SEC automatically encrypts the key as a black key (encrypted key) using as the encryption key the current value in the JDKEKR or TDKEKR, modified via the appropriate TZ/SDID value. Thus, each security domain (and TrustZone SecureWorld) has its own private black keys, which cannot be decrypted by the user of a different security domain identifier. When SEC is instructed to use a black key as an encryption key, SEC automatically decrypts the black key and places it directly into a key register before using the decrypted value in the user-specified cryptographic operation.

12.4.1 Black key encapsulation schemes

SEC supports two different black key encapsulation schemes, one intended for quick decryption, and another intended for high assurance.

- The quick decryption scheme uses AES-ECB encryption.
- The high-assurance black key scheme uses AES-CCM encryption. The AES-CCM mode is not as fast as AES-ECB mode, but AES-CCM includes an "MAC tag" (integrity check value) that ensures the integrity of the encapsulated key. SEC does not mix the length of the encrypted key into the value of the key encryption key when using the high assurance black key scheme, because the MAC-tag prevents misrepresenting the length of the encrypted key. In AES-CCM encryption the AES algorithm is always used in the "encryption" direction regardless of whether the key is being encrypted or decrypted, so in the high-assurance black key scheme encapsulation and decapsulation require approximately the same amount of time.

12.4.2 Differences between black and red keys

Differences between black keys and red keys include the following:

- Black keys are encrypted, while red keys are un-encrypted.
- A black key is usually longer than the red key that is encapsulated. ECB encrypted data is a multiple of 16 bytes long, because ECB is a block cipher with a block length of 16 bytes. So if the red key that is to be encapsulated in an ECB-black key is not already a multiple of 16 bytes long, it is padded with zeros to make it a multiple of 16 bytes long before it is encrypted, and the resulting black key is this length.
- A CCM-encrypted black key is always at least 12 bytes longer than the encapsulated red key, because the encapsulation uses a 6-byte nonce and adds a 6-byte ICV. If the key is not already a multiple of 8 bytes long, it is padded as necessary so that it is a multiple of 8 bytes long. The nonce and ICV add another 12 bytes to the length.

12.4.3 Loading red keys

Red keys can be loaded into Key Registers using either a LOAD command or a KEY command with ENC = 0. But keys cannot be stored from Key Registers back to memory in red form. The only way to store keys back out to memory is in black form. This is accomplished by using the FIFO STORE command with an appropriate OUTPUT DATA TYPE value (see [Table 7-30](#), values 10h-27h).

12.4.4 Loading black keys

The only way that black keys can be successfully loaded is by using a KEY command with ENC = 1 and the proper setting of the EKT bit. The EKT bit in the KEY command indicates which encryption algorithm (AES-ECB or AES-CCM) should be used to decrypt the key. An ECB-encrypted black key can be successfully loaded only with EKT = 0 (ECB mode), and a CCM-encrypted black key can be successfully loaded only with EKT = 1 (CCM mode).

12.4.5 Avoiding errors when loading red and black keys

There are many ways to load red and black keys. Some of these ways will be successful, and other ways will not be successful. In many cases an unsuccessful key load will generate an error message, but there are a few cases that SEC cannot detect, so these do not produce error messages. In those cases special care should be taken.

Key Type	Error	Error Indication	Comments
Black Key	Black Key loaded as Red Key (Black Key loaded into a key register using a LOAD command or a KEY command with ENC = 0)	The LOAD or KEY operation will complete normally, but since the key register will contain a Black Key rather than a Red Key, the key value will be incorrect. If the key is then used in an encryption or decryption operation, the operation will complete normally, but incorrect results will be produced.	Since all possible values are permissible in a Red Key, SEC cannot distinguish between a Black Key and a Red Key. Software should take care to distinguish Red Keys from Black Keys either by location or via software tags.
Red Key or ECB-encrypted Black Key	non-ECB-Encrypted Key loaded as an ECB-Encrypted Black Key (Key loaded into a key register using a a KEY command with ENC = 1 and EKT = 0)	The KEY command will decrypt the key as if it were an ECB-encrypted Black Key. The Key command will complete normally, but since the key register will contain a Black Key rather than a Red Key, the key value will be incorrect. if the key is then used in an encryption or decryption operation, the operation will complete normally, but incorrect results will be produced.	Since all possible values are permissible in an ECB-encrypted Black Key, SEC cannot distinguish between a CCM-encrypted Black Key or a Red Key and an ECB-encrypted Black Key. Software should take care to distinguish Red Keys and CCM-encrypted Black Keys from ECB-encrypted Black Keys either by location or via software tags.
CCM-encrypted Black Key	CCM-encrypted Black Key loaded as CCM-encrypted Black Key, but key value is corrupted (Key loaded into a key register using a a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	The MAC check failed when the CCM-encrypted Black Key was decrypted.
Red Key or ECB-encrypted Black Key	Non-CCM-encrypted Black Key loaded as CCM-encrypted Black Key (Key loaded into a key register using a a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	When SEC tried to verify the MAC tag, the check failed because the key did not have a MAC tag.
CCM-encrypted Black Key	CCM-encrypted Black Key loaded by the wrong security domain (Key loaded into a key register using a a KEY command with ENC = 1 and EKT = 1)	MAC Check Error Job termination status word: Source = 2h (CCB); ERRID = Ah (ICV check failed)	When SEC tried to verify the MAC tag, the check failed because the wrong SDID value was used in deriving the key encryption key.
Any Key	Specified size too large for key register Value in KEY command's LENGTH field is larger than the Class 1 Key Register or Class 2 Key Register (depending upon the destination specified in the Key command).	Key Size Error Job termination status word: Source = 2h (CCB); ERRID = 3h (Key size error)	LENGTH is less than 16 when reading a key from a Secure Memory key partitionSince all possible values are permissible in an ECB-encrypted Black Key, SEC cannot distinguish between a CCM-encrypted Black Key or a Red Key and an ECB-encrypted Black Key. Software should take care to distinguish Red Keys and CCM-encrypted Black Keys from ECB-encrypted Black Keys either by location or via software tags.

12.4.6 Encapsulating and decapsulating black keys

SEC's key-protection policy imposes restrictions on creating black keys and converting between black key types. When loading a red or black key into a Key Register, it is possible to prohibit the key from being written back out to memory at all. Executing a KEY command with $NWB = 1$ prohibits writing the key out, whereas $NWB = 0$ permits the key to be stored to memory as a black key. If a red key is loaded into a key register, it can be stored as either an ECB or CCM-encrypted black key (assuming $NWB = 0$). But if a black key is loaded into a key register, it can be stored out only as the same type of black key as was loaded, as shown in this figure.

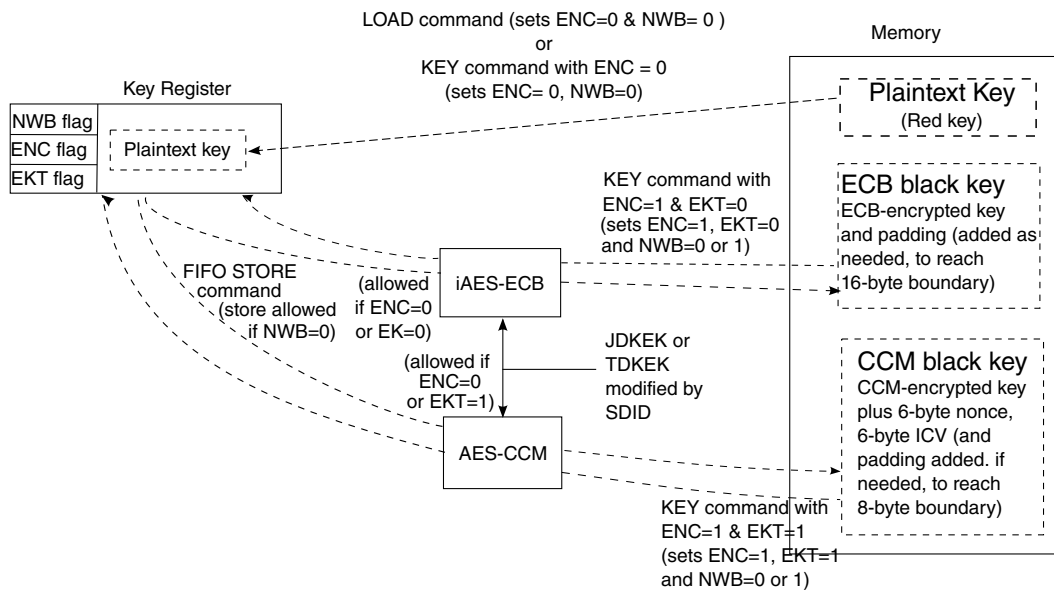


Figure 12-1. Encapsulating and decapsulating SEC black keys

The cryptographic key used to encrypt or decrypt black keys is held in the 256-bit JDKEKR or TDKEKR, and this key's value is modified via the appropriate TZ/SDID value before being used in a black key operation on behalf of some descriptor. The TZ/SDID value is taken from the job ring or Queue Manager Interface SDID register, depending upon where the descriptor is executed. Job descriptors or their shared descriptors always use the JDKEKR key, but trusted descriptors, or shared descriptors referenced by trusted descriptors, can use either the JDKEKR key or the TDKEKR key. Use of the TDKEKR allows trusted descriptors to encapsulate keys that cannot be decrypted by job descriptors.

The black keys used by each SDID value are encrypted using a different modification of the JDKEK or TDKEK in order to provide cryptographic separation between the keys used in different security domains. Note that TrustZone trusted descriptors always use TZ

= 1b, SDID = 000h for the JDKEK/TDKEK modification, regardless of the Job Ring in which the descriptor is executed. All QI queues use the same SDID value, so all queues can share black keys.

Because black keys are not intended for storage of keys across chip power cycles (SEC's blob mechanism (section [Blobs](#)) is intended for this purpose), the values in the JDKEKR and TDKEKR are not preserved at chip power-down. Instead, new 256-bit secret values are loaded into the JDKEKR and TDKEKR from the RNG following power-on for use during the current power-on session. That means that a black key created during one power-on session cannot be decrypted on subsequent power-on sessions.

12.4.7 Types of black keys and their use

The four types of black keys that SEC's black-key mechanism implements are listed in this table.

Table 12-2. Black key types

Black key type	Key used	Encryption mode
JDKEK-ECB	JDKEKR	AES-ECB
TDKEK-ECB	TDKEKR	
JDKEK-CCM	JDKEKR	AES-CCM
TDKEK-CCM	TDKEKR	

Note that it is possible to inadvertently load a black key as the wrong type, resulting in an incorrect key value in the key register. No error message is generated when any of the black key types listed in this table are loaded in ECB mode. But an ICV check failure error message is generated if the wrong black key type (or a red key) is loaded in CCM mode.

It is possible to load a JDKEK-encrypted black key and save it out as a TDKEK-encrypted black key, or vice versa. This is permitted because only trusted descriptors have access to TDKEK encryption, and they are trusted to operate only in a secure manner. Such conversions might be used during a key provisioning procedure. (But as noted earlier, conversion between ECB-black keys and CCM-black keys is not permitted.)

12.4.8 Types of blobs for key storage

As described in [Blobs](#), SEC implements different types of blobs that are intended for storage of keys across power cycles. Because encapsulation or decapsulation of blobs takes longer than encapsulation and decapsulation of black keys, if a long-term key is stored in a blob and must be used multiple times during a power-on session, for performance reasons it is preferable to decapsulate the blob at power-up and re-encapsulate the key as a black key.

SEC implements operations that convert between blob encapsulation and black-key encapsulation without exposing the key in plaintext. There are several different blob types dedicated to key storage that correspond to the different types of black keys. A specific type of black key converts into a specific type of black-key blob. If this were not enforced by SEC, a hacker could attempt to convert one black key type to another black key type by first exporting the black key as a black key blob, and then re-importing the blob as if it were a different type of black key blob. But because each blob type uses a different derivation for the blob key encryption key, such an attempt at misrepresenting the blob type fails with a MAC-tag error when the blob is decapsulated.

12.5 Trusted descriptors

Trusted descriptors provide a means for trustworthy software to create trusted "applets" that can be safely executed by less trustworthy software.

12.5.1 Why trusted descriptors are needed

Software utilizes the cryptographic features of SEC by building a descriptor, and then adding this descriptor to a Job Ring. Usually the same software entity performs both operations, that is, building the descriptor and adding it to a Job Ring. But there are cases in which different software entities perform the two operations. One important case is when the descriptor builder is more trustworthy than the Job Ring owner. For example, the boot software or TrustZone SecureWorld software might be trusted to properly handle particularly sensitive data, such as digital-rights management keys, but the content-rendering software that needs to use those keys may not be as trustworthy.

SEC implements a trusted descriptor mechanism to be used in these cases. These trusted descriptors are granted special privileges that ordinary job descriptors are not, and to ensure that these special privileges are not abused by tampering with the trusted descriptor, SEC ensures the integrity of the trusted descriptor with a cryptographic signature.

12.5.2 Trusted-descriptor key types and uses

When SEC is in trusted mode or secure mode, the hardware ([Special-purpose cryptographic keys](#)) allows SEC to use the trusted-descriptor key encryption key and the trusted-descriptor signing key. These keys are available only to SEC and cannot be read or written. (For testing purposes, these registers, but not the trusted mode or secure mode values of these keys, can be read and written in non-secure mode.) Furthermore, these keys can be used only for key encryption/decryption or signing/signature verification; users cannot use them for anything else. In addition, these keys are changed every boot cycle so that any keys encrypted with the trusted-descriptor key encryption key are lost when the system is rebooted. Likewise, following a reboot, any trusted descriptors signed (HMAC'd) during the previous power-on cycle fail the integrity check and do not execute.

12.5.3 Trusted descriptors encrypting/decrypting black keys

SEC implements both trusted and normal (non-trusted) black keys, which are encrypted with different key-encryption keys. Both trusted and normal descriptors are allowed to encrypt or decrypt normal black keys, but only trusted descriptors are allowed to encrypt or decrypt trusted black keys. Note that if any black keys are included as immediate data within the trusted descriptor, it is the encrypted version of the key that is verified when computing the signature. When executing the trusted descriptor, the black key is not decrypted unless the signature is valid.

Trusted software can decapsulate master secrets from trusted-descriptor blobs and can use these master secrets to derive keys that it embeds as trusted black keys within trusted descriptors. Untrusted software can then cause SEC to execute these trusted descriptors to encrypt or decrypt data, without the master secrets or derived keys ever being directly accessible to the untrusted software.

In addition, trusted descriptors can be written to ensure that these keys cannot be misused. This mechanism would be useful in certain IKE key exchange processes, or for supporting trusted-computing group, trusted-platform module operations, or various data rights-management standards.

See [Black keys](#) for more information.

12.5.4 Trusted-descriptor blob types and uses

SEC implements both trusted-descriptor blobs and normal (non-trusted descriptor) blobs, which use different key derivations for the blob-key encryption keys. Both trusted and normal descriptors are allowed to encapsulate or decapsulate normal blobs, but only trusted descriptors are allowed to encapsulate or decapsulate trusted blobs. When executing the trusted descriptor, the blob is not decapsulated unless the integrity check is valid.

See [Blobs](#) for more information.

12.5.5 Configuring the system to create trusted descriptors properly

NOTE

Trusted descriptors use the descriptor commands defined in [Using descriptor commands](#). The SIGNATURE command ([SIGNATURE command](#)) is used only by trusted descriptors.

Although trusted descriptors cannot be forged or altered in unauthorized ways after they are generated and signed, to be truly considered "trusted," the system must be configured so that trusted descriptors can be created only by trusted software. Trusted descriptors can be created only via a Job Ring that has the Allow Make Trusted Descriptor (AMTD) bit set in the Job Ring's JRaICID register. Proper configuration is required to ensure that only trusted software can write to any JRaICID register (because this would allow the AMTD bit to be set). This can be ensured in any of the following ways:

- The register is written and then locked (via its LAMTD bit) by trusted boot software.
- The system uses the operating system or hypervisor to control access to the address block that includes the JRaICID registers.

Proper configuration for the use of trusted descriptors must also ensure control of access to the trusted-descriptor-creation Job Rings, that is, those Job Rings whose JRaICID registers have been configured with the AMTD bit set. The operating system or hypervisor can provide access control by granting certain processes access to the register address block containing a particular Job Ring's control registers, and denying access to that block to other processes.

12.5.6 Creating trusted descriptors

To create a trusted descriptor, trustworthy software builds a candidate trusted descriptor that uses the extra privileges properly. For example, the trusted descriptor might utilize cryptographic keys that an ordinary job descriptor cannot access, but the trusted descriptor would be designed so that the key values cannot be exposed.

The candidate trusted-descriptor is converted to a trusted descriptor by executing the candidate trusted-descriptor in a specially-privileged Job Ring. This causes SEC to cryptographically sign the descriptor. The trusted descriptor can later be executed by less trustworthy software. When the trusted descriptor is executed, SEC executes the commands within the trusted descriptor only if the signature is correct. This ensures that the trusted descriptor has not been tampered with after it was created.

12.5.6.1 Trusted descriptors and descriptor-header bits

Descriptor headers contain a 2-bit field related to trusted descriptors. See [HEADER command](#) for a full explanation of the descriptor header.

The 00 value in the TDES field indicates an ordinary job descriptor. The 11 value indicates a candidate trusted descriptor, that is, a descriptor that SEC should convert into a trusted descriptor by affixing a signature. When SEC executes a candidate trusted descriptor, it checks to see if the AMTD (allow make trusted descriptor) bit is set in the Job Ring's JRaCID register. If not, the candidate trusted descriptor is not converted to a trusted descriptor and the job terminates with an error. If AMTD=1, SEC changes the TDES field value to 10 if the candidate trusted descriptor is being created in a Job Ring owned by TrustZone nonSecureWorld, but changes the TDES field to 01 if the Job Ring is owned by TrustZone SecureWorld. SEC then either affixes a signature to the new trusted descriptor, or executes the trusted descriptor, or both, depending upon the option in the SIGNATURE command at the end of the descriptor.

12.5.6.2 Trusted-descriptor execution considerations

Important rules of use and things to consider when executing trusted descriptors are as follows:

- When a trusted descriptor is executed, SEC first checks the signature (HMAC) to verify that the trusted descriptor has not been modified. If the trusted descriptor references a shared descriptor, it is included in the computation of the signature. If the signature is valid, the trusted descriptor is executed. If the signature is invalid, the job is aborted with an error indication.

- A TrustZone non-SecureWorld trusted descriptor can be executed only within a Job Ring that has the same SDID value as the Job Ring in which the trusted descriptor was created. The reason for this restriction is that Job Rings may be owned by different security domains that do not trust each other's TrustZone non-SecureWorld trusted descriptors. This restriction is enforced by including the SDID of the Job Ring's JRaICID register in the signature computation, both when creating the trusted descriptor and before executing the trusted descriptor.
- A TrustZone SecureWorld trusted descriptor can be executed within any Job Ring, regardless of that Job Ring's SDID or TZ value. TrustZone SecureWorld trusted descriptors can be created only in a Job Ring owned by TrustZone SecureWorld. This allows TrustZone SecureWorld to create special trusted descriptors that are trusted by all security domains. The TDES field in the trusted descriptor's HEADER command is used to distinguish SecureWorld trusted descriptors from non-SecureWorld trusted descriptors.
- If a trusted descriptor contains a jump to another descriptor, it must also be trusted. Jumping from a trusted descriptor to a job descriptor results in an error and processing stops. Because all CHAs, all MODEs, and the Class 2 Key and Key Size Registers are reset before a trusted descriptor's signature is checked, care must be taken when transferring to a trusted descriptor from another descriptor (whether trusted or not) via Non-Local JUMP (see Section [JUMP \(HALT\) command](#)), In-Line Descriptor (see INL field in [Table 7-96](#)), or Replacement Job Descriptor (see RJD field in [Table 7-96](#)).
- Note that although address pointers within a trusted descriptor are protected against modification, any data referenced by an address pointer is not protected against modification. Therefore, keys and other information that must be protected against modification should be contained as immediate data within the trusted descriptor. When a trusted descriptor executes, it is permitted to modify itself just like a non-trusted descriptor can. This ability can be useful if the trusted descriptor is maintaining an integrity-protected value that changes, such as a usage count, sequence number, and so on. Because modifying the trusted descriptor renders the signature invalid, the signature must be recomputed after the modification. This can be accomplished by placing a SIGNATURE Command at the end of the trusted descriptor. This directs SEC to recompute the trusted descriptor's signature.

12.6 Blobs

SEC can protect data in a cryptographic data structure called a blob, which provides both confidentiality and integrity protection.

12.6.1 Blob protocol

SEC's built-in blob protocol provides a method for protecting user-defined data across system power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Each time that the blob protocol is used to protect data, a different randomly generated key is used to encrypt the data. This random key is itself encrypted using a key encryption key and the resulting encrypted key is then stored along with the encrypted data. The key-encryption key is derived from the chip's master secret key so the key-encryption key can be recreated when the chip powers up again. The combination of encrypted key and encrypted data is called a blob.

[Table 7-58](#) shows the format of the PROTINFO field for the blob protocol, and [Table 7-59](#) describes the bit values.

12.6.2 Why blobs are needed

To retain data across power cycles, the data must be stored in non-volatile memory. But data stored in this manner is potentially vulnerable to disclosure or modification when the SoC's software and hardware security-mechanisms are not functioning, for example, during debug operations. SEC is able to protect data for long term storage by encrypting that data using a secure non-volatile key. ¹ Using a unique non-volatile key for each device prevents data encrypted on one device from being copied and decrypted on a different device, which might compromise the secrecy of the data.

12.6.3 Blob conformance considerations

Generation of private blobs is not considered in any governmental security specification. However, there are several steps in the process that can be viewed as having approved methods. These methods were chosen to conform to the following specifications, (except where noted).

- FIPS PUB 197, *Advanced Encryption Standard (AES)*, November 26, 2001.
- FIPS PUB 180-2, *SECURE HASH STANDARD*, August 1, 2002.
- SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012. Draft SP800-90B, *Recommendation of the Entropy Sources Used for Random Bit Generation*, August 2012.

1. The data is actually encrypted with a randomly generated blob key, and it is that blob key that is encrypted using the secure non-volatile key

- SP800-38c, *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, May 2004.
- SP800-56A, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, March, 2007.
- SP800-57, *Recommendation for Key Management - Part 1: General*, March, 2007.

In the context of SEC, a blob is encrypted data that is bound to a specific device by virtue of using a secret non-volatile, device-specific master key. This master key is used only for the purpose of creating and extracting blob data, and the value of this key cannot itself be extracted from a device. To protect data requiring high-security strength, blob creation is performed in hardware using 256-bit security strength. AES-256 is used as the encryption algorithm. SHA-256 is used for key derivation (SP800-57 specifies that SHA-256 has 256-bit security strength when used in key derivation).

The random number generator is specified in SP800-90A, using the Hash_DRBG with SHA-256 as the hash function. It gets entropy from a live entropy source intended to comply with SP800-90B. The random number generator has a security strength of 256 bits.

SEC blobs provide both confidentiality and integrity protection for the encapsulated data. Because a blob protects both confidentiality and integrity, it may be stored in external long-term storage such as flash. Counter with cipher block chaining-message authentication code (AES-CCM) is used as the bulk encryption algorithm. Note that the MAC associated with a blob provides integrity protection not only for the encrypted data the blob contains, but also for all intermediate keys used in the creation of a blob.

There may be many different blobs existing at the same time, used for many different purposes, and subject to different security policies. To guarantee that blobs are not inadvertently or intentionally swapped, SEC encrypts different blobs with different keys. Two mechanisms are used to guarantee that a single key is not used to encrypt unrelated data and to ensure that each key is used to encrypt as little data as possible. One of these mechanisms is random-key generation. Each time that a blob is created, SEC generates a different, random 256-bit key using SEC's internal hardware random-number generator (RNG). This blob key is used to encrypt the blob data using AES-CCM, which provides both confidentiality and integrity protection. The second mechanism is key derivation, using a device-unique, non-volatile master key as the key-derivation key. The (volatile) random blob key is encrypted with the non-volatile key derived from the master key, and then stored with the blob so that the blob data can be decrypted during subsequent power-on cycles. Different types of blobs are encrypted using different keys derived from the master key. The derived keys are further differentiated by a key modifier supplied by software, which can be used to guarantee that one blob cannot be inadvertently or

maliciously substituted for another blob. Software can use these key modifiers to differentiate specific data, or to prevent replay attacks (the replacement of the current blob with an out-of-date version of the blob).

The master key is used in a key derivation function (KDF) similar to that specified in SP800-56 (sec. 5.8.1). That function includes two parties U and V, who both add information for use in deriving their shared key. Here the derived key is used for storage, and so there is only a single involved party, and hence only one block of public and private information. In the current key-derivation function, only a single iteration is required, because the size of the derived key is the same as the hash function used. Therefore, the counter is not used. The master key is concatenated with a key modifier (which may be a public or private nonce), an AlgorithmID (the blob type) and a security state indicator (that is, non-secure, secure or trusted). This message is then hashed with SHA-256, and the output is used as a blob-key encryption key.

AES-CCM mode uses a nonce and initial counter value as inputs, along with the key and data. SP800-38c requires that the nonce and counter values be unique across all invocations of AES-CCM under a given key. This requirement is met by virtue of a random key being generated for each blob. Because a key is never used more than once, there are no requirements on the nonce and initial counter value. Therefore, both nonce and initial counter value are fully specified, and the same values are used for all blobs. Blob creation uses the formatting function specified in SP800-38c, Appendix A.

The entire 16-byte MAC is stored along with the encrypted data, to provide a strong assurance of integrity. Note that due to the design of the blobs, the MAC provides integrity protection for the data, blob key and blob-key encryption key.

12.6.4 Encapsulating and decapsulating blobs

When encapsulating a blob, SEC:

1. Obtains a random blob key (BK) value from the RNG
2. Encrypts the data with that BK
3. Derives a blob-key encryption key (BKEK) from the master key
4. Encrypts the BK using that BKEK

When decapsulating a blob, SEC:

1. Derives a BKEK from the master key
2. Decrypts the BK using that BKEK
3. Decrypts the data with the BK

12.6.5 Blob types

SEC supports different types of blobs, and a coded value of the blob type is used as an input to the key-derivation function. This prevents a blob that was exported as one type from being imported as another type because it would decrypt improperly and so would fail the MAC tag check. This table lists the types of blobs that SEC supports. Note that the type categories are orthogonal, that is, a blob has one type from each type category. For instance, one blob may be a (normal format/black key/secure state blob), while another blob may be a (test format/general data/trusted state) blob. In addition, black key blobs are differentiated by encryption mode and encryption key, so one black key blob may be a (AES-ECB/TDKEK) type and another black key blob may be a (AES-CCM/JDKEK) type.

Table 12-3. Blob types

Type Category	Type	Cross-reference
Formats	Normal format	Blob types differentiated by format
	Test format	
	Master key verification format	
Contents	General data (that is, red blobs)	Blob types differentiated by content
	Black keys (that is, black blobs) <ul style="list-style-type: none"> • Encryption modes: AES-ECB and AES-CCM • Encryption keys: JDKEK and TDKEK 	
Security states	Trusted state	Blob types differentiated by security state
	Secure state	
	Non-secure state	

12.6.5.1 Blob types differentiated by format

SEC supports three different formats for blobs, usable for all blob content types, and all blob security state types. This figure describes the blob formats and how they work.

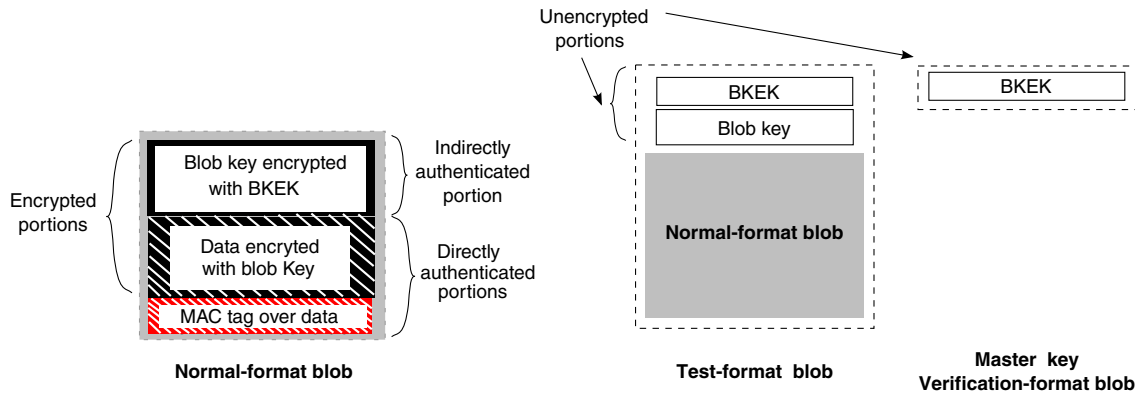


Figure 12-2. Formats of SEC blobs

- A normal-format blob consists of the encrypted blob key, the encrypted data, and a message authentication code (MAC) tag, as shown on the left side of the figure. A randomly-generated, 256-bit blob key is used to encrypt the data using the AES-CCM cryptographic algorithm. AES-CCM encrypts the data and also yields a MAC tag that is used to protect the data's integrity. The blob key itself is encrypted in AES-ECB mode using a 256-bit blob-key encryption key (BKEK). Checking the MAC directly authenticates the data encapsulated in the blob. The blob key is indirectly authenticated because substitution or corruption of the encrypted blob key yields an incorrect plaintext blob key, which causes the blob content to be decrypted incorrectly, which is detected by the MAC check. Because a normal-format blob is used to protect actual data, the blob-key encryption key (BKEK) that is used to encrypt the blob key for a normal format blob is secret, by virtue of having been derived from the secret master key.
- As shown in the middle of the figure, a test-format blob consists of a normal-format blob, with the unencrypted BKEK and unencrypted blob key prepended. Because the purpose of a test-format blob is to facilitate testing blob encapsulation and decapsulation, the BKEK for a test-format blob is derived from a known test key. SEC permits test-format blobs to be encapsulated or decapsulated only when SEC is in non-secure mode.
- As shown on the right side of the figure, a master key verification format blob consists of only the unencrypted BKEK. Because the purpose of a master key verification format blob is to verify that the master key has been properly programmed, the BKEK for a master key verification format blob is derived from the secret master key. In order to ensure the secrecy of BKEKs used for normal format blobs, the derivation is different from the derivation used for normal format blobs. This ensures that the BKEKs used to protect data cannot be exposed by examining the BKEK values in master key verification format blobs.

12.6.5.2 Blob types differentiated by content

One of the blob content types is intended for general data (see [Red blobs \(for general data\)](#)), and four content types are intended for cryptographic keys (see [Black blobs \(for cryptographic keys\)](#)).

12.6.5.2.1 Red blobs (for general data)

Unencrypted data that should be protected is sometimes referred to as "red data", so the type of blob intended for general data (which is left unencrypted when the blob is decapsulated) is called a red blob. When SEC is instructed to encapsulate data as a red blob, it assumes that the data to be encapsulated is unencrypted and it proceeds to encrypt the data with the blob key. Likewise, when SEC is instructed to decapsulate a red blob, it assumes that the data that is decapsulated is to be left in memory unencrypted. Other mechanisms, such as an operating system or hypervisor acting in conjunction with a memory management unit, may be used to protect the data before it is encapsulated into a blob and after it is decapsulated from a blob.

12.6.5.2.2 Black blobs (for cryptographic keys)

SEC's black blob mechanism is a means for translating between black key encapsulation and blob encapsulation without exposing the key during the translation process. A black blob is simply a blob whose input during blob encapsulation is assumed to be a black key, and whose output during blob decapsulation is either a black key that is written into memory, or an unencrypted key that is placed directly into a Key Register.

SEC supports the protection of cryptographic session keys by encrypting these keys in a "black key" encapsulation format when storing them in memory via a FIFO STORE command, and then decapsulating them "on-the-fly" as they are referenced by a job descriptor with a descriptor KEY command. Black key encapsulation or decapsulation is very quick, but black keys are intended only for protection during the current SoC power-on session. Black keys encapsulated during one chip power-on session cannot be decapsulated on subsequent power-on sessions because the key encryption key (JDKEK or TDKEK) is erased during power-down and is replaced by a new randomly-generated key encryption key at power-up. To protect a key so that it can be recovered on subsequent power cycles, the key must be encapsulated as a blob. A key could be encapsulated as a red blob, but this would require exposing the key in memory in unencrypted form. To avoid exposing keys in unencrypted form, SEC supports the concept of black blobs. (Data that is not sensitive to disclosure, either because it is inherently nonsensitive or because it always remains encrypted, is sometimes referred to as "black data".)

12.6.5.2.3 Enforcing blob content type

When SEC is instructed to encapsulate a black blob, it first decapsulates the black key that was specified as input and then encapsulates the resulting key as a Black blob. The black blob itself is exactly the same as a red blob, except that the BKEK derivation is different from red blobs. This prevents a black blob from being decapsulated as a red blob, which would leave the key exposed in memory. Because black keys can be encrypted under either the JDKEK or the TDKEK, and can be encrypted in either AES-ECB mode or AES-CCM mode, SEC first decrypts the black key data with the appropriate KEK using the appropriate mode and then re-encrypts the key data with the BK using AES-CCM. During this process the key that is temporarily unencrypted is safely protected within SEC's hardware storage. To prevent mixing up the different types of black blobs (JDKEK vs. TDKEK and ECB vs. CCM), the BKEK for each type is derived differently.

12.6.5.3 Blob types differentiated by security state

SEC also supports different types of blobs for use in different security states. All of the blob-format types and blob-content types are available in each of the following different security states:

- Trusted state
- Secure state
- Non-secure state

However, the BKEKs for the blobs are derived differently for each of these states. Therefore, a blob encapsulated while operating in a particular state cannot be decapsulated while SEC is operating in another of these states:

- During trusted and secure states, the BKEK is derived from the secret master key (but using different key derivation functions in the two states).
- While SEC is operating in non-secure state, the BKEK is derived from the known test key. This latter type of blob is intended to facilitate testing using known-answer tests.

12.6.6 Blob encapsulation

A data blob is encrypted using a blob key (BK), which is a random number used as an AES-CCM key. The NIST AES-CCM specification states that for any key, all invocations must use distinct nonces and counter blocks. Although SEC uses the same nonce and initial counter block values for all data blobs, SEC satisfies the AES-CCM

requirement because each encryption operation uses a different key (that is, a random number generated by the RNG). The nonce is given as all zeros, and so the initial block $B_0 = 3B00_0000_0000_0000_0000_0000_xxxxh$, where $xxxx$ is the number of bytes of plaintext (maximum length is 65535 bytes), while the initial counter value $Ctrl_0 = 0300_0000_0000_0000_0000_0000_0000_0000h$. These values are automatically generated during the encapsulation operation.

Figure 12-3 shows the entire blob-encryption operation. B_0 is generated internally and stored in the Class 1 Context DWords 0 and 1, while $Ctrl_0$, also generated internally, is stored in Class 1 Context DWords 2 and 3 (see Table 11-100). The random BK value is stored in the Class 1 Key Register, and the operation mode is set to AES-CCM.

At the blob pointer, the first 32 bytes contain the key blob, which is the encrypted value of the random blob key. Output ciphertext data (data blob) is stored at the blob pointer + 32. The generated message-authentication code (MAC, the signature over the data blob) is stored in the final 16 bytes of the blob.

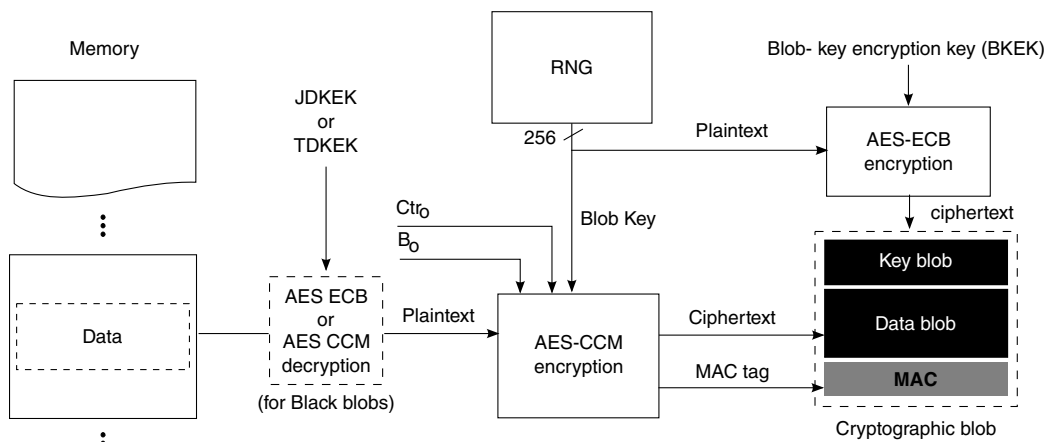


Figure 12-3. Encapsulating a blob

12.6.7 Blob decapsulation

Before decrypting a data blob, the associated key blob must be decrypted to obtain the blob key. The key blob resides at the blob pointer. AES-ECB mode is used to decrypt the key blob using the BKEK. Generation of the BKEK for blobs is described below.

$Ctrl_0$ and B_0 are generated internally, and are stored in the Class 1 Context 1 and Context 2 registers, respectively (see Figure 12-4). AES-CCM mode is used to decrypt the data blob (starting at the blob pointer + 32), using the decrypted blob key.

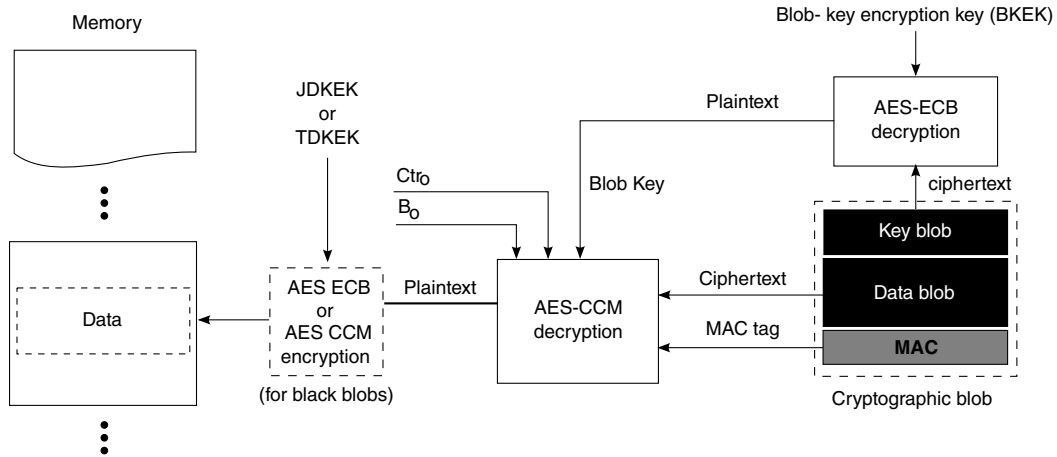


Figure 12-4. Decapsulating a blob

12.7 Critical security parameters

SEC contains several encryption and authentication keys that are identified as being critical security parameters (CSPs), as defined in FIPS140-2. Each of these CSPs are zeroized (cleared) upon the SecMon state machine entering the FAIL state. This FAIL state indicator is an input to SEC and can be observed via the SEC Status Register.

Upon receiving an indication that the security state machine has entered the FAIL state, all register-based CSPs are zeroized via the asynchronous hardware reset. SEC can be restarted after the chip has transitioned from FAIL to non-secure state; however, all critical security parameters are lost forever.

This table lists the critical security parameters included in SEC.

Table 12-4. Critical security parameters

CSP	Notes	Related cross-reference
Zeroizable master key	Inside of security power island; loaded and locked once at provisioning time	—
CCB Class 1 Key Register	—	See register appendix
CCB Class 2 Key Register	—	See register appendix
PKHA E register	Exponent	See register appendix
Trusted descriptor signing key	Loaded at boot time from RNG	Trusted descriptors
Trusted descriptor key encryption key		
Job descriptor key encryption key	Loaded at boot time from RNG	Keys available in different security modes

Table continues on the next page...

Table 12-4. Critical security parameters (continued)

CSP	Notes	Related cross-reference
SNOW f8 internal state	—	SNOW 3G f8 accelerator functionality
SNOW f9 internal state	—	SNOW 3G f9 accelerator functionality
ZUCE internal state	—	ZUC encryption accelerator (ZUCE) functionality
ZUCA internal state	—	ZUC authentication accelerator (ZUCA) functionality
Crypto-engine internal datapath registers	—	See register appendix
Output data FIFO	—	—

12.8 Manufacturing-protection chip-authentication process

The manufacturing-protection authentication process is used to authenticate the chip to the OEM's server. This authentication process can ensure that the chip:

- Is a genuine NXP part
- Is the correct part type
- Has been properly configured by means of fuses
- Is running authenticated OEM software
- Is currently in the secure or trusted mode

These are inputs to a key derivation function, to create a private ECC key that is available only to the crypto hardware. The public ECC key can be generated and used to later authenticate the chip and verify the security status of the chip. These properties are verified by digitally signing a message using this private ECC key. The message may be verified by a server using the public ECC key. Because only a genuine NXP part, configured correctly, and running in the proper security state can correctly sign the message, assurance of all of this is provided by the verification of the message signature. The message cannot be spoofed by untrustworthy software, because the private-key generation, public-key generation and signature functions are all implemented in hardware, and the chip-specific data is supplied by secure-boot firmware. After the signature over the message has been verified, the server can be assured that it is safe to download proprietary data to the chip over a secured connection.

The authentication process takes place in three stages, implemented via three functions built into the SEC hardware.

Table 12-5. Manufacturing-protection chip-authentication functions

Function name	Abbreviation	Authentication steps implemented by function	Cross-reference
Manufacturing-protection private-key generation function	MPPrivK	<ul style="list-style-type: none"> Takes input data to be authenticated and hashes that data with a secret value embedded in the silicon. The result is an ECDSA private key that is securely stored in the MPPKR. 	MPPrivK-generation function
Manufacturing-protection public-key generation function	MPPubK	<ul style="list-style-type: none"> Generates an ECDSA public key that matches the private key in MPPKR and outputs that public key.¹ 	MPPubK-generation function
Manufacturing-protection sign function	MPSign	<ul style="list-style-type: none"> Takes the value in the MPMsg register and concatenates any additional data supplied as ordinary input to the MPSign function Signs the concatenated message data using the private key that was stored in the MPPKR by the MPPrivK Generation function Outputs the signed message, along with the message representative. Software running on the chip sends this signed message to the OEM's server, which then verifies the signature by means of the public key output earlier by the MPPubK Generation function. 	MPSign function

1. The MPPubK-generation function is run once on a single chip at the OEM's facility, and the OEM's server retains a copy of this public key to be used later in the authentication process.

12.8.1 Providing data to the manufacturing-protection authentication process

The purpose of the manufacturing-protection authentication process is to authenticate certain information, such as the chip's part number, serial number, and the super root key hash, by signing it with a private key that can be used only in a legitimate NXP chip of the correct type running in the secure or trusted states.

The following sections describe how data is input to the manufacturing-protection process.

12.8.1.1 Providing data to the MPPrivK-generation function

The MPPrivK-generation function is expected to be run only by the secure boot firmware.

12.8.1.2 Providing data to the MPPubK-generation function

The only inputs to the MPPubK-generation function are the manufacturing protection private key from the MPPKR, and the elliptic curve selection from the CSEL field in the PDB. The hardware guarantees the correctness of both of these inputs because the MPPKR is accessible only to hardware, and the value in the CSEL field must match the value used in the MPPrivK-generation function.

12.8.1.3 Providing data to the MPSign function

To provide data to the MPSign function, the secure boot firmware writes some or all of the data into the MPMR, then locks it by setting the MPMRL bit in the Security Configuration Register. Additional data can be provided as ordinary message input to the MPSign function. This additional data will be appended to the content of the MPMR before the data is hashed and signed. All this data is authenticated as having originated from a legitimate NXP chip of a specific type, because the data is signed with the manufacturing-protection private key when the MPSign function is invoked. But only the portion of the data that was written into the MPMR is guaranteed to have originated from trusted firmware. Since the MPSign function is intended to be invoked by software that has not yet been authenticated, the extra data supplied as ordinary message input to the MPSign function should be treated skeptically until the authentication process is complete.

12.8.1.4 Role of the ROM-resident secure boot firmware

Because the ROM-resident secure boot firmware is the only software that is known to be trusted prior to authentication, it plays a crucial role in the manufacturing-protection authentication process. The ROM-resident boot firmware reads fuse-resident data that needs to be authenticated and either supplies some or all of it as data to the MPPrivK

generation function or writes some or all of the data to the MPMR. Note that all of the data needed to authenticate the software image that will be booted must be supplied by the ROM-resident firmware using either the MPPrivK generation function or the MPMR.

The MPSign function is intended to be invoked by untrusted software that has just booted, which is why the data to be authenticated via the MPSign function must be supplied in advance by trusted ROM-resident secure boot software and then securely conveyed to the MPSign function via the MPMR. After the operating system has booted and is able to run a network-protocol stack, application software establishes a communication session with the OEM's server. The application software then runs a descriptor that invokes the MPSign protocol, which uses the ECDSA private key stored in MPPKR to sign a message composed of the content of MPMR followed by other optional data. Note that this additional optional data is supplied by potential untrustworthy software, so it can be relied on only if data authenticated via the MPPrivK-generation function or via the MPMR has demonstrated that the software that supplied the data was properly authenticated via the super root key hash.

12.8.2 MPPrivK-generation function

The MPPrivK-generation function uses supplied input data together with a secret value embedded in the silicon to generate an elliptic-curve DSA private key. The function stores the private key in the MPPKR and then the MPPKR is locked to prevent reading or writing from the register bus. The private key is later used in the MPPubK-generation function and the MPSign function. Note that an error is generated if the MPPrivK Generation function is run a second time in the same power-on session.

12.8.2.1 Differences between the MPPrivK-generation function and the DL KEY PAIR GEN function

The MPPrivK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPrivK generation function generates only ECDSA private keys, not DSA keypairs.
- The MPPrivK generation function generates the private key by applying a key generation function to the input message data and a secret value embedded in the silicon. The secret value is different in each chip type. The DL KEY PAIR GEN function cannot use the secret value embedded in the silicon.

- The MPPrivK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the CSEL field in the PDB. The DL KEY PAIR GEN function uses curve parameters supplied via the PDB.
- The MPPrivK generation function keeps the private key secret by storing it in the MPPrivK register. The DL KEY PAIR GEN function outputs the private key to memory (along with the public key).

12.8.2.2 MPPrivK-generation function parameters and operation

This table describes the MPPrivK-generation parameters.

Table 12-6. MPPrivK-generation function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	$2*L$	ECC curve parameters.
$G_{x,y}$	Built-in	$2*L$	Generator point
m	Input	-	The message data to be input to the private-key generator function
s	Stored in MPPKR	N	Private key

This table describes the inputs, outputs and operation of the MPPrivK function.

Table 12-7. MPPrivK-generation function inputs, outputs, and operation

Property	Value
Inputs	<ul style="list-style-type: none"> • Message data to be input to the private key generation function. • The Csel field in the PDB, selecting a predefined ECC curve.
Outputs	<ul style="list-style-type: none"> • The manufacturing protection private key s, which is stored in the MPPrivK register.
Operation	<ul style="list-style-type: none"> • Generate a private key s, in the range $1 \leq s < r$. (Hash the supplied message data and the built-in secret value to yield s. If $s=0$, alter the input to the generation function by a constant and generate a new s.) • Store s in MPKeyR as the private key.

12.8.2.3 Protocol data block (PDB) for the MPPrivK-generation function

This figure shows the PDB for the MPPrivK-generation function.

Table 12-8. MPPrivK-generation PDB

SGF (1 bit)	Reserved (10 bits)	Csel (4 bits)	Reserved (17 bits)
Pointer to m			
Message length			

This figure shows the format of the SGF field.

Table 12-9. MPPrivK-generation function PDB-format of the SGF field

31
ref m SGF (Scatter Gather Flag) If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.

This figure shows the format of the CSEL field.

Table 12-10. MPPrivK-generation function PDB-format of the CSEL field

20-17
CSEL (Curve Select) <ul style="list-style-type: none"> • 0011 = P256 • 0100 = P384 • 0101 = P521
All other values are reserved.

12.8.3 MPPubK-generation function

The MPPubK-generation function uses the private key value stored in the MPPrivK register by the MPPrivK-generation function to generate a matching elliptic-curve DSA public key. The curve selected via the Csel field in the PDB must match the curve used by the MPPrivK-generation function, else an error is generated. The public key created by the MPPubK-generation function is written out to the specified results destination. Note that the MPPubK Generation function is intended to be run just once, at the OEM's facility, but no harm is done if it is run at other times.

12.8.3.1 Differences between the MPPubK-generation function and the DL KEY PAIR GEN function

The MPPubK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPubK generation function generates only an ECDSA public key, not DSA or ECDSA keypairs.
- The MPPubK generation function creates a public key to match the private key value that was stored in the MPPrivK register by the MPPrivK generation function.
- The MPPubK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the Csel field in the PDB.
- The MPPubK generation function outputs only the public key. Unlike the DL KEY PAIR GEN function it does not output the private key.
- The private key stored in the MPPrivK register by the MPPrivK generation function is not altered, and remains available for use in the MPSign function.

12.8.3.2 MPPubK-generation function parameters and operation

This table describes the MPPubK-generation parameters.

Table 12-11. MPPubK-generation function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	$2*L$	ECC curve parameters.
$G_{x,y}$	Built-in	$2*L$	Generator point
s	Read from MPPKR	N	Private key
$W_{x,y}$	Output	$2*L$	Public key

This table describes the inputs, outputs and operation of the MPPubK function.

Table 12-12. MPPubK-generation function inputs, outputs, and operation

Property	Value
Inputs	<ul style="list-style-type: none"> • The Csel field in the PDB, selecting a predefined ECC curve. • The manufacturing protection private key s, which is read from the MPPrivK register
Outputs	<ul style="list-style-type: none"> • The manufacturing protection public key $W_{x,y}$, which is output to memory.
Operation	<ul style="list-style-type: none"> • Compute $W_{x,y} = sG_{x,y}$ • Output $W_{x,y}$ as the public key.

12.8.3.3 Protocol data block (PDB) for the MPPubK-generation function

This figure shows the PDB for the MPPubK-generation function.

Table 12-13. MPPubK-generation PDB

SGF (1 bits)	Reserved (10 bits)	Csel (4 bits)	Reserved (17 bits)
Pointer to Wx,y			
Message length			

This figure shows the format of the SGF field.

Table 12-14. MPPubK-generation function PDB-format of the SGF field

31
ref Wx,y SGF (Scatter Gather Flag) - If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.

This figure shows the format of the CSEL field.

Table 12-15. MPPubK-generation function PDB-format of the CSEL field

20-17
CSEL (Curve Select) <ul style="list-style-type: none"> • 0011 = P256 • 0100 = P384 • 0101 = P521
All other values are reserved.

12.8.3.4 Running the MPPubK generation function at the OEM's facility

When a chip is first adopted by an OEM, the OEM runs the MPPubK-generation function on a sample of the chip and saves the public key of the manufacturing protection keypair on the OEM's server. Running the MPPubK-generation function at the OEM's facility this one time guarantees that the public key is authentic, that is, that it matches the private key that is used to sign messages generated by properly configured NXP chips of this type. The OEM will first have programmed the trusted root public key into fuses, and then reboot the chip. The secure boot firmware will run the MPPrivK-generation function at POR and store the manufacturing protection private key in the MPPrivK register. The MPPubK-generation function will read the manufacturing protection private key from the MPPrivK register and generate a matching public key. Later when the identically configured chips are booted within the contract manufacturing facility, the identical private key will be generated by the MPPrivK generation function. The MPPrivK-

generation function stores the private key in the MPPKR for use in the MPSign function. The message signed by the MPSign function can be authenticated against the manufacturing protection public key stored on the OEM's seerver.

12.8.4 MPSign function

MPSign is the elliptic-curve, digital-signature algorithm (ECDSA) signing function used in the manufacturing protection authentication process. See [Manufacturing-protection chip-authentication process](#) for a discussion of this process. MPSign supports only ECDSA in prime fields. This function takes message data as input, and outputs a signature over a message composed of the content of the MPMR, followed by the input-data message.

Note that the curve specified via the Csel field in the PDB must match the curve used in the MPPrivK-generation function. This table lists the MPSign protocol parameters.

12.8.4.1 MPSign function parameters and operation

This table describes the MPSign function parameters.

Table 12-16. MPSign function parameters

Parameter	Source/Destination	Length	Definition
q	Built-in	L	Prime number or irreducible polynomial that creates the field
r	Built-in	N	Order of the field of private keys
a,b	Built-in	$2*L$	ECC curve parameters
$G_{x,y}$	Built-in	$2*L$	Generator point
s	Read from MPPKR	N	Private key
m	Input	-	The message data to be signed.
C	Output	N	First part of digital signature
d	Output	N	Second part of digital signature. The buffer for d must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding.
$mes-rep$	Output	256	The hash of the MPMR concatenated with m .

This table describes the inputs, outputs and operation of the MPSign function.

Table 12-17. MPSign function inputs, outputs, and operation

Property	Value
Inputs	<ul style="list-style-type: none"> m, the message data to be signed

Table continues on the next page...

Table 12-17. MPSign function inputs, outputs, and operation (continued)

Property	Value
	<ul style="list-style-type: none"> • u, the private key (from the MPPrivK register) • a, b, the curve parameters (selected via the Csel field in the PDB)
Outputs	<ul style="list-style-type: none"> • The signature over the signed message. • mes-rep, the hash of MPMR concatenated with the message data
Operation	<ul style="list-style-type: none"> • Compute $V_{x,y} = u G_{x,y}$, $c = V_x \bmod r$. If $c=0$, try again with a new u. • Compute $d = u^{-1}(f+sc) \bmod r$. If $d=0$, try again with a new u. • Output (C, d) as the signature.

12.8.4.2 Protocol data block (PDB) MPSign function

This figure shows the MPSign function PDB.

Table 12-18. MPSign function PDB

SGF (4 bits)	Reserved (7 bits)	Csel (4 bits)	Reserved (17 bits)
Pointer to m			
Pointer to mes-rep			
Pointer to C			
Pointer to d			
Message length			

This figure shows the format of the SGF field.

Table 12-19. MPSign function PDB-format of the SGF field

31	30	29	28
ref m	ref mes-rep	ref C	ref d
NOTE: If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer.			

This figure shows the format of the CSEL field.

Table 12-20. MPSign function function PDB-format of the CSEL field

20-17
CSEL (Curve Select) <ul style="list-style-type: none"> • 0011 = P256 • 0100 = P384 • 0101 = P521 All other values are reserved.

Chapter 13

SEC service error detection, recovery (reset), and reconfiguration

SEC can be concurrently used by multiple users through use-case optimized [Service interfaces](#) that efficiently coordinate the use of SEC hardware resources and address different user needs. The sharing of SEC hardware is generally managed through privileged software providing means to assign physical or virtualized SEC service access to less privileged software. While this use model enables efficient sharing of hardware resources, it requires that errors introduced by one user must not significantly affect other users.

SEC addresses this requirement by enabling service interface users to handle most of their own service management needs and by enabling management software to monitor and, if necessary, take corrective action if individual SEC users do not 'behave' in a cooperative manner, have fatally failed and need to be terminated or restarted, or simply have finished and their assigned SEC service access capability can be made available to other users. In addition, management software may detect fatal error conditions within SoC hardware components and requires support for a limited reset of SEC and/or the connected components to maximize SoC operation availability, e.g., perform a sub-system reset to minimize SoC out-of-service times.

The following describes SEC's facilities available to both (ordinary) user and (privileged) management software to detect SEC service problems and take corrective action, or enable and reassign SEC services to alternate users.

13.1 Software SEC Reset

The software SEC reset will cause most registers and state machines in SEC to reset. Registers that are not affected by software reset are listed in the description of the SWRST field in the Master Config Register (MCFGR). A software SEC reset is initiated by writing 1 to the SWRST bit in the Master Configuration Register (MCFGR). Note that SWRST will remain 1 (and the registers will be held in reset) until any outstanding SEC DMA transactions complete. Writing a 1 to SWRST will not cause a reset of the SEC

DMA unless SWRST is already 1 and a 1 is also written to DMARST. Note that writing to MCFGR will overwrite the values in LARGE_BURST, AXIPIPE, AWCACHE and ARCACHE, so to avoid disrupting outstanding DMA transactions when initiating a SWRST, these fields should be written with their current values.

SEC's Queue Manager Interface (QI) should be stopped prior to initiating a software SEC reset. This ensures that SEC will not break the dequeue or enqueue signaling handshake with QMan. Stopping QI is accomplished by writing a 1 to the STOP bit in the QI Control Register (QICTL) and waiting for the STOPD bit to assert in the QI Status Register (QISTA).

SEC's AIOP Interface (AI) should be stopped prior to initiating a software SEC reset. This ensures that SEC will not break the dequeue or enqueue signaling handshake with AIOP. Stopping AI is accomplished by writing a 1 to the STOP bit in the AI Control Register (AICTL) and waiting for the STOPD bit to assert in the AI Status Register (AISTA).

13.2 Job ring error detection, recovery, reset and reconfiguration

SEC 's Job Ring interface can be independently assigned (and re-assigned) to different users. Examples include, but are not limited to, Arm TZ software, hypervisor software, (guest) operating systems, and (ordinary) user processes. In all cases privileged software is either using a Job Ring itself or is granting Job Ring use to less privileged software.

The following describes the error detection, recovery, reset features available to less privileged software being granted use of the Job Ring (the 'user') and the features available to the privileged software having granted Job Ring use to the user (the 'manager'). A 'manager' may need to utilize Job Ring services and thus also play the role of a 'user'.

13.2.1 Job ring user services

This section describes the error detection, recovery, reset, and reconfiguration features available to Job Ring users.

13.2.1.1 Error recovery

The functional errors that can be detected by Job Rings are the ones that are reported in the ERR_TYPE field of the Job Ring Interrupt Status register (JRINTR_JRa). Almost all of these errors are caused by incorrect programming of the Job Ring.

- 00001b - Memory access error writing status to Output Ring
- 00011b - Bad input ring base address (not on a 4-byte boundary)
- 00100b - Bad output ring base address (not on a 4-byte boundary)
- 00101b - Invalid write to Input Ring Base Address register (IRBAR_JRa) or Input Ring Size register (IRSR_JRa)
- 00110b - Invalid write to Output Ring Base Address register (ORBAR_JRa) or Output Ring Size register (ORSR_JRa)
- 00111b - Job ring reset released before Job Ring is halted.
- 01000b - Removed too many jobs (ORJRR larger than ORSFR)
- 01001b - Added too many jobs (IRJAR larger than IRSAR)
- 01010b - Writing ORSF > ORS
- 01011b - Writing IRSA > IRS
- 01100b - Writing ORWI > ORS in bytes
- 01101b - Writing IRRI > IRS in bytes
- 01110b - Writing IRSA when ring is not empty
- 01111b - Writing IRRI when ring is not empty
- 10000b - Writing ORSF when ring is not empty
- 10001b - Writing ORWI when ring is not empty

Each of these errors cause the JRI bit, if IMSK is not set in the Job Ring Configuration register (JRCFG_RJa_LS), the JRE bit, and the ERR_TYPE field to be set in the Job Ring Interrupt Status register (JRINTR_JRa). Also, if IMSK is not set, the SEC interrupt output for this Job Ring will assert.

The recovery procedure for error type 00011b is to reprogram the Input Ring Base Address register (IRBAR_JRa) with an address that is aligned to a 4-byte boundary. Then clear the JRI and JRE bits in the Job Ring Interrupt Status register (JRINTR_JRa).

The recovery procedure for error type 00100b is to reprogram the Output Ring Base Address register (ORBAR_JRa) with an address that is aligned to a 4-byte boundary. Then clear the JRI and JRE bits in the Job Ring Interrupt Status register (JRINTR_JRa).

The error codes 01010b, 01011b, 01100b, and 01101b are caused by attempting to write out of range values to the Output Ring Slots Full register (ORSFR_JRa), Input Ring Slots Available register (IRSAR_JRa), Output Ring Write Index register (ORWIR_JRa) or Input Ring Read Index register (IRRIR_JRa). The write operation to the ORSFR_JRa, IRSAR_JRa, ORWIR_JRa, or IRRIR_JRa register that caused the error will be ignored. In order to recover from these errors, simply clear the JRE and JRI bits in the Job Ring

Interrupt Status register (JRINTR_JRa) and write a valid value in the ORSFR_JRa, IRSAR_JRa, ORWIR_JRa, or IRRIR_JRa register. If a valid value cannot be determined, a Job Ring reset may be required.

All other error types are more serious and may cause some jobs to never run or cause loss of some completed job information. The recovery procedure for these errors is perform a Job Ring reset (see Job Ring Command Register JRCCR_JRa RESET field). A Job Ring reset will clear all registers for that particular Job Ring except the REIRxJRa, IRBAR_JRa, IRSR_JRa, ORBAR_JRa, ORSR_JRa, and JRCFGR registers. The REIRxJRa registers should be manually reset after a Job Ring reset. The IRBAR_JRa, IRSR_JRa, ORBAR_JRa, ORSR_JRa, and JRCFGR_JRa registers can be rewritten or not, as appropriate, after a Job Ring reset. If a Job Ring reset cannot be performed for some reason, then a management service will be required.

13.2.1.2 Unrecoverable conditions

An unrecoverable condition is one from which the Job Ring user will not be able to recover or will not be able to recover without assistance from the manager. A security violation is an unrecoverable error condition. A second type of unrecoverable condition is when the manager has stopped SEC to prevent further jobs from processing.

If SEC detects a security violation, this will be reported in the Job Ring Interrupt Status register (JRINTR_JRa). The ENTER_FAIL bit in the Job Ring Interrupt Status register (JRINTR_JRa) sets when security violation occurs and the EXIT_FAIL bit in that same register sets when the security violation is cleared. If the interrupt is not masked and the FAIL_MODE bit is set in the Job Ring Configuration register (JRCFGR_JRa_MS), the interrupt for this Job Ring will assert when the ENTER_FAIL bit sets and again when the EXIT_FAIL bit sets. Any jobs running while the security violation is active terminate with a DECO fail mode error status (see [Job termination status/error codes](#)). If the FAIL_MODE bit in JRCFGR_JRa_MS is not set when the security violation is detected, the Job Ring will begin to halt. During the time while jobs from this Job Ring are still in progress, the HALT field in the JRINTR_JRa will return 01b, indicating that the Job Ring is halting. When all jobs are complete, the HALT field of the JRINTR_JRa will return 10b.

Until the security violation is resolved, the Job Ring will remain halted or will continue to process jobs, but return them with a fail mode error status, depending on the setting of the FAIL_MODE bit in JRCFGR_JRa_MS. After the security violation is resolved, the job ring will process jobs normally again. However, certain features of SEC may remain disabled until the manager performs a software SECreset or a power on reset.

If the manager stops SEC either as a result of a recoverable error interrupt or to begin reading various debug registers, no further jobs for any user will run until the manager restarts SEC to allow processing to resume.

13.2.1.3 User reconfiguration options

It is possible for the user to reconfigure the Job Ring. This can be done by performing the following steps:

- Stop adding new jobs to the input ring.
- Wait until all jobs issued to the input ring are complete or perform a Job Ring reset or park and wait for halt. A park (initiated by writing the PARK bit of the JRCR_JRa register) will allow the jobs to complete normally, while a reset will cause the jobs to terminate with a DECO Job Ring reset error.
- Process all jobs in the output ring.
- Clear any interrupt or error status in the Job Ring Interrupt Status register (JRINTR_JRa).
- Clear the recoverable error interrupt registers (REIRxJRa).
- Reprogram the Job Ring (IRBAR_JRa, IRSR_JRa, ORBAR_JRa, ORSR_JRa, and JRCFGR_JRa) registers and issue new jobs to the input ring.

13.2.2 Job ring management services

This section describes the error detection, recovery, reset, and reconfiguration features available to manage Job Ring users and their assigned resources.

13.2.2.1 Recoverable error interrupt/status notifications

The Job Ring recoverable error interrupt registers (REIR0JRa, REIR2JRa, REIR4JRa, and REIR5JRa) provide additional information in the event of a memory access error reading or writing the Job Ring, or while one of the Job Ring jobs is executing. If the bit for this Job Ring is set in the Recoverable Error Interrupt Enable register (REIE), the SEC recoverable error interrupt output will assert when a memory access error occurs for this Job Ring.

In the event of a memory access error while writing the output ring, error status will also be reported in the Job Ring Interrupt Status register (JRINTR_JRa). In the event of memory access errors reading the input ring, reading a Job Ring job descriptor, or during the execution of a Job Ring job descriptor, the error code written to the output ring will indicate a DMA error or an error reading the Descriptor or Descriptor address. See [Job termination status/error codes](#).

13.2.2.2 Ring user access termination procedure

When the manager wants to terminate the access for the current ring user, the manager must first prevent the current user from further accessing the Job Ring. Then the manager must reset the Job Ring to terminate any in progress jobs using the Job Ring Command register (JRCCR). This also prevents any new jobs that were already programmed in the input ring from starting. When the HALT field in the Job Ring Interrupt Status register (JRINTR_JRa) indicates that the Job Ring has stopped, perform a Job Ring soft reset, also using JRCCR, or, if virtualization is enabled, stop the Job Ring. This procedure does not save any state information for the Job Ring, so the ring will need to be completely reconfigured for the next user.

13.2.2.3 Ring user (re-)assignment procedure

If the Job Ring is currently assigned to a user, terminate that user's access as described above. Stop the Job Ring if it is not already stopped, if virtualization is enabled. Program the Job Ring ICID register. Start the job ring, if virtualization is enabled. Enable user Job Ring register access so that the user can program the IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers.

13.3 QMan interface error detection, recovery, reset, and reconfiguration

SEC's QMan Interface (QI) can be utilized to provide service to any software or hardware entity able to utilize a QMan portal and having access to at least 1 pair of frame queues linking the entity's portal to SEC's QMan portal. In all use cases privileged software is either configuring and using one or more portals and frame queues itself, or is granting access and use of those QMan resources to less privileged hardware or software. The availability of access to a QMan portal and an appropriately configured pair of frame queues enables the user to send service requests in the form of frame descriptors to SEC and receive SEC response frame descriptors on its QMan portal in return. Examples of possible SEC's QI users include, but are not limited to, Arm TZ software, hypervisor software, (guest) operating systems, and (ordinary) user processes.

The following describes the error detection, recovery, reset features available to less privileged software being granted use of SEC's QI (the 'user') and the features available to the privileged software having granted use of SEC's QI to the user (the 'manager'). As stated above, a 'manager' may need to utilize SEC's QI user services and thus also play the role of a 'user'.

13.3.1 QI user services

This section describes the error detection, recovery, reset, and reconfiguration features available to QI users.

13.3.1.1 Recoverable error conditions

Errors and warnings are reported to QI users in the ERR and FRC fields of the response FDs. A non-zero ERR value indicates an error or warning, plus specific bits of ERR identify a few key error types. The FRC value identifies the specific error or warning. [Table 5-3](#) lists SEC error and warning codes and shows the FRC field formatting. ERR field coding is described in the DPAA2RM "Frame Descriptor and Frame Description" chapter, and summarized below:

- ERR[7], the accelerator specific error bit, is set for all SEC errors and warnings.
- ERR[6], the frame annotation error bit, is set when there is an error handling PTA or ASA. For example, if FLC programming enables ASA output but there is no space for ASA in the output frame, ERR[6] will be set.
- ERR[5], the frame size error bit, is set when the output frame format or length is not valid. For example, if the F bit could not be set in the last entry of the output frame SGT due to a system bus access error, ERR[5] will be set.
- ERR[4], the FLC error bit, is set when the FLC in the response FD does not contain the expected value. For example, if there is an FLC read error, the response FLC address is not available and cannot be supplied in the response FD.
- ERR[3], the system bus access error bit, is set whenever there is a memory read or write error.
- ERR[2], the FD format or frame format error bit, is set whenever an FD or Frame List format error is detected. For example, if reserved fields in the FD are not zero, ERR[2] will be set.
- ERR[1:0], the QMan error bits, are never set by SEC.

When the user sends an FD to QI, the ERR field must be zero. If ERR is non-zero, the associated frame is considered misdirected since the error condition should have been processed elsewhere. In this case, QI discards the FD and reports the error condition to

system management software. No frame processing is done and no FD is returned to the QI user. Misdirected frames are handled this way to prevent error reporting loops. System management software may take corrective action, like shutting off user access to QI.

Serious errors like the misdirected frame are reported to system management software since they may indicate a system failure, user failure, or user attempt to subvert security restrictions. System management software may use QI management services to flush user jobs from QI, or even stop or reset QI, to recover QI resources and restore normal operations.

Error notification in the response FDs is the only indication of an error seen by QI users. No changes to QI programming are required to continue using SEC services via QI after a recoverable error, but a user may need to request help from QI management services to prevent additional errors. For example, a BMan buffer pool may need to be replenished. If the user determines that a reported error will affect other frames from the same flow or frames sent to SEC through the same QMan FQ, the user can issue a flush command to QI to abort SEC processing of other frames from the same FQ and/or determine when SEC is done with other frames from the same flow or FQ.

13.3.1.2 Unrecoverable error conditions

If a SEC security violation or other serious error occurs, management services may abort QI job processing, suspend QI operation, or reset QI. When a job is aborted, its response FD will be marked with an error. When QI operation is suspended, the flow of response FDs to the user will stop until management services re-enables QI. If QI is reset, QI job data is lost, so no response FDs will be sent to the user for those jobs.

13.3.1.3 User reconfiguration options

There are no QI user reconfiguration options. See [QI management services](#) for a description of QI management services that impact some or all QI users.

13.3.1.4 Flow and queue flush procedures and options

QI users can issue a flush command to QI to mark specific processing milestones or to accelerate service termination to free resources (e.g., free flow context structures or deallocate queues). Flush commands available to the user are described in the following sections.

- Flow flush

A flow flush FD can be used to determine when SEC has finished processing all frames from a specific flow. A response FD for the flow flush is returned to the user after the response FDs for all preceding frames from the flow specified in the flush FD.

- FQ flush

A FQ flush FD can be used to determine when SEC has finished processing all frames from a specific FQ assigned to the user. A response FD for the FQ flush is returned to the user after the response FDs for all other frames from the same FQ as the flush FD.

- FQ flush with abort

A FQ flush with abort is used like a FQ flush, except that it will abort the processing of frames that have not been completed yet from the same FQ as the flush FD.

13.3.2 QI management services

This section describes the error detection, recovery, reset, and reconfiguration features available to manage QI users and resources.

13.3.2.1 Recoverable error conditions

Three types of recoverable errors are reported to QI management services via the QI Recoverable Error Interrupt Record Registers and the associated interrupt. These are potentially serious errors since they may indicate system failures, user failures, or user attempts to subvert security restrictions. The three error types are misdirected frames, system bus access errors, and partition isolation errors. When the error occurs, QI will capture relevant information in the REIR Registers. Management services must write to the REIROQI register to enable capture of data for a subsequent error. This ensures that data for the first error is preserved until management services have a chance to read the registers. QI will set the MISS bit in REIROQI to indicate that another error occurred before capture was re-enabled.

13.3.2.2 Unrecoverable error conditions

If a SEC security violation or other serious error occurs, management services may abort QI job processing, suspend QI operation, or reset QI. When jobs are aborted, response FDs will be returned to the users with an error. When QI operation is suspended, the flow of response FDs to users will stop until management services re-enables QI. If QI is reset, QI job data is lost, so no response FDs will be sent to users for those jobs.

13.3.2.3 Single QI user SEC queue use termination procedure

To terminate use of a specific pair of QMan request and response queues assigned to a user for SEC services, system management software must first disable the user's access to the request queue. It can then issue a FQ flush to determine when all processing associated with the queue pair is done and the related resources can be safely allocated to other users.

13.3.2.4 Single QI user SEC access termination procedure

To terminate all access by a user to SEC services, system management software must first disable the user's access to QMan request queues linked to SEC. It can then issue an ICID flush to determine when all processing for the user is done and user resources (queues and memory) can be safely allocated to other users.

13.3.2.5 Single QI user access (re-)enable procedure

To enable and re-enable user access to SEC, system management software must assign QMan request and response queues linked to SEC, set up the associated QMan FQD, and set up the Flow Context and Shared Descriptor that control SEC processing. QI must be programmed to enable frame dequeues and stop controls or error conditions must be cleared.

13.3.2.6 Global QI user access controls

QI provides several registers for controlling dequeue and enqueue operations that are accessible only by QI management services. See the QI registers starting with Queue Interface Configuration (QICFG) in SEC Register Descriptions for details about the various configuration and control functions. These include debug features that allow QI operation to be stopped when an error occurs or when a control register bit is set so that QI state and job data can be examined. It's expected that these features will only be used

during software development. For normal operation, system management software will configure QI, enable QI dequeue operation, and then enable access for one or more users, as briefly explained in the preceding section.

Procedures for dealing with QI error interrupts or requests from QI users for error handling support are varied. QI management services can flush single user or all user jobs, change QI dequeue and enqueue configuration settings, modify Shared Descriptors and Flow Contexts, etc. It can use the same flush commands as QI users, plus ICID flush and ICID flush with abort commands.

If error recovery requires resetting SEC, QI dequeues should be disabled and then all user jobs should be flushed using controls provided in Queue Interface Control (QICTL) in SEC Register Descriptions.

13.3.2.7 Queue and ICID flush procedures and options

QI management services can issue a flush command to QI to mark specific processing milestones or to recover from an error. Flush commands available to the management services are listed below.

- FQ flush

A FQ flush can be used to determine when SEC has finished processing all frames from a specific QMan request queue. A response FD for the FQ flush is returned to the user after the response FDs for all other frames from the request queue.

- FQ flush with abort

A FQ flush with abort is used like a FQ flush, except that it will abort processing of frames from the request queue that are not already done.

- ICID flush

An ICID flush is used to determine when SEC has finished processing all frames associated with the user assigned the specified ICID for input.

- ICID flush with abort

An ICID flush with abort is used like an ICID flush, except that it will abort processing of frames from with the specified input ICID that are not already done.

13.4 AIOP interface error detection, recovery, reset, and reconfiguration

SEC's AIOP Interface (AI) can be utilized by any AIOP software task. AIOP software is privileged software and as such has full control over the configuration, management, and use of SEC's AI resources as well as the services provided through AI. Similar to the SEC QMan Interface (QI), AI provides both 'user' and 'management' services. While QI's user and management services are typically utilized by different entities, AIOP tasks can utilize both type of services due to their privileged status. SEC's AI management services by their nature may impact other AI service users. For that reason it is a requirement that AIOP tasks coordinate the use of AI management services between each other.

The following describes the error detection, recovery, reset features available to AIOP tasks utilizing SEC's AI 'user' and 'manager' services.

13.4.1 Single AI task services

This section describes the error detection, recovery, reset, and reconfiguration features applicable to individual AIOP tasks.

13.4.1.1 Recoverable error conditions

Errors and warnings are reported by AI to AIOP tasks in the same way QI reports errors to QI users, with the following exceptions:

- The default error code format is different for AIOP tasks than for QI users.
- AI handles misdirected frames like other recoverable errors. It returns the response FD with an error rather than dropping the FD like QI. This is done to prevent AIOP tasks from hanging.
- Errors pertaining to the use of FQs are not applicable to AIOP tasks.

13.4.1.2 Unrecoverable error conditions

The description in [Unrecoverable error conditions](#) applies to AIOP tasks.

13.4.1.3 Task reconfiguration options

Tasks can override the storage profile specified in the Flow Context that may have been provided by AIOP management software. AAP is commanded to append the task-specific storage profile to the dequeue data sent to AI.

13.4.2 AI management services

This section describes the error detection, recovery, reset, and reconfiguration features applicable to multiple AIOP tasks.

The AI interface is shared between privileged user tasks and each task has full service interface management capability. Access to SEC registers is coordinated between tasks on a voluntary basis, access coordination is not enforced.

13.4.2.1 Global AIOP task services

AI provides a few registers for configuration and control. See AI registers starting with AIOP Interface Configuration (AICFG) in SEC Register Descriptions. Debug registers are also available that allow AI operation to be stopped when an error occurs or when a control register bit is set so that AI state and job data can be examined. It's expected that debug features will only be used during software development.

Procedures for dealing error conditions are varied. AIOP tasks can flush jobs from a specific flow, jobs using a specific ICID for input, or all jobs. If error recovery requires resetting SEC, AI dequeues should be disabled and then all jobs should be flushed using the AI Control Register FLUSH bit (see AIOP Interface Control (AICTL) in SEC Register Descriptions).

13.4.2.2 Flow and ICID flush procedures and options

Flow and ICID flushes work the same in AI as described in [QI user services](#) and [QI management services](#) for QI.

13.5 RTIC error detection, recovery, reset, and reconfiguration

SEC's Run-Time Integrity Checker (RTIC) services are designed to be utilized by privileged software because they may affect all SEC service users. Privileged RTIC software (e.g., an OS driver) may make select (including no) RTIC services available to unprivileged (ordinary) software. The following describes the error detection, recovery, reset features available to privileged software utilizing SEC's RTIC services.

13.5.1 RTIC user services

As stated above, the RTIC feature set is designed to be utilized only by privileged software, i.e., any RTIC services made available to unprivileged software need to be made available by the privileged software. Privileged software may provide a subset (including none) of the available services available to privileged software to the unprivileged software. For a list of the available services see the following RTIC management services section.

13.5.2 RTIC management services

The RTIC interface is used by management software to offer both user and privileged Run Time Integrity Checking. Management software can initially use RTIC to perform One-time hash generation, and offer this service to other user software. This is intended for initial boot services only. After boot, the management software is expected to configure RTIC for Run-time integrity checking. RTIC will stay in the Run-time integrity check mode until a system reset occurs.

There are two run-time services which may be offered: Temporary Run-time integrity checking; and permanent Run-time Integrity checking. Temporary Run-time integrity checking can be disabled by software, or will terminate with an error interrupt. This is a recoverable error. Permanent Run-time integrity checking cannot be disabled, and will run until a system reset occurs, or an error is detected. An error will be reported as a hardware security violation to the SecMon module. This is a non-recoverable error. SecMon must be in the Trusted/Secure state to use the permanent Run-time integrity checking.

RTIC can monitor, or hash, up to four independent regions of memory. The management software can offer four services simultaneously. The completion of one monitoring, or hashing, operation does not affect the operation of those other three services. When RTIC is not monitoring, or hashing, any of the four memory regions, it can be configured for another operation, independently of any other operations it is currently performing.

13.5.2.1 Recoverable error conditions

The only error condition for One-time hash generation is a bus error; an access to memory is not allowed. This may be caused by an invalid address, or perhaps a hardware error (e.g., ECC error). When an interrupt is generated, manager software will read the RTIC status register to determine whether the hash generation completed properly, or terminated with a bus error. In either case, the One-time hash generation has completed, and RTIC can be re-configured to perform a new operation.

Temporary Run-time Integrity checking can be disabled by management software, or it will receive an interrupt specifying that a memory integrity check, or address error has occurred. In this case, an interrupt can only mean that an error has occurred, and RTIC will stop hashing that region of memory. Management software can determine which memory region had the error by reading the RTIC status register. The status register has the status of all four memory regions, and the act of reading the status register clears all errors. RTIC can then be re-configured to perform its next operation, or that region can remain idle. The management software will need to deal with the corrupted memory, perhaps by terminating the process being monitored.

13.5.2.2 Unrecoverable error conditions

Permanent Run-time Integrity checking can only terminate with a non-recoverable error. In this mode, RTIC is locked to prevent software turning off the checking. RTIC will typically be checking a critical piece of software, such as the trusted base component of the manager software itself. This software is expected to be so critical that any detected modification of code memory can only be fixed through a system shutdown and re-boot. In this case, RTIC will report the detected error by issuing a security violation directly to the SecMon. If the SecMon is in the Trusted or Secure state, it will immediately transition to the Fail state, signaling an un-recoverable error, and clearing critical key registers. The SecMon will generate an interrupt to the management software indicating that it should perform a halt or abort due to the security violation.

After issuing a security violation, RTIC will not be usable until the next system reset. There is no recovery from this error condition.

13.5.2.3 Reconfiguration procedure

A recoverable error will disable Run-time integrity checking of the memory region that had the error. Management software reads the RTIC status register to determine which memory had a recoverable error. The RTIC configuration for that region will go back to an idle state, and it can simply be configured for a new monitoring operation.

13.6 Global and DECO error detection, recovery, reset, and reconfiguration

SEC's global and DECO services are designed to be utilized by privileged software because they may affect all SEC service users. Privileged SEC software (e.g., an OS driver) may make select (including no) global SEC or DECO services available to unprivileged (ordinary) software. The following describes the error detection, recovery, reset features available to privileged software utilizing SEC's global and DECO services.

13.6.1 Global and DECO user services

As stated above, the global SEC and DECO services feature set is designed to be utilized only by privileged software, i.e., any global SEC or DECO services made available to unprivileged (ordinary) software need to be made available by privileged software. Privileged software may provide a subset (including none) of the available services available to privileged software to the unprivileged software. For a list of the available services see the following Global and DECO management services section.

13.6.2 Global SEC and DECO management services

This section describes the error detection, recovery, reset, and reconfiguration features available to manage global SEC and DECO services and resources.

13.6.2.1 Error detection

Descriptors which do not run properly in a DECO can encounter two classes of problems. The first class of problems consist of errors detected DECO, the CCB, or a CHA. In such cases, DECO reacts to the error by terminating execution and returning the corresponding error status. Note that such errors include execution errors, e.g. illegal DECO commands, and functional errors, e.g. bad data or ICV failures. Such errors are handled via normal channels and no additional access to the DECO is required.

The second class of problems is due to a hang. The vast majority of hangs are detected by the watchdog timer. If the watchdog timer fires, then the hang turns into a problem of the first class and no further action is required. However, there are some hangs which the timer either can not detect or, having detected the hang, DECO still can't recover. (The most common case which the watchdog can't detect is a loop which doesn't terminate.) In such cases, software must intervene through the recovery method.

Although it is possible to detect that a DECO has hung by noticing that the status for a job has not been returned after an unusually long time, there is a better method. The DECO Availability Register can be used to detect a DECO that is potentially hung. Each DECO automatically clears its corresponding bit in this register whenever it is idle. (Note that DECOs always report themselves as idle between jobs.) If it is suspected that a job has hung in a DECO, then software can write a 1 to each bit in this register which corresponds to a DECO. (It is safe to write all 32 bits of this register to 1 since unimplemented bits are not written and will remain 0 when read.) Software would then wait long enough for most jobs to finish and then read the register. Any bit position that is still a 1 would correspond to a DECO which has not completed its job. Software can use this method proactively. For example, software could set all the bits in the DECO Activity Register to 1, then wait a second, and check to see if any DECO was not idle during that second. If not, then set all the bits to 1 again and repeat. The actual length of time used between checks is a function of the type of descriptors being run in a particular application, the likelihood that such descriptors will hang, and the availability of a processor to check on progress.

If the DECO Activity Register indicates that a DECO has not completed a job in an excessive amount of time, further work can be done to identify the descriptor running in that DECO by reading debug registers. These registers, which are always accessible for reading, include the DECO Debug Job Register, the DECO Debug Debug Register, the DECO Debug Job Pointer, and DECO Debug Share Pointer Register, and can be used to identify which job and/or shared descriptor is running. Furthermore, the first two of these registers can identify whether progress is being made. This information can help determine whether one of the recovery processes needs to be utilized. If progress is being made, or if this particular job is expected to run for a very long time, then no action needs to be taken.

13.6.2.2 Recover procedure

In order to recover the DECO identified as hung in the DECO Activity Register, write a 1 to the corresponding bit in the DECO Reset Register. Note that this forces the DECO into an error state from which that DECO should recover. However, that recovery may not be instantaneous as the DECO needs to wait for outstanding DMA transactions to complete

and may also have to wait for buffers to be released. Once the DECO completes the reset process, the corresponding bit in the DECO Availability Register will clear. The bit in the DECO Reset Register is automatically cleared.

Chapter 14

SEC register descriptions

The SEC's memory map is divided into the following register address blocks, listed in the table below. Each block is aligned to a 64 KB page boundary.

Table 14-1. SEC Register Address Block Identifiers

Block Identifier	Included registers
0	General registers (for example, configuration, control, debugging, and RNG)
1-4	Job Ring registers (JR0-3)
5	AIOP interface registers
6	Real-time integrity check registers
7	Queue interface registers
8-13	Descriptor controller DECO 0-5 and CHA control block CCB 0-5

All reads of undefined and write-only addresses always return zero. Writes to undefined and read-only addresses are ignored. SEC will never generate a transfer error on the register bus. Although many of the SEC registers hold more than 32 bits, the register addresses shown in the Memory Map below represent how these registers are accessed over the register bus as 32-bit words.

NOTE

SEC performs certain actions automatically immediately after POR, and SEC may be used by the boot firmware at boot time. As a consequence, by the time software reads the SEC registers their reset values may already have been changed from the POR values.

NOTE

The SEC address space is divided into 16 64 KB pages to match the access granularity of the MMU. Registers that are intended to be accessed by a specific processor or process are grouped into one of these 16 pages so that access to these registers can be restricted via SMMUs or via the CPU's MMU. For instance, many configuration and status registers are located within page 0 and these are intended to be accessed only by privileged software. The settings in many of these page 0 registers affect the behavior of SEC hardware for all users of SEC. The CPU MMUs and SMMU should be configured to prevent page 0 from being accessed by untrusted software. The registers that control each Job Ring are located in separate address blocks so that access to each Job Ring can be restricted to a particular process. Some registers, such as the version ID registers, are intended to be shared among processes. Rather than require each SEC driver process to have two MMU page entries, one page for its private registers and one for the shared registers, SEC "aliases" these shared registers into the upper section of each of the 16 address blocks. Reading any one of the address aliases for the same register returns the same information. Some of these aliased registers are writable, so access to these registers may require that software implement a concurrency control construct, as would be the case with any register that is read/write accessible by multiple processes.

NOTE

The reset value of some registers differs between different versions of SEC. To ensure driver compatibility across different versions of SEC, when updating fields within registers the registers should first be read, the required fields updated, and then the register should be written. This will avoid inadvertently changing the settings of other fields in the same register.

Most of SEC's configuration registers are accessible in block 0 of SEC's register space, as indicated in the following table. These registers are intended to be accessed by privileged software (e.g. boot software, hypervisor, secure operating system).

The format and fields in each SEC register are defined below. Some of the register format figures apply to several different registers. In such cases a different register name will be associated with each of the register offset addresses that appear at the top of the register format figure. Although these registers share the same format, they are independent

registers. In addition, many registers can be accessed at multiple addresses. In these cases there will be a single register name and the list of addresses at which that register is accessible will be indicated as aliases. Unless noted in the individual register descriptions, registers are reset only at Power-On Reset (POR).

Although many of the SEC registers hold more than 32 bits, these registers are accessed over the register bus as 32-bit words. Note that all registers other than the CCB/DECO registers must be accessed only as full 32-bit words. Byte enables are permitted only for the CCB/DECO registers. All addresses not shown are reserved.

NOTE

A register whose width is shown as 32 bits in the Memory Map should be accessed as a single 32-bit bus transaction, even if there is an adjacent related 32-bit register (e.g. SECVID_MS and SECVID_LS). SEC does not double-word swap the addresses of such register pairs, so accessing them via 32-bit bus transactions will facilitate software portability across big-endian and little-endian SoCs.

14.1 SEC Memory map

SEC base address: 800_0000h

Offset	Register	Width (In bits)	Access	Reset value
4h	Master Configuration Register (MCFGR)	32	RW	0000_B701h
Ch	Security Configuration Register (SCFGR)	32	RW	0000_0000h
10h	Job Ring 0 ICID Register - most significant half (JR0ICID_MS)	32	RW	0000_0000h
14h	Job Ring 0 ICID Register - least significant half (JR0ICID_LS)	32	RW	0000_0000h
18h	Job Ring 1 ICID Register - most significant half (JR1ICID_MS)	32	RW	0000_0000h
1Ch	Job Ring 1 ICID Register - least significant half (JR1ICID_LS)	32	RW	0000_0000h
20h	Job Ring 2 ICID Register - most significant half (JR2ICID_MS)	32	RW	0000_0000h
24h	Job Ring 2 ICID Register - least significant half (JR2ICID_LS)	32	RW	0000_0000h
28h	Job Ring 3 ICID Register - most significant half (JR3ICID_MS)	32	RW	0000_0000h
2Ch	Job Ring 3 ICID Register - least significant half (JR3ICID_LS)	32	RW	0000_0000h
50h	Queue Manager Interface SDID Register (QISDID)	32	RW	0000_0000h
58h	Debug Control Register (DEBUGCTL)	32	RW	0000_0000h
5Ch	Job Ring Start Register (JRSTARTR)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
60h	RTIC ICID Register for Block A - most significant half (RTICAICID_MS)	32	RW	0000_0000h
64h	RTIC ICID Register for Block A - least significant half (RTICAICID_LS)	32	RW	0000_0000h
68h	RTIC ICID Register for Block B - most significant half (RTICBICID_MS)	32	RW	0000_0000h
6Ch	RTIC ICID Register for Block B - least significant half (RTICBICID_LS)	32	RW	0000_0000h
70h	RTIC ICID Register for Block C - most significant half (RTICCICID_MS)	32	RW	0000_0000h
74h	RTIC ICID Register for Block C - least significant half (RTICCICID_LS)	32	RW	0000_0000h
78h	RTIC ICID Register for Block D - most significant half (RTICDICID_MS)	32	RW	0000_0000h
7Ch	RTIC ICID Register for Block D - least significant half (RTICDICID_LS)	32	RW	0000_0000h
84h	Protocol Configuration Register (PROTCFG)	32	RW	0000_0001h
94h	DECO Request Source Register (DECORSR)	32	RW	0000_0000h
9Ch	DECO Request Register (DECORR)	32	RW	0000_0000h
A0h	DECO0 ICID Register - most significant half (DECO0ICID_MS)	32	RW	0000_0000h
A4h	DECO0 ICID Register - least significant half (DECO0ICID_LS)	32	RW	0000_0000h
A8h	DECO1 ICID Register - most significant half (DECO1ICID_MS)	32	RW	0000_0000h
ACh	DECO1 ICID Register - least significant half (DECO1ICID_LS)	32	RW	0000_0000h
B0h	DECO2 ICID Register - most significant half (DECO2ICID_MS)	32	RW	0000_0000h
B4h	DECO2 ICID Register - least significant half (DECO2ICID_LS)	32	RW	0000_0000h
B8h	DECO3 ICID Register - most significant half (DECO3ICID_MS)	32	RW	0000_0000h
BCh	DECO3 ICID Register - least significant half (DECO3ICID_LS)	32	RW	0000_0000h
C0h	DECO4 ICID Register - most significant half (DECO4ICID_MS)	32	RW	0000_0000h
C4h	DECO4 ICID Register - least significant half (DECO4ICID_LS)	32	RW	0000_0000h
C8h	DECO5 ICID Register - most significant half (DECO5ICID_MS)	32	RW	0000_0000h
CCh	DECO5 ICID Register - least significant half (DECO5ICID_LS)	32	RW	0000_0000h
120h	DECO Availability Register (DAR)	32	RW	0000_0000h
124h	DECO Reset Register (DRR)	32	WO	0000_0000h
1F0h	Adaptive Resource Reservation Register (ARRR)	32	RW	0000_0000h
204h	DMA Control Register (DMAC)	32	RW	0000_0003h
220h	Peak Bandwidth Smoothing Limit Register (PBSL)	32	RW	0000_000Fh
240h	DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS)	32	RO	See description.
244h	DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS)	32	RO	See description.
248h	DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS)	32	RO	See description.

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
24Ch	DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS)	32	RO	See description.
250h	DMA0 AXI ID Enable Register (DMA0_AID_ENB)	32	RO	See description.
260h	DMA0 AXI Read Timing Check Register (DMA0_ARD_TC)	64	RW	0000_0000_00_00_0000h
26Ch	DMA0 Read Timing Check Latency Register (DMA0_ARD_LAT)	32	RW	0000_0000h
270h	DMA0 AXI Write Timing Check Register (DMA0_AWR_TC)	64	RW	0000_0000_00_00_0000h
27Ch	DMA0 Write Timing Check Latency Register (DMA0_AWR_LAT)	32	RW	0000_0000h
280h	DMA1_AIDL_MAP_MS (DMA1_AIDL_MAP_MS)	32	RO	See description.
284h	DMA1_AIDL_MAP_LS (DMA1_AIDL_MAP_LS)	32	RO	See description.
288h	DMA1_AIDM_MAP_MS (DMA1_AIDM_MAP_MS)	32	RO	See description.
28Ch	DMA1_AIDM_MAP_LS (DMA1_AIDM_MAP_LS)	32	RO	See description.
290h	DMA1 AXI ID Enable Register (DMA1_AID_ENB)	32	RO	See description.
2A0h	DMA1 AXI Read Timing Check Register (DMA1_ARD_TC)	64	RW	0000_0000_00_00_0000h
2ACh	DMA1 Read Timing Check Latency Register (DMA1_ARD_LAT)	32	RW	0000_0000h
2B0h	DMA1 AXI Write Timing Check Register (DMA1_AWR_TC)	64	RW	0000_0000_00_00_0000h
2BCh	DMA1 Write Timing Check Latency Register (DMA1_AWR_LAT)	32	RW	0000_0000h
300h - 33Fh	Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)	8	RW	00h
380h - 39Fh	Manufacturing Protection Message Register (MPMR0 - MPMR31)	8	RW	00h
3C0h - 3DFh	Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)	8	RO	00h
400h - 41Ch	Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7)	32	RW	See description.
420h - 43Ch	Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)	32	RW	See description.
440h - 45Ch	Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)	32	RW	See description.
4E0h	Secure Key Nonce Register (SKNR)	64	RW	0000_0000_00_00_0000h
504h	DMA Control Register (DMA_CTRL)	32	RW	0000_0003h
50Ch	DMA Status Register (DMA_STA)	32	RO	0000_0080h
510h	DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)	32	RO	See description.
514h	DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)	32	RO	See description.

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
518h	DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)	32	RO	See description.
51Ch	DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)	32	RO	See description.
524h	DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN)	32	RO	See description.
530h	DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)	32	RW	0000_0000h
534h	DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)	32	RW	0000_0000h
538h	DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)	32	RW	0000_0000h
53Ch	DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)	32	RW	0000_0000h
540h	DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)	32	RW	0000_0000h
544h	DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)	32	RW	0000_0000h
548h	DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)	32	RW	0000_0000h
54Ch	DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)	32	RW	0000_0000h
600h	RNG TRNG Miscellaneous Control Register (RTMCTL)	32	RW	0000_0001h
604h	RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)	32	RW	0001_0022h
608h	RNG TRNG Poker Range Register (RTPKRRNG)	32	RW	0000_09A3h
60Ch	RNG TRNG Poker Maximum Limit Register (RTPKRMAX)	32	RW	0000_6920h
60Ch	RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)	32	RO	0000_0000h
610h	RNG TRNG Seed Control Register (RTSDCTL)	32	RW	0C80_09C4h
614h	RNG TRNG Sparse Bit Limit Register (RTSBLIM)	32	RW	0000_003Fh
614h	RNG TRNG Total Samples Register (RTTOTSAM)	32	RO	0000_0000h
618h	RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)	32	RW	0000_0190h
61Ch	RNG TRNG Frequency Count Register (RTFRQCNT)	32	RO	0000_0000h
61Ch	RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)	32	RW	0000_1900h
620h	RNG TRNG Statistical Check Monobit Count Register (RTSCMC)	32	RO	0000_0000h
620h	RNG TRNG Statistical Check Monobit Limit Register (RTSCML)	32	RW	010C_0568h
624h	RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)	32	RO	0000_0000h
624h	RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)	32	RW	00B2_0195h
628h	RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)	32	RO	0000_0000h
628h	RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)	32	RW	007A_00DCh
62Ch	RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)	32	RO	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
62Ch	RNG TRNG Statistical Check Run Length 3 Limit Register (RTSC R3L)	32	RW	0058_007Dh
630h	RNG TRNG Statistical Check Run Length 4 Count Register (RTSC R4C)	32	RO	0000_0000h
630h	RNG TRNG Statistical Check Run Length 4 Limit Register (RTSC R4L)	32	RW	0040_004Bh
634h	RNG TRNG Statistical Check Run Length 5 Count Register (RTSC R5C)	32	RO	0000_0000h
634h	RNG TRNG Statistical Check Run Length 5 Limit Register (RTSC R5L)	32	RW	002E_002Fh
638h	RNG TRNG Statistical Check Run Length 6+ Count Register (RTSC R6PC)	32	RO	0000_0000h
638h	RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSC R6PL)	32	RW	002E_002Fh
63Ch	RNG TRNG Status Register (RTSTATUS)	32	RO	0000_0000h
640h - 67Ch	RNG TRNG Entropy Read Register (RTENT0 - RTENT15)	32	RO	0000_0000h
680h	RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPK RCNT10)	32	RO	0000_0000h
684h	RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPK RCNT32)	32	RO	0000_0000h
688h	RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPK RCNT54)	32	RO	0000_0000h
68Ch	RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPK RCNT76)	32	RO	0000_0000h
690h	RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPK RCNT98)	32	RO	0000_0000h
694h	RNG TRNG Statistical Check Poker Count B and A Register (RTPK RCNTBA)	32	RO	0000_0000h
698h	RNG TRNG Statistical Check Poker Count D and C Register (RTPK RCNTDC)	32	RO	0000_0000h
69Ch	RNG TRNG Statistical Check Poker Count F and E Register (RTPK RCNTFE)	32	RO	0000_0000h
6C0h	RNG DRNG Status Register (RDSTA)	32	RO	0000_0000h
6D0h	RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)	32	RO	0000_0000h
6D4h	RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)	32	RO	0000_0000h
6E0h	RNG DRNG Hash Control Register (RDHCNTL)	32	RW	0000_0000h
6E4h	RNG DRNG Hash Digest Register (RDHDIG)	32	RO	0000_0000h
6E8h	RNG DRNG Hash Buffer Register (RDHBUF)	32	WO	0000_0000h
B00h	Recoverable Error Interrupt Status (REIS)	32	W1C	0000_0000h
B04h	Recoverable Error Interrupt Enable (REIE)	32	RW	0000_0000h
B08h	Recoverable Error Interrupt Force (REIF)	32	RO	0000_0000h
B0Ch	Recoverable Error Interrupt Halt (REIH)	32	RW	0000_0000h
BF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
BFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
C00h	Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR)	64	RO	0000_0000_0000_0000h
C08h	Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)	64	RO	0000_0000_0000_0000h
C10h	Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS)	32	RO	0000_0000h
C14h	Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)	32	RO	0000_0000h
C1Ch	Holding Tank Status (HT0_STATUS)	32	RO	0000_0000h
C24h	Job Queue Debug Select Register (JQ_DEBUG_SEL)	32	RW	0000_0000h
DBCh	Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS)	32	RO	0000_0000h
DC0h	Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)	32	RO	0000_0000h
DC4h	Job Ring Job-Done Job ID FIFO (JRJDJIF)	32	RO	0000_0000h
DE4h	Job Ring Job-Done Source 1 (JRJDS1)	32	RO	0000_0000h
E00h	Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)	64	RO	0000_0000_0000_0000h
F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OUT_ENC_REQ)	64	RW	0000_0000_0000_0000h
F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_INB_DEC_REQ)	64	RW	0000_0000_0000_0000h
F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OUTB_ENCRYPT)	64	RW	0000_0000_0000_0000h
F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OUTB_PROTECT)	64	RW	0000_0000_0000_0000h
F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_INB_DECRYPT)	64	RW	0000_0000_0000_0000h
F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_INB_VALIDATED)	64	RW	0000_0000_0000_0000h
FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
FACCh (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
FCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
1_0000h	Input Ring Base Address Register for Job Ring 0 (IRBAR_JR0)	64	RW	0000_0000_00 00_0000h
1_000Ch	Input Ring Size Register for Job Ring 0 (IRSR_JR0)	32	RW	0000_0000h
1_0014h	Input Ring Slots Available Register for Job Ring 0 (IRSAR_JR0)	32	RW	0000_0000h
1_001Ch	Input Ring Jobs Added Register for Job Ring0 (IRJAR_JR0)	32	RW	0000_0000h
1_0020h	Output Ring Base Address Register for Job Ring 0 (ORBAR_JR0)	64	RW	0000_0000_00 00_0000h
1_002Ch	Output Ring Size Register for Job Ring 0 (ORSR_JR0)	32	RW	0000_0000h
1_0034h	Output Ring Jobs Removed Register for Job Ring 0 (ORJRR_JR0)	32	RW	0000_0000h
1_003Ch	Output Ring Slots Full Register for Job Ring 0 (ORSFR_JR0)	32	RW	0000_0000h
1_0044h	Job Ring Output Status Register for Job Ring 0 (JRSTAR_JR0)	32	RO	0000_0000h
1_004Ch	Job Ring Interrupt Status Register for Job Ring 0 (JRINTR_JR0)	32	W1C	0000_0000h
1_0050h	Job Ring Configuration Register for Job Ring 0, most-significant half (JRCFGR_JR0_MS)	32	RW	0000_0000h
1_0054h	Job Ring Configuration Register for Job Ring 0, least-significant half (JRCFGR_JR0_LS)	32	RW	0000_0000h
1_005Ch	Input Ring Read Index Register for Job Ring 0 (IRRIR_JR0)	32	RW	0000_0000h
1_0064h	Output Ring Write Index Register for Job Ring 0 (ORWIR_JR0)	32	RW	0000_0000h
1_006Ch	Job Ring Command Register for Job Ring 0 (JR0CR_JR0)	32	WO	0000_0000h
1_0704h	Job Ring 0 Address-Array Valid Register (JR0AAV)	32	RO	0000_0000h
1_0800h	Job Ring 0 Address-Array Address 0 Register (JR0AAA0)	64	RO	0000_0000_00 00_0000h
1_0808h	Job Ring 0 Address-Array Address 1 Register (JR0AAA1)	64	RO	0000_0000_00 00_0000h
1_0810h	Job Ring 0 Address-Array Address 2 Register (JR0AAA2)	64	RO	0000_0000_00 00_0000h
1_0818h	Job Ring 0 Address-Array Address 3 Register (JR0AAA3)	64	RO	0000_0000_00 00_0000h
1_0820h	Job Ring 0 Address-Array Address 4 Register (JR0AAA4)	64	RO	0000_0000_00 00_0000h
1_0828h	Job Ring 0 Address-Array Address 5 Register (JR0AAA5)	64	RO	0000_0000_00 00_0000h
1_0830h	Job Ring 0 Address-Array Address 6 Register (JR0AAA6)	64	RO	0000_0000_00 00_0000h
1_0838h	Job Ring 0 Address-Array Address 7 Register (JR0AAA7)	64	RO	0000_0000_00 00_0000h
1_0E00h	Recoverable Error Interrupt Record 0 for Job Ring 0 (REIR0JR0)	32	RO	0000_0000h
1_0E08h	Recoverable Error Interrupt Record 2 for Job Ring 0 (REIR2JR0)	64	RO	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
1_0E10h	Recoverable Error Interrupt Record 4 for Job Ring 0 (REIR4JR0)	32	RO	0000_0000h
1_0E14h	Recoverable Error Interrupt Record 5 for Job Ring 0 (REIR5JR0)	32	RO	0000_0000h
1_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
1_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
1_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
1_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
1_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
1_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
1_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
1_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
1_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
1_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
1_0FAC h (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
1_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
1_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
1_0FCC h (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
1_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
1_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
1_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
1_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
1_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
1_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
1_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
1_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
1_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
2_0000h	Input Ring Base Address Register for Job Ring 1 (IRBAR_JR1)	64	RW	0000_0000_00 00_0000h
2_000Ch	Input Ring Size Register for Job Ring 1 (IRSR_JR1)	32	RW	0000_0000h
2_0014h	Input Ring Slots Available Register for Job Ring 1 (IRSAR_JR1)	32	RW	0000_0000h
2_001Ch	Input Ring Jobs Added Register for Job Ring1 (IRJAR_JR1)	32	RW	0000_0000h
2_0020h	Output Ring Base Address Register for Job Ring 1 (ORBAR_JR1)	64	RW	0000_0000_00 00_0000h
2_002Ch	Output Ring Size Register for Job Ring 1 (ORSR_JR1)	32	RW	0000_0000h
2_0034h	Output Ring Jobs Removed Register for Job Ring 1 (ORJRR_JR1)	32	RW	0000_0000h
2_003Ch	Output Ring Slots Full Register for Job Ring 1 (ORSFR_JR1)	32	RW	0000_0000h
2_0044h	Job Ring Output Status Register for Job Ring 1 (JRSTAR_JR1)	32	RO	0000_0000h
2_004Ch	Job Ring Interrupt Status Register for Job Ring 1 (JRINTR_JR1)	32	W1C	0000_0000h
2_0050h	Job Ring Configuration Register for Job Ring 1, most-significant half (JRCFGR_JR1_MS)	32	RW	0000_0000h
2_0054h	Job Ring Configuration Register for Job Ring 1, least-significant half (JRCFGR_JR1_LS)	32	RW	0000_0000h
2_005Ch	Input Ring Read Index Register for Job Ring 1 (IRRIR_JR1)	32	RW	0000_0000h
2_0064h	Output Ring Write Index Register for Job Ring 1 (ORWIR_JR1)	32	RW	0000_0000h
2_006Ch	Job Ring Command Register for Job Ring 1 (JRCR_JR1)	32	WO	0000_0000h
2_0704h	Job Ring 1 Address-Array Valid Register (JR1AAV)	32	RO	0000_0000h
2_0800h	Job Ring 1 Address-Array Address 0 Register (JR1AAA0)	64	RO	0000_0000_00 00_0000h
2_0808h	Job Ring 1 Address-Array Address 1 Register (JR1AAA1)	64	RO	0000_0000_00 00_0000h
2_0810h	Job Ring 1 Address-Array Address 2 Register (JR1AAA2)	64	RO	0000_0000_00 00_0000h
2_0818h	Job Ring 1 Address-Array Address 3 Register (JR1AAA3)	64	RO	0000_0000_00 00_0000h
2_0820h	Job Ring 1 Address-Array Address 4 Register (JR1AAA4)	64	RO	0000_0000_00 00_0000h
2_0828h	Job Ring 1 Address-Array Address 5 Register (JR1AAA5)	64	RO	0000_0000_00 00_0000h
2_0830h	Job Ring 1 Address-Array Address 6 Register (JR1AAA6)	64	RO	0000_0000_00 00_0000h
2_0838h	Job Ring 1 Address-Array Address 7 Register (JR1AAA7)	64	RO	0000_0000_00 00_0000h
2_0E00h	Recoverable Error Interrupt Record 0 for Job Ring 1 (REIR0JR1)	32	RO	0000_0000h
2_0E08h	Recoverable Error Interrupt Record 2 for Job Ring 1 (REIR2JR1)	64	RO	0000_0000_00 00_0000h
2_0E10h	Recoverable Error Interrupt Record 4 for Job Ring 1 (REIR4JR1)	32	RO	0000_0000h
2_0E14h	Recoverable Error Interrupt Record 5 for Job Ring 1 (REIR5JR1)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
2_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
2_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
2_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
2_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
2_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
2_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
2_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
2_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
2_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
2_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
2_0FACH (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
2_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
2_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
2_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
2_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
2_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
2_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
2_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
2_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
2_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
2_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
2_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
2_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
3_0000h	Input Ring Base Address Register for Job Ring 2 (IRBAR_JR2)	64	RW	0000_0000_00 00_0000h
3_000Ch	Input Ring Size Register for Job Ring 2 (IRSR_JR2)	32	RW	0000_0000h
3_0014h	Input Ring Slots Available Register for Job Ring 2 (IRSAR_JR2)	32	RW	0000_0000h
3_001Ch	Input Ring Jobs Added Register for Job Ring2 (IRJAR_JR2)	32	RW	0000_0000h
3_0020h	Output Ring Base Address Register for Job Ring 2 (ORBAR_JR2)	64	RW	0000_0000_00 00_0000h
3_002Ch	Output Ring Size Register for Job Ring 2 (ORSR_JR2)	32	RW	0000_0000h
3_0034h	Output Ring Jobs Removed Register for Job Ring 2 (ORJRR_JR2)	32	RW	0000_0000h
3_003Ch	Output Ring Slots Full Register for Job Ring 2 (ORSFR_JR2)	32	RW	0000_0000h
3_0044h	Job Ring Output Status Register for Job Ring 2 (JRSTAR_JR2)	32	RO	0000_0000h
3_004Ch	Job Ring Interrupt Status Register for Job Ring 2 (JRINTR_JR2)	32	W1C	0000_0000h
3_0050h	Job Ring Configuration Register for Job Ring 2, most-significant half (JR2CFG_JR2_MS)	32	RW	0000_0000h
3_0054h	Job Ring Configuration Register for Job Ring 2, least-significant half (JR2CFG_JR2_LS)	32	RW	0000_0000h
3_005Ch	Input Ring Read Index Register for Job Ring 2 (IRRIIR_JR2)	32	RW	0000_0000h
3_0064h	Output Ring Write Index Register for Job Ring 2 (ORWIR_JR2)	32	RW	0000_0000h
3_006Ch	Job Ring Command Register for Job Ring 2 (JR2CMD_JR2)	32	WO	0000_0000h
3_0704h	Job Ring 2 Address-Array Valid Register (JR2AAV)	32	RO	0000_0000h
3_0800h	Job Ring 2 Address-Array Address 0 Register (JR2AAA0)	64	RO	0000_0000_00 00_0000h
3_0808h	Job Ring 2 Address-Array Address 1 Register (JR2AAA1)	64	RO	0000_0000_00 00_0000h
3_0810h	Job Ring 2 Address-Array Address 2 Register (JR2AAA2)	64	RO	0000_0000_00 00_0000h
3_0818h	Job Ring 2 Address-Array Address 3 Register (JR2AAA3)	64	RO	0000_0000_00 00_0000h
3_0820h	Job Ring 2 Address-Array Address 4 Register (JR2AAA4)	64	RO	0000_0000_00 00_0000h
3_0828h	Job Ring 2 Address-Array Address 5 Register (JR2AAA5)	64	RO	0000_0000_00 00_0000h
3_0830h	Job Ring 2 Address-Array Address 6 Register (JR2AAA6)	64	RO	0000_0000_00 00_0000h
3_0838h	Job Ring 2 Address-Array Address 7 Register (JR2AAA7)	64	RO	0000_0000_00 00_0000h
3_0E00h	Recoverable Error Interrupt Record 0 for Job Ring 2 (REIR0JR2)	32	RO	0000_0000h
3_0E08h	Recoverable Error Interrupt Record 2 for Job Ring 2 (REIR2JR2)	64	RO	0000_0000_00 00_0000h
3_0E10h	Recoverable Error Interrupt Record 4 for Job Ring 2 (REIR4JR2)	32	RO	0000_0000h
3_0E14h	Recoverable Error Interrupt Record 5 for Job Ring 2 (REIR5JR2)	32	RO	0000_0000h
3_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_D EQ)	64	RW	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
3_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
3_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
3_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
3_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
3_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
3_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
3_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
3_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
3_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
3_0FACh (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
3_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
3_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
3_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
3_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
3_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
3_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
3_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
3_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
3_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
3_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
3_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
3_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
4_0000h	Input Ring Base Address Register for Job Ring 3 (IRBAR_JR3)	64	RW	0000_0000_0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
4_000Ch	Input Ring Size Register for Job Ring 3 (IRSR_JR3)	32	RW	0000_0000h
4_0014h	Input Ring Slots Available Register for Job Ring 3 (IRSAR_JR3)	32	RW	0000_0000h
4_001Ch	Input Ring Jobs Added Register for Job Ring3 (IRJAR_JR3)	32	RW	0000_0000h
4_0020h	Output Ring Base Address Register for Job Ring 3 (ORBAR_JR3)	64	RW	0000_0000_00 00_0000h
4_002Ch	Output Ring Size Register for Job Ring 3 (ORSR_JR3)	32	RW	0000_0000h
4_0034h	Output Ring Jobs Removed Register for Job Ring 3 (ORJRR_JR3)	32	RW	0000_0000h
4_003Ch	Output Ring Slots Full Register for Job Ring 3 (ORSFR_JR3)	32	RW	0000_0000h
4_0044h	Job Ring Output Status Register for Job Ring 3 (JRSTAR_JR3)	32	RO	0000_0000h
4_004Ch	Job Ring Interrupt Status Register for Job Ring 3 (JRINTR_JR3)	32	W1C	0000_0000h
4_0050h	Job Ring Configuration Register for Job Ring 3, most-significant half (JR3CFG_JR3_MS)	32	RW	0000_0000h
4_0054h	Job Ring Configuration Register for Job Ring 3, least-significant half (JR3CFG_JR3_LS)	32	RW	0000_0000h
4_005Ch	Input Ring Read Index Register for Job Ring 3 (IRRIR_JR3)	32	RW	0000_0000h
4_0064h	Output Ring Write Index Register for Job Ring 3 (ORWIR_JR3)	32	RW	0000_0000h
4_006Ch	Job Ring Command Register for Job Ring 3 (JR3CMD_JR3)	32	WO	0000_0000h
4_0704h	Job Ring 3 Address-Array Valid Register (JR3AAV)	32	RO	0000_0000h
4_0800h	Job Ring 3 Address-Array Address 0 Register (JR3AAA0)	64	RO	0000_0000_00 00_0000h
4_0808h	Job Ring 3 Address-Array Address 1 Register (JR3AAA1)	64	RO	0000_0000_00 00_0000h
4_0810h	Job Ring 3 Address-Array Address 2 Register (JR3AAA2)	64	RO	0000_0000_00 00_0000h
4_0818h	Job Ring 3 Address-Array Address 3 Register (JR3AAA3)	64	RO	0000_0000_00 00_0000h
4_0820h	Job Ring 3 Address-Array Address 4 Register (JR3AAA4)	64	RO	0000_0000_00 00_0000h
4_0828h	Job Ring 3 Address-Array Address 5 Register (JR3AAA5)	64	RO	0000_0000_00 00_0000h
4_0830h	Job Ring 3 Address-Array Address 6 Register (JR3AAA6)	64	RO	0000_0000_00 00_0000h
4_0838h	Job Ring 3 Address-Array Address 7 Register (JR3AAA7)	64	RO	0000_0000_00 00_0000h
4_0E00h	Recoverable Error Interrupt Record 0 for Job Ring 3 (REIR0JR3)	32	RO	0000_0000h
4_0E08h	Recoverable Error Interrupt Record 2 for Job Ring 3 (REIR2JR3)	64	RO	0000_0000_00 00_0000h
4_0E10h	Recoverable Error Interrupt Record 4 for Job Ring 3 (REIR4JR3)	32	RO	0000_0000h
4_0E14h	Recoverable Error Interrupt Record 5 for Job Ring 3 (REIR5JR3)	32	RO	0000_0000h
4_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_00 00_0000h
4_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
4_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
4_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
4_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
4_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
4_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
4_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
4_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
4_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
4_0FAC h (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
4_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
4_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
4_0FCC h (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
4_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
4_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
4_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
4_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
4_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
4_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
4_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
4_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
4_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
5_0000h	AIOP Interface Configuration Register (AICFG)	32	RW	0100_0000h
5_0004h	AIOP Interface Control Register (AICTL)	32	RW	0000_0000h
5_000Ch	AIOP Interface Status Register (AISTA)	32	RO	0000_0000h
5_0030h	AIOP Interface Dequeue Event Filter Control Register 0 (ADQEFC0)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
5_0034h	AIOP Interface Dequeue Event Filter Control Register 1 (ADQEFC1)	32	RW	0000_0000h
5_0038h	AIOP Interface Dequeue Event Filter Control Register 2 (ADQEFC2)	32	RW	0000_0000h
5_0050h	AIOP Interface Enqueue Event Filter Control Register 0 (AEQEFC0)	32	RW	0000_0000h
5_0054h	AIOP Interface Enqueue Event Filter Control Register 1 (AEQEFC1)	32	RW	0000_0000h
5_0058h	AIOP Interface Enqueue Event Filter Control Register 2 (AEQEFC2)	32	RW	0000_0000h
5_0500h	Jobs in Use Register for AIOP Interface (JOBS_IN_USE_AI)	32	RO	0000_0000h
5_0508h	Jobs Ready Register for AIOP Interface (JOBS_READY_AI)	32	RO	0000_0000h
5_0510h	Jobs Transfer Blocking Disabled Register for AIOP Interface (JOBS_XFR_BLK_DIS_AI)	32	RO	0000_0000h
5_0518h	Jobs Transferred Register for AIOP Interface (JOBS_XFRD_AI)	32	RO	0000_0000h
5_0520h	Jobs Executing Register for AIOP Interface (JOBS_EXEC_AI)	32	RO	0000_0000h
5_0528h	Jobs Done Register for AIOP Interface (JOBS_DONE_AI)	32	RO	0000_0000h
5_0570h	Job Select Register for AIOP Interface (JOB_SELECT_AI)	32	RW	0000_1800h
5_0574h	AIOP Interface Job Data Register 0 (AIJOB0)	32	RO	0000_0000h
5_0578h	AIOP Interface Job Data Register 1 (AIJOB1)	32	RO	0000_0000h
5_057Ch	AIOP Interface Job Data Register 2 (AIJOB2)	32	RU	0000_0000h
5_0580h	AIOP Interface Job Data Register 3 (AIJOB3)	32	RU	0000_0000h
5_0584h	AIOP Interface Job Data Register 4 (AIJOB4)	32	RU	0000_0000h
5_0588h	AIOP Interface Job Data Register 5 (AIJOB5)	32	RO	0000_0000h
5_058Ch	AIOP Interface Job Data Register 6 (AIJOB6)	32	RO	0000_0000h
5_0590h	AIOP Interface Job Data Register 7 (AIJOB7)	32	RO	0000_0000h
5_0594h	AIOP Interface Job Data Register 8 (AIJOB8)	32	RO	0000_0000h
5_0598h	AIOP Interface Job Data Register 9 (AIJOB9)	32	RO	0000_0000h
5_059Ch	AIOP Interface Job Data Register 10 (AIJOB10)	32	RO	0000_0000h
5_05A0h	AIOP Interface Job Data Register 11 (AIJOB11)	32	RO	0000_0000h
5_05A4h	AIOP Interface Job Data Register 12 (AIJOB12)	32	RO	0000_0000h
5_05A8h	AIOP Interface Job Data Register 13 (AIJOB13)	32	RO	0000_0000h
5_05ACh	AIOP Interface Job Data Register 14 (AIJOB14)	32	RO	0000_0000h
5_05B0h	AIOP Interface Job Data Register 15 (AIJOB15)	32	RO	0000_0000h
5_05B4h	AIOP Interface Job Data Register 16 (AIJOB16)	32	RO	0000_0000h
5_05B8h	AIOP Interface Job Data Register 17 (AIJOB17)	32	RO	0000_0000h
5_05BCh	AIOP Interface Job Data Register 18 (AIJOB18)	32	RO	0000_0000h
5_05C0h	AIOP Interface Job Data Register 19 (AIJOB19)	32	RO	0000_0000h
5_05C4h	AIOP Interface Job Data Register 20 (AIJOB20)	32	RO	0000_0000h
5_05C8h	AIOP Interface Job Data Register 21 (AIJOB21)	32	RO	0000_0000h
5_05CCh	AIOP Interface Job Data Register 22 (AIJOB22)	32	RO	0000_0000h
5_05D0h	AIOP Interface Job Data Register 23 (AIJOB23)	32	RO	0000_0000h
5_05D4h	AIOP Interface Job Data Register 24 (AIJOB24)	32	RO	0000_0000h
5_05D8h	AIOP Interface Job Data Register 25 (AIJOB25)	32	RO	0000_0000h
5_05DCh	AIOP Interface Job Data Register 26 (AIJOB26)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
5_05E0h	AIOP Interface Job Data Register 27 (AIJOB27)	32	RU	0000_0000h
5_05E4h	AIOP Interface Job Data Register 28 (AIJOB28)	32	RU	0000_0000h
5_05E8h	AIOP Interface Job Data Register 29 (AIJOB29)	32	RU	0000_0000h
5_05ECh	AIOP Interface Job Data Register 30 (AIJOB30)	32	RU	0000_0000h
5_05F0h	AIOP Interface Job Data Register 31 (AIJOB31)	32	RU	0000_0000h
5_05F4h	AIOP Interface Job Data Register 32 (AIJOB32)	32	RO	0000_0000h
5_05F8h	AIOP Interface Job Data Register 33 (AIJOB33)	32	RO	0000_0000h
5_05FCh	AIOP Interface Job Data Register 34 (AIJOB34)	32	RO	0000_0000h
5_0700h	Recoverable Error Interrupt Record 0 for the AIOP Interface (REIR 0AI)	32	RO	0000_0000h
5_0704h	Recoverable Error Interrupt Record 1 for the AIOP Interface (REIR 1AI)	32	RO	0000_0000h
5_0708h	Recoverable Error Interrupt Record 2 for the AIOP Interface (REIR 2AI)	64	RO	0000_0000_0000_0000h
5_0710h	Recoverable Error Interrupt Record 4 for the AIOP Interface (REIR 4AI)	32	RO	0000_0000h
5_0714h	Recoverable Error Interrupt Record 5 for the AIOP Interface (REIR 5AI)	32	RO	0000_0000h
5_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
5_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
5_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
5_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
5_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
5_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
5_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
5_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
5_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
5_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
5_0FAC h (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
5_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
5_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
5_0FCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
5_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
5_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
5_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
6_0004h	RTIC Status Register (RSTA)	32	RO	0004_0000h
6_000Ch	RTIC Command Register (RCMD)	32	RW	0000_0000h
6_0014h	RTIC Control Register (RCTL)	32	RW	0000_0000h
6_001Ch	RTIC Throttle Register (RTHR)	32	RW	0000_0000h
6_0028h	RTIC Watchdog Timer (RWDOG)	64	RW	0000_0000_00 00_0000h
6_0100h	RTIC Memory Block A Address 0 Register (RMAA0)	64	RW	0000_0000_00 00_0000h
6_010Ch	RTIC Memory Block A Length 0 Register (RMAL0)	32	RW	0000_0000h
6_0110h	RTIC Memory Block A Address 1 Register (RMAA1)	64	RW	0000_0000_00 00_0000h
6_011Ch	RTIC Memory Block A Length 1 Register (RMAL1)	32	RW	0000_0000h
6_0120h	RTIC Memory Block B Address 0 Register (RMBA0)	64	RW	0000_0000_00 00_0000h
6_012Ch	RTIC Memory Block B Length 0 Register (RMBL0)	32	RW	0000_0000h
6_0130h	RTIC Memory Block B Address 1 Register (RMBA1)	64	RW	0000_0000_00 00_0000h
6_013Ch	RTIC Memory Block B Length 1 Register (RMBL1)	32	RW	0000_0000h
6_0140h	RTIC Memory Block C Address 0 Register (RMCA0)	64	RW	0000_0000_00 00_0000h
6_014Ch	RTIC Memory Block C Length 0 Register (RMCL0)	32	RW	0000_0000h
6_0150h	RTIC Memory Block C Address 1 Register (RMCA1)	64	RW	0000_0000_00 00_0000h
6_015Ch	RTIC Memory Block C Length 1 Register (RMCL1)	32	RW	0000_0000h
6_0160h	RTIC Memory Block D Address 0 Register (RMDA0)	64	RW	0000_0000_00 00_0000h
6_016Ch	RTIC Memory Block D Length 0 Register (RMDL0)	32	RW	0000_0000h
6_0170h	RTIC Memory Block D Address 1 Register (RMDA1)	64	RW	0000_0000_00 00_0000h
6_017Ch	RTIC Memory Block D Length 1 Register (RMDL1)	32	RW	0000_0000h
6_0200h	RTIC Memory Block A Big Endian Hash Result Word 0 (RAMDB_0)	32	RW	0000_0000h
6_0204h	RTIC Memory Block A Big Endian Hash Result Word 1 (RAMDB_1)	32	RW	0000_0000h
6_0208h	RTIC Memory Block A Big Endian Hash Result Word 2 (RAMDB_2)	32	RW	0000_0000h
6_020Ch	RTIC Memory Block A Big Endian Hash Result Word 3 (RAMDB_3)	32	RW	0000_0000h
6_0210h	RTIC Memory Block A Big Endian Hash Result Word 4 (RAMDB_4)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_0214h	RTIC Memory Block A Big Endian Hash Result Word 5 (RAMDB_5)	32	RW	0000_0000h
6_0218h	RTIC Memory Block A Big Endian Hash Result Word 6 (RAMDB_6)	32	RW	0000_0000h
6_021Ch	RTIC Memory Block A Big Endian Hash Result Word 7 (RAMDB_7)	32	RW	0000_0000h
6_0220h	RTIC Memory Block A Big Endian Hash Result Word 8 (RAMDB_8)	32	RW	0000_0000h
6_0224h	RTIC Memory Block A Big Endian Hash Result Word 9 (RAMDB_9)	32	RW	0000_0000h
6_0228h	RTIC Memory Block A Big Endian Hash Result Word 10 (RAMDB_10)	32	RW	0000_0000h
6_022Ch	RTIC Memory Block A Big Endian Hash Result Word 11 (RAMDB_11)	32	RW	0000_0000h
6_0230h	RTIC Memory Block A Big Endian Hash Result Word 12 (RAMDB_12)	32	RW	0000_0000h
6_0234h	RTIC Memory Block A Big Endian Hash Result Word 13 (RAMDB_13)	32	RW	0000_0000h
6_0238h	RTIC Memory Block A Big Endian Hash Result Word 14 (RAMDB_14)	32	RW	0000_0000h
6_023Ch	RTIC Memory Block A Big Endian Hash Result Word 15 (RAMDB_15)	32	RW	0000_0000h
6_0240h	RTIC Memory Block A Big Endian Hash Result Word 16 (RAMDB_16)	32	RW	0000_0000h
6_0244h	RTIC Memory Block A Big Endian Hash Result Word 17 (RAMDB_17)	32	RW	0000_0000h
6_0248h	RTIC Memory Block A Big Endian Hash Result Word 18 (RAMDB_18)	32	RW	0000_0000h
6_024Ch	RTIC Memory Block A Big Endian Hash Result Word 19 (RAMDB_19)	32	RW	0000_0000h
6_0250h	RTIC Memory Block A Big Endian Hash Result Word 20 (RAMDB_20)	32	RW	0000_0000h
6_0254h	RTIC Memory Block A Big Endian Hash Result Word 21 (RAMDB_21)	32	RW	0000_0000h
6_0258h	RTIC Memory Block A Big Endian Hash Result Word 22 (RAMDB_22)	32	RW	0000_0000h
6_025Ch	RTIC Memory Block A Big Endian Hash Result Word 23 (RAMDB_23)	32	RW	0000_0000h
6_0260h	RTIC Memory Block A Big Endian Hash Result Word 24 (RAMDB_24)	32	RW	0000_0000h
6_0264h	RTIC Memory Block A Big Endian Hash Result Word 25 (RAMDB_25)	32	RW	0000_0000h
6_0268h	RTIC Memory Block A Big Endian Hash Result Word 26 (RAMDB_26)	32	RW	0000_0000h
6_026Ch	RTIC Memory Block A Big Endian Hash Result Word 27 (RAMDB_27)	32	RW	0000_0000h
6_0270h	RTIC Memory Block A Big Endian Hash Result Word 28 (RAMDB_28)	32	RW	0000_0000h
6_0274h	RTIC Memory Block A Big Endian Hash Result Word 29 (RAMDB_29)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
6_0278h	RTIC Memory Block A Big Endian Hash Result Word 30 (RAMDB_30)	32	RW	0000_0000h
6_027Ch	RTIC Memory Block A Big Endian Hash Result Word 31 (RAMDB_31)	32	RW	0000_0000h
6_0280h	RTIC Memory Block A Little Endian Hash Result Word 0 (RAMDL_0)	32	RW	0000_0000h
6_0284h	RTIC Memory Block A Little Endian Hash Result Word 1 (RAMDL_1)	32	RW	0000_0000h
6_0288h	RTIC Memory Block A Little Endian Hash Result Word 2 (RAMDL_2)	32	RW	0000_0000h
6_028Ch	RTIC Memory Block A Little Endian Hash Result Word 3 (RAMDL_3)	32	RW	0000_0000h
6_0290h	RTIC Memory Block A Little Endian Hash Result Word 4 (RAMDL_4)	32	RW	0000_0000h
6_0294h	RTIC Memory Block A Little Endian Hash Result Word 5 (RAMDL_5)	32	RW	0000_0000h
6_0298h	RTIC Memory Block A Little Endian Hash Result Word 6 (RAMDL_6)	32	RW	0000_0000h
6_029Ch	RTIC Memory Block A Little Endian Hash Result Word 7 (RAMDL_7)	32	RW	0000_0000h
6_02A0h	RTIC Memory Block A Little Endian Hash Result Word 8 (RAMDL_8)	32	RW	0000_0000h
6_02A4h	RTIC Memory Block A Little Endian Hash Result Word 9 (RAMDL_9)	32	RW	0000_0000h
6_02A8h	RTIC Memory Block A Little Endian Hash Result Word 10 (RAMDL_10)	32	RW	0000_0000h
6_02ACh	RTIC Memory Block A Little Endian Hash Result Word 11 (RAMDL_11)	32	RW	0000_0000h
6_02B0h	RTIC Memory Block A Little Endian Hash Result Word 12 (RAMDL_12)	32	RW	0000_0000h
6_02B4h	RTIC Memory Block A Little Endian Hash Result Word 13 (RAMDL_13)	32	RW	0000_0000h
6_02B8h	RTIC Memory Block A Little Endian Hash Result Word 14 (RAMDL_14)	32	RW	0000_0000h
6_02BCh	RTIC Memory Block A Little Endian Hash Result Word 15 (RAMDL_15)	32	RW	0000_0000h
6_02C0h	RTIC Memory Block A Little Endian Hash Result Word 16 (RAMDL_16)	32	RW	0000_0000h
6_02C4h	RTIC Memory Block A Little Endian Hash Result Word 17 (RAMDL_17)	32	RW	0000_0000h
6_02C8h	RTIC Memory Block A Little Endian Hash Result Word 18 (RAMDL_18)	32	RW	0000_0000h
6_02CCh	RTIC Memory Block A Little Endian Hash Result Word 19 (RAMDL_19)	32	RW	0000_0000h
6_02D0h	RTIC Memory Block A Little Endian Hash Result Word 20 (RAMDL_20)	32	RW	0000_0000h
6_02D4h	RTIC Memory Block A Little Endian Hash Result Word 21 (RAMDL_21)	32	RW	0000_0000h
6_02D8h	RTIC Memory Block A Little Endian Hash Result Word 22 (RAMDL_22)	32	RW	0000_0000h
6_02DCh	RTIC Memory Block A Little Endian Hash Result Word 23 (RAMDL_23)	32	RW	0000_0000h
6_02E0h	RTIC Memory Block A Little Endian Hash Result Word 24 (RAMDL_24)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_02E4h	RTIC Memory Block A Little Endian Hash Result Word 25 (RAMDL_25)	32	RW	0000_0000h
6_02E8h	RTIC Memory Block A Little Endian Hash Result Word 26 (RAMDL_26)	32	RW	0000_0000h
6_02ECh	RTIC Memory Block A Little Endian Hash Result Word 27 (RAMDL_27)	32	RW	0000_0000h
6_02F0h	RTIC Memory Block A Little Endian Hash Result Word 28 (RAMDL_28)	32	RW	0000_0000h
6_02F4h	RTIC Memory Block A Little Endian Hash Result Word 29 (RAMDL_29)	32	RW	0000_0000h
6_02F8h	RTIC Memory Block A Little Endian Hash Result Word 30 (RAMDL_30)	32	RW	0000_0000h
6_02FCh	RTIC Memory Block A Little Endian Hash Result Word 31 (RAMDL_31)	32	RW	0000_0000h
6_0300h	RTIC Memory Block B Big Endian Hash Result Word 0 (RBMDB_0)	32	RW	0000_0000h
6_0304h	RTIC Memory Block B Big Endian Hash Result Word 1 (RBMDB_1)	32	RW	0000_0000h
6_0308h	RTIC Memory Block B Big Endian Hash Result Word 2 (RBMDB_2)	32	RW	0000_0000h
6_030Ch	RTIC Memory Block B Big Endian Hash Result Word 3 (RBMDB_3)	32	RW	0000_0000h
6_0310h	RTIC Memory Block B Big Endian Hash Result Word 4 (RBMDB_4)	32	RW	0000_0000h
6_0314h	RTIC Memory Block B Big Endian Hash Result Word 5 (RBMDB_5)	32	RW	0000_0000h
6_0318h	RTIC Memory Block B Big Endian Hash Result Word 6 (RBMDB_6)	32	RW	0000_0000h
6_031Ch	RTIC Memory Block B Big Endian Hash Result Word 7 (RBMDB_7)	32	RW	0000_0000h
6_0320h	RTIC Memory Block B Big Endian Hash Result Word 8 (RBMDB_8)	32	RW	0000_0000h
6_0324h	RTIC Memory Block B Big Endian Hash Result Word 9 (RBMDB_9)	32	RW	0000_0000h
6_0328h	RTIC Memory Block B Big Endian Hash Result Word 10 (RBMDB_10)	32	RW	0000_0000h
6_032Ch	RTIC Memory Block B Big Endian Hash Result Word 11 (RBMDB_11)	32	RW	0000_0000h
6_0330h	RTIC Memory Block B Big Endian Hash Result Word 12 (RBMDB_12)	32	RW	0000_0000h
6_0334h	RTIC Memory Block B Big Endian Hash Result Word 13 (RBMDB_13)	32	RW	0000_0000h
6_0338h	RTIC Memory Block B Big Endian Hash Result Word 14 (RBMDB_14)	32	RW	0000_0000h
6_033Ch	RTIC Memory Block B Big Endian Hash Result Word 15 (RBMDB_15)	32	RW	0000_0000h
6_0340h	RTIC Memory Block B Big Endian Hash Result Word 16 (RBMDB_16)	32	RW	0000_0000h
6_0344h	RTIC Memory Block B Big Endian Hash Result Word 17 (RBMDB_17)	32	RW	0000_0000h
6_0348h	RTIC Memory Block B Big Endian Hash Result Word 18 (RBMDB_18)	32	RW	0000_0000h
6_034Ch	RTIC Memory Block B Big Endian Hash Result Word 19 (RBMDB_19)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
6_0350h	RTIC Memory Block B Big Endian Hash Result Word 20 (RBMDB_20)	32	RW	0000_0000h
6_0354h	RTIC Memory Block B Big Endian Hash Result Word 21 (RBMDB_21)	32	RW	0000_0000h
6_0358h	RTIC Memory Block B Big Endian Hash Result Word 22 (RBMDB_22)	32	RW	0000_0000h
6_035Ch	RTIC Memory Block B Big Endian Hash Result Word 23 (RBMDB_23)	32	RW	0000_0000h
6_0360h	RTIC Memory Block B Big Endian Hash Result Word 24 (RBMDB_24)	32	RW	0000_0000h
6_0364h	RTIC Memory Block B Big Endian Hash Result Word 25 (RBMDB_25)	32	RW	0000_0000h
6_0368h	RTIC Memory Block B Big Endian Hash Result Word 26 (RBMDB_26)	32	RW	0000_0000h
6_036Ch	RTIC Memory Block B Big Endian Hash Result Word 27 (RBMDB_27)	32	RW	0000_0000h
6_0370h	RTIC Memory Block B Big Endian Hash Result Word 28 (RBMDB_28)	32	RW	0000_0000h
6_0374h	RTIC Memory Block B Big Endian Hash Result Word 29 (RBMDB_29)	32	RW	0000_0000h
6_0378h	RTIC Memory Block B Big Endian Hash Result Word 30 (RBMDB_30)	32	RW	0000_0000h
6_037Ch	RTIC Memory Block B Big Endian Hash Result Word 31 (RBMDB_31)	32	RW	0000_0000h
6_0380h	RTIC Memory Block B Little Endian Hash Result Word 0 (RBMDL_0)	32	RW	0000_0000h
6_0384h	RTIC Memory Block B Little Endian Hash Result Word 1 (RBMDL_1)	32	RW	0000_0000h
6_0388h	RTIC Memory Block B Little Endian Hash Result Word 2 (RBMDL_2)	32	RW	0000_0000h
6_038Ch	RTIC Memory Block B Little Endian Hash Result Word 3 (RBMDL_3)	32	RW	0000_0000h
6_0390h	RTIC Memory Block B Little Endian Hash Result Word 4 (RBMDL_4)	32	RW	0000_0000h
6_0394h	RTIC Memory Block B Little Endian Hash Result Word 5 (RBMDL_5)	32	RW	0000_0000h
6_0398h	RTIC Memory Block B Little Endian Hash Result Word 6 (RBMDL_6)	32	RW	0000_0000h
6_039Ch	RTIC Memory Block B Little Endian Hash Result Word 7 (RBMDL_7)	32	RW	0000_0000h
6_03A0h	RTIC Memory Block B Little Endian Hash Result Word 8 (RBMDL_8)	32	RW	0000_0000h
6_03A4h	RTIC Memory Block B Little Endian Hash Result Word 9 (RBMDL_9)	32	RW	0000_0000h
6_03A8h	RTIC Memory Block B Little Endian Hash Result Word 10 (RBMDL_10)	32	RW	0000_0000h
6_03ACh	RTIC Memory Block B Little Endian Hash Result Word 11 (RBMDL_11)	32	RW	0000_0000h
6_03B0h	RTIC Memory Block B Little Endian Hash Result Word 12 (RBMDL_12)	32	RW	0000_0000h
6_03B4h	RTIC Memory Block B Little Endian Hash Result Word 13 (RBMDL_13)	32	RW	0000_0000h
6_03B8h	RTIC Memory Block B Little Endian Hash Result Word 14 (RBMDL_14)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_03BCh	RTIC Memory Block B Little Endian Hash Result Word 15 (RBMDL_15)	32	RW	0000_0000h
6_03C0h	RTIC Memory Block B Little Endian Hash Result Word 16 (RBMDL_16)	32	RW	0000_0000h
6_03C4h	RTIC Memory Block B Little Endian Hash Result Word 17 (RBMDL_17)	32	RW	0000_0000h
6_03C8h	RTIC Memory Block B Little Endian Hash Result Word 18 (RBMDL_18)	32	RW	0000_0000h
6_03CCh	RTIC Memory Block B Little Endian Hash Result Word 19 (RBMDL_19)	32	RW	0000_0000h
6_03D0h	RTIC Memory Block B Little Endian Hash Result Word 20 (RBMDL_20)	32	RW	0000_0000h
6_03D4h	RTIC Memory Block B Little Endian Hash Result Word 21 (RBMDL_21)	32	RW	0000_0000h
6_03D8h	RTIC Memory Block B Little Endian Hash Result Word 22 (RBMDL_22)	32	RW	0000_0000h
6_03DCh	RTIC Memory Block B Little Endian Hash Result Word 23 (RBMDL_23)	32	RW	0000_0000h
6_03E0h	RTIC Memory Block B Little Endian Hash Result Word 24 (RBMDL_24)	32	RW	0000_0000h
6_03E4h	RTIC Memory Block B Little Endian Hash Result Word 25 (RBMDL_25)	32	RW	0000_0000h
6_03E8h	RTIC Memory Block B Little Endian Hash Result Word 26 (RBMDL_26)	32	RW	0000_0000h
6_03ECh	RTIC Memory Block B Little Endian Hash Result Word 27 (RBMDL_27)	32	RW	0000_0000h
6_03F0h	RTIC Memory Block B Little Endian Hash Result Word 28 (RBMDL_28)	32	RW	0000_0000h
6_03F4h	RTIC Memory Block B Little Endian Hash Result Word 29 (RBMDL_29)	32	RW	0000_0000h
6_03F8h	RTIC Memory Block B Little Endian Hash Result Word 30 (RBMDL_30)	32	RW	0000_0000h
6_03FCh	RTIC Memory Block B Little Endian Hash Result Word 31 (RBMDL_31)	32	RW	0000_0000h
6_0400h	RTIC Memory Block C Big Endian Hash Result Word 0 (RCMDB_0)	32	RW	0000_0000h
6_0404h	RTIC Memory Block C Big Endian Hash Result Word 1 (RCMDB_1)	32	RW	0000_0000h
6_0408h	RTIC Memory Block C Big Endian Hash Result Word 2 (RCMDB_2)	32	RW	0000_0000h
6_040Ch	RTIC Memory Block C Big Endian Hash Result Word 3 (RCMDB_3)	32	RW	0000_0000h
6_0410h	RTIC Memory Block C Big Endian Hash Result Word 4 (RCMDB_4)	32	RW	0000_0000h
6_0414h	RTIC Memory Block C Big Endian Hash Result Word 5 (RCMDB_5)	32	RW	0000_0000h
6_0418h	RTIC Memory Block C Big Endian Hash Result Word 6 (RCMDB_6)	32	RW	0000_0000h
6_041Ch	RTIC Memory Block C Big Endian Hash Result Word 7 (RCMDB_7)	32	RW	0000_0000h
6_0420h	RTIC Memory Block C Big Endian Hash Result Word 8 (RCMDB_8)	32	RW	0000_0000h
6_0424h	RTIC Memory Block C Big Endian Hash Result Word 9 (RCMDB_9)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
6_0428h	RTIC Memory Block C Big Endian Hash Result Word 10 (RCMDB_10)	32	RW	0000_0000h
6_042Ch	RTIC Memory Block C Big Endian Hash Result Word 11 (RCMDB_11)	32	RW	0000_0000h
6_0430h	RTIC Memory Block C Big Endian Hash Result Word 12 (RCMDB_12)	32	RW	0000_0000h
6_0434h	RTIC Memory Block C Big Endian Hash Result Word 13 (RCMDB_13)	32	RW	0000_0000h
6_0438h	RTIC Memory Block C Big Endian Hash Result Word 14 (RCMDB_14)	32	RW	0000_0000h
6_043Ch	RTIC Memory Block C Big Endian Hash Result Word 15 (RCMDB_15)	32	RW	0000_0000h
6_0440h	RTIC Memory Block C Big Endian Hash Result Word 16 (RCMDB_16)	32	RW	0000_0000h
6_0444h	RTIC Memory Block C Big Endian Hash Result Word 17 (RCMDB_17)	32	RW	0000_0000h
6_0448h	RTIC Memory Block C Big Endian Hash Result Word 18 (RCMDB_18)	32	RW	0000_0000h
6_044Ch	RTIC Memory Block C Big Endian Hash Result Word 19 (RCMDB_19)	32	RW	0000_0000h
6_0450h	RTIC Memory Block C Big Endian Hash Result Word 20 (RCMDB_20)	32	RW	0000_0000h
6_0454h	RTIC Memory Block C Big Endian Hash Result Word 21 (RCMDB_21)	32	RW	0000_0000h
6_0458h	RTIC Memory Block C Big Endian Hash Result Word 22 (RCMDB_22)	32	RW	0000_0000h
6_045Ch	RTIC Memory Block C Big Endian Hash Result Word 23 (RCMDB_23)	32	RW	0000_0000h
6_0460h	RTIC Memory Block C Big Endian Hash Result Word 24 (RCMDB_24)	32	RW	0000_0000h
6_0464h	RTIC Memory Block C Big Endian Hash Result Word 25 (RCMDB_25)	32	RW	0000_0000h
6_0468h	RTIC Memory Block C Big Endian Hash Result Word 26 (RCMDB_26)	32	RW	0000_0000h
6_046Ch	RTIC Memory Block C Big Endian Hash Result Word 27 (RCMDB_27)	32	RW	0000_0000h
6_0470h	RTIC Memory Block C Big Endian Hash Result Word 28 (RCMDB_28)	32	RW	0000_0000h
6_0474h	RTIC Memory Block C Big Endian Hash Result Word 29 (RCMDB_29)	32	RW	0000_0000h
6_0478h	RTIC Memory Block C Big Endian Hash Result Word 30 (RCMDB_30)	32	RW	0000_0000h
6_047Ch	RTIC Memory Block C Big Endian Hash Result Word 31 (RCMDB_31)	32	RW	0000_0000h
6_0480h	RTIC Memory Block C Little Endian Hash Result Word 0 (RCMDL_0)	32	RW	0000_0000h
6_0484h	RTIC Memory Block C Little Endian Hash Result Word 1 (RCMDL_1)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_0488h	RTIC Memory Block C Little Endian Hash Result Word 2 (RCMDL_2)	32	RW	0000_0000h
6_048Ch	RTIC Memory Block C Little Endian Hash Result Word 3 (RCMDL_3)	32	RW	0000_0000h
6_0490h	RTIC Memory Block C Little Endian Hash Result Word 4 (RCMDL_4)	32	RW	0000_0000h
6_0494h	RTIC Memory Block C Little Endian Hash Result Word 5 (RCMDL_5)	32	RW	0000_0000h
6_0498h	RTIC Memory Block C Little Endian Hash Result Word 6 (RCMDL_6)	32	RW	0000_0000h
6_049Ch	RTIC Memory Block C Little Endian Hash Result Word 7 (RCMDL_7)	32	RW	0000_0000h
6_04A0h	RTIC Memory Block C Little Endian Hash Result Word 8 (RCMDL_8)	32	RW	0000_0000h
6_04A4h	RTIC Memory Block C Little Endian Hash Result Word 9 (RCMDL_9)	32	RW	0000_0000h
6_04A8h	RTIC Memory Block C Little Endian Hash Result Word 10 (RCMDL_10)	32	RW	0000_0000h
6_04ACh	RTIC Memory Block C Little Endian Hash Result Word 11 (RCMDL_11)	32	RW	0000_0000h
6_04B0h	RTIC Memory Block C Little Endian Hash Result Word 12 (RCMDL_12)	32	RW	0000_0000h
6_04B4h	RTIC Memory Block C Little Endian Hash Result Word 13 (RCMDL_13)	32	RW	0000_0000h
6_04B8h	RTIC Memory Block C Little Endian Hash Result Word 14 (RCMDL_14)	32	RW	0000_0000h
6_04BCh	RTIC Memory Block C Little Endian Hash Result Word 15 (RCMDL_15)	32	RW	0000_0000h
6_04C0h	RTIC Memory Block C Little Endian Hash Result Word 16 (RCMDL_16)	32	RW	0000_0000h
6_04C4h	RTIC Memory Block C Little Endian Hash Result Word 17 (RCMDL_17)	32	RW	0000_0000h
6_04C8h	RTIC Memory Block C Little Endian Hash Result Word 18 (RCMDL_18)	32	RW	0000_0000h
6_04CCh	RTIC Memory Block C Little Endian Hash Result Word 19 (RCMDL_19)	32	RW	0000_0000h
6_04D0h	RTIC Memory Block C Little Endian Hash Result Word 20 (RCMDL_20)	32	RW	0000_0000h
6_04D4h	RTIC Memory Block C Little Endian Hash Result Word 21 (RCMDL_21)	32	RW	0000_0000h
6_04D8h	RTIC Memory Block C Little Endian Hash Result Word 22 (RCMDL_22)	32	RW	0000_0000h
6_04DCh	RTIC Memory Block C Little Endian Hash Result Word 23 (RCMDL_23)	32	RW	0000_0000h
6_04E0h	RTIC Memory Block C Little Endian Hash Result Word 24 (RCMDL_24)	32	RW	0000_0000h
6_04E4h	RTIC Memory Block C Little Endian Hash Result Word 25 (RCMDL_25)	32	RW	0000_0000h
6_04E8h	RTIC Memory Block C Little Endian Hash Result Word 26 (RCMDL_26)	32	RW	0000_0000h
6_04ECh	RTIC Memory Block C Little Endian Hash Result Word 27 (RCMDL_27)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
6_04F0h	RTIC Memory Block C Little Endian Hash Result Word 28 (RCMDL_28)	32	RW	0000_0000h
6_04F4h	RTIC Memory Block C Little Endian Hash Result Word 29 (RCMDL_29)	32	RW	0000_0000h
6_04F8h	RTIC Memory Block C Little Endian Hash Result Word 30 (RCMDL_30)	32	RW	0000_0000h
6_04FCh	RTIC Memory Block C Little Endian Hash Result Word 31 (RCMDL_31)	32	RW	0000_0000h
6_0500h	RTIC Memory Block D Big Endian Hash Result Word 0 (RDMDB_0)	32	RW	0000_0000h
6_0504h	RTIC Memory Block D Big Endian Hash Result Word 1 (RDMDB_1)	32	RW	0000_0000h
6_0508h	RTIC Memory Block D Big Endian Hash Result Word 2 (RDMDB_2)	32	RW	0000_0000h
6_050Ch	RTIC Memory Block D Big Endian Hash Result Word 3 (RDMDB_3)	32	RW	0000_0000h
6_0510h	RTIC Memory Block D Big Endian Hash Result Word 4 (RDMDB_4)	32	RW	0000_0000h
6_0514h	RTIC Memory Block D Big Endian Hash Result Word 5 (RDMDB_5)	32	RW	0000_0000h
6_0518h	RTIC Memory Block D Big Endian Hash Result Word 6 (RDMDB_6)	32	RW	0000_0000h
6_051Ch	RTIC Memory Block D Big Endian Hash Result Word 7 (RDMDB_7)	32	RW	0000_0000h
6_0520h	RTIC Memory Block D Big Endian Hash Result Word 8 (RDMDB_8)	32	RW	0000_0000h
6_0524h	RTIC Memory Block D Big Endian Hash Result Word 9 (RDMDB_9)	32	RW	0000_0000h
6_0528h	RTIC Memory Block D Big Endian Hash Result Word 10 (RDMDB_10)	32	RW	0000_0000h
6_052Ch	RTIC Memory Block D Big Endian Hash Result Word 11 (RDMDB_11)	32	RW	0000_0000h
6_0530h	RTIC Memory Block D Big Endian Hash Result Word 12 (RDMDB_12)	32	RW	0000_0000h
6_0534h	RTIC Memory Block D Big Endian Hash Result Word 13 (RDMDB_13)	32	RW	0000_0000h
6_0538h	RTIC Memory Block D Big Endian Hash Result Word 14 (RDMDB_14)	32	RW	0000_0000h
6_053Ch	RTIC Memory Block D Big Endian Hash Result Word 15 (RDMDB_15)	32	RW	0000_0000h
6_0540h	RTIC Memory Block D Big Endian Hash Result Word 16 (RDMDB_16)	32	RW	0000_0000h
6_0544h	RTIC Memory Block D Big Endian Hash Result Word 17 (RDMDB_17)	32	RW	0000_0000h
6_0548h	RTIC Memory Block D Big Endian Hash Result Word 18 (RDMDB_18)	32	RW	0000_0000h
6_054Ch	RTIC Memory Block D Big Endian Hash Result Word 19 (RDMDB_19)	32	RW	0000_0000h
6_0550h	RTIC Memory Block D Big Endian Hash Result Word 20 (RDMDB_20)	32	RW	0000_0000h
6_0554h	RTIC Memory Block D Big Endian Hash Result Word 21 (RDMDB_21)	32	RW	0000_0000h
6_0558h	RTIC Memory Block D Big Endian Hash Result Word 22 (RDMDB_22)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_055Ch	RTIC Memory Block D Big Endian Hash Result Word 23 (RDMDB_23)	32	RW	0000_0000h
6_0560h	RTIC Memory Block D Big Endian Hash Result Word 24 (RDMDB_24)	32	RW	0000_0000h
6_0564h	RTIC Memory Block D Big Endian Hash Result Word 25 (RDMDB_25)	32	RW	0000_0000h
6_0568h	RTIC Memory Block D Big Endian Hash Result Word 26 (RDMDB_26)	32	RW	0000_0000h
6_056Ch	RTIC Memory Block D Big Endian Hash Result Word 27 (RDMDB_27)	32	RW	0000_0000h
6_0570h	RTIC Memory Block D Big Endian Hash Result Word 28 (RDMDB_28)	32	RW	0000_0000h
6_0574h	RTIC Memory Block D Big Endian Hash Result Word 29 (RDMDB_29)	32	RW	0000_0000h
6_0578h	RTIC Memory Block D Big Endian Hash Result Word 30 (RDMDB_30)	32	RW	0000_0000h
6_057Ch	RTIC Memory Block D Big Endian Hash Result Word 31 (RDMDB_31)	32	RW	0000_0000h
6_0580h	RTIC Memory Block D Little Endian Hash Result Word 0 (RDMDL_0)	32	RW	0000_0000h
6_0584h	RTIC Memory Block D Little Endian Hash Result Word 1 (RDMDL_1)	32	RW	0000_0000h
6_0588h	RTIC Memory Block D Little Endian Hash Result Word 2 (RDMDL_2)	32	RW	0000_0000h
6_058Ch	RTIC Memory Block D Little Endian Hash Result Word 3 (RDMDL_3)	32	RW	0000_0000h
6_0590h	RTIC Memory Block D Little Endian Hash Result Word 4 (RDMDL_4)	32	RW	0000_0000h
6_0594h	RTIC Memory Block D Little Endian Hash Result Word 5 (RDMDL_5)	32	RW	0000_0000h
6_0598h	RTIC Memory Block D Little Endian Hash Result Word 6 (RDMDL_6)	32	RW	0000_0000h
6_059Ch	RTIC Memory Block D Little Endian Hash Result Word 7 (RDMDL_7)	32	RW	0000_0000h
6_05A0h	RTIC Memory Block D Little Endian Hash Result Word 8 (RDMDL_8)	32	RW	0000_0000h
6_05A4h	RTIC Memory Block D Little Endian Hash Result Word 9 (RDMDL_9)	32	RW	0000_0000h
6_05A8h	RTIC Memory Block D Little Endian Hash Result Word 10 (RDMDL_10)	32	RW	0000_0000h
6_05ACh	RTIC Memory Block D Little Endian Hash Result Word 11 (RDMDL_11)	32	RW	0000_0000h
6_05B0h	RTIC Memory Block D Little Endian Hash Result Word 12 (RDMDL_12)	32	RW	0000_0000h
6_05B4h	RTIC Memory Block D Little Endian Hash Result Word 13 (RDMDL_13)	32	RW	0000_0000h
6_05B8h	RTIC Memory Block D Little Endian Hash Result Word 14 (RDMDL_14)	32	RW	0000_0000h
6_05BCh	RTIC Memory Block D Little Endian Hash Result Word 15 (RDMDL_15)	32	RW	0000_0000h
6_05C0h	RTIC Memory Block D Little Endian Hash Result Word 16 (RDMDL_16)	32	RW	0000_0000h
6_05C4h	RTIC Memory Block D Little Endian Hash Result Word 17 (RDMDL_17)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
6_05C8h	RTIC Memory Block D Little Endian Hash Result Word 18 (RDMDL_18)	32	RW	0000_0000h
6_05CCh	RTIC Memory Block D Little Endian Hash Result Word 19 (RDMDL_19)	32	RW	0000_0000h
6_05D0h	RTIC Memory Block D Little Endian Hash Result Word 20 (RDMDL_20)	32	RW	0000_0000h
6_05D4h	RTIC Memory Block D Little Endian Hash Result Word 21 (RDMDL_21)	32	RW	0000_0000h
6_05D8h	RTIC Memory Block D Little Endian Hash Result Word 22 (RDMDL_22)	32	RW	0000_0000h
6_05DCh	RTIC Memory Block D Little Endian Hash Result Word 23 (RDMDL_23)	32	RW	0000_0000h
6_05E0h	RTIC Memory Block D Little Endian Hash Result Word 24 (RDMDL_24)	32	RW	0000_0000h
6_05E4h	RTIC Memory Block D Little Endian Hash Result Word 25 (RDMDL_25)	32	RW	0000_0000h
6_05E8h	RTIC Memory Block D Little Endian Hash Result Word 26 (RDMDL_26)	32	RW	0000_0000h
6_05ECh	RTIC Memory Block D Little Endian Hash Result Word 27 (RDMDL_27)	32	RW	0000_0000h
6_05F0h	RTIC Memory Block D Little Endian Hash Result Word 28 (RDMDL_28)	32	RW	0000_0000h
6_05F4h	RTIC Memory Block D Little Endian Hash Result Word 29 (RDMDL_29)	32	RW	0000_0000h
6_05F8h	RTIC Memory Block D Little Endian Hash Result Word 30 (RDMDL_30)	32	RW	0000_0000h
6_05FCh	RTIC Memory Block D Little Endian Hash Result Word 31 (RDMDL_31)	32	RW	0000_0000h
6_0E00h	Recoverable Error Interrupt Record 0 for RTIC (REIR0RTIC)	32	RO	0000_0000h
6_0E08h	Recoverable Error Interrupt Record 2 for RTIC (REIR2RTIC)	64	RO	0000_0000_0000_0000h
6_0E10h	Recoverable Error Interrupt Record 4 for RTIC (REIR4RTIC)	32	RO	0000_0000h
6_0E14h	Recoverable Error Interrupt Record 5 for RTIC (REIR5RTIC)	32	RO	0000_0000h
6_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
6_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
6_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
6_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
6_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
6_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
6_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
6_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
6_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
6_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
6_0FACH (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
6_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
6_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
6_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
6_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
6_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
6_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
6_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
6_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
6_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
6_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
6_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
6_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
7_0000h	Queue Interface Configuration Register (QICFG)	32	RW	0000_0000h
7_0004h	Queue Interface Control Register (QICTL)	32	RW	0000_0000h
7_000Ch	Queue Interface Status Register (QISTA)	32	RO	0000_0000h
7_0014h	Queue Interface Dequeue Configuration Register (QIDQC)	32	RW	FFFF_0411h
7_0030h	Queue Interface Dequeue Event Filter Control Register 0 (QDQE FC0)	32	RW	0000_0000h
7_0034h	Queue Interface Dequeue Event Filter Control Register 1 (QDQE FC1)	32	RW	0000_0000h
7_0038h	Queue Interface Dequeue Event Filter Control Register 2 (QDQE FC2)	32	RW	0000_0000h
7_0050h	Queue Interface Enqueue Event Filter Control Register 0 (QEQEFC0)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
7_0054h	Queue Interface Enqueue Event Filter Control Register 1 (QEFC1)	32	RW	0000_0000h
7_0058h	Queue Interface Enqueue Event Filter Control Register 2 (QEFC2)	32	RW	0000_0000h
7_0500h	Jobs in Use Register for QM Interface (JOBS_IN_USE_QI)	32	RO	0000_0000h
7_0508h	Jobs Ready Register for QM Interface (JOBS_READY_QI)	32	RO	0000_0000h
7_0510h	Jobs Transfer Blocking Disabled Register for QM Interface (JOBS_XFR_BLK_DIS_QI)	32	RO	0000_0000h
7_0518h	Jobs Transferred Register for QM Interface (JOBS_XFRD_QI)	32	RO	0000_0000h
7_0520h	Jobs Executing Register for QM Interface (JOBS_EXEC_QI)	32	RO	0000_0000h
7_0528h	Jobs Done Register for QM Interface (JOBS_DONE_QI)	32	RO	0000_0000h
7_0530h	Subportal Frame Count Register for Queue Interface (SUBPORT_FC_QI)	32	RO	0000_0000h
7_0570h	Job Select Register for QI Interface (JOB_SELECT_QI)	32	RW	0000_1800h
7_0574h	Queue Interface Job Data Register 0 (QIJOB0)	32	RO	0000_0000h
7_0578h	Queue Interface Job Data Register 1 (QIJOB1)	32	RO	0000_0000h
7_057Ch	Queue Interface Job Data Register 2 (QIJOB2)	32	RO	0000_0000h
7_0580h	Queue Interface Job Data Register 3 (QIJOB3)	32	RO	0000_0000h
7_0584h	Queue Interface Job Data Register 4 (QIJOB4)	32	RO	0000_0000h
7_0588h	Queue Interface Job Data Register 5 (QIJOB5)	32	RO	0000_0000h
7_058Ch	Queue Interface Job Data Register 6 (QIJOB6)	32	RO	0000_0000h
7_0590h	Queue Interface Job Data Register 7 (QIJOB7)	32	RO	0000_0000h
7_0594h	Queue Interface Job Data Register 8 (QIJOB8)	32	RO	0000_0000h
7_0598h	Queue Interface Job Data Register 9 (QIJOB9)	32	RO	0000_0000h
7_059Ch	Queue Interface Job Data Register 10 (QIJOB10)	32	RO	0000_0000h
7_05A0h	Queue Interface Job Data Register 11 (QIJOB11)	32	RO	0000_0000h
7_05A4h	Queue Interface Job Data Register 12 (QIJOB12)	32	RO	0000_0000h
7_05A8h	Queue Interface Job Data Register 13 (QIJOB13)	32	RO	0000_0000h
7_05ACh	Queue Interface Job Data Register 14 (QIJOB14)	32	RO	0000_0000h
7_05B0h	Queue Interface Job Data Register 15 (QIJOB15)	32	RO	0000_0000h
7_05B4h	Queue Interface Job Data Register 16 (QIJOB16)	32	RO	0000_0000h
7_05B8h	Queue Interface Job Data Register 17 (QIJOB17)	32	RO	0000_0000h
7_05BCh	Queue Interface Job Data Register 18 (QIJOB18)	32	RO	0000_0000h
7_05C0h	Queue Interface Job Data Register 19 (QIJOB19)	32	RO	0000_0000h
7_05C4h	Queue Interface Job Data Register 20 (QIJOB20)	32	RO	0000_0000h
7_05C8h	Queue Interface Job Data Register 21 (QIJOB21)	32	RO	0000_0000h
7_05CCh	Queue Interface Job Data Register 22 (QIJOB22)	32	RO	0000_0000h
7_05D0h	Queue Interface Job Data Register 23 (QIJOB23)	32	RO	0000_0000h
7_05D4h	Queue Interface Job Data Register 24 (QIJOB24)	32	RO	0000_0000h
7_05D8h	Queue Interface Job Data Register 25 (QIJOB25)	32	RO	0000_0000h
7_05DCh	Queue Interface Job Data Register 26 (QIJOB26)	32	RO	0000_0000h
7_05E0h	Queue Interface Job Data Register 27 (QIJOB27)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
7_05E4h	Queue Interface Job Data Register 28 (QIJOB28)	32	RO	0000_0000h
7_05E8h	Queue Interface Job Data Register 29 (QIJOB29)	32	RO	0000_0000h
7_05ECh	Queue Interface Job Data Register 30 (QIJOB30)	32	RO	0000_0000h
7_05F0h	Queue Interface Job Data Register 31 (QIJOB31)	32	RO	0000_0000h
7_05F4h	Queue Interface Job Data Register 32 (QIJOB32)	32	RO	0000_0000h
7_05F8h	Queue Interface Job Data Register 33 (QIJOB33)	32	RO	0000_0000h
7_0700h	Recoverable Error Interrupt Record 0 for the Queue Interface (REIR 0QI)	32	RO	0000_0000h
7_0704h	Recoverable Error Interrupt Record 1 for the Queue Interface (REIR 1QI)	32	RO	0000_0000h
7_0708h	Recoverable Error Interrupt Record 2 for the Queue Interface (REIR 2QI)	64	RO	0000_0000_0000_0000h
7_0710h	Recoverable Error Interrupt Record 4 for the Queue Interface (REIR 4QI)	32	RO	0000_0000h
7_0714h	Recoverable Error Interrupt Record 5 for the Queue Interface (REIR 5QI)	32	RO	0000_0000h
7_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
7_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
7_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
7_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
7_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
7_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
7_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
7_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
7_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
7_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
7_0FAC h (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
7_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
7_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
7_0FCC h (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
7_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
7_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
7_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
7_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
7_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
7_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
7_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
7_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
7_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
8_0004h	CCB 0 Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK)	32	RW	0000_0000h
8_0004h	CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK)	32	RW	0000_0000h
8_0004h	CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG)	32	RW	0000_0000h
8_000Ch	CCB 0 Class 1 Key Size Register (C0C1KSR)	32	RW	0000_0000h
8_0010h	CCB 0 Class 1 Data Size Register (C0C1DSR)	64	RW	0000_0000_00 00_0000h
8_001Ch	CCB 0 Class 1 ICV Size Register (C0C1ICVSR)	32	RW	0000_0000h
8_0034h	CCB 0 CHA Control Register (C0CCTRL)	32	WO	0000_0000h
8_003Ch	CCB 0 Interrupt Control Register (C0ICTL)	32	W1C	0000_0000h
8_0044h	CCB 0 Clear Written Register (C0CWR)	32	WO	0000_0000h
8_0048h	CCB 0 Status and Error Register, most-significant half (C0CSTA_MS)	32	RO	0000_0000h
8_004Ch	CCB 0 Status and Error Register, least-significant half (C0CSTA_LS)	32	RO	0000_0000h
8_005Ch	CCB 0 AAD Size Register (C0AADSZR)	32	RW	0000_0000h
8_0064h	Class 1 IV Size Register (C0C1IVSZR)	32	RW	0000_0000h
8_0084h	PKHA A Size Register (C0PKASZR)	32	RW	0000_0000h
8_008Ch	PKHA B Size Register (C0PKBSZR)	32	RW	0000_0000h
8_0094h	PKHA N Size Register (C0PKNSZR)	32	RW	0000_0000h
8_009Ch	PKHA E Size Register (C0PKESZR)	32	RW	0000_0000h
8_0100h	CCB 0 Class 1 Context Register Word 0 (C0C1CTXR0)	32	RW	0000_0000h
8_0104h	CCB 0 Class 1 Context Register Word 1 (C0C1CTXR1)	32	RW	0000_0000h
8_0108h	CCB 0 Class 1 Context Register Word 2 (C0C1CTXR2)	32	RW	0000_0000h
8_010Ch	CCB 0 Class 1 Context Register Word 3 (C0C1CTXR3)	32	RW	0000_0000h
8_0110h	CCB 0 Class 1 Context Register Word 4 (C0C1CTXR4)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
8_0114h	CCB 0 Class 1 Context Register Word 5 (C0C1CTXR5)	32	RW	0000_0000h
8_0118h	CCB 0 Class 1 Context Register Word 6 (C0C1CTXR6)	32	RW	0000_0000h
8_011Ch	CCB 0 Class 1 Context Register Word 7 (C0C1CTXR7)	32	RW	0000_0000h
8_0120h	CCB 0 Class 1 Context Register Word 8 (C0C1CTXR8)	32	RW	0000_0000h
8_0124h	CCB 0 Class 1 Context Register Word 9 (C0C1CTXR9)	32	RW	0000_0000h
8_0128h	CCB 0 Class 1 Context Register Word 10 (C0C1CTXR10)	32	RW	0000_0000h
8_012Ch	CCB 0 Class 1 Context Register Word 11 (C0C1CTXR11)	32	RW	0000_0000h
8_0130h	CCB 0 Class 1 Context Register Word 12 (C0C1CTXR12)	32	RW	0000_0000h
8_0134h	CCB 0 Class 1 Context Register Word 13 (C0C1CTXR13)	32	RW	0000_0000h
8_0138h	CCB 0 Class 1 Context Register Word 14 (C0C1CTXR14)	32	RW	0000_0000h
8_013Ch	CCB 0 Class 1 Context Register Word 15 (C0C1CTXR15)	32	RW	0000_0000h
8_0200h	CCB 0 Class 1 Key Registers Word 0 (C0C1KR0)	32	RW	0000_0000h
8_0204h	CCB 0 Class 1 Key Registers Word 1 (C0C1KR1)	32	RW	0000_0000h
8_0208h	CCB 0 Class 1 Key Registers Word 2 (C0C1KR2)	32	RW	0000_0000h
8_020Ch	CCB 0 Class 1 Key Registers Word 3 (C0C1KR3)	32	RW	0000_0000h
8_0210h	CCB 0 Class 1 Key Registers Word 4 (C0C1KR4)	32	RW	0000_0000h
8_0214h	CCB 0 Class 1 Key Registers Word 5 (C0C1KR5)	32	RW	0000_0000h
8_0218h	CCB 0 Class 1 Key Registers Word 6 (C0C1KR6)	32	RW	0000_0000h
8_021Ch	CCB 0 Class 1 Key Registers Word 7 (C0C1KR7)	32	RW	0000_0000h
8_0404h	CCB 0 Class 2 Mode Register (C0C2MR)	32	RW	0000_0000h
8_040Ch	CCB 0 Class 2 Key Size Register (C0C2KSR)	32	RW	0000_0000h
8_0410h	CCB 0 Class 2 Data Size Register (C0C2DSR)	64	RW	0000_0000_00 00_0000h
8_041Ch	CCB 0 Class 2 ICV Size Register (C0C2ICVSZR)	32	RW	0000_0000h
8_0500h	CCB 0 Class 2 Context Register Word 0 (C0C2CTXR0)	32	RW	0000_0000h
8_0504h	CCB 0 Class 2 Context Register Word 1 (C0C2CTXR1)	32	RW	0000_0000h
8_0508h	CCB 0 Class 2 Context Register Word 2 (C0C2CTXR2)	32	RW	0000_0000h
8_050Ch	CCB 0 Class 2 Context Register Word 3 (C0C2CTXR3)	32	RW	0000_0000h
8_0510h	CCB 0 Class 2 Context Register Word 4 (C0C2CTXR4)	32	RW	0000_0000h
8_0514h	CCB 0 Class 2 Context Register Word 5 (C0C2CTXR5)	32	RW	0000_0000h
8_0518h	CCB 0 Class 2 Context Register Word 6 (C0C2CTXR6)	32	RW	0000_0000h
8_051Ch	CCB 0 Class 2 Context Register Word 7 (C0C2CTXR7)	32	RW	0000_0000h
8_0520h	CCB 0 Class 2 Context Register Word 8 (C0C2CTXR8)	32	RW	0000_0000h
8_0524h	CCB 0 Class 2 Context Register Word 9 (C0C2CTXR9)	32	RW	0000_0000h
8_0528h	CCB 0 Class 2 Context Register Word 10 (C0C2CTXR10)	32	RW	0000_0000h
8_052Ch	CCB 0 Class 2 Context Register Word 11 (C0C2CTXR11)	32	RW	0000_0000h
8_0530h	CCB 0 Class 2 Context Register Word 12 (C0C2CTXR12)	32	RW	0000_0000h
8_0534h	CCB 0 Class 2 Context Register Word 13 (C0C2CTXR13)	32	RW	0000_0000h
8_0538h	CCB 0 Class 2 Context Register Word 14 (C0C2CTXR14)	32	RW	0000_0000h
8_053Ch	CCB 0 Class 2 Context Register Word 15 (C0C2CTXR15)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
8_0540h	CCB 0 Class 2 Context Register Word 16 (C0C2CTXR16)	32	RW	0000_0000h
8_0544h	CCB 0 Class 2 Context Register Word 17 (C0C2CTXR17)	32	RW	0000_0000h
8_0600h	CCB 0 Class 2 Key Register Word 0 (C0C2KEYR0)	32	RW	0000_0000h
8_0604h	CCB 0 Class 2 Key Register Word 1 (C0C2KEYR1)	32	RW	0000_0000h
8_0608h	CCB 0 Class 2 Key Register Word 2 (C0C2KEYR2)	32	RW	0000_0000h
8_060Ch	CCB 0 Class 2 Key Register Word 3 (C0C2KEYR3)	32	RW	0000_0000h
8_0610h	CCB 0 Class 2 Key Register Word 4 (C0C2KEYR4)	32	RW	0000_0000h
8_0614h	CCB 0 Class 2 Key Register Word 5 (C0C2KEYR5)	32	RW	0000_0000h
8_0618h	CCB 0 Class 2 Key Register Word 6 (C0C2KEYR6)	32	RW	0000_0000h
8_061Ch	CCB 0 Class 2 Key Register Word 7 (C0C2KEYR7)	32	RW	0000_0000h
8_0620h	CCB 0 Class 2 Key Register Word 8 (C0C2KEYR8)	32	RW	0000_0000h
8_0624h	CCB 0 Class 2 Key Register Word 9 (C0C2KEYR9)	32	RW	0000_0000h
8_0628h	CCB 0 Class 2 Key Register Word 10 (C0C2KEYR10)	32	RW	0000_0000h
8_062Ch	CCB 0 Class 2 Key Register Word 11 (C0C2KEYR11)	32	RW	0000_0000h
8_0630h	CCB 0 Class 2 Key Register Word 12 (C0C2KEYR12)	32	RW	0000_0000h
8_0634h	CCB 0 Class 2 Key Register Word 13 (C0C2KEYR13)	32	RW	0000_0000h
8_0638h	CCB 0 Class 2 Key Register Word 14 (C0C2KEYR14)	32	RW	0000_0000h
8_063Ch	CCB 0 Class 2 Key Register Word 15 (C0C2KEYR15)	32	RW	0000_0000h
8_0640h	CCB 0 Class 2 Key Register Word 16 (C0C2KEYR16)	32	RW	0000_0000h
8_0644h	CCB 0 Class 2 Key Register Word 17 (C0C2KEYR17)	32	RW	0000_0000h
8_0648h	CCB 0 Class 2 Key Register Word 18 (C0C2KEYR18)	32	RW	0000_0000h
8_064Ch	CCB 0 Class 2 Key Register Word 19 (C0C2KEYR19)	32	RW	0000_0000h
8_0650h	CCB 0 Class 2 Key Register Word 20 (C0C2KEYR20)	32	RW	0000_0000h
8_0654h	CCB 0 Class 2 Key Register Word 21 (C0C2KEYR21)	32	RW	0000_0000h
8_0658h	CCB 0 Class 2 Key Register Word 22 (C0C2KEYR22)	32	RW	0000_0000h
8_065Ch	CCB 0 Class 2 Key Register Word 23 (C0C2KEYR23)	32	RW	0000_0000h
8_0660h	CCB 0 Class 2 Key Register Word 24 (C0C2KEYR24)	32	RW	0000_0000h
8_0664h	CCB 0 Class 2 Key Register Word 25 (C0C2KEYR25)	32	RW	0000_0000h
8_0668h	CCB 0 Class 2 Key Register Word 26 (C0C2KEYR26)	32	RW	0000_0000h
8_066Ch	CCB 0 Class 2 Key Register Word 27 (C0C2KEYR27)	32	RW	0000_0000h
8_0670h	CCB 0 Class 2 Key Register Word 28 (C0C2KEYR28)	32	RW	0000_0000h
8_0674h	CCB 0 Class 2 Key Register Word 29 (C0C2KEYR29)	32	RW	0000_0000h
8_0678h	CCB 0 Class 2 Key Register Word 30 (C0C2KEYR30)	32	RW	0000_0000h
8_067Ch	CCB 0 Class 2 Key Register Word 31 (C0C2KEYR31)	32	RW	0000_0000h
8_07C0h	CCB 0 FIFO Status (C0FIFOSTA)	32	RO	0000_0000h
8_07D0h	CCB 0 iNformation FIFO When STYPE Is Not 10 (C0NFIFO)	32	WO	0000_0000h
8_07D0h	CCB 0 iNformation FIFO When STYPE Is 10 (C0NFIFO_2)	32	WO	0000_0000h
8_07E0h	CCB 0 Input Data FIFO (C0IFIFO)	32	WO	0000_0000h
8_07F0h	CCB 0 Output Data FIFO (C0OFIFO)	64	RO	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
8_0800h	DECO0 Job Queue Control Register, most-significant half (D0JQCR_MS)	32	RW	0000_0000h
8_0804h	DECO0 Job Queue Control Register, least-significant half (D0JQCR_LS)	32	RO	0000_0000h
8_0808h	DECO0 Descriptor Address Register (D0DAR)	64	RO	0000_0000_00_00_0000h
8_0810h	DECO0 Operation Status Register, most-significant half (D0OPSTA_MS)	32	RO	0000_0000h
8_0814h	DECO0 Operation Status Register, least-significant half (D0OPSTA_LS)	32	RO	0000_0000h
8_0818h	DECO0 Checksum Register (D0CKSUMR)	32	RW	0000_0000h
8_0820h	DECO0 Control and Output ICID Status Register (D0COICIDSR)	32	RO	0000_0000h
8_0824h	DECO0 SDID and Input ICID Status Register (D0SIICIDSR)	32	RO	0000_0000h
8_0840h	DECO0 Math Register 0_MS (D0MTH0_MS)	32	RW	0000_0000h
8_0844h	DECO0 Math Register 0_LS (D0MTH0_LS)	32	RW	0000_0000h
8_0848h	DECO0 Math Register 1_MS (D0MTH1_MS)	32	RW	0000_0000h
8_084Ch	DECO0 Math Register 1_LS (D0MTH1_LS)	32	RW	0000_0000h
8_0850h	DECO0 Math Register 2_MS (D0MTH2_MS)	32	RW	0000_0000h
8_0854h	DECO0 Math Register 2_LS (D0MTH2_LS)	32	RW	0000_0000h
8_0858h	DECO0 Math Register 3_MS (D0MTH3_MS)	32	RW	0000_0000h
8_085Ch	DECO0 Math Register 3_LS (D0MTH3_LS)	32	RW	0000_0000h
8_0860h	DECO0 Math Register 4_MS (D0MTH4_MS)	32	RW	0000_0000h
8_0864h	DECO0 Math Register 4_LS (D0MTH4_LS)	32	RW	0000_0000h
8_0868h	DECO0 Math Register 5_MS (D0MTH5_MS)	32	RW	0000_0000h
8_086Ch	DECO0 Math Register 5_LS (D0MTH5_LS)	32	RW	0000_0000h
8_0870h	DECO0 Math Register 6_MS (D0MTH6_MS)	32	RW	0000_0000h
8_0874h	DECO0 Math Register 6_LS (D0MTH6_LS)	32	RW	0000_0000h
8_0878h	DECO0 Math Register 7_MS (D0MTH7_MS)	32	RW	0000_0000h
8_087Ch	DECO0 Math Register 7_LS (D0MTH7_LS)	32	RW	0000_0000h
8_0880h	DECO0 Gather Table Register 0 (D0GTR0)	128	RW	See description.
8_0890h	DECO0 Gather Table Register 1 (D0GTR1)	128	RW	See description.
8_08A0h	DECO0 Gather Table Register 2 (D0GTR2)	128	RW	See description.
8_08B0h	DECO0 Gather Table Register 3 (D0GTR3)	128	RW	See description.
8_0900h	DECO0 Scatter Table Register 0 (D0STR0)	128	RW	See description.
8_0910h	DECO0 Scatter Table Register 1 (D0STR1)	128	RW	See description.
8_0920h	DECO0 Scatter Table Register 2 (D0STR2)	128	RW	See description.

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
8_0930h	DECO0 Scatter Table Register 3 (D0STR3)	128	RW	See description.
8_0A00h	DECO0 Descriptor Buffer Word 0 (D0DESB0)	32	RW	0000_0000h
8_0A04h	DECO0 Descriptor Buffer Word 1 (D0DESB1)	32	RW	0000_0000h
8_0A08h	DECO0 Descriptor Buffer Word 2 (D0DESB2)	32	RW	0000_0000h
8_0A0Ch	DECO0 Descriptor Buffer Word 3 (D0DESB3)	32	RW	0000_0000h
8_0A10h	DECO0 Descriptor Buffer Word 4 (D0DESB4)	32	RW	0000_0000h
8_0A14h	DECO0 Descriptor Buffer Word 5 (D0DESB5)	32	RW	0000_0000h
8_0A18h	DECO0 Descriptor Buffer Word 6 (D0DESB6)	32	RW	0000_0000h
8_0A1Ch	DECO0 Descriptor Buffer Word 7 (D0DESB7)	32	RW	0000_0000h
8_0A20h	DECO0 Descriptor Buffer Word 8 (D0DESB8)	32	RW	0000_0000h
8_0A24h	DECO0 Descriptor Buffer Word 9 (D0DESB9)	32	RW	0000_0000h
8_0A28h	DECO0 Descriptor Buffer Word 10 (D0DESB10)	32	RW	0000_0000h
8_0A2Ch	DECO0 Descriptor Buffer Word 11 (D0DESB11)	32	RW	0000_0000h
8_0A30h	DECO0 Descriptor Buffer Word 12 (D0DESB12)	32	RW	0000_0000h
8_0A34h	DECO0 Descriptor Buffer Word 13 (D0DESB13)	32	RW	0000_0000h
8_0A38h	DECO0 Descriptor Buffer Word 14 (D0DESB14)	32	RW	0000_0000h
8_0A3Ch	DECO0 Descriptor Buffer Word 15 (D0DESB15)	32	RW	0000_0000h
8_0A40h	DECO0 Descriptor Buffer Word 16 (D0DESB16)	32	RW	0000_0000h
8_0A44h	DECO0 Descriptor Buffer Word 17 (D0DESB17)	32	RW	0000_0000h
8_0A48h	DECO0 Descriptor Buffer Word 18 (D0DESB18)	32	RW	0000_0000h
8_0A4Ch	DECO0 Descriptor Buffer Word 19 (D0DESB19)	32	RW	0000_0000h
8_0A50h	DECO0 Descriptor Buffer Word 20 (D0DESB20)	32	RW	0000_0000h
8_0A54h	DECO0 Descriptor Buffer Word 21 (D0DESB21)	32	RW	0000_0000h
8_0A58h	DECO0 Descriptor Buffer Word 22 (D0DESB22)	32	RW	0000_0000h
8_0A5Ch	DECO0 Descriptor Buffer Word 23 (D0DESB23)	32	RW	0000_0000h
8_0A60h	DECO0 Descriptor Buffer Word 24 (D0DESB24)	32	RW	0000_0000h
8_0A64h	DECO0 Descriptor Buffer Word 25 (D0DESB25)	32	RW	0000_0000h
8_0A68h	DECO0 Descriptor Buffer Word 26 (D0DESB26)	32	RW	0000_0000h
8_0A6Ch	DECO0 Descriptor Buffer Word 27 (D0DESB27)	32	RW	0000_0000h
8_0A70h	DECO0 Descriptor Buffer Word 28 (D0DESB28)	32	RW	0000_0000h
8_0A74h	DECO0 Descriptor Buffer Word 29 (D0DESB29)	32	RW	0000_0000h
8_0A78h	DECO0 Descriptor Buffer Word 30 (D0DESB30)	32	RW	0000_0000h
8_0A7Ch	DECO0 Descriptor Buffer Word 31 (D0DESB31)	32	RW	0000_0000h
8_0A80h	DECO0 Descriptor Buffer Word 32 (D0DESB32)	32	RW	0000_0000h
8_0A84h	DECO0 Descriptor Buffer Word 33 (D0DESB33)	32	RW	0000_0000h
8_0A88h	DECO0 Descriptor Buffer Word 34 (D0DESB34)	32	RW	0000_0000h
8_0A8Ch	DECO0 Descriptor Buffer Word 35 (D0DESB35)	32	RW	0000_0000h
8_0A90h	DECO0 Descriptor Buffer Word 36 (D0DESB36)	32	RW	0000_0000h
8_0A94h	DECO0 Descriptor Buffer Word 37 (D0DESB37)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
8_0A98h	DECO0 Descriptor Buffer Word 38 (D0DESB38)	32	RW	0000_0000h
8_0A9Ch	DECO0 Descriptor Buffer Word 39 (D0DESB39)	32	RW	0000_0000h
8_0AA0h	DECO0 Descriptor Buffer Word 40 (D0DESB40)	32	RW	0000_0000h
8_0AA4h	DECO0 Descriptor Buffer Word 41 (D0DESB41)	32	RW	0000_0000h
8_0AA8h	DECO0 Descriptor Buffer Word 42 (D0DESB42)	32	RW	0000_0000h
8_0AACh	DECO0 Descriptor Buffer Word 43 (D0DESB43)	32	RW	0000_0000h
8_0AB0h	DECO0 Descriptor Buffer Word 44 (D0DESB44)	32	RW	0000_0000h
8_0AB4h	DECO0 Descriptor Buffer Word 45 (D0DESB45)	32	RW	0000_0000h
8_0AB8h	DECO0 Descriptor Buffer Word 46 (D0DESB46)	32	RW	0000_0000h
8_0ABCh	DECO0 Descriptor Buffer Word 47 (D0DESB47)	32	RW	0000_0000h
8_0AC0h	DECO0 Descriptor Buffer Word 48 (D0DESB48)	32	RW	0000_0000h
8_0AC4h	DECO0 Descriptor Buffer Word 49 (D0DESB49)	32	RW	0000_0000h
8_0AC8h	DECO0 Descriptor Buffer Word 50 (D0DESB50)	32	RW	0000_0000h
8_0ACCh	DECO0 Descriptor Buffer Word 51 (D0DESB51)	32	RW	0000_0000h
8_0AD0h	DECO0 Descriptor Buffer Word 52 (D0DESB52)	32	RW	0000_0000h
8_0AD4h	DECO0 Descriptor Buffer Word 53 (D0DESB53)	32	RW	0000_0000h
8_0AD8h	DECO0 Descriptor Buffer Word 54 (D0DESB54)	32	RW	0000_0000h
8_0ADCh	DECO0 Descriptor Buffer Word 55 (D0DESB55)	32	RW	0000_0000h
8_0AE0h	DECO0 Descriptor Buffer Word 56 (D0DESB56)	32	RW	0000_0000h
8_0AE4h	DECO0 Descriptor Buffer Word 57 (D0DESB57)	32	RW	0000_0000h
8_0AE8h	DECO0 Descriptor Buffer Word 58 (D0DESB58)	32	RW	0000_0000h
8_0AECh	DECO0 Descriptor Buffer Word 59 (D0DESB59)	32	RW	0000_0000h
8_0AF0h	DECO0 Descriptor Buffer Word 60 (D0DESB60)	32	RW	0000_0000h
8_0AF4h	DECO0 Descriptor Buffer Word 61 (D0DESB61)	32	RW	0000_0000h
8_0AF8h	DECO0 Descriptor Buffer Word 62 (D0DESB62)	32	RW	0000_0000h
8_0AFCh	DECO0 Descriptor Buffer Word 63 (D0DESB63)	32	RW	0000_0000h
8_0E00h	DECO0 Debug Job (D0DJR)	32	RO	0000_0000h
8_0E04h	DECO0 Debug DECO (D0DDR)	32	RO	0000_0000h
8_0E08h	DECO0 Debug Job Pointer (D0DJP)	64	RO	0000_0000_00 00_0000h
8_0E10h	DECO0 Debug Shared Pointer (D0SDP)	64	RO	0000_0000_00 00_0000h
8_0E18h	DECO0 Debug_ICID, most-significant half (D0DIR_MS)	32	RO	0000_0000h
8_0E1Ch	DECO0 Debug ICID, least-significant half (D0DIR_LS)	32	RO	0000_0000h
8_0E20h	Sequence Output Length Register (SOL0)	32	RW	0000_0000h
8_0E24h	Variable Sequence Output Length Register (VSOL0)	32	RW	0000_0000h
8_0E28h	Sequence Input Length Register (SIL0)	32	RW	0000_0000h
8_0E2Ch	Variable Sequence Input Length Register (VSIL0)	32	RW	0000_0000h
8_0E30h	Protocol Override Register (D0POVRD)	32	RW	0000_0000h
8_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL0)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
8_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL0)	32	RW	0000_0000h
8_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
8_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
8_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
8_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
8_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
8_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
8_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
8_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
8_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
8_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
8_0FAC h (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
8_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
8_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
8_0FCC h (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
8_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
8_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
8_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
8_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
8_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
8_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
8_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
8_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
8_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
9_0004h	CCB 1 Class 1 Mode Register Format for Non-Public Key Algorithms (C1C1MR_NPK)	32	RW	0000_0000h
9_0004h	CCB 1 Class 1 Mode Register Format for Public Key Algorithms (C1C1MR_PK)	32	RW	0000_0000h
9_0004h	CCB 1 Class 1 Mode Register Format for RNG4 (C1C1MR_RNG)	32	RW	0000_0000h
9_000Ch	CCB 1 Class 1 Key Size Register (C1C1KSR)	32	RW	0000_0000h
9_0010h	CCB 1 Class 1 Data Size Register (C1C1DSR)	64	RW	0000_0000_00 00_0000h
9_001Ch	CCB 1 Class 1 ICV Size Register (C1C1ICVSR)	32	RW	0000_0000h
9_0034h	CCB 1 CHA Control Register (C1CCTRL)	32	WO	0000_0000h
9_003Ch	CCB 1 Interrupt Control Register (C1ICTL)	32	W1C	0000_0000h
9_0044h	CCB 1 Clear Written Register (C1CWR)	32	WO	0000_0000h
9_0048h	CCB 1 Status and Error Register, most-significant half (C1CSTA_MS)	32	RO	0000_0000h
9_004Ch	CCB 1 Status and Error Register, least-significant half (C1CSTA_LS)	32	RO	0000_0000h
9_005Ch	CCB 1 AAD Size Register (C1AADSZR)	32	RW	0000_0000h
9_0064h	Class 1 IV Size Register (C1C1IVSZR)	32	RW	0000_0000h
9_0084h	PKHA A Size Register (C1PKASZR)	32	RW	0000_0000h
9_008Ch	PKHA B Size Register (C1PKBSZR)	32	RW	0000_0000h
9_0094h	PKHA N Size Register (C1PKNSZR)	32	RW	0000_0000h
9_009Ch	PKHA E Size Register (C1PKESZR)	32	RW	0000_0000h
9_0100h	CCB 1 Class 1 Context Register Word 0 (C1C1CTXR0)	32	RW	0000_0000h
9_0104h	CCB 1 Class 1 Context Register Word 1 (C1C1CTXR1)	32	RW	0000_0000h
9_0108h	CCB 1 Class 1 Context Register Word 2 (C1C1CTXR2)	32	RW	0000_0000h
9_010Ch	CCB 1 Class 1 Context Register Word 3 (C1C1CTXR3)	32	RW	0000_0000h
9_0110h	CCB 1 Class 1 Context Register Word 4 (C1C1CTXR4)	32	RW	0000_0000h
9_0114h	CCB 1 Class 1 Context Register Word 5 (C1C1CTXR5)	32	RW	0000_0000h
9_0118h	CCB 1 Class 1 Context Register Word 6 (C1C1CTXR6)	32	RW	0000_0000h
9_011Ch	CCB 1 Class 1 Context Register Word 7 (C1C1CTXR7)	32	RW	0000_0000h
9_0120h	CCB 1 Class 1 Context Register Word 8 (C1C1CTXR8)	32	RW	0000_0000h
9_0124h	CCB 1 Class 1 Context Register Word 9 (C1C1CTXR9)	32	RW	0000_0000h
9_0128h	CCB 1 Class 1 Context Register Word 10 (C1C1CTXR10)	32	RW	0000_0000h
9_012Ch	CCB 1 Class 1 Context Register Word 11 (C1C1CTXR11)	32	RW	0000_0000h
9_0130h	CCB 1 Class 1 Context Register Word 12 (C1C1CTXR12)	32	RW	0000_0000h
9_0134h	CCB 1 Class 1 Context Register Word 13 (C1C1CTXR13)	32	RW	0000_0000h
9_0138h	CCB 1 Class 1 Context Register Word 14 (C1C1CTXR14)	32	RW	0000_0000h
9_013Ch	CCB 1 Class 1 Context Register Word 15 (C1C1CTXR15)	32	RW	0000_0000h
9_0200h	CCB 1 Class 1 Key Registers Word 0 (C1C1KR0)	32	RW	0000_0000h
9_0204h	CCB 1 Class 1 Key Registers Word 1 (C1C1KR1)	32	RW	0000_0000h
9_0208h	CCB 1 Class 1 Key Registers Word 2 (C1C1KR2)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
9_020Ch	CCB 1 Class 1 Key Registers Word 3 (C1C1KR3)	32	RW	0000_0000h
9_0210h	CCB 1 Class 1 Key Registers Word 4 (C1C1KR4)	32	RW	0000_0000h
9_0214h	CCB 1 Class 1 Key Registers Word 5 (C1C1KR5)	32	RW	0000_0000h
9_0218h	CCB 1 Class 1 Key Registers Word 6 (C1C1KR6)	32	RW	0000_0000h
9_021Ch	CCB 1 Class 1 Key Registers Word 7 (C1C1KR7)	32	RW	0000_0000h
9_0404h	CCB 1 Class 2 Mode Register (C1C2MR)	32	RW	0000_0000h
9_040Ch	CCB 1 Class 2 Key Size Register (C1C2KSR)	32	RW	0000_0000h
9_0410h	CCB 1 Class 2 Data Size Register (C1C2DSR)	64	RW	0000_0000_00 00_0000h
9_041Ch	CCB 1 Class 2 ICV Size Register (C1C2ICVSZR)	32	RW	0000_0000h
9_0500h	CCB 1 Class 2 Context Register Word 0 (C1C2CTXR0)	32	RW	0000_0000h
9_0504h	CCB 1 Class 2 Context Register Word 1 (C1C2CTXR1)	32	RW	0000_0000h
9_0508h	CCB 1 Class 2 Context Register Word 2 (C1C2CTXR2)	32	RW	0000_0000h
9_050Ch	CCB 1 Class 2 Context Register Word 3 (C1C2CTXR3)	32	RW	0000_0000h
9_0510h	CCB 1 Class 2 Context Register Word 4 (C1C2CTXR4)	32	RW	0000_0000h
9_0514h	CCB 1 Class 2 Context Register Word 5 (C1C2CTXR5)	32	RW	0000_0000h
9_0518h	CCB 1 Class 2 Context Register Word 6 (C1C2CTXR6)	32	RW	0000_0000h
9_051Ch	CCB 1 Class 2 Context Register Word 7 (C1C2CTXR7)	32	RW	0000_0000h
9_0520h	CCB 1 Class 2 Context Register Word 8 (C1C2CTXR8)	32	RW	0000_0000h
9_0524h	CCB 1 Class 2 Context Register Word 9 (C1C2CTXR9)	32	RW	0000_0000h
9_0528h	CCB 1 Class 2 Context Register Word 10 (C1C2CTXR10)	32	RW	0000_0000h
9_052Ch	CCB 1 Class 2 Context Register Word 11 (C1C2CTXR11)	32	RW	0000_0000h
9_0530h	CCB 1 Class 2 Context Register Word 12 (C1C2CTXR12)	32	RW	0000_0000h
9_0534h	CCB 1 Class 2 Context Register Word 13 (C1C2CTXR13)	32	RW	0000_0000h
9_0538h	CCB 1 Class 2 Context Register Word 14 (C1C2CTXR14)	32	RW	0000_0000h
9_053Ch	CCB 1 Class 2 Context Register Word 15 (C1C2CTXR15)	32	RW	0000_0000h
9_0540h	CCB 1 Class 2 Context Register Word 16 (C1C2CTXR16)	32	RW	0000_0000h
9_0544h	CCB 1 Class 2 Context Register Word 17 (C1C2CTXR17)	32	RW	0000_0000h
9_0600h	CCB 1 Class 2 Key Register Word 0 (C1C2KEYR0)	32	RW	0000_0000h
9_0604h	CCB 1 Class 2 Key Register Word 1 (C1C2KEYR1)	32	RW	0000_0000h
9_0608h	CCB 1 Class 2 Key Register Word 2 (C1C2KEYR2)	32	RW	0000_0000h
9_060Ch	CCB 1 Class 2 Key Register Word 3 (C1C2KEYR3)	32	RW	0000_0000h
9_0610h	CCB 1 Class 2 Key Register Word 4 (C1C2KEYR4)	32	RW	0000_0000h
9_0614h	CCB 1 Class 2 Key Register Word 5 (C1C2KEYR5)	32	RW	0000_0000h
9_0618h	CCB 1 Class 2 Key Register Word 6 (C1C2KEYR6)	32	RW	0000_0000h
9_061Ch	CCB 1 Class 2 Key Register Word 7 (C1C2KEYR7)	32	RW	0000_0000h
9_0620h	CCB 1 Class 2 Key Register Word 8 (C1C2KEYR8)	32	RW	0000_0000h
9_0624h	CCB 1 Class 2 Key Register Word 9 (C1C2KEYR9)	32	RW	0000_0000h
9_0628h	CCB 1 Class 2 Key Register Word 10 (C1C2KEYR10)	32	RW	0000_0000h
9_062Ch	CCB 1 Class 2 Key Register Word 11 (C1C2KEYR11)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
9_0630h	CCB 1 Class 2 Key Register Word 12 (C1C2KEYR12)	32	RW	0000_0000h
9_0634h	CCB 1 Class 2 Key Register Word 13 (C1C2KEYR13)	32	RW	0000_0000h
9_0638h	CCB 1 Class 2 Key Register Word 14 (C1C2KEYR14)	32	RW	0000_0000h
9_063Ch	CCB 1 Class 2 Key Register Word 15 (C1C2KEYR15)	32	RW	0000_0000h
9_0640h	CCB 1 Class 2 Key Register Word 16 (C1C2KEYR16)	32	RW	0000_0000h
9_0644h	CCB 1 Class 2 Key Register Word 17 (C1C2KEYR17)	32	RW	0000_0000h
9_0648h	CCB 1 Class 2 Key Register Word 18 (C1C2KEYR18)	32	RW	0000_0000h
9_064Ch	CCB 1 Class 2 Key Register Word 19 (C1C2KEYR19)	32	RW	0000_0000h
9_0650h	CCB 1 Class 2 Key Register Word 20 (C1C2KEYR20)	32	RW	0000_0000h
9_0654h	CCB 1 Class 2 Key Register Word 21 (C1C2KEYR21)	32	RW	0000_0000h
9_0658h	CCB 1 Class 2 Key Register Word 22 (C1C2KEYR22)	32	RW	0000_0000h
9_065Ch	CCB 1 Class 2 Key Register Word 23 (C1C2KEYR23)	32	RW	0000_0000h
9_0660h	CCB 1 Class 2 Key Register Word 24 (C1C2KEYR24)	32	RW	0000_0000h
9_0664h	CCB 1 Class 2 Key Register Word 25 (C1C2KEYR25)	32	RW	0000_0000h
9_0668h	CCB 1 Class 2 Key Register Word 26 (C1C2KEYR26)	32	RW	0000_0000h
9_066Ch	CCB 1 Class 2 Key Register Word 27 (C1C2KEYR27)	32	RW	0000_0000h
9_0670h	CCB 1 Class 2 Key Register Word 28 (C1C2KEYR28)	32	RW	0000_0000h
9_0674h	CCB 1 Class 2 Key Register Word 29 (C1C2KEYR29)	32	RW	0000_0000h
9_0678h	CCB 1 Class 2 Key Register Word 30 (C1C2KEYR30)	32	RW	0000_0000h
9_067Ch	CCB 1 Class 2 Key Register Word 31 (C1C2KEYR31)	32	RW	0000_0000h
9_07C0h	CCB 1 FIFO Status (C1FIFOSTA)	32	RO	0000_0000h
9_07D0h	CCB 1 iNformation FIFO When STYPE Is Not 10 (C1NFIFO)	32	WO	0000_0000h
9_07D0h	CCB 1 iNformation FIFO When STYPE Is 10 (C1NFIFO_2)	32	WO	0000_0000h
9_07E0h	CCB 1 Input Data FIFO (C1IFIFO)	32	WO	0000_0000h
9_07F0h	CCB 1 Output Data FIFO (C1OFIFO)	64	RO	0000_0000_00 00_0000h
9_0800h	DECO1 Job Queue Control Register, most-significant half (D1JQCR_MS)	32	RW	0000_0000h
9_0804h	DECO1 Job Queue Control Register, least-significant half (D1JQCR_LS)	32	RO	0000_0000h
9_0808h	DECO1 Descriptor Address Register (D1DAR)	64	RO	0000_0000_00 00_0000h
9_0810h	DECO1 Operation Status Register, most-significant half (D1OPSTA_MS)	32	RO	0000_0000h
9_0814h	DECO1 Operation Status Register, least-significant half (D1OPSTA_LS)	32	RO	0000_0000h
9_0818h	DECO1 Checksum Register (D1CKSUMR)	32	RW	0000_0000h
9_0820h	DECO1 Control and Output ICID Status Register (D1COICIDSR)	32	RO	0000_0000h
9_0824h	DECO1 SDID and Input ICID Status Register (D1SIICIDSR)	32	RO	0000_0000h
9_0840h	DECO1 Math Register 0_MS (D1MTH0_MS)	32	RW	0000_0000h
9_0844h	DECO1 Math Register 0_LS (D1MTH0_LS)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
9_0848h	DECO1 Math Register 1_MS (D1MTH1_MS)	32	RW	0000_0000h
9_084Ch	DECO1 Math Register 1_LS (D1MTH1_LS)	32	RW	0000_0000h
9_0850h	DECO1 Math Register 2_MS (D1MTH2_MS)	32	RW	0000_0000h
9_0854h	DECO1 Math Register 2_LS (D1MTH2_LS)	32	RW	0000_0000h
9_0858h	DECO1 Math Register 3_MS (D1MTH3_MS)	32	RW	0000_0000h
9_085Ch	DECO1 Math Register 3_LS (D1MTH3_LS)	32	RW	0000_0000h
9_0860h	DECO1 Math Register 4_MS (D1MTH4_MS)	32	RW	0000_0000h
9_0864h	DECO1 Math Register 4_LS (D1MTH4_LS)	32	RW	0000_0000h
9_0868h	DECO1 Math Register 5_MS (D1MTH5_MS)	32	RW	0000_0000h
9_086Ch	DECO1 Math Register 5_LS (D1MTH5_LS)	32	RW	0000_0000h
9_0870h	DECO1 Math Register 6_MS (D1MTH6_MS)	32	RW	0000_0000h
9_0874h	DECO1 Math Register 6_LS (D1MTH6_LS)	32	RW	0000_0000h
9_0878h	DECO1 Math Register 7_MS (D1MTH7_MS)	32	RW	0000_0000h
9_087Ch	DECO1 Math Register 7_LS (D1MTH7_LS)	32	RW	0000_0000h
9_0880h	DECO1 Gather Table Register 0 (D1GTR0)	128	RW	See description.
9_0890h	DECO1 Gather Table Register 1 (D1GTR1)	128	RW	See description.
9_08A0h	DECO1 Gather Table Register 2 (D1GTR2)	128	RW	See description.
9_08B0h	DECO1 Gather Table Register 3 (D1GTR3)	128	RW	See description.
9_0900h	DECO1 Scatter Table Register 0 (D1STR0)	128	RW	See description.
9_0910h	DECO1 Scatter Table Register 1 (D1STR1)	128	RW	See description.
9_0920h	DECO1 Scatter Table Register 2 (D1STR2)	128	RW	See description.
9_0930h	DECO1 Scatter Table Register 3 (D1STR3)	128	RW	See description.
9_0A00h	DECO1 Descriptor Buffer Word 0 (D1DESB0)	32	RW	0000_0000h
9_0A04h	DECO1 Descriptor Buffer Word 1 (D1DESB1)	32	RW	0000_0000h
9_0A08h	DECO1 Descriptor Buffer Word 2 (D1DESB2)	32	RW	0000_0000h
9_0A0Ch	DECO1 Descriptor Buffer Word 3 (D1DESB3)	32	RW	0000_0000h
9_0A10h	DECO1 Descriptor Buffer Word 4 (D1DESB4)	32	RW	0000_0000h
9_0A14h	DECO1 Descriptor Buffer Word 5 (D1DESB5)	32	RW	0000_0000h
9_0A18h	DECO1 Descriptor Buffer Word 6 (D1DESB6)	32	RW	0000_0000h
9_0A1Ch	DECO1 Descriptor Buffer Word 7 (D1DESB7)	32	RW	0000_0000h
9_0A20h	DECO1 Descriptor Buffer Word 8 (D1DESB8)	32	RW	0000_0000h
9_0A24h	DECO1 Descriptor Buffer Word 9 (D1DESB9)	32	RW	0000_0000h
9_0A28h	DECO1 Descriptor Buffer Word 10 (D1DESB10)	32	RW	0000_0000h
9_0A2Ch	DECO1 Descriptor Buffer Word 11 (D1DESB11)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
9_0A30h	DECO1 Descriptor Buffer Word 12 (D1DESB12)	32	RW	0000_0000h
9_0A34h	DECO1 Descriptor Buffer Word 13 (D1DESB13)	32	RW	0000_0000h
9_0A38h	DECO1 Descriptor Buffer Word 14 (D1DESB14)	32	RW	0000_0000h
9_0A3Ch	DECO1 Descriptor Buffer Word 15 (D1DESB15)	32	RW	0000_0000h
9_0A40h	DECO1 Descriptor Buffer Word 16 (D1DESB16)	32	RW	0000_0000h
9_0A44h	DECO1 Descriptor Buffer Word 17 (D1DESB17)	32	RW	0000_0000h
9_0A48h	DECO1 Descriptor Buffer Word 18 (D1DESB18)	32	RW	0000_0000h
9_0A4Ch	DECO1 Descriptor Buffer Word 19 (D1DESB19)	32	RW	0000_0000h
9_0A50h	DECO1 Descriptor Buffer Word 20 (D1DESB20)	32	RW	0000_0000h
9_0A54h	DECO1 Descriptor Buffer Word 21 (D1DESB21)	32	RW	0000_0000h
9_0A58h	DECO1 Descriptor Buffer Word 22 (D1DESB22)	32	RW	0000_0000h
9_0A5Ch	DECO1 Descriptor Buffer Word 23 (D1DESB23)	32	RW	0000_0000h
9_0A60h	DECO1 Descriptor Buffer Word 24 (D1DESB24)	32	RW	0000_0000h
9_0A64h	DECO1 Descriptor Buffer Word 25 (D1DESB25)	32	RW	0000_0000h
9_0A68h	DECO1 Descriptor Buffer Word 26 (D1DESB26)	32	RW	0000_0000h
9_0A6Ch	DECO1 Descriptor Buffer Word 27 (D1DESB27)	32	RW	0000_0000h
9_0A70h	DECO1 Descriptor Buffer Word 28 (D1DESB28)	32	RW	0000_0000h
9_0A74h	DECO1 Descriptor Buffer Word 29 (D1DESB29)	32	RW	0000_0000h
9_0A78h	DECO1 Descriptor Buffer Word 30 (D1DESB30)	32	RW	0000_0000h
9_0A7Ch	DECO1 Descriptor Buffer Word 31 (D1DESB31)	32	RW	0000_0000h
9_0A80h	DECO1 Descriptor Buffer Word 32 (D1DESB32)	32	RW	0000_0000h
9_0A84h	DECO1 Descriptor Buffer Word 33 (D1DESB33)	32	RW	0000_0000h
9_0A88h	DECO1 Descriptor Buffer Word 34 (D1DESB34)	32	RW	0000_0000h
9_0A8Ch	DECO1 Descriptor Buffer Word 35 (D1DESB35)	32	RW	0000_0000h
9_0A90h	DECO1 Descriptor Buffer Word 36 (D1DESB36)	32	RW	0000_0000h
9_0A94h	DECO1 Descriptor Buffer Word 37 (D1DESB37)	32	RW	0000_0000h
9_0A98h	DECO1 Descriptor Buffer Word 38 (D1DESB38)	32	RW	0000_0000h
9_0A9Ch	DECO1 Descriptor Buffer Word 39 (D1DESB39)	32	RW	0000_0000h
9_0AA0h	DECO1 Descriptor Buffer Word 40 (D1DESB40)	32	RW	0000_0000h
9_0AA4h	DECO1 Descriptor Buffer Word 41 (D1DESB41)	32	RW	0000_0000h
9_0AA8h	DECO1 Descriptor Buffer Word 42 (D1DESB42)	32	RW	0000_0000h
9_0AACh	DECO1 Descriptor Buffer Word 43 (D1DESB43)	32	RW	0000_0000h
9_0AB0h	DECO1 Descriptor Buffer Word 44 (D1DESB44)	32	RW	0000_0000h
9_0AB4h	DECO1 Descriptor Buffer Word 45 (D1DESB45)	32	RW	0000_0000h
9_0AB8h	DECO1 Descriptor Buffer Word 46 (D1DESB46)	32	RW	0000_0000h
9_0ABCh	DECO1 Descriptor Buffer Word 47 (D1DESB47)	32	RW	0000_0000h
9_0AC0h	DECO1 Descriptor Buffer Word 48 (D1DESB48)	32	RW	0000_0000h
9_0AC4h	DECO1 Descriptor Buffer Word 49 (D1DESB49)	32	RW	0000_0000h
9_0AC8h	DECO1 Descriptor Buffer Word 50 (D1DESB50)	32	RW	0000_0000h
9_0ACCh	DECO1 Descriptor Buffer Word 51 (D1DESB51)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
9_0AD0h	DECO1 Descriptor Buffer Word 52 (D1DESB52)	32	RW	0000_0000h
9_0AD4h	DECO1 Descriptor Buffer Word 53 (D1DESB53)	32	RW	0000_0000h
9_0AD8h	DECO1 Descriptor Buffer Word 54 (D1DESB54)	32	RW	0000_0000h
9_0ADCCh	DECO1 Descriptor Buffer Word 55 (D1DESB55)	32	RW	0000_0000h
9_0AE0h	DECO1 Descriptor Buffer Word 56 (D1DESB56)	32	RW	0000_0000h
9_0AE4h	DECO1 Descriptor Buffer Word 57 (D1DESB57)	32	RW	0000_0000h
9_0AE8h	DECO1 Descriptor Buffer Word 58 (D1DESB58)	32	RW	0000_0000h
9_0AECCh	DECO1 Descriptor Buffer Word 59 (D1DESB59)	32	RW	0000_0000h
9_0AF0h	DECO1 Descriptor Buffer Word 60 (D1DESB60)	32	RW	0000_0000h
9_0AF4h	DECO1 Descriptor Buffer Word 61 (D1DESB61)	32	RW	0000_0000h
9_0AF8h	DECO1 Descriptor Buffer Word 62 (D1DESB62)	32	RW	0000_0000h
9_0AFCh	DECO1 Descriptor Buffer Word 63 (D1DESB63)	32	RW	0000_0000h
9_0E00h	DECO1 Debug Job (D1DJR)	32	RO	0000_0000h
9_0E04h	DECO1 Debug DECO (D1DDR)	32	RO	0000_0000h
9_0E08h	DECO1 Debug Job Pointer (D1DJP)	64	RO	0000_0000_00 00_0000h
9_0E10h	DECO1 Debug Shared Pointer (D1SDP)	64	RO	0000_0000_00 00_0000h
9_0E18h	DECO1 Debug_ICID, most-significant half (D1DIR_MS)	32	RO	0000_0000h
9_0E1Ch	DECO1 Debug ICID, least-significant half (D1DIR_LS)	32	RO	0000_0000h
9_0E20h	Sequence Output Length Register (SOL1)	32	RW	0000_0000h
9_0E24h	Variable Sequence Output Length Register (VSOL1)	32	RW	0000_0000h
9_0E28h	Sequence Input Length Register (SIL1)	32	RW	0000_0000h
9_0E2Ch	Variable Sequence Input Length Register (VSIL1)	32	RW	0000_0000h
9_0E30h	Protocol Override Register (D1POVRD)	32	RW	0000_0000h
9_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL1)	32	RW	0000_0000h
9_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL1)	32	RW	0000_0000h
9_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_00 00_0000h
9_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_00 00_0000h
9_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_00 00_0000h
9_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_00 00_0000h
9_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_00 00_0000h
9_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_00 00_0000h
9_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
9_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
9_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
9_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
9_0FACh (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
9_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_00 00_0000h
9_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
9_0FCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
9_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
9_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
9_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
9_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
9_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
9_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
9_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
9_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
9_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
A_0004h	CCB 2 Class 1 Mode Register Format for Non-Public Key Algorithms (C2C1MR_NPK)	32	RW	0000_0000h
A_0004h	CCB 2 Class 1 Mode Register Format for Public Key Algorithms (C2C1MR_PK)	32	RW	0000_0000h
A_0004h	CCB 2 Class 1 Mode Register Format for RNG4 (C2C1MR_RNG)	32	RW	0000_0000h
A_000Ch	CCB 2 Class 1 Key Size Register (C2C1KSR)	32	RW	0000_0000h
A_0010h	CCB 2 Class 1 Data Size Register (C2C1DSR)	64	RW	0000_0000_00 00_0000h
A_001Ch	CCB 2 Class 1 ICV Size Register (C2C1ICVSR)	32	RW	0000_0000h
A_0034h	CCB 2 CHA Control Register (C2CCTRL)	32	WO	0000_0000h
A_003Ch	CCB 2 Interrupt Control Register (C2ICTL)	32	W1C	0000_0000h
A_0044h	CCB 2 Clear Written Register (C2CWR)	32	WO	0000_0000h
A_0048h	CCB 2 Status and Error Register, most-significant half (C2CSTA_MS)	32	RO	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
A_004Ch	CCB 2 Status and Error Register, least-significant half (C2CSTA_LS)	32	RO	0000_0000h
A_005Ch	CCB 2 AAD Size Register (C2AADSZR)	32	RW	0000_0000h
A_0064h	Class 1 IV Size Register (C2C1IVSZR)	32	RW	0000_0000h
A_0084h	PKHA A Size Register (C2PKASZR)	32	RW	0000_0000h
A_008Ch	PKHA B Size Register (C2PKBSZR)	32	RW	0000_0000h
A_0094h	PKHA N Size Register (C2PKNSZR)	32	RW	0000_0000h
A_009Ch	PKHA E Size Register (C2PKESZR)	32	RW	0000_0000h
A_0100h	CCB 2 Class 1 Context Register Word 0 (C2C1CTXR0)	32	RW	0000_0000h
A_0104h	CCB 2 Class 1 Context Register Word 1 (C2C1CTXR1)	32	RW	0000_0000h
A_0108h	CCB 2 Class 1 Context Register Word 2 (C2C1CTXR2)	32	RW	0000_0000h
A_010Ch	CCB 2 Class 1 Context Register Word 3 (C2C1CTXR3)	32	RW	0000_0000h
A_0110h	CCB 2 Class 1 Context Register Word 4 (C2C1CTXR4)	32	RW	0000_0000h
A_0114h	CCB 2 Class 1 Context Register Word 5 (C2C1CTXR5)	32	RW	0000_0000h
A_0118h	CCB 2 Class 1 Context Register Word 6 (C2C1CTXR6)	32	RW	0000_0000h
A_011Ch	CCB 2 Class 1 Context Register Word 7 (C2C1CTXR7)	32	RW	0000_0000h
A_0120h	CCB 2 Class 1 Context Register Word 8 (C2C1CTXR8)	32	RW	0000_0000h
A_0124h	CCB 2 Class 1 Context Register Word 9 (C2C1CTXR9)	32	RW	0000_0000h
A_0128h	CCB 2 Class 1 Context Register Word 10 (C2C1CTXR10)	32	RW	0000_0000h
A_012Ch	CCB 2 Class 1 Context Register Word 11 (C2C1CTXR11)	32	RW	0000_0000h
A_0130h	CCB 2 Class 1 Context Register Word 12 (C2C1CTXR12)	32	RW	0000_0000h
A_0134h	CCB 2 Class 1 Context Register Word 13 (C2C1CTXR13)	32	RW	0000_0000h
A_0138h	CCB 2 Class 1 Context Register Word 14 (C2C1CTXR14)	32	RW	0000_0000h
A_013Ch	CCB 2 Class 1 Context Register Word 15 (C2C1CTXR15)	32	RW	0000_0000h
A_0200h	CCB 2 Class 1 Key Registers Word 0 (C2C1KR0)	32	RW	0000_0000h
A_0204h	CCB 2 Class 1 Key Registers Word 1 (C2C1KR1)	32	RW	0000_0000h
A_0208h	CCB 2 Class 1 Key Registers Word 2 (C2C1KR2)	32	RW	0000_0000h
A_020Ch	CCB 2 Class 1 Key Registers Word 3 (C2C1KR3)	32	RW	0000_0000h
A_0210h	CCB 2 Class 1 Key Registers Word 4 (C2C1KR4)	32	RW	0000_0000h
A_0214h	CCB 2 Class 1 Key Registers Word 5 (C2C1KR5)	32	RW	0000_0000h
A_0218h	CCB 2 Class 1 Key Registers Word 6 (C2C1KR6)	32	RW	0000_0000h
A_021Ch	CCB 2 Class 1 Key Registers Word 7 (C2C1KR7)	32	RW	0000_0000h
A_0404h	CCB 2 Class 2 Mode Register (C2C2MR)	32	RW	0000_0000h
A_040Ch	CCB 2 Class 2 Key Size Register (C2C2KSR)	32	RW	0000_0000h
A_0410h	CCB 2 Class 2 Data Size Register (C2C2DSR)	64	RW	0000_0000_00 00_0000h
A_041Ch	CCB 2 Class 2 ICV Size Register (C2C2ICVSZR)	32	RW	0000_0000h
A_0500h	CCB 2 Class 2 Context Register Word 0 (C2C2CTXR0)	32	RW	0000_0000h
A_0504h	CCB 2 Class 2 Context Register Word 1 (C2C2CTXR1)	32	RW	0000_0000h
A_0508h	CCB 2 Class 2 Context Register Word 2 (C2C2CTXR2)	32	RW	0000_0000h
A_050Ch	CCB 2 Class 2 Context Register Word 3 (C2C2CTXR3)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
A_0510h	CCB 2 Class 2 Context Register Word 4 (C2C2CTXR4)	32	RW	0000_0000h
A_0514h	CCB 2 Class 2 Context Register Word 5 (C2C2CTXR5)	32	RW	0000_0000h
A_0518h	CCB 2 Class 2 Context Register Word 6 (C2C2CTXR6)	32	RW	0000_0000h
A_051Ch	CCB 2 Class 2 Context Register Word 7 (C2C2CTXR7)	32	RW	0000_0000h
A_0520h	CCB 2 Class 2 Context Register Word 8 (C2C2CTXR8)	32	RW	0000_0000h
A_0524h	CCB 2 Class 2 Context Register Word 9 (C2C2CTXR9)	32	RW	0000_0000h
A_0528h	CCB 2 Class 2 Context Register Word 10 (C2C2CTXR10)	32	RW	0000_0000h
A_052Ch	CCB 2 Class 2 Context Register Word 11 (C2C2CTXR11)	32	RW	0000_0000h
A_0530h	CCB 2 Class 2 Context Register Word 12 (C2C2CTXR12)	32	RW	0000_0000h
A_0534h	CCB 2 Class 2 Context Register Word 13 (C2C2CTXR13)	32	RW	0000_0000h
A_0538h	CCB 2 Class 2 Context Register Word 14 (C2C2CTXR14)	32	RW	0000_0000h
A_053Ch	CCB 2 Class 2 Context Register Word 15 (C2C2CTXR15)	32	RW	0000_0000h
A_0540h	CCB 2 Class 2 Context Register Word 16 (C2C2CTXR16)	32	RW	0000_0000h
A_0544h	CCB 2 Class 2 Context Register Word 17 (C2C2CTXR17)	32	RW	0000_0000h
A_0600h	CCB 2 Class 2 Key Register Word 0 (C2C2KEYR0)	32	RW	0000_0000h
A_0604h	CCB 2 Class 2 Key Register Word 1 (C2C2KEYR1)	32	RW	0000_0000h
A_0608h	CCB 2 Class 2 Key Register Word 2 (C2C2KEYR2)	32	RW	0000_0000h
A_060Ch	CCB 2 Class 2 Key Register Word 3 (C2C2KEYR3)	32	RW	0000_0000h
A_0610h	CCB 2 Class 2 Key Register Word 4 (C2C2KEYR4)	32	RW	0000_0000h
A_0614h	CCB 2 Class 2 Key Register Word 5 (C2C2KEYR5)	32	RW	0000_0000h
A_0618h	CCB 2 Class 2 Key Register Word 6 (C2C2KEYR6)	32	RW	0000_0000h
A_061Ch	CCB 2 Class 2 Key Register Word 7 (C2C2KEYR7)	32	RW	0000_0000h
A_0620h	CCB 2 Class 2 Key Register Word 8 (C2C2KEYR8)	32	RW	0000_0000h
A_0624h	CCB 2 Class 2 Key Register Word 9 (C2C2KEYR9)	32	RW	0000_0000h
A_0628h	CCB 2 Class 2 Key Register Word 10 (C2C2KEYR10)	32	RW	0000_0000h
A_062Ch	CCB 2 Class 2 Key Register Word 11 (C2C2KEYR11)	32	RW	0000_0000h
A_0630h	CCB 2 Class 2 Key Register Word 12 (C2C2KEYR12)	32	RW	0000_0000h
A_0634h	CCB 2 Class 2 Key Register Word 13 (C2C2KEYR13)	32	RW	0000_0000h
A_0638h	CCB 2 Class 2 Key Register Word 14 (C2C2KEYR14)	32	RW	0000_0000h
A_063Ch	CCB 2 Class 2 Key Register Word 15 (C2C2KEYR15)	32	RW	0000_0000h
A_0640h	CCB 2 Class 2 Key Register Word 16 (C2C2KEYR16)	32	RW	0000_0000h
A_0644h	CCB 2 Class 2 Key Register Word 17 (C2C2KEYR17)	32	RW	0000_0000h
A_0648h	CCB 2 Class 2 Key Register Word 18 (C2C2KEYR18)	32	RW	0000_0000h
A_064Ch	CCB 2 Class 2 Key Register Word 19 (C2C2KEYR19)	32	RW	0000_0000h
A_0650h	CCB 2 Class 2 Key Register Word 20 (C2C2KEYR20)	32	RW	0000_0000h
A_0654h	CCB 2 Class 2 Key Register Word 21 (C2C2KEYR21)	32	RW	0000_0000h
A_0658h	CCB 2 Class 2 Key Register Word 22 (C2C2KEYR22)	32	RW	0000_0000h
A_065Ch	CCB 2 Class 2 Key Register Word 23 (C2C2KEYR23)	32	RW	0000_0000h
A_0660h	CCB 2 Class 2 Key Register Word 24 (C2C2KEYR24)	32	RW	0000_0000h
A_0664h	CCB 2 Class 2 Key Register Word 25 (C2C2KEYR25)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
A_0668h	CCB 2 Class 2 Key Register Word 26 (C2C2KEYR26)	32	RW	0000_0000h
A_066Ch	CCB 2 Class 2 Key Register Word 27 (C2C2KEYR27)	32	RW	0000_0000h
A_0670h	CCB 2 Class 2 Key Register Word 28 (C2C2KEYR28)	32	RW	0000_0000h
A_0674h	CCB 2 Class 2 Key Register Word 29 (C2C2KEYR29)	32	RW	0000_0000h
A_0678h	CCB 2 Class 2 Key Register Word 30 (C2C2KEYR30)	32	RW	0000_0000h
A_067Ch	CCB 2 Class 2 Key Register Word 31 (C2C2KEYR31)	32	RW	0000_0000h
A_07C0h	CCB 2 FIFO Status (C2FIFOSTA)	32	RO	0000_0000h
A_07D0h	CCB 2 iNformation FIFO When STYPE Is Not 10 (C2NFIFO)	32	WO	0000_0000h
A_07D0h	CCB 2 iNformation FIFO When STYPE Is 10 (C2NFIFO_2)	32	WO	0000_0000h
A_07E0h	CCB 2 Input Data FIFO (C2IFIFO)	32	WO	0000_0000h
A_07F0h	CCB 2 Output Data FIFO (C2OFIFO)	64	RO	0000_0000_00 00_0000h
A_0800h	DECO2 Job Queue Control Register, most-significant half (D2JQCR_MS)	32	RW	0000_0000h
A_0804h	DECO2 Job Queue Control Register, least-significant half (D2JQCR_LS)	32	RO	0000_0000h
A_0808h	DECO2 Descriptor Address Register (D2DAR)	64	RO	0000_0000_00 00_0000h
A_0810h	DECO2 Operation Status Register, most-significant half (D2OPSTA_MS)	32	RO	0000_0000h
A_0814h	DECO2 Operation Status Register, least-significant half (D2OPSTA_LS)	32	RO	0000_0000h
A_0818h	DECO2 Checksum Register (D2CKSUMR)	32	RW	0000_0000h
A_0820h	DECO2 Control and Output ICID Status Register (D2COICIDSR)	32	RO	0000_0000h
A_0824h	DECO2 SDID and Input ICID Status Register (D2SIICIDSR)	32	RO	0000_0000h
A_0840h	DECO2 Math Register 0_MS (D2MTH0_MS)	32	RW	0000_0000h
A_0844h	DECO2 Math Register 0_LS (D2MTH0_LS)	32	RW	0000_0000h
A_0848h	DECO2 Math Register 1_MS (D2MTH1_MS)	32	RW	0000_0000h
A_084Ch	DECO2 Math Register 1_LS (D2MTH1_LS)	32	RW	0000_0000h
A_0850h	DECO2 Math Register 2_MS (D2MTH2_MS)	32	RW	0000_0000h
A_0854h	DECO2 Math Register 2_LS (D2MTH2_LS)	32	RW	0000_0000h
A_0858h	DECO2 Math Register 3_MS (D2MTH3_MS)	32	RW	0000_0000h
A_085Ch	DECO2 Math Register 3_LS (D2MTH3_LS)	32	RW	0000_0000h
A_0860h	DECO2 Math Register 4_MS (D2MTH4_MS)	32	RW	0000_0000h
A_0864h	DECO2 Math Register 4_LS (D2MTH4_LS)	32	RW	0000_0000h
A_0868h	DECO2 Math Register 5_MS (D2MTH5_MS)	32	RW	0000_0000h
A_086Ch	DECO2 Math Register 5_LS (D2MTH5_LS)	32	RW	0000_0000h
A_0870h	DECO2 Math Register 6_MS (D2MTH6_MS)	32	RW	0000_0000h
A_0874h	DECO2 Math Register 6_LS (D2MTH6_LS)	32	RW	0000_0000h
A_0878h	DECO2 Math Register 7_MS (D2MTH7_MS)	32	RW	0000_0000h
A_087Ch	DECO2 Math Register 7_LS (D2MTH7_LS)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
A_0880h	DECO2 Gather Table Register 0 (D2GTR0)	128	RW	See description.
A_0890h	DECO2 Gather Table Register 1 (D2GTR1)	128	RW	See description.
A_08A0h	DECO2 Gather Table Register 2 (D2GTR2)	128	RW	See description.
A_08B0h	DECO2 Gather Table Register 3 (D2GTR3)	128	RW	See description.
A_0900h	DECO2 Scatter Table Register 0 (D2STR0)	128	RW	See description.
A_0910h	DECO2 Scatter Table Register 1 (D2STR1)	128	RW	See description.
A_0920h	DECO2 Scatter Table Register 2 (D2STR2)	128	RW	See description.
A_0930h	DECO2 Scatter Table Register 3 (D2STR3)	128	RW	See description.
A_0A00h	DECO2 Descriptor Buffer Word 0 (D2DESB0)	32	RW	0000_0000h
A_0A04h	DECO2 Descriptor Buffer Word 1 (D2DESB1)	32	RW	0000_0000h
A_0A08h	DECO2 Descriptor Buffer Word 2 (D2DESB2)	32	RW	0000_0000h
A_0A0Ch	DECO2 Descriptor Buffer Word 3 (D2DESB3)	32	RW	0000_0000h
A_0A10h	DECO2 Descriptor Buffer Word 4 (D2DESB4)	32	RW	0000_0000h
A_0A14h	DECO2 Descriptor Buffer Word 5 (D2DESB5)	32	RW	0000_0000h
A_0A18h	DECO2 Descriptor Buffer Word 6 (D2DESB6)	32	RW	0000_0000h
A_0A1Ch	DECO2 Descriptor Buffer Word 7 (D2DESB7)	32	RW	0000_0000h
A_0A20h	DECO2 Descriptor Buffer Word 8 (D2DESB8)	32	RW	0000_0000h
A_0A24h	DECO2 Descriptor Buffer Word 9 (D2DESB9)	32	RW	0000_0000h
A_0A28h	DECO2 Descriptor Buffer Word 10 (D2DESB10)	32	RW	0000_0000h
A_0A2Ch	DECO2 Descriptor Buffer Word 11 (D2DESB11)	32	RW	0000_0000h
A_0A30h	DECO2 Descriptor Buffer Word 12 (D2DESB12)	32	RW	0000_0000h
A_0A34h	DECO2 Descriptor Buffer Word 13 (D2DESB13)	32	RW	0000_0000h
A_0A38h	DECO2 Descriptor Buffer Word 14 (D2DESB14)	32	RW	0000_0000h
A_0A3Ch	DECO2 Descriptor Buffer Word 15 (D2DESB15)	32	RW	0000_0000h
A_0A40h	DECO2 Descriptor Buffer Word 16 (D2DESB16)	32	RW	0000_0000h
A_0A44h	DECO2 Descriptor Buffer Word 17 (D2DESB17)	32	RW	0000_0000h
A_0A48h	DECO2 Descriptor Buffer Word 18 (D2DESB18)	32	RW	0000_0000h
A_0A4Ch	DECO2 Descriptor Buffer Word 19 (D2DESB19)	32	RW	0000_0000h
A_0A50h	DECO2 Descriptor Buffer Word 20 (D2DESB20)	32	RW	0000_0000h
A_0A54h	DECO2 Descriptor Buffer Word 21 (D2DESB21)	32	RW	0000_0000h
A_0A58h	DECO2 Descriptor Buffer Word 22 (D2DESB22)	32	RW	0000_0000h
A_0A5Ch	DECO2 Descriptor Buffer Word 23 (D2DESB23)	32	RW	0000_0000h
A_0A60h	DECO2 Descriptor Buffer Word 24 (D2DESB24)	32	RW	0000_0000h
A_0A64h	DECO2 Descriptor Buffer Word 25 (D2DESB25)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
A_0A68h	DECO2 Descriptor Buffer Word 26 (D2DESB26)	32	RW	0000_0000h
A_0A6Ch	DECO2 Descriptor Buffer Word 27 (D2DESB27)	32	RW	0000_0000h
A_0A70h	DECO2 Descriptor Buffer Word 28 (D2DESB28)	32	RW	0000_0000h
A_0A74h	DECO2 Descriptor Buffer Word 29 (D2DESB29)	32	RW	0000_0000h
A_0A78h	DECO2 Descriptor Buffer Word 30 (D2DESB30)	32	RW	0000_0000h
A_0A7Ch	DECO2 Descriptor Buffer Word 31 (D2DESB31)	32	RW	0000_0000h
A_0A80h	DECO2 Descriptor Buffer Word 32 (D2DESB32)	32	RW	0000_0000h
A_0A84h	DECO2 Descriptor Buffer Word 33 (D2DESB33)	32	RW	0000_0000h
A_0A88h	DECO2 Descriptor Buffer Word 34 (D2DESB34)	32	RW	0000_0000h
A_0A8Ch	DECO2 Descriptor Buffer Word 35 (D2DESB35)	32	RW	0000_0000h
A_0A90h	DECO2 Descriptor Buffer Word 36 (D2DESB36)	32	RW	0000_0000h
A_0A94h	DECO2 Descriptor Buffer Word 37 (D2DESB37)	32	RW	0000_0000h
A_0A98h	DECO2 Descriptor Buffer Word 38 (D2DESB38)	32	RW	0000_0000h
A_0A9Ch	DECO2 Descriptor Buffer Word 39 (D2DESB39)	32	RW	0000_0000h
A_0AA0h	DECO2 Descriptor Buffer Word 40 (D2DESB40)	32	RW	0000_0000h
A_0AA4h	DECO2 Descriptor Buffer Word 41 (D2DESB41)	32	RW	0000_0000h
A_0AA8h	DECO2 Descriptor Buffer Word 42 (D2DESB42)	32	RW	0000_0000h
A_0AACh	DECO2 Descriptor Buffer Word 43 (D2DESB43)	32	RW	0000_0000h
A_0AB0h	DECO2 Descriptor Buffer Word 44 (D2DESB44)	32	RW	0000_0000h
A_0AB4h	DECO2 Descriptor Buffer Word 45 (D2DESB45)	32	RW	0000_0000h
A_0AB8h	DECO2 Descriptor Buffer Word 46 (D2DESB46)	32	RW	0000_0000h
A_0ABCh	DECO2 Descriptor Buffer Word 47 (D2DESB47)	32	RW	0000_0000h
A_0AC0h	DECO2 Descriptor Buffer Word 48 (D2DESB48)	32	RW	0000_0000h
A_0AC4h	DECO2 Descriptor Buffer Word 49 (D2DESB49)	32	RW	0000_0000h
A_0AC8h	DECO2 Descriptor Buffer Word 50 (D2DESB50)	32	RW	0000_0000h
A_0ACCh	DECO2 Descriptor Buffer Word 51 (D2DESB51)	32	RW	0000_0000h
A_0AD0h	DECO2 Descriptor Buffer Word 52 (D2DESB52)	32	RW	0000_0000h
A_0AD4h	DECO2 Descriptor Buffer Word 53 (D2DESB53)	32	RW	0000_0000h
A_0AD8h	DECO2 Descriptor Buffer Word 54 (D2DESB54)	32	RW	0000_0000h
A_0ADCh	DECO2 Descriptor Buffer Word 55 (D2DESB55)	32	RW	0000_0000h
A_0AE0h	DECO2 Descriptor Buffer Word 56 (D2DESB56)	32	RW	0000_0000h
A_0AE4h	DECO2 Descriptor Buffer Word 57 (D2DESB57)	32	RW	0000_0000h
A_0AE8h	DECO2 Descriptor Buffer Word 58 (D2DESB58)	32	RW	0000_0000h
A_0AECh	DECO2 Descriptor Buffer Word 59 (D2DESB59)	32	RW	0000_0000h
A_0AF0h	DECO2 Descriptor Buffer Word 60 (D2DESB60)	32	RW	0000_0000h
A_0AF4h	DECO2 Descriptor Buffer Word 61 (D2DESB61)	32	RW	0000_0000h
A_0AF8h	DECO2 Descriptor Buffer Word 62 (D2DESB62)	32	RW	0000_0000h
A_0AFCh	DECO2 Descriptor Buffer Word 63 (D2DESB63)	32	RW	0000_0000h
A_0E00h	DECO2 Debug Job (D2DJR)	32	RO	0000_0000h
A_0E04h	DECO2 Debug DECO (D2DDR)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
A_0E08h	DECO2 Debug Job Pointer (D2DJP)	64	RO	0000_0000_00_0000h
A_0E10h	DECO2 Debug Shared Pointer (D2SDP)	64	RO	0000_0000_00_0000h
A_0E18h	DECO2 Debug_ICID, most-significant half (D2DIR_MS)	32	RO	0000_0000h
A_0E1Ch	DECO2 Debug ICID, least-significant half (D2DIR_LS)	32	RO	0000_0000h
A_0E20h	Sequence Output Length Register (SOL2)	32	RW	0000_0000h
A_0E24h	Variable Sequence Output Length Register (VSOL2)	32	RW	0000_0000h
A_0E28h	Sequence Input Length Register (SIL2)	32	RW	0000_0000h
A_0E2Ch	Variable Sequence Input Length Register (VSIL2)	32	RW	0000_0000h
A_0E30h	Protocol Override Register (D2POVRD)	32	RW	0000_0000h
A_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL2)	32	RW	0000_0000h
A_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL2)	32	RW	0000_0000h
A_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_00_0000h
A_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_00_0000h
A_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_00_0000h
A_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_00_0000h
A_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_00_0000h
A_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_00_0000h
A_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_00_0000h
A_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
A_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
A_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
A_0FACH (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
A_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_00_0000h
A_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
A_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
A_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
A_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
A_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
A_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
A_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
A_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
A_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
A_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
A_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
B_0004h	CCB 3 Class 1 Mode Register Format for Non-Public Key Algorithms (C3C1MR_NPK)	32	RW	0000_0000h
B_0004h	CCB 3 Class 1 Mode Register Format for Public Key Algorithms (C3C1MR_PK)	32	RW	0000_0000h
B_0004h	CCB 3 Class 1 Mode Register Format for RNG4 (C3C1MR_RNG)	32	RW	0000_0000h
B_000Ch	CCB 3 Class 1 Key Size Register (C3C1KSR)	32	RW	0000_0000h
B_0010h	CCB 3 Class 1 Data Size Register (C3C1DSR)	64	RW	0000_0000_00 00_0000h
B_001Ch	CCB 3 Class 1 ICV Size Register (C3C1ICVSR)	32	RW	0000_0000h
B_0034h	CCB 3 CHA Control Register (C3CCTRL)	32	WO	0000_0000h
B_003Ch	CCB 3 Interrupt Control Register (C3ICTL)	32	W1C	0000_0000h
B_0044h	CCB 3 Clear Written Register (C3CWR)	32	WO	0000_0000h
B_0048h	CCB 3 Status and Error Register, most-significant half (C3CSTA_MS)	32	RO	0000_0000h
B_004Ch	CCB 3 Status and Error Register, least-significant half (C3CSTA_LS)	32	RO	0000_0000h
B_005Ch	CCB 3 AAD Size Register (C3AADSZR)	32	RW	0000_0000h
B_0064h	Class 1 IV Size Register (C3C1IVSZR)	32	RW	0000_0000h
B_0084h	PKHA A Size Register (C3PKASZR)	32	RW	0000_0000h
B_008Ch	PKHA B Size Register (C3PKBSZR)	32	RW	0000_0000h
B_0094h	PKHA N Size Register (C3PKNSZR)	32	RW	0000_0000h
B_009Ch	PKHA E Size Register (C3PKESZR)	32	RW	0000_0000h
B_0100h	CCB 3 Class 1 Context Register Word 0 (C3C1CTXR0)	32	RW	0000_0000h
B_0104h	CCB 3 Class 1 Context Register Word 1 (C3C1CTXR1)	32	RW	0000_0000h
B_0108h	CCB 3 Class 1 Context Register Word 2 (C3C1CTXR2)	32	RW	0000_0000h
B_010Ch	CCB 3 Class 1 Context Register Word 3 (C3C1CTXR3)	32	RW	0000_0000h
B_0110h	CCB 3 Class 1 Context Register Word 4 (C3C1CTXR4)	32	RW	0000_0000h
B_0114h	CCB 3 Class 1 Context Register Word 5 (C3C1CTXR5)	32	RW	0000_0000h
B_0118h	CCB 3 Class 1 Context Register Word 6 (C3C1CTXR6)	32	RW	0000_0000h
B_011Ch	CCB 3 Class 1 Context Register Word 7 (C3C1CTXR7)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
B_0120h	CCB 3 Class 1 Context Register Word 8 (C3C1CTXR8)	32	RW	0000_0000h
B_0124h	CCB 3 Class 1 Context Register Word 9 (C3C1CTXR9)	32	RW	0000_0000h
B_0128h	CCB 3 Class 1 Context Register Word 10 (C3C1CTXR10)	32	RW	0000_0000h
B_012Ch	CCB 3 Class 1 Context Register Word 11 (C3C1CTXR11)	32	RW	0000_0000h
B_0130h	CCB 3 Class 1 Context Register Word 12 (C3C1CTXR12)	32	RW	0000_0000h
B_0134h	CCB 3 Class 1 Context Register Word 13 (C3C1CTXR13)	32	RW	0000_0000h
B_0138h	CCB 3 Class 1 Context Register Word 14 (C3C1CTXR14)	32	RW	0000_0000h
B_013Ch	CCB 3 Class 1 Context Register Word 15 (C3C1CTXR15)	32	RW	0000_0000h
B_0200h	CCB 3 Class 1 Key Registers Word 0 (C3C1KR0)	32	RW	0000_0000h
B_0204h	CCB 3 Class 1 Key Registers Word 1 (C3C1KR1)	32	RW	0000_0000h
B_0208h	CCB 3 Class 1 Key Registers Word 2 (C3C1KR2)	32	RW	0000_0000h
B_020Ch	CCB 3 Class 1 Key Registers Word 3 (C3C1KR3)	32	RW	0000_0000h
B_0210h	CCB 3 Class 1 Key Registers Word 4 (C3C1KR4)	32	RW	0000_0000h
B_0214h	CCB 3 Class 1 Key Registers Word 5 (C3C1KR5)	32	RW	0000_0000h
B_0218h	CCB 3 Class 1 Key Registers Word 6 (C3C1KR6)	32	RW	0000_0000h
B_021Ch	CCB 3 Class 1 Key Registers Word 7 (C3C1KR7)	32	RW	0000_0000h
B_0404h	CCB 3 Class 2 Mode Register (C3C2MR)	32	RW	0000_0000h
B_040Ch	CCB 3 Class 2 Key Size Register (C3C2KSR)	32	RW	0000_0000h
B_0410h	CCB 3 Class 2 Data Size Register (C3C2DSR)	64	RW	0000_0000_00 00_0000h
B_041Ch	CCB 3 Class 2 ICV Size Register (C3C2ICVSZR)	32	RW	0000_0000h
B_0500h	CCB 3 Class 2 Context Register Word 0 (C3C2CTXR0)	32	RW	0000_0000h
B_0504h	CCB 3 Class 2 Context Register Word 1 (C3C2CTXR1)	32	RW	0000_0000h
B_0508h	CCB 3 Class 2 Context Register Word 2 (C3C2CTXR2)	32	RW	0000_0000h
B_050Ch	CCB 3 Class 2 Context Register Word 3 (C3C2CTXR3)	32	RW	0000_0000h
B_0510h	CCB 3 Class 2 Context Register Word 4 (C3C2CTXR4)	32	RW	0000_0000h
B_0514h	CCB 3 Class 2 Context Register Word 5 (C3C2CTXR5)	32	RW	0000_0000h
B_0518h	CCB 3 Class 2 Context Register Word 6 (C3C2CTXR6)	32	RW	0000_0000h
B_051Ch	CCB 3 Class 2 Context Register Word 7 (C3C2CTXR7)	32	RW	0000_0000h
B_0520h	CCB 3 Class 2 Context Register Word 8 (C3C2CTXR8)	32	RW	0000_0000h
B_0524h	CCB 3 Class 2 Context Register Word 9 (C3C2CTXR9)	32	RW	0000_0000h
B_0528h	CCB 3 Class 2 Context Register Word 10 (C3C2CTXR10)	32	RW	0000_0000h
B_052Ch	CCB 3 Class 2 Context Register Word 11 (C3C2CTXR11)	32	RW	0000_0000h
B_0530h	CCB 3 Class 2 Context Register Word 12 (C3C2CTXR12)	32	RW	0000_0000h
B_0534h	CCB 3 Class 2 Context Register Word 13 (C3C2CTXR13)	32	RW	0000_0000h
B_0538h	CCB 3 Class 2 Context Register Word 14 (C3C2CTXR14)	32	RW	0000_0000h
B_053Ch	CCB 3 Class 2 Context Register Word 15 (C3C2CTXR15)	32	RW	0000_0000h
B_0540h	CCB 3 Class 2 Context Register Word 16 (C3C2CTXR16)	32	RW	0000_0000h
B_0544h	CCB 3 Class 2 Context Register Word 17 (C3C2CTXR17)	32	RW	0000_0000h
B_0600h	CCB 3 Class 2 Key Register Word 0 (C3C2KEYR0)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
B_0604h	CCB 3 Class 2 Key Register Word 1 (C3C2KEYR1)	32	RW	0000_0000h
B_0608h	CCB 3 Class 2 Key Register Word 2 (C3C2KEYR2)	32	RW	0000_0000h
B_060Ch	CCB 3 Class 2 Key Register Word 3 (C3C2KEYR3)	32	RW	0000_0000h
B_0610h	CCB 3 Class 2 Key Register Word 4 (C3C2KEYR4)	32	RW	0000_0000h
B_0614h	CCB 3 Class 2 Key Register Word 5 (C3C2KEYR5)	32	RW	0000_0000h
B_0618h	CCB 3 Class 2 Key Register Word 6 (C3C2KEYR6)	32	RW	0000_0000h
B_061Ch	CCB 3 Class 2 Key Register Word 7 (C3C2KEYR7)	32	RW	0000_0000h
B_0620h	CCB 3 Class 2 Key Register Word 8 (C3C2KEYR8)	32	RW	0000_0000h
B_0624h	CCB 3 Class 2 Key Register Word 9 (C3C2KEYR9)	32	RW	0000_0000h
B_0628h	CCB 3 Class 2 Key Register Word 10 (C3C2KEYR10)	32	RW	0000_0000h
B_062Ch	CCB 3 Class 2 Key Register Word 11 (C3C2KEYR11)	32	RW	0000_0000h
B_0630h	CCB 3 Class 2 Key Register Word 12 (C3C2KEYR12)	32	RW	0000_0000h
B_0634h	CCB 3 Class 2 Key Register Word 13 (C3C2KEYR13)	32	RW	0000_0000h
B_0638h	CCB 3 Class 2 Key Register Word 14 (C3C2KEYR14)	32	RW	0000_0000h
B_063Ch	CCB 3 Class 2 Key Register Word 15 (C3C2KEYR15)	32	RW	0000_0000h
B_0640h	CCB 3 Class 2 Key Register Word 16 (C3C2KEYR16)	32	RW	0000_0000h
B_0644h	CCB 3 Class 2 Key Register Word 17 (C3C2KEYR17)	32	RW	0000_0000h
B_0648h	CCB 3 Class 2 Key Register Word 18 (C3C2KEYR18)	32	RW	0000_0000h
B_064Ch	CCB 3 Class 2 Key Register Word 19 (C3C2KEYR19)	32	RW	0000_0000h
B_0650h	CCB 3 Class 2 Key Register Word 20 (C3C2KEYR20)	32	RW	0000_0000h
B_0654h	CCB 3 Class 2 Key Register Word 21 (C3C2KEYR21)	32	RW	0000_0000h
B_0658h	CCB 3 Class 2 Key Register Word 22 (C3C2KEYR22)	32	RW	0000_0000h
B_065Ch	CCB 3 Class 2 Key Register Word 23 (C3C2KEYR23)	32	RW	0000_0000h
B_0660h	CCB 3 Class 2 Key Register Word 24 (C3C2KEYR24)	32	RW	0000_0000h
B_0664h	CCB 3 Class 2 Key Register Word 25 (C3C2KEYR25)	32	RW	0000_0000h
B_0668h	CCB 3 Class 2 Key Register Word 26 (C3C2KEYR26)	32	RW	0000_0000h
B_066Ch	CCB 3 Class 2 Key Register Word 27 (C3C2KEYR27)	32	RW	0000_0000h
B_0670h	CCB 3 Class 2 Key Register Word 28 (C3C2KEYR28)	32	RW	0000_0000h
B_0674h	CCB 3 Class 2 Key Register Word 29 (C3C2KEYR29)	32	RW	0000_0000h
B_0678h	CCB 3 Class 2 Key Register Word 30 (C3C2KEYR30)	32	RW	0000_0000h
B_067Ch	CCB 3 Class 2 Key Register Word 31 (C3C2KEYR31)	32	RW	0000_0000h
B_07C0h	CCB 3 FIFO Status (C3FIFOSTA)	32	RO	0000_0000h
B_07D0h	CCB 3 iNformation FIFO When STYPE Is Not 10 (C3NFIFO)	32	WO	0000_0000h
B_07D0h	CCB 3 iNformation FIFO When STYPE Is 10 (C3NFIFO_2)	32	WO	0000_0000h
B_07E0h	CCB 3 Input Data FIFO (C3IFIFO)	32	WO	0000_0000h
B_07F0h	CCB 3 Output Data FIFO (C3OFIFO)	64	RO	0000_0000_00 00_0000h
B_0800h	DECO3 Job Queue Control Register, most-significant half (D3JQCR_MS)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
B_0804h	DECO3 Job Queue Control Register, least-significant half (D3JQCR_LS)	32	RO	0000_0000h
B_0808h	DECO3 Descriptor Address Register (D3DAR)	64	RO	0000_0000_0000_0000h
B_0810h	DECO3 Operation Status Register, most-significant half (D3OPSTA_MS)	32	RO	0000_0000h
B_0814h	DECO3 Operation Status Register, least-significant half (D3OPSTA_LS)	32	RO	0000_0000h
B_0818h	DECO3 Checksum Register (D3CKSUMR)	32	RW	0000_0000h
B_0820h	DECO3 Control and Output ICID Status Register (D3COICIDSR)	32	RO	0000_0000h
B_0824h	DECO3 SDID and Input ICID Status Register (D3SIICIDSR)	32	RO	0000_0000h
B_0840h	DECO3 Math Register 0_MS (D3MTH0_MS)	32	RW	0000_0000h
B_0844h	DECO3 Math Register 0_LS (D3MTH0_LS)	32	RW	0000_0000h
B_0848h	DECO3 Math Register 1_MS (D3MTH1_MS)	32	RW	0000_0000h
B_084Ch	DECO3 Math Register 1_LS (D3MTH1_LS)	32	RW	0000_0000h
B_0850h	DECO3 Math Register 2_MS (D3MTH2_MS)	32	RW	0000_0000h
B_0854h	DECO3 Math Register 2_LS (D3MTH2_LS)	32	RW	0000_0000h
B_0858h	DECO3 Math Register 3_MS (D3MTH3_MS)	32	RW	0000_0000h
B_085Ch	DECO3 Math Register 3_LS (D3MTH3_LS)	32	RW	0000_0000h
B_0860h	DECO3 Math Register 4_MS (D3MTH4_MS)	32	RW	0000_0000h
B_0864h	DECO3 Math Register 4_LS (D3MTH4_LS)	32	RW	0000_0000h
B_0868h	DECO3 Math Register 5_MS (D3MTH5_MS)	32	RW	0000_0000h
B_086Ch	DECO3 Math Register 5_LS (D3MTH5_LS)	32	RW	0000_0000h
B_0870h	DECO3 Math Register 6_MS (D3MTH6_MS)	32	RW	0000_0000h
B_0874h	DECO3 Math Register 6_LS (D3MTH6_LS)	32	RW	0000_0000h
B_0878h	DECO3 Math Register 7_MS (D3MTH7_MS)	32	RW	0000_0000h
B_087Ch	DECO3 Math Register 7_LS (D3MTH7_LS)	32	RW	0000_0000h
B_0880h	DECO3 Gather Table Register 0 (D3GTR0)	128	RW	See description.
B_0890h	DECO3 Gather Table Register 1 (D3GTR1)	128	RW	See description.
B_08A0h	DECO3 Gather Table Register 2 (D3GTR2)	128	RW	See description.
B_08B0h	DECO3 Gather Table Register 3 (D3GTR3)	128	RW	See description.
B_0900h	DECO3 Scatter Table Register 0 (D3STR0)	128	RW	See description.
B_0910h	DECO3 Scatter Table Register 1 (D3STR1)	128	RW	See description.
B_0920h	DECO3 Scatter Table Register 2 (D3STR2)	128	RW	See description.
B_0930h	DECO3 Scatter Table Register 3 (D3STR3)	128	RW	See description.

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
B_0A00h	DECO3 Descriptor Buffer Word 0 (D3DESB0)	32	RW	0000_0000h
B_0A04h	DECO3 Descriptor Buffer Word 1 (D3DESB1)	32	RW	0000_0000h
B_0A08h	DECO3 Descriptor Buffer Word 2 (D3DESB2)	32	RW	0000_0000h
B_0A0Ch	DECO3 Descriptor Buffer Word 3 (D3DESB3)	32	RW	0000_0000h
B_0A10h	DECO3 Descriptor Buffer Word 4 (D3DESB4)	32	RW	0000_0000h
B_0A14h	DECO3 Descriptor Buffer Word 5 (D3DESB5)	32	RW	0000_0000h
B_0A18h	DECO3 Descriptor Buffer Word 6 (D3DESB6)	32	RW	0000_0000h
B_0A1Ch	DECO3 Descriptor Buffer Word 7 (D3DESB7)	32	RW	0000_0000h
B_0A20h	DECO3 Descriptor Buffer Word 8 (D3DESB8)	32	RW	0000_0000h
B_0A24h	DECO3 Descriptor Buffer Word 9 (D3DESB9)	32	RW	0000_0000h
B_0A28h	DECO3 Descriptor Buffer Word 10 (D3DESB10)	32	RW	0000_0000h
B_0A2Ch	DECO3 Descriptor Buffer Word 11 (D3DESB11)	32	RW	0000_0000h
B_0A30h	DECO3 Descriptor Buffer Word 12 (D3DESB12)	32	RW	0000_0000h
B_0A34h	DECO3 Descriptor Buffer Word 13 (D3DESB13)	32	RW	0000_0000h
B_0A38h	DECO3 Descriptor Buffer Word 14 (D3DESB14)	32	RW	0000_0000h
B_0A3Ch	DECO3 Descriptor Buffer Word 15 (D3DESB15)	32	RW	0000_0000h
B_0A40h	DECO3 Descriptor Buffer Word 16 (D3DESB16)	32	RW	0000_0000h
B_0A44h	DECO3 Descriptor Buffer Word 17 (D3DESB17)	32	RW	0000_0000h
B_0A48h	DECO3 Descriptor Buffer Word 18 (D3DESB18)	32	RW	0000_0000h
B_0A4Ch	DECO3 Descriptor Buffer Word 19 (D3DESB19)	32	RW	0000_0000h
B_0A50h	DECO3 Descriptor Buffer Word 20 (D3DESB20)	32	RW	0000_0000h
B_0A54h	DECO3 Descriptor Buffer Word 21 (D3DESB21)	32	RW	0000_0000h
B_0A58h	DECO3 Descriptor Buffer Word 22 (D3DESB22)	32	RW	0000_0000h
B_0A5Ch	DECO3 Descriptor Buffer Word 23 (D3DESB23)	32	RW	0000_0000h
B_0A60h	DECO3 Descriptor Buffer Word 24 (D3DESB24)	32	RW	0000_0000h
B_0A64h	DECO3 Descriptor Buffer Word 25 (D3DESB25)	32	RW	0000_0000h
B_0A68h	DECO3 Descriptor Buffer Word 26 (D3DESB26)	32	RW	0000_0000h
B_0A6Ch	DECO3 Descriptor Buffer Word 27 (D3DESB27)	32	RW	0000_0000h
B_0A70h	DECO3 Descriptor Buffer Word 28 (D3DESB28)	32	RW	0000_0000h
B_0A74h	DECO3 Descriptor Buffer Word 29 (D3DESB29)	32	RW	0000_0000h
B_0A78h	DECO3 Descriptor Buffer Word 30 (D3DESB30)	32	RW	0000_0000h
B_0A7Ch	DECO3 Descriptor Buffer Word 31 (D3DESB31)	32	RW	0000_0000h
B_0A80h	DECO3 Descriptor Buffer Word 32 (D3DESB32)	32	RW	0000_0000h
B_0A84h	DECO3 Descriptor Buffer Word 33 (D3DESB33)	32	RW	0000_0000h
B_0A88h	DECO3 Descriptor Buffer Word 34 (D3DESB34)	32	RW	0000_0000h
B_0A8Ch	DECO3 Descriptor Buffer Word 35 (D3DESB35)	32	RW	0000_0000h
B_0A90h	DECO3 Descriptor Buffer Word 36 (D3DESB36)	32	RW	0000_0000h
B_0A94h	DECO3 Descriptor Buffer Word 37 (D3DESB37)	32	RW	0000_0000h
B_0A98h	DECO3 Descriptor Buffer Word 38 (D3DESB38)	32	RW	0000_0000h
B_0A9Ch	DECO3 Descriptor Buffer Word 39 (D3DESB39)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
B_0AA0h	DECO3 Descriptor Buffer Word 40 (D3DESB40)	32	RW	0000_0000h
B_0AA4h	DECO3 Descriptor Buffer Word 41 (D3DESB41)	32	RW	0000_0000h
B_0AA8h	DECO3 Descriptor Buffer Word 42 (D3DESB42)	32	RW	0000_0000h
B_0AACh	DECO3 Descriptor Buffer Word 43 (D3DESB43)	32	RW	0000_0000h
B_0AB0h	DECO3 Descriptor Buffer Word 44 (D3DESB44)	32	RW	0000_0000h
B_0AB4h	DECO3 Descriptor Buffer Word 45 (D3DESB45)	32	RW	0000_0000h
B_0AB8h	DECO3 Descriptor Buffer Word 46 (D3DESB46)	32	RW	0000_0000h
B_0ABCCh	DECO3 Descriptor Buffer Word 47 (D3DESB47)	32	RW	0000_0000h
B_0AC0h	DECO3 Descriptor Buffer Word 48 (D3DESB48)	32	RW	0000_0000h
B_0AC4h	DECO3 Descriptor Buffer Word 49 (D3DESB49)	32	RW	0000_0000h
B_0AC8h	DECO3 Descriptor Buffer Word 50 (D3DESB50)	32	RW	0000_0000h
B_0ACCh	DECO3 Descriptor Buffer Word 51 (D3DESB51)	32	RW	0000_0000h
B_0AD0h	DECO3 Descriptor Buffer Word 52 (D3DESB52)	32	RW	0000_0000h
B_0AD4h	DECO3 Descriptor Buffer Word 53 (D3DESB53)	32	RW	0000_0000h
B_0AD8h	DECO3 Descriptor Buffer Word 54 (D3DESB54)	32	RW	0000_0000h
B_0ADCh	DECO3 Descriptor Buffer Word 55 (D3DESB55)	32	RW	0000_0000h
B_0AE0h	DECO3 Descriptor Buffer Word 56 (D3DESB56)	32	RW	0000_0000h
B_0AE4h	DECO3 Descriptor Buffer Word 57 (D3DESB57)	32	RW	0000_0000h
B_0AE8h	DECO3 Descriptor Buffer Word 58 (D3DESB58)	32	RW	0000_0000h
B_0AECh	DECO3 Descriptor Buffer Word 59 (D3DESB59)	32	RW	0000_0000h
B_0AF0h	DECO3 Descriptor Buffer Word 60 (D3DESB60)	32	RW	0000_0000h
B_0AF4h	DECO3 Descriptor Buffer Word 61 (D3DESB61)	32	RW	0000_0000h
B_0AF8h	DECO3 Descriptor Buffer Word 62 (D3DESB62)	32	RW	0000_0000h
B_0AFCh	DECO3 Descriptor Buffer Word 63 (D3DESB63)	32	RW	0000_0000h
B_0E00h	DECO3 Debug Job (D3DJR)	32	RO	0000_0000h
B_0E04h	DECO3 Debug DECO (D3DDR)	32	RO	0000_0000h
B_0E08h	DECO3 Debug Job Pointer (D3DJP)	64	RO	0000_0000_0000_0000h
B_0E10h	DECO3 Debug Shared Pointer (D3SDP)	64	RO	0000_0000_0000_0000h
B_0E18h	DECO3 Debug_ICID, most-significant half (D3DIR_MS)	32	RO	0000_0000h
B_0E1Ch	DECO3 Debug ICID, least-significant half (D3DIR_LS)	32	RO	0000_0000h
B_0E20h	Sequence Output Length Register (SOL3)	32	RW	0000_0000h
B_0E24h	Variable Sequence Output Length Register (VSOL3)	32	RW	0000_0000h
B_0E28h	Sequence Input Length Register (SIL3)	32	RW	0000_0000h
B_0E2Ch	Variable Sequence Input Length Register (VSIL3)	32	RW	0000_0000h
B_0E30h	Protocol Override Register (D3POVRD)	32	RW	0000_0000h
B_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL3)	32	RW	0000_0000h
B_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL3)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
B_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_0000_0000h
B_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_0000_0000h
B_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_0000_0000h
B_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_0000_0000h
B_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_0000_0000h
B_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_0000_0000h
B_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_0000_0000h
B_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
B_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
B_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
B_0FACH (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
B_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_0000_0000h
B_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
B_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
B_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
B_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
B_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
B_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
B_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
B_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
B_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
B_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
B_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
C_0004h	CCB 4 Class 1 Mode Register Format for Non-Public Key Algorithms (C4C1MR_NPK)	32	RW	0000_0000h
C_0004h	CCB 4 Class 1 Mode Register Format for Public Key Algorithms (C4C1MR_PK)	32	RW	0000_0000h
C_0004h	CCB 4 Class 1 Mode Register Format for RNG4 (C4C1MR_RNG)	32	RW	0000_0000h
C_000Ch	CCB 4 Class 1 Key Size Register (C4C1KSR)	32	RW	0000_0000h
C_0010h	CCB 4 Class 1 Data Size Register (C4C1DSR)	64	RW	0000_0000_0000_0000h
C_001Ch	CCB 4 Class 1 ICV Size Register (C4C1ICVSR)	32	RW	0000_0000h
C_0034h	CCB 4 CHA Control Register (C4CCTRL)	32	WO	0000_0000h
C_003Ch	CCB 4 Interrupt Control Register (C4ICTL)	32	W1C	0000_0000h
C_0044h	CCB 4 Clear Written Register (C4CWR)	32	WO	0000_0000h
C_0048h	CCB 4 Status and Error Register, most-significant half (C4CSTA_MS)	32	RO	0000_0000h
C_004Ch	CCB 4 Status and Error Register, least-significant half (C4CSTA_LS)	32	RO	0000_0000h
C_005Ch	CCB 4 AAD Size Register (C4AADSZR)	32	RW	0000_0000h
C_0064h	Class 1 IV Size Register (C4C1IVSZR)	32	RW	0000_0000h
C_0084h	PKHA A Size Register (C4PKASZR)	32	RW	0000_0000h
C_008Ch	PKHA B Size Register (C4PKBSZR)	32	RW	0000_0000h
C_0094h	PKHA N Size Register (C4PKNSZR)	32	RW	0000_0000h
C_009Ch	PKHA E Size Register (C4PKESZR)	32	RW	0000_0000h
C_0100h	CCB 4 Class 1 Context Register Word 0 (C4C1CTXR0)	32	RW	0000_0000h
C_0104h	CCB 4 Class 1 Context Register Word 1 (C4C1CTXR1)	32	RW	0000_0000h
C_0108h	CCB 4 Class 1 Context Register Word 2 (C4C1CTXR2)	32	RW	0000_0000h
C_010Ch	CCB 4 Class 1 Context Register Word 3 (C4C1CTXR3)	32	RW	0000_0000h
C_0110h	CCB 4 Class 1 Context Register Word 4 (C4C1CTXR4)	32	RW	0000_0000h
C_0114h	CCB 4 Class 1 Context Register Word 5 (C4C1CTXR5)	32	RW	0000_0000h
C_0118h	CCB 4 Class 1 Context Register Word 6 (C4C1CTXR6)	32	RW	0000_0000h
C_011Ch	CCB 4 Class 1 Context Register Word 7 (C4C1CTXR7)	32	RW	0000_0000h
C_0120h	CCB 4 Class 1 Context Register Word 8 (C4C1CTXR8)	32	RW	0000_0000h
C_0124h	CCB 4 Class 1 Context Register Word 9 (C4C1CTXR9)	32	RW	0000_0000h
C_0128h	CCB 4 Class 1 Context Register Word 10 (C4C1CTXR10)	32	RW	0000_0000h
C_012Ch	CCB 4 Class 1 Context Register Word 11 (C4C1CTXR11)	32	RW	0000_0000h
C_0130h	CCB 4 Class 1 Context Register Word 12 (C4C1CTXR12)	32	RW	0000_0000h
C_0134h	CCB 4 Class 1 Context Register Word 13 (C4C1CTXR13)	32	RW	0000_0000h
C_0138h	CCB 4 Class 1 Context Register Word 14 (C4C1CTXR14)	32	RW	0000_0000h
C_013Ch	CCB 4 Class 1 Context Register Word 15 (C4C1CTXR15)	32	RW	0000_0000h
C_0200h	CCB 4 Class 1 Key Registers Word 0 (C4C1KR0)	32	RW	0000_0000h
C_0204h	CCB 4 Class 1 Key Registers Word 1 (C4C1KR1)	32	RW	0000_0000h
C_0208h	CCB 4 Class 1 Key Registers Word 2 (C4C1KR2)	32	RW	0000_0000h
C_020Ch	CCB 4 Class 1 Key Registers Word 3 (C4C1KR3)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
C_0210h	CCB 4 Class 1 Key Registers Word 4 (C4C1KR4)	32	RW	0000_0000h
C_0214h	CCB 4 Class 1 Key Registers Word 5 (C4C1KR5)	32	RW	0000_0000h
C_0218h	CCB 4 Class 1 Key Registers Word 6 (C4C1KR6)	32	RW	0000_0000h
C_021Ch	CCB 4 Class 1 Key Registers Word 7 (C4C1KR7)	32	RW	0000_0000h
C_0404h	CCB 4 Class 2 Mode Register (C4C2MR)	32	RW	0000_0000h
C_040Ch	CCB 4 Class 2 Key Size Register (C4C2KSR)	32	RW	0000_0000h
C_0410h	CCB 4 Class 2 Data Size Register (C4C2DSR)	64	RW	0000_0000_00 00_0000h
C_041Ch	CCB 4 Class 2 ICV Size Register (C4C2ICVSZR)	32	RW	0000_0000h
C_0500h	CCB 4 Class 2 Context Register Word 0 (C4C2CTXR0)	32	RW	0000_0000h
C_0504h	CCB 4 Class 2 Context Register Word 1 (C4C2CTXR1)	32	RW	0000_0000h
C_0508h	CCB 4 Class 2 Context Register Word 2 (C4C2CTXR2)	32	RW	0000_0000h
C_050Ch	CCB 4 Class 2 Context Register Word 3 (C4C2CTXR3)	32	RW	0000_0000h
C_0510h	CCB 4 Class 2 Context Register Word 4 (C4C2CTXR4)	32	RW	0000_0000h
C_0514h	CCB 4 Class 2 Context Register Word 5 (C4C2CTXR5)	32	RW	0000_0000h
C_0518h	CCB 4 Class 2 Context Register Word 6 (C4C2CTXR6)	32	RW	0000_0000h
C_051Ch	CCB 4 Class 2 Context Register Word 7 (C4C2CTXR7)	32	RW	0000_0000h
C_0520h	CCB 4 Class 2 Context Register Word 8 (C4C2CTXR8)	32	RW	0000_0000h
C_0524h	CCB 4 Class 2 Context Register Word 9 (C4C2CTXR9)	32	RW	0000_0000h
C_0528h	CCB 4 Class 2 Context Register Word 10 (C4C2CTXR10)	32	RW	0000_0000h
C_052Ch	CCB 4 Class 2 Context Register Word 11 (C4C2CTXR11)	32	RW	0000_0000h
C_0530h	CCB 4 Class 2 Context Register Word 12 (C4C2CTXR12)	32	RW	0000_0000h
C_0534h	CCB 4 Class 2 Context Register Word 13 (C4C2CTXR13)	32	RW	0000_0000h
C_0538h	CCB 4 Class 2 Context Register Word 14 (C4C2CTXR14)	32	RW	0000_0000h
C_053Ch	CCB 4 Class 2 Context Register Word 15 (C4C2CTXR15)	32	RW	0000_0000h
C_0540h	CCB 4 Class 2 Context Register Word 16 (C4C2CTXR16)	32	RW	0000_0000h
C_0544h	CCB 4 Class 2 Context Register Word 17 (C4C2CTXR17)	32	RW	0000_0000h
C_0600h	CCB 4 Class 2 Key Register Word 0 (C4C2KEYR0)	32	RW	0000_0000h
C_0604h	CCB 4 Class 2 Key Register Word 1 (C4C2KEYR1)	32	RW	0000_0000h
C_0608h	CCB 4 Class 2 Key Register Word 2 (C4C2KEYR2)	32	RW	0000_0000h
C_060Ch	CCB 4 Class 2 Key Register Word 3 (C4C2KEYR3)	32	RW	0000_0000h
C_0610h	CCB 4 Class 2 Key Register Word 4 (C4C2KEYR4)	32	RW	0000_0000h
C_0614h	CCB 4 Class 2 Key Register Word 5 (C4C2KEYR5)	32	RW	0000_0000h
C_0618h	CCB 4 Class 2 Key Register Word 6 (C4C2KEYR6)	32	RW	0000_0000h
C_061Ch	CCB 4 Class 2 Key Register Word 7 (C4C2KEYR7)	32	RW	0000_0000h
C_0620h	CCB 4 Class 2 Key Register Word 8 (C4C2KEYR8)	32	RW	0000_0000h
C_0624h	CCB 4 Class 2 Key Register Word 9 (C4C2KEYR9)	32	RW	0000_0000h
C_0628h	CCB 4 Class 2 Key Register Word 10 (C4C2KEYR10)	32	RW	0000_0000h
C_062Ch	CCB 4 Class 2 Key Register Word 11 (C4C2KEYR11)	32	RW	0000_0000h
C_0630h	CCB 4 Class 2 Key Register Word 12 (C4C2KEYR12)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
C_0634h	CCB 4 Class 2 Key Register Word 13 (C4C2KEYR13)	32	RW	0000_0000h
C_0638h	CCB 4 Class 2 Key Register Word 14 (C4C2KEYR14)	32	RW	0000_0000h
C_063Ch	CCB 4 Class 2 Key Register Word 15 (C4C2KEYR15)	32	RW	0000_0000h
C_0640h	CCB 4 Class 2 Key Register Word 16 (C4C2KEYR16)	32	RW	0000_0000h
C_0644h	CCB 4 Class 2 Key Register Word 17 (C4C2KEYR17)	32	RW	0000_0000h
C_0648h	CCB 4 Class 2 Key Register Word 18 (C4C2KEYR18)	32	RW	0000_0000h
C_064Ch	CCB 4 Class 2 Key Register Word 19 (C4C2KEYR19)	32	RW	0000_0000h
C_0650h	CCB 4 Class 2 Key Register Word 20 (C4C2KEYR20)	32	RW	0000_0000h
C_0654h	CCB 4 Class 2 Key Register Word 21 (C4C2KEYR21)	32	RW	0000_0000h
C_0658h	CCB 4 Class 2 Key Register Word 22 (C4C2KEYR22)	32	RW	0000_0000h
C_065Ch	CCB 4 Class 2 Key Register Word 23 (C4C2KEYR23)	32	RW	0000_0000h
C_0660h	CCB 4 Class 2 Key Register Word 24 (C4C2KEYR24)	32	RW	0000_0000h
C_0664h	CCB 4 Class 2 Key Register Word 25 (C4C2KEYR25)	32	RW	0000_0000h
C_0668h	CCB 4 Class 2 Key Register Word 26 (C4C2KEYR26)	32	RW	0000_0000h
C_066Ch	CCB 4 Class 2 Key Register Word 27 (C4C2KEYR27)	32	RW	0000_0000h
C_0670h	CCB 4 Class 2 Key Register Word 28 (C4C2KEYR28)	32	RW	0000_0000h
C_0674h	CCB 4 Class 2 Key Register Word 29 (C4C2KEYR29)	32	RW	0000_0000h
C_0678h	CCB 4 Class 2 Key Register Word 30 (C4C2KEYR30)	32	RW	0000_0000h
C_067Ch	CCB 4 Class 2 Key Register Word 31 (C4C2KEYR31)	32	RW	0000_0000h
C_07C0h	CCB 4 FIFO Status (C4FIFOSTA)	32	RO	0000_0000h
C_07D0h	CCB 4 iNformation FIFO When STYPE Is Not 10 (C4NFIFO)	32	WO	0000_0000h
C_07D0h	CCB 4 iNformation FIFO When STYPE Is 10 (C4NFIFO_2)	32	WO	0000_0000h
C_07E0h	CCB 4 Input Data FIFO (C4IFIFO)	32	WO	0000_0000h
C_07F0h	CCB 4 Output Data FIFO (C4OFIFO)	64	RO	0000_0000_00 00_0000h
C_0800h	DECO4 Job Queue Control Register, most-significant half (D4JQCR_ MS)	32	RW	0000_0000h
C_0804h	DECO4 Job Queue Control Register, least-significant half (D4JQCR_ LS)	32	RO	0000_0000h
C_0808h	DECO4 Descriptor Address Register (D4DAR)	64	RO	0000_0000_00 00_0000h
C_0810h	DECO4 Operation Status Register, most-significant half (D4OPSTA_ MS)	32	RO	0000_0000h
C_0814h	DECO4 Operation Status Register, least-significant half (D4OPSTA_ LS)	32	RO	0000_0000h
C_0818h	DECO4 Checksum Register (D4CKSUMR)	32	RW	0000_0000h
C_0820h	DECO4 Control and Output ICID Status Register (D4COICIDSR)	32	RO	0000_0000h
C_0824h	DECO4 SDID and Input ICID Status Register (D4SIICIDSR)	32	RO	0000_0000h
C_0840h	DECO4 Math Register 0_MS (D4MTH0_MS)	32	RW	0000_0000h
C_0844h	DECO4 Math Register 0_LS (D4MTH0_LS)	32	RW	0000_0000h
C_0848h	DECO4 Math Register 1_MS (D4MTH1_MS)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
C_084Ch	DECO4 Math Register 1_LS (D4MTH1_LS)	32	RW	0000_0000h
C_0850h	DECO4 Math Register 2_MS (D4MTH2_MS)	32	RW	0000_0000h
C_0854h	DECO4 Math Register 2_LS (D4MTH2_LS)	32	RW	0000_0000h
C_0858h	DECO4 Math Register 3_MS (D4MTH3_MS)	32	RW	0000_0000h
C_085Ch	DECO4 Math Register 3_LS (D4MTH3_LS)	32	RW	0000_0000h
C_0860h	DECO4 Math Register 4_MS (D4MTH4_MS)	32	RW	0000_0000h
C_0864h	DECO4 Math Register 4_LS (D4MTH4_LS)	32	RW	0000_0000h
C_0868h	DECO4 Math Register 5_MS (D4MTH5_MS)	32	RW	0000_0000h
C_086Ch	DECO4 Math Register 5_LS (D4MTH5_LS)	32	RW	0000_0000h
C_0870h	DECO4 Math Register 6_MS (D4MTH6_MS)	32	RW	0000_0000h
C_0874h	DECO4 Math Register 6_LS (D4MTH6_LS)	32	RW	0000_0000h
C_0878h	DECO4 Math Register 7_MS (D4MTH7_MS)	32	RW	0000_0000h
C_087Ch	DECO4 Math Register 7_LS (D4MTH7_LS)	32	RW	0000_0000h
C_0880h	DECO4 Gather Table Register 0 (D4GTR0)	128	RW	See description.
C_0890h	DECO4 Gather Table Register 1 (D4GTR1)	128	RW	See description.
C_08A0h	DECO4 Gather Table Register 2 (D4GTR2)	128	RW	See description.
C_08B0h	DECO4 Gather Table Register 3 (D4GTR3)	128	RW	See description.
C_0900h	DECO4 Scatter Table Register 0 (D4STR0)	128	RW	See description.
C_0910h	DECO4 Scatter Table Register 1 (D4STR1)	128	RW	See description.
C_0920h	DECO4 Scatter Table Register 2 (D4STR2)	128	RW	See description.
C_0930h	DECO4 Scatter Table Register 3 (D4STR3)	128	RW	See description.
C_0A00h	DECO4 Descriptor Buffer Word 0 (D4DESB0)	32	RW	0000_0000h
C_0A04h	DECO4 Descriptor Buffer Word 1 (D4DESB1)	32	RW	0000_0000h
C_0A08h	DECO4 Descriptor Buffer Word 2 (D4DESB2)	32	RW	0000_0000h
C_0A0Ch	DECO4 Descriptor Buffer Word 3 (D4DESB3)	32	RW	0000_0000h
C_0A10h	DECO4 Descriptor Buffer Word 4 (D4DESB4)	32	RW	0000_0000h
C_0A14h	DECO4 Descriptor Buffer Word 5 (D4DESB5)	32	RW	0000_0000h
C_0A18h	DECO4 Descriptor Buffer Word 6 (D4DESB6)	32	RW	0000_0000h
C_0A1Ch	DECO4 Descriptor Buffer Word 7 (D4DESB7)	32	RW	0000_0000h
C_0A20h	DECO4 Descriptor Buffer Word 8 (D4DESB8)	32	RW	0000_0000h
C_0A24h	DECO4 Descriptor Buffer Word 9 (D4DESB9)	32	RW	0000_0000h
C_0A28h	DECO4 Descriptor Buffer Word 10 (D4DESB10)	32	RW	0000_0000h
C_0A2Ch	DECO4 Descriptor Buffer Word 11 (D4DESB11)	32	RW	0000_0000h
C_0A30h	DECO4 Descriptor Buffer Word 12 (D4DESB12)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
C_0A34h	DECO4 Descriptor Buffer Word 13 (D4DESB13)	32	RW	0000_0000h
C_0A38h	DECO4 Descriptor Buffer Word 14 (D4DESB14)	32	RW	0000_0000h
C_0A3Ch	DECO4 Descriptor Buffer Word 15 (D4DESB15)	32	RW	0000_0000h
C_0A40h	DECO4 Descriptor Buffer Word 16 (D4DESB16)	32	RW	0000_0000h
C_0A44h	DECO4 Descriptor Buffer Word 17 (D4DESB17)	32	RW	0000_0000h
C_0A48h	DECO4 Descriptor Buffer Word 18 (D4DESB18)	32	RW	0000_0000h
C_0A4Ch	DECO4 Descriptor Buffer Word 19 (D4DESB19)	32	RW	0000_0000h
C_0A50h	DECO4 Descriptor Buffer Word 20 (D4DESB20)	32	RW	0000_0000h
C_0A54h	DECO4 Descriptor Buffer Word 21 (D4DESB21)	32	RW	0000_0000h
C_0A58h	DECO4 Descriptor Buffer Word 22 (D4DESB22)	32	RW	0000_0000h
C_0A5Ch	DECO4 Descriptor Buffer Word 23 (D4DESB23)	32	RW	0000_0000h
C_0A60h	DECO4 Descriptor Buffer Word 24 (D4DESB24)	32	RW	0000_0000h
C_0A64h	DECO4 Descriptor Buffer Word 25 (D4DESB25)	32	RW	0000_0000h
C_0A68h	DECO4 Descriptor Buffer Word 26 (D4DESB26)	32	RW	0000_0000h
C_0A6Ch	DECO4 Descriptor Buffer Word 27 (D4DESB27)	32	RW	0000_0000h
C_0A70h	DECO4 Descriptor Buffer Word 28 (D4DESB28)	32	RW	0000_0000h
C_0A74h	DECO4 Descriptor Buffer Word 29 (D4DESB29)	32	RW	0000_0000h
C_0A78h	DECO4 Descriptor Buffer Word 30 (D4DESB30)	32	RW	0000_0000h
C_0A7Ch	DECO4 Descriptor Buffer Word 31 (D4DESB31)	32	RW	0000_0000h
C_0A80h	DECO4 Descriptor Buffer Word 32 (D4DESB32)	32	RW	0000_0000h
C_0A84h	DECO4 Descriptor Buffer Word 33 (D4DESB33)	32	RW	0000_0000h
C_0A88h	DECO4 Descriptor Buffer Word 34 (D4DESB34)	32	RW	0000_0000h
C_0A8Ch	DECO4 Descriptor Buffer Word 35 (D4DESB35)	32	RW	0000_0000h
C_0A90h	DECO4 Descriptor Buffer Word 36 (D4DESB36)	32	RW	0000_0000h
C_0A94h	DECO4 Descriptor Buffer Word 37 (D4DESB37)	32	RW	0000_0000h
C_0A98h	DECO4 Descriptor Buffer Word 38 (D4DESB38)	32	RW	0000_0000h
C_0A9Ch	DECO4 Descriptor Buffer Word 39 (D4DESB39)	32	RW	0000_0000h
C_0AA0h	DECO4 Descriptor Buffer Word 40 (D4DESB40)	32	RW	0000_0000h
C_0AA4h	DECO4 Descriptor Buffer Word 41 (D4DESB41)	32	RW	0000_0000h
C_0AA8h	DECO4 Descriptor Buffer Word 42 (D4DESB42)	32	RW	0000_0000h
C_0AACh	DECO4 Descriptor Buffer Word 43 (D4DESB43)	32	RW	0000_0000h
C_0AB0h	DECO4 Descriptor Buffer Word 44 (D4DESB44)	32	RW	0000_0000h
C_0AB4h	DECO4 Descriptor Buffer Word 45 (D4DESB45)	32	RW	0000_0000h
C_0AB8h	DECO4 Descriptor Buffer Word 46 (D4DESB46)	32	RW	0000_0000h
C_0ABCh	DECO4 Descriptor Buffer Word 47 (D4DESB47)	32	RW	0000_0000h
C_0AC0h	DECO4 Descriptor Buffer Word 48 (D4DESB48)	32	RW	0000_0000h
C_0AC4h	DECO4 Descriptor Buffer Word 49 (D4DESB49)	32	RW	0000_0000h
C_0AC8h	DECO4 Descriptor Buffer Word 50 (D4DESB50)	32	RW	0000_0000h
C_0ACCh	DECO4 Descriptor Buffer Word 51 (D4DESB51)	32	RW	0000_0000h
C_0AD0h	DECO4 Descriptor Buffer Word 52 (D4DESB52)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
C_0AD4h	DECO4 Descriptor Buffer Word 53 (D4DESB53)	32	RW	0000_0000h
C_0AD8h	DECO4 Descriptor Buffer Word 54 (D4DESB54)	32	RW	0000_0000h
C_0ADCh	DECO4 Descriptor Buffer Word 55 (D4DESB55)	32	RW	0000_0000h
C_0AE0h	DECO4 Descriptor Buffer Word 56 (D4DESB56)	32	RW	0000_0000h
C_0AE4h	DECO4 Descriptor Buffer Word 57 (D4DESB57)	32	RW	0000_0000h
C_0AE8h	DECO4 Descriptor Buffer Word 58 (D4DESB58)	32	RW	0000_0000h
C_0AECCh	DECO4 Descriptor Buffer Word 59 (D4DESB59)	32	RW	0000_0000h
C_0AF0h	DECO4 Descriptor Buffer Word 60 (D4DESB60)	32	RW	0000_0000h
C_0AF4h	DECO4 Descriptor Buffer Word 61 (D4DESB61)	32	RW	0000_0000h
C_0AF8h	DECO4 Descriptor Buffer Word 62 (D4DESB62)	32	RW	0000_0000h
C_0AFCh	DECO4 Descriptor Buffer Word 63 (D4DESB63)	32	RW	0000_0000h
C_0E00h	DECO4 Debug Job (D4DJR)	32	RO	0000_0000h
C_0E04h	DECO4 Debug DECO (D4DDR)	32	RO	0000_0000h
C_0E08h	DECO4 Debug Job Pointer (D4DJP)	64	RO	0000_0000_00 00_0000h
C_0E10h	DECO4 Debug Shared Pointer (D4SDP)	64	RO	0000_0000_00 00_0000h
C_0E18h	DECO4 Debug_ICID, most-significant half (D4DIR_MS)	32	RO	0000_0000h
C_0E1Ch	DECO4 Debug ICID, least-significant half (D4DIR_LS)	32	RO	0000_0000h
C_0E20h	Sequence Output Length Register (SOL4)	32	RW	0000_0000h
C_0E24h	Variable Sequence Output Length Register (VSOL4)	32	RW	0000_0000h
C_0E28h	Sequence Input Length Register (SIL4)	32	RW	0000_0000h
C_0E2Ch	Variable Sequence Input Length Register (VSIL4)	32	RW	0000_0000h
C_0E30h	Protocol Override Register (D4POVRD)	32	RW	0000_0000h
C_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL4)	32	RW	0000_0000h
C_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL4)	32	RW	0000_0000h
C_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_00 00_0000h
C_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_00 00_0000h
C_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_00 00_0000h
C_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_00 00_0000h
C_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_00 00_0000h
C_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_00 00_0000h
C_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_00 00_0000h
C_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
C_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
C_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
C_0FACh (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
C_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_00 00_0000h
C_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
C_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
C_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
C_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
C_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h
C_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
C_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
C_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
C_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
C_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
C_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h
D_0004h	CCB 5 Class 1 Mode Register Format for Non-Public Key Algorithms (C5C1MR_NPK)	32	RW	0000_0000h
D_0004h	CCB 5 Class 1 Mode Register Format for Public Key Algorithms (C5C1MR_PK)	32	RW	0000_0000h
D_0004h	CCB 5 Class 1 Mode Register Format for RNG4 (C5C1MR_RNG)	32	RW	0000_0000h
D_000Ch	CCB 5 Class 1 Key Size Register (C5C1KSR)	32	RW	0000_0000h
D_0010h	CCB 5 Class 1 Data Size Register (C5C1DSR)	64	RW	0000_0000_00 00_0000h
D_001Ch	CCB 5 Class 1 ICV Size Register (C5C1ICVSR)	32	RW	0000_0000h
D_0034h	CCB 5 CHA Control Register (C5CCTRL)	32	WO	0000_0000h
D_003Ch	CCB 5 Interrupt Control Register (C5ICTL)	32	W1C	0000_0000h
D_0044h	CCB 5 Clear Written Register (C5CWR)	32	WO	0000_0000h
D_0048h	CCB 5 Status and Error Register, most-significant half (C5CSTA_MS)	32	RO	0000_0000h
D_004Ch	CCB 5 Status and Error Register, least-significant half (C5CSTA_LS)	32	RO	0000_0000h
D_005Ch	CCB 5 AAD Size Register (C5AADSZR)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
D_0064h	Class 1 IV Size Register (C5C1IVSZR)	32	RW	0000_0000h
D_0084h	PKHA A Size Register (C5PKASZR)	32	RW	0000_0000h
D_008Ch	PKHA B Size Register (C5PKBSZR)	32	RW	0000_0000h
D_0094h	PKHA N Size Register (C5PKNSZR)	32	RW	0000_0000h
D_009Ch	PKHA E Size Register (C5PKESZR)	32	RW	0000_0000h
D_0100h	CCB 5 Class 1 Context Register Word 0 (C5C1CTXR0)	32	RW	0000_0000h
D_0104h	CCB 5 Class 1 Context Register Word 1 (C5C1CTXR1)	32	RW	0000_0000h
D_0108h	CCB 5 Class 1 Context Register Word 2 (C5C1CTXR2)	32	RW	0000_0000h
D_010Ch	CCB 5 Class 1 Context Register Word 3 (C5C1CTXR3)	32	RW	0000_0000h
D_0110h	CCB 5 Class 1 Context Register Word 4 (C5C1CTXR4)	32	RW	0000_0000h
D_0114h	CCB 5 Class 1 Context Register Word 5 (C5C1CTXR5)	32	RW	0000_0000h
D_0118h	CCB 5 Class 1 Context Register Word 6 (C5C1CTXR6)	32	RW	0000_0000h
D_011Ch	CCB 5 Class 1 Context Register Word 7 (C5C1CTXR7)	32	RW	0000_0000h
D_0120h	CCB 5 Class 1 Context Register Word 8 (C5C1CTXR8)	32	RW	0000_0000h
D_0124h	CCB 5 Class 1 Context Register Word 9 (C5C1CTXR9)	32	RW	0000_0000h
D_0128h	CCB 5 Class 1 Context Register Word 10 (C5C1CTXR10)	32	RW	0000_0000h
D_012Ch	CCB 5 Class 1 Context Register Word 11 (C5C1CTXR11)	32	RW	0000_0000h
D_0130h	CCB 5 Class 1 Context Register Word 12 (C5C1CTXR12)	32	RW	0000_0000h
D_0134h	CCB 5 Class 1 Context Register Word 13 (C5C1CTXR13)	32	RW	0000_0000h
D_0138h	CCB 5 Class 1 Context Register Word 14 (C5C1CTXR14)	32	RW	0000_0000h
D_013Ch	CCB 5 Class 1 Context Register Word 15 (C5C1CTXR15)	32	RW	0000_0000h
D_0200h	CCB 5 Class 1 Key Registers Word 0 (C5C1KR0)	32	RW	0000_0000h
D_0204h	CCB 5 Class 1 Key Registers Word 1 (C5C1KR1)	32	RW	0000_0000h
D_0208h	CCB 5 Class 1 Key Registers Word 2 (C5C1KR2)	32	RW	0000_0000h
D_020Ch	CCB 5 Class 1 Key Registers Word 3 (C5C1KR3)	32	RW	0000_0000h
D_0210h	CCB 5 Class 1 Key Registers Word 4 (C5C1KR4)	32	RW	0000_0000h
D_0214h	CCB 5 Class 1 Key Registers Word 5 (C5C1KR5)	32	RW	0000_0000h
D_0218h	CCB 5 Class 1 Key Registers Word 6 (C5C1KR6)	32	RW	0000_0000h
D_021Ch	CCB 5 Class 1 Key Registers Word 7 (C5C1KR7)	32	RW	0000_0000h
D_0404h	CCB 5 Class 2 Mode Register (C5C2MR)	32	RW	0000_0000h
D_040Ch	CCB 5 Class 2 Key Size Register (C5C2KSR)	32	RW	0000_0000h
D_0410h	CCB 5 Class 2 Data Size Register (C5C2DSR)	64	RW	0000_0000_00 00_0000h
D_041Ch	CCB 5 Class 2 ICV Size Register (C5C2ICVSR)	32	RW	0000_0000h
D_0500h	CCB 5 Class 2 Context Register Word 0 (C5C2CTXR0)	32	RW	0000_0000h
D_0504h	CCB 5 Class 2 Context Register Word 1 (C5C2CTXR1)	32	RW	0000_0000h
D_0508h	CCB 5 Class 2 Context Register Word 2 (C5C2CTXR2)	32	RW	0000_0000h
D_050Ch	CCB 5 Class 2 Context Register Word 3 (C5C2CTXR3)	32	RW	0000_0000h
D_0510h	CCB 5 Class 2 Context Register Word 4 (C5C2CTXR4)	32	RW	0000_0000h
D_0514h	CCB 5 Class 2 Context Register Word 5 (C5C2CTXR5)	32	RW	0000_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
D_0518h	CCB 5 Class 2 Context Register Word 6 (C5C2CTXR6)	32	RW	0000_0000h
D_051Ch	CCB 5 Class 2 Context Register Word 7 (C5C2CTXR7)	32	RW	0000_0000h
D_0520h	CCB 5 Class 2 Context Register Word 8 (C5C2CTXR8)	32	RW	0000_0000h
D_0524h	CCB 5 Class 2 Context Register Word 9 (C5C2CTXR9)	32	RW	0000_0000h
D_0528h	CCB 5 Class 2 Context Register Word 10 (C5C2CTXR10)	32	RW	0000_0000h
D_052Ch	CCB 5 Class 2 Context Register Word 11 (C5C2CTXR11)	32	RW	0000_0000h
D_0530h	CCB 5 Class 2 Context Register Word 12 (C5C2CTXR12)	32	RW	0000_0000h
D_0534h	CCB 5 Class 2 Context Register Word 13 (C5C2CTXR13)	32	RW	0000_0000h
D_0538h	CCB 5 Class 2 Context Register Word 14 (C5C2CTXR14)	32	RW	0000_0000h
D_053Ch	CCB 5 Class 2 Context Register Word 15 (C5C2CTXR15)	32	RW	0000_0000h
D_0540h	CCB 5 Class 2 Context Register Word 16 (C5C2CTXR16)	32	RW	0000_0000h
D_0544h	CCB 5 Class 2 Context Register Word 17 (C5C2CTXR17)	32	RW	0000_0000h
D_0600h	CCB 5 Class 2 Key Register Word 0 (C5C2KEYR0)	32	RW	0000_0000h
D_0604h	CCB 5 Class 2 Key Register Word 1 (C5C2KEYR1)	32	RW	0000_0000h
D_0608h	CCB 5 Class 2 Key Register Word 2 (C5C2KEYR2)	32	RW	0000_0000h
D_060Ch	CCB 5 Class 2 Key Register Word 3 (C5C2KEYR3)	32	RW	0000_0000h
D_0610h	CCB 5 Class 2 Key Register Word 4 (C5C2KEYR4)	32	RW	0000_0000h
D_0614h	CCB 5 Class 2 Key Register Word 5 (C5C2KEYR5)	32	RW	0000_0000h
D_0618h	CCB 5 Class 2 Key Register Word 6 (C5C2KEYR6)	32	RW	0000_0000h
D_061Ch	CCB 5 Class 2 Key Register Word 7 (C5C2KEYR7)	32	RW	0000_0000h
D_0620h	CCB 5 Class 2 Key Register Word 8 (C5C2KEYR8)	32	RW	0000_0000h
D_0624h	CCB 5 Class 2 Key Register Word 9 (C5C2KEYR9)	32	RW	0000_0000h
D_0628h	CCB 5 Class 2 Key Register Word 10 (C5C2KEYR10)	32	RW	0000_0000h
D_062Ch	CCB 5 Class 2 Key Register Word 11 (C5C2KEYR11)	32	RW	0000_0000h
D_0630h	CCB 5 Class 2 Key Register Word 12 (C5C2KEYR12)	32	RW	0000_0000h
D_0634h	CCB 5 Class 2 Key Register Word 13 (C5C2KEYR13)	32	RW	0000_0000h
D_0638h	CCB 5 Class 2 Key Register Word 14 (C5C2KEYR14)	32	RW	0000_0000h
D_063Ch	CCB 5 Class 2 Key Register Word 15 (C5C2KEYR15)	32	RW	0000_0000h
D_0640h	CCB 5 Class 2 Key Register Word 16 (C5C2KEYR16)	32	RW	0000_0000h
D_0644h	CCB 5 Class 2 Key Register Word 17 (C5C2KEYR17)	32	RW	0000_0000h
D_0648h	CCB 5 Class 2 Key Register Word 18 (C5C2KEYR18)	32	RW	0000_0000h
D_064Ch	CCB 5 Class 2 Key Register Word 19 (C5C2KEYR19)	32	RW	0000_0000h
D_0650h	CCB 5 Class 2 Key Register Word 20 (C5C2KEYR20)	32	RW	0000_0000h
D_0654h	CCB 5 Class 2 Key Register Word 21 (C5C2KEYR21)	32	RW	0000_0000h
D_0658h	CCB 5 Class 2 Key Register Word 22 (C5C2KEYR22)	32	RW	0000_0000h
D_065Ch	CCB 5 Class 2 Key Register Word 23 (C5C2KEYR23)	32	RW	0000_0000h
D_0660h	CCB 5 Class 2 Key Register Word 24 (C5C2KEYR24)	32	RW	0000_0000h
D_0664h	CCB 5 Class 2 Key Register Word 25 (C5C2KEYR25)	32	RW	0000_0000h
D_0668h	CCB 5 Class 2 Key Register Word 26 (C5C2KEYR26)	32	RW	0000_0000h
D_066Ch	CCB 5 Class 2 Key Register Word 27 (C5C2KEYR27)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
D_0670h	CCB 5 Class 2 Key Register Word 28 (C5C2KEYR28)	32	RW	0000_0000h
D_0674h	CCB 5 Class 2 Key Register Word 29 (C5C2KEYR29)	32	RW	0000_0000h
D_0678h	CCB 5 Class 2 Key Register Word 30 (C5C2KEYR30)	32	RW	0000_0000h
D_067Ch	CCB 5 Class 2 Key Register Word 31 (C5C2KEYR31)	32	RW	0000_0000h
D_07C0h	CCB 5 FIFO Status (C5FIFOSTA)	32	RO	0000_0000h
D_07D0h	CCB 5 iNformation FIFO When STYPE Is Not 10 (C5NFIFO)	32	WO	0000_0000h
D_07D0h	CCB 5 iNformation FIFO When STYPE Is 10 (C5NFIFO_2)	32	WO	0000_0000h
D_07E0h	CCB 5 Input Data FIFO (C5IFIFO)	32	WO	0000_0000h
D_07F0h	CCB 5 Output Data FIFO (C5OFIFO)	64	RO	0000_0000_00 00_0000h
D_0800h	DECO5 Job Queue Control Register, most-significant half (D5JQCR_MS)	32	RW	0000_0000h
D_0804h	DECO5 Job Queue Control Register, least-significant half (D5JQCR_LS)	32	RO	0000_0000h
D_0808h	DECO5 Descriptor Address Register (D5DAR)	64	RO	0000_0000_00 00_0000h
D_0810h	DECO5 Operation Status Register, most-significant half (D5OPSTA_MS)	32	RO	0000_0000h
D_0814h	DECO5 Operation Status Register, least-significant half (D5OPSTA_LS)	32	RO	0000_0000h
D_0818h	DECO5 Checksum Register (D5CKSUMR)	32	RW	0000_0000h
D_0820h	DECO5 Control and Output ICID Status Register (D5COICIDSR)	32	RO	0000_0000h
D_0824h	DECO5 SDID and Input ICID Status Register (D5SIICIDSR)	32	RO	0000_0000h
D_0840h	DECO5 Math Register 0_MS (D5MTH0_MS)	32	RW	0000_0000h
D_0844h	DECO5 Math Register 0_LS (D5MTH0_LS)	32	RW	0000_0000h
D_0848h	DECO5 Math Register 1_MS (D5MTH1_MS)	32	RW	0000_0000h
D_084Ch	DECO5 Math Register 1_LS (D5MTH1_LS)	32	RW	0000_0000h
D_0850h	DECO5 Math Register 2_MS (D5MTH2_MS)	32	RW	0000_0000h
D_0854h	DECO5 Math Register 2_LS (D5MTH2_LS)	32	RW	0000_0000h
D_0858h	DECO5 Math Register 3_MS (D5MTH3_MS)	32	RW	0000_0000h
D_085Ch	DECO5 Math Register 3_LS (D5MTH3_LS)	32	RW	0000_0000h
D_0860h	DECO5 Math Register 4_MS (D5MTH4_MS)	32	RW	0000_0000h
D_0864h	DECO5 Math Register 4_LS (D5MTH4_LS)	32	RW	0000_0000h
D_0868h	DECO5 Math Register 5_MS (D5MTH5_MS)	32	RW	0000_0000h
D_086Ch	DECO5 Math Register 5_LS (D5MTH5_LS)	32	RW	0000_0000h
D_0870h	DECO5 Math Register 6_MS (D5MTH6_MS)	32	RW	0000_0000h
D_0874h	DECO5 Math Register 6_LS (D5MTH6_LS)	32	RW	0000_0000h
D_0878h	DECO5 Math Register 7_MS (D5MTH7_MS)	32	RW	0000_0000h
D_087Ch	DECO5 Math Register 7_LS (D5MTH7_LS)	32	RW	0000_0000h
D_0880h	DECO5 Gather Table Register 0 (D5GTR0)	128	RW	See description.

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
D_0890h	DECO5 Gather Table Register 1 (D5GTR1)	128	RW	See description.
D_08A0h	DECO5 Gather Table Register 2 (D5GTR2)	128	RW	See description.
D_08B0h	DECO5 Gather Table Register 3 (D5GTR3)	128	RW	See description.
D_0900h	DECO5 Scatter Table Register 0 (D5STR0)	128	RW	See description.
D_0910h	DECO5 Scatter Table Register 1 (D5STR1)	128	RW	See description.
D_0920h	DECO5 Scatter Table Register 2 (D5STR2)	128	RW	See description.
D_0930h	DECO5 Scatter Table Register 3 (D5STR3)	128	RW	See description.
D_0A00h	DECO5 Descriptor Buffer Word 0 (D5DESB0)	32	RW	0000_0000h
D_0A04h	DECO5 Descriptor Buffer Word 1 (D5DESB1)	32	RW	0000_0000h
D_0A08h	DECO5 Descriptor Buffer Word 2 (D5DESB2)	32	RW	0000_0000h
D_0A0Ch	DECO5 Descriptor Buffer Word 3 (D5DESB3)	32	RW	0000_0000h
D_0A10h	DECO5 Descriptor Buffer Word 4 (D5DESB4)	32	RW	0000_0000h
D_0A14h	DECO5 Descriptor Buffer Word 5 (D5DESB5)	32	RW	0000_0000h
D_0A18h	DECO5 Descriptor Buffer Word 6 (D5DESB6)	32	RW	0000_0000h
D_0A1Ch	DECO5 Descriptor Buffer Word 7 (D5DESB7)	32	RW	0000_0000h
D_0A20h	DECO5 Descriptor Buffer Word 8 (D5DESB8)	32	RW	0000_0000h
D_0A24h	DECO5 Descriptor Buffer Word 9 (D5DESB9)	32	RW	0000_0000h
D_0A28h	DECO5 Descriptor Buffer Word 10 (D5DESB10)	32	RW	0000_0000h
D_0A2Ch	DECO5 Descriptor Buffer Word 11 (D5DESB11)	32	RW	0000_0000h
D_0A30h	DECO5 Descriptor Buffer Word 12 (D5DESB12)	32	RW	0000_0000h
D_0A34h	DECO5 Descriptor Buffer Word 13 (D5DESB13)	32	RW	0000_0000h
D_0A38h	DECO5 Descriptor Buffer Word 14 (D5DESB14)	32	RW	0000_0000h
D_0A3Ch	DECO5 Descriptor Buffer Word 15 (D5DESB15)	32	RW	0000_0000h
D_0A40h	DECO5 Descriptor Buffer Word 16 (D5DESB16)	32	RW	0000_0000h
D_0A44h	DECO5 Descriptor Buffer Word 17 (D5DESB17)	32	RW	0000_0000h
D_0A48h	DECO5 Descriptor Buffer Word 18 (D5DESB18)	32	RW	0000_0000h
D_0A4Ch	DECO5 Descriptor Buffer Word 19 (D5DESB19)	32	RW	0000_0000h
D_0A50h	DECO5 Descriptor Buffer Word 20 (D5DESB20)	32	RW	0000_0000h
D_0A54h	DECO5 Descriptor Buffer Word 21 (D5DESB21)	32	RW	0000_0000h
D_0A58h	DECO5 Descriptor Buffer Word 22 (D5DESB22)	32	RW	0000_0000h
D_0A5Ch	DECO5 Descriptor Buffer Word 23 (D5DESB23)	32	RW	0000_0000h
D_0A60h	DECO5 Descriptor Buffer Word 24 (D5DESB24)	32	RW	0000_0000h
D_0A64h	DECO5 Descriptor Buffer Word 25 (D5DESB25)	32	RW	0000_0000h
D_0A68h	DECO5 Descriptor Buffer Word 26 (D5DESB26)	32	RW	0000_0000h
D_0A6Ch	DECO5 Descriptor Buffer Word 27 (D5DESB27)	32	RW	0000_0000h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
D_0A70h	DECO5 Descriptor Buffer Word 28 (D5DESB28)	32	RW	0000_0000h
D_0A74h	DECO5 Descriptor Buffer Word 29 (D5DESB29)	32	RW	0000_0000h
D_0A78h	DECO5 Descriptor Buffer Word 30 (D5DESB30)	32	RW	0000_0000h
D_0A7Ch	DECO5 Descriptor Buffer Word 31 (D5DESB31)	32	RW	0000_0000h
D_0A80h	DECO5 Descriptor Buffer Word 32 (D5DESB32)	32	RW	0000_0000h
D_0A84h	DECO5 Descriptor Buffer Word 33 (D5DESB33)	32	RW	0000_0000h
D_0A88h	DECO5 Descriptor Buffer Word 34 (D5DESB34)	32	RW	0000_0000h
D_0A8Ch	DECO5 Descriptor Buffer Word 35 (D5DESB35)	32	RW	0000_0000h
D_0A90h	DECO5 Descriptor Buffer Word 36 (D5DESB36)	32	RW	0000_0000h
D_0A94h	DECO5 Descriptor Buffer Word 37 (D5DESB37)	32	RW	0000_0000h
D_0A98h	DECO5 Descriptor Buffer Word 38 (D5DESB38)	32	RW	0000_0000h
D_0A9Ch	DECO5 Descriptor Buffer Word 39 (D5DESB39)	32	RW	0000_0000h
D_0AA0h	DECO5 Descriptor Buffer Word 40 (D5DESB40)	32	RW	0000_0000h
D_0AA4h	DECO5 Descriptor Buffer Word 41 (D5DESB41)	32	RW	0000_0000h
D_0AA8h	DECO5 Descriptor Buffer Word 42 (D5DESB42)	32	RW	0000_0000h
D_0AACh	DECO5 Descriptor Buffer Word 43 (D5DESB43)	32	RW	0000_0000h
D_0AB0h	DECO5 Descriptor Buffer Word 44 (D5DESB44)	32	RW	0000_0000h
D_0AB4h	DECO5 Descriptor Buffer Word 45 (D5DESB45)	32	RW	0000_0000h
D_0AB8h	DECO5 Descriptor Buffer Word 46 (D5DESB46)	32	RW	0000_0000h
D_0ABCh	DECO5 Descriptor Buffer Word 47 (D5DESB47)	32	RW	0000_0000h
D_0AC0h	DECO5 Descriptor Buffer Word 48 (D5DESB48)	32	RW	0000_0000h
D_0AC4h	DECO5 Descriptor Buffer Word 49 (D5DESB49)	32	RW	0000_0000h
D_0AC8h	DECO5 Descriptor Buffer Word 50 (D5DESB50)	32	RW	0000_0000h
D_0ACCh	DECO5 Descriptor Buffer Word 51 (D5DESB51)	32	RW	0000_0000h
D_0AD0h	DECO5 Descriptor Buffer Word 52 (D5DESB52)	32	RW	0000_0000h
D_0AD4h	DECO5 Descriptor Buffer Word 53 (D5DESB53)	32	RW	0000_0000h
D_0AD8h	DECO5 Descriptor Buffer Word 54 (D5DESB54)	32	RW	0000_0000h
D_0ADCh	DECO5 Descriptor Buffer Word 55 (D5DESB55)	32	RW	0000_0000h
D_0AE0h	DECO5 Descriptor Buffer Word 56 (D5DESB56)	32	RW	0000_0000h
D_0AE4h	DECO5 Descriptor Buffer Word 57 (D5DESB57)	32	RW	0000_0000h
D_0AE8h	DECO5 Descriptor Buffer Word 58 (D5DESB58)	32	RW	0000_0000h
D_0AECh	DECO5 Descriptor Buffer Word 59 (D5DESB59)	32	RW	0000_0000h
D_0AF0h	DECO5 Descriptor Buffer Word 60 (D5DESB60)	32	RW	0000_0000h
D_0AF4h	DECO5 Descriptor Buffer Word 61 (D5DESB61)	32	RW	0000_0000h
D_0AF8h	DECO5 Descriptor Buffer Word 62 (D5DESB62)	32	RW	0000_0000h
D_0AFCh	DECO5 Descriptor Buffer Word 63 (D5DESB63)	32	RW	0000_0000h
D_0E00h	DECO5 Debug Job (D5DJR)	32	RO	0000_0000h
D_0E04h	DECO5 Debug DECO (D5DDR)	32	RO	0000_0000h
D_0E08h	DECO5 Debug Job Pointer (D5DJP)	64	RO	0000_0000_00 00_0000h

Table continues on the next page...

SEC Memory map

Offset	Register	Width (In bits)	Access	Reset value
D_0E10h	DECO5 Debug Shared Pointer (D5SDP)	64	RO	0000_0000_00_00_0000h
D_0E18h	DECO5 Debug_ICID, most-significant half (D5DIR_MS)	32	RO	0000_0000h
D_0E1Ch	DECO5 Debug ICID, least-significant half (D5DIR_LS)	32	RO	0000_0000h
D_0E20h	Sequence Output Length Register (SOL5)	32	RW	0000_0000h
D_0E24h	Variable Sequence Output Length Register (VSOL5)	32	RW	0000_0000h
D_0E28h	Sequence Input Length Register (SIL5)	32	RW	0000_0000h
D_0E2Ch	Variable Sequence Input Length Register (VSIL5)	32	RW	0000_0000h
D_0E30h	Protocol Override Register (D5POVRD)	32	RW	0000_0000h
D_0E34h	Variable Sequence Output Length Register; Upper 32 bits (UVSOL5)	32	RW	0000_0000h
D_0E38h	Variable Sequence Input Length Register; Upper 32 bits (UVSIL5)	32	RW	0000_0000h
D_0F00h (alias)	Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)	64	RW	0000_0000_00_00_0000h
D_0F08h (alias)	Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)	64	RW	0000_0000_00_00_0000h
D_0F10h (alias)	Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)	64	RW	0000_0000_00_00_0000h
D_0F18h (alias)	Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)	64	RW	0000_0000_00_00_0000h
D_0F20h (alias)	Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)	64	RW	0000_0000_00_00_0000h
D_0F28h (alias)	Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)	64	RW	0000_0000_00_00_0000h
D_0F30h (alias)	Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)	64	RW	0000_0000_00_00_0000h
D_0FA0h (alias)	CHA Revision Number Register, most-significant half (CRNR_MS)	32	RO	1000_1026h
D_0FA4h (alias)	CHA Revision Number Register, least-significant half (CRNR_LS)	32	RO	4413_0017h
D_0FA8h (alias)	Compile Time Parameters Register, most-significant half (CTPR_MS)	32	RO	02BF_2A11h
D_0FACh (alias)	Compile Time Parameters Register, least-significant half (CTPR_LS)	32	RO	0000_7FFFh
D_0FC0h (alias)	Fault Address Register (FAR)	64	RO	0000_0000_00_00_0000h
D_0FC8h (alias)	Fault Address ICID Register (FAICID)	32	RO	0000_0000h
D_0FCCCh (alias)	Fault Address Detail Register (FADR)	32	RO	0000_0000h
D_0FD4h (alias)	SEC Status Register (SSTA)	32	RO	0000_0002h
D_0FE0h (alias)	RTIC Version ID Register (RVID)	32	RO	0F0A_0003h
D_0FE4h (alias)	CHA Cluster Block Version ID Register (CCBVID)	32	RO	0800_0005h

Table continues on the next page...

Offset	Register	Width (In bits)	Access	Reset value
D_0FE8h (alias)	CHA Version ID Register, most-significant half (CHAVID_MS)	32	RO	3400_0001h
D_0FECh (alias)	CHA Version ID Register, least-significant half (CHAVID_LS)	32	RO	3014_3004h
D_0FF0h (alias)	CHA Number Register, most-significant half (CHANUM_MS)	32	RO	4600_1616h
D_0FF4h (alias)	CHA Number Register, least-significant half (CHANUM_LS)	32	RO	1661_6066h
D_0FF8h (alias)	SEC Version ID Register, most-significant half (SECVID_MS)	32	RO	0A10_0401h
D_0FFCh (alias)	SEC Version ID Register, least-significant half (SECVID_LS)	32	RO	0000_0000h

14.2 Master Configuration Register (MCFGR)

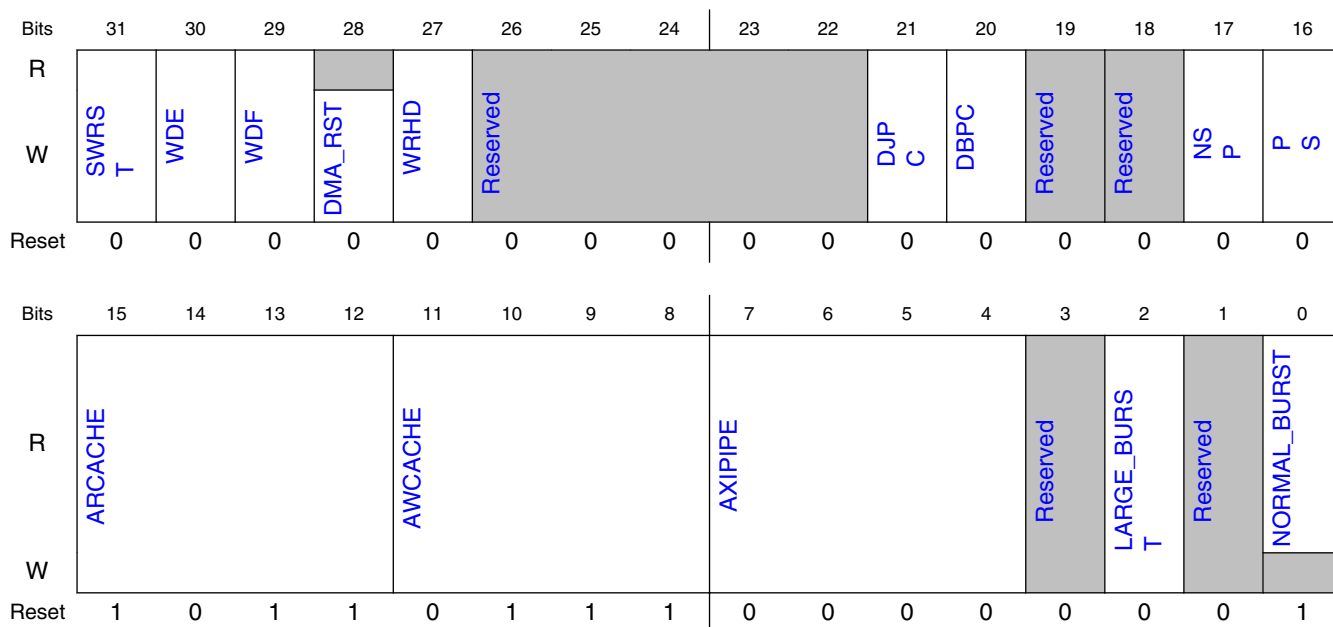
14.2.1 Offset

Register	Offset
MCFGR	4h

14.2.2 Function

The Master Configuration Register is used to set some bus master configurations. This register is typically written at boot time, and in some debug scenarios.

14.2.3 Diagram



14.2.4 Fields

Field	Function
31 SWRST	Software Reset. Prior to issuing a software SEC reset, QI and AI should be stopped as described in the Software SEC Reset section. Writing a 1 to this bit will cause most registers and state machines in SEC to reset. The following SEC registers are not reset: MCFGR, SCFGR, JR0ICID_MS, JR0ICID_LS, JR1ICID_MS, JR1ICID_LS, JR2ICID_MS, JR2ICID_LS, JR3ICID_MS, JR3ICID_LS, QIICID, JRSTARTR, RTICAICID_MS, RTICAICID_LS, RTICBICID_MS, RTICBICID_LS, RTICCID_MS, RTICCID_LS, RTICDICID_MS, RTICDICID_LS, DECO0ICID_MS, DECO0ICID_LS, DECO1ICID_MS, DECO1ICID_LS, DECO2ICID_MS, DECO2ICID_LS, DECO3ICID_MS, DECO3ICID_LS, DECO4ICID_MS, DECO4ICID_LS, DECO5ICID_MS, DECO5ICID_LS, DECO6ICID_MS, DECO6ICID_LS, DAR, PBSL, JDKEKR_0 - JDKEKR_7, TDKEKR_0 - TDKEKR_7, TDSKR_0 - TDSKR_7, SKNR, RTMCTL, RTSCMISC, RTPKRRNG, RTPKRMAX, RTPKRSQ, RTSDCTL, RTTOTSAM, RTSBLIM, RTFRQMIN, RTFRQCNT, RTFRQMAX, RTSCML, RTSCMC, RTSCR1L, RTSCR1C, RTSCR2C, RTSCR2L, RTSCR3L, RTSCR3C, RTSCR4L, RTSCR4C, RTSCR5L, RTSCR5C, RTSCR6PC, RTSCR6PL, RTSTATUS, RTENT0 - RTENT11, RTPKRCNT10, RTPKRCNT32, RTPKRCNT54, RTPKRCNT76, RTPKRCNT98, RTPKRCNTBA, RTPKRCNTDC, RTPKRCNTFE, RDSTA, RDINT0, RDINT1, RDHCNTL, RDHDIG, RDHBUF, but the remaining registers in SEC register page 0 are reset by SWRST. The Job Ring registers in SEC register pages 1 .. 4 are reset by SWRST. The AIOP Interface registers in SEC register page 5 are reset by SWRST. The RTIC registers in SEC register page 6 are not reset by SWRST. (If an RTIC descriptor is in execution or is waiting for execution when SWRST is requested, RTIC will abandon the current sweep through all the hash blocks and restart hashing at the first hash block.) The Queue Manager Interface registers in SEC register page 7 are reset by SWRST. The DECO and CCB registers in SEC register pages 8 ..13 are reset by SWRST.

Table continues on the next page...

Field	Function
	Note that SWRST will remain 1 (and the registers will be held in reset) until any outstanding SEC DMA transactions complete. Writing a 1 to SWRST will not cause a reset of the SEC DMA unless SWRST is already 1 and a 1 is also written to DMARST. Note that writing to MCFGR will overwrite the values in LARGE_BURST, AXIPIPE, AWCACHE and ARCACHE, so to avoid disrupting outstanding DMA transactions when initiating a SWRST, these fields should be written with their current values.
30 WDE	DECO Watchdog Enable. Enables the DECO Watchdog Timer to run. The Timer is used to detect and flush a job that has caused a DECO to hang. If the DECO Watchdog Timer expires, the hung job is usually flushed from the DECO with an error status indication. In those cases in which a hung job is not flushed automatically, software can reset the DECO via the DECO Reset Register. Note that the watchdog expiration period is extended for certain DECO operations and for RNG reseeding, because these can take longer than the normal watchdog expiration period.
29 WDF	Watchdog Fast. Causes the DECO Watchdog Timer to expire prematurely for testing purposes. To facilitate testing the upper bytes of the encrypted byte count logic, when WDF is 1 the encrypted byte count increments by 2^{16} per byte that is encrypted. When WDF is 0 the encrypted byte count increments by 1 per byte that is encrypted.
28 DMA_RST	DMA Reset. If SWRST is already 1, writing a 1 to DMARST and SWRST on the same cycle will cause both DMAs to be reset. The DMAs will not be reset if SWRST is not already a 1 (i.e. a 1 was previously written to SWRST but DMA transactions have not completed). Following a DMA reset, system software should delay long enough for outstanding AXI transaction responses to finish. These orphaned responses will be ignored.
27 WRHD	Write Handoff Disable. If WRHD=0, when DECO has initiated the last write transaction of the current job DECO will go idle without waiting for the bus slave's response to that write transaction. This allows DECO to start another job while awaiting the slave's response. If an error response is eventually received SEC will update the transaction status appropriately. If WRHD=1 DECO will wait for the bus slave's response to the last write transaction before DECO goes to the idle state. Setting WRHD=1 is intended for product testing, so WRHD should normally be left at its PO reset value.
26-22 —	Reserved
21 DJPC	Disable Job Performance Counters. When DJPC=1 counting is disabled in the following Performance Counter registers: PC_REQ_DEQ PC_OB_ENC_REQ PC_IB_DEC_REQ Note that these registers can still be read or written even if DJPC=1.
20 DBPC	Disable Byte Performance Counters. When DJPC=1 counting is disabled in the following Performance Counter registers: PC_OB_ENCRYPT PC_OB_PROTECT PC_IB_DECRYPT PC_IB_VALIDATED Note that these registers can still be read or written even if DBPC=1.
19 —	Reserved
18 —	Reserved

Table continues on the next page...

Master Configuration Register (MCFGR)

Field	Function
17 NSP	<p>No Snoop. This bit controls whether SEC marks all subsequent memory access transactions by default as sharable or not sharable (coherent or non-coherent).</p> <p>NOTE: The NSP configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation.</p> <p>0b - All SEC transactions are marked to be snooped by the coherency control logic of the SoC. 1b - All SEC transactions are marked to be not snooped by the coherency control logic of the SoC.</p>
16 PS	<p>Pointer Size. This bit determines the size of address pointers. (see Address pointers).</p> <p>0b - Pointers fit in one 32-bit word (pointers are 32-bit addresses). 1b - Pointers require two 32-bit words (pointers are 49-bit addresses).</p>
15-12 ARCACHE	<p>AXI Read Transaction Attributes. This field provides default values for the generation of the ARCACHE[3:0] interface signals for read transactions. For a general description of ARCACHE signals refer to the AXI3/4 protocol specification.</p> <p>The following functionality, limitations, and extensions exist in this version of SEC:</p> <ul style="list-style-type: none"> • ARCACHE[0] (Bufferable): This bit is intended to indicate whether read data may be fetched from an intermediate point in the interconnect or must be fetched from the transaction target (for details see AXI4 specification). • ARCACHE[1] (Cacheable/Modifiable): A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance. NOTE: ARCACHE[1] must be set if any of the ARCACHE[3:2] configuration bits are set or if the transaction is marked as sharable (coherent). NOTE: SEC may drive the ARCACHE[1] signal to 1 if read-safe is enabled, independent of the value of the ARCACHE[1] bit configuration. • ARCACHE[2] (Read data allocate recommended): A setting of 1 recommends to allocate the associated data in a downstream cache. NOTE: Setting ARCACHE[2] to 1 requires that ARCACHE[1] also be set to 1 (AXI requirement to enable caching). • ARCACHE[3] (Check for already cached): A setting of 1 indicates the transaction must be checked whether any data located at the target address range is cached so that the correct data can be fetched. NOTE: Setting ARCACHE[3] to 1 requires that ARCACHE[1] also be set to 1 (AXI requirement to enable caching). <p>NOTE: The ARCACHE[3:0] configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation.</p>
11-8 AWCACHE	<p>AXI Write Transaction Attributes. This field provides default values for the generation of the AWCACHE[3:0] interface signals for write transactions. For a general description of AWCACHE signals refer to the AXI3/4 protocol specifications.</p> <p>The following functionality, limitations, and extensions exist for this version of SEC:</p> <ul style="list-style-type: none"> • AWCACHE[0] (Bufferable): A setting of 1 indicates the transaction response may be generated from an intermediate point and the transaction may be delayed reaching its final destination (this setting is intended to reduce transaction latency and improve performance). NOTE: This SoC ignores SEC-generated AWCACHE[0] signals and marks all write transactions as bufferable while guaranteeing that a response is generated

Table continues on the next page...

Field	Function
	<p>only after write data using the same AWID has been made visible to other masters and in the order the transactions were issued.</p> <ul style="list-style-type: none"> • AWCACHE[1] (Cacheable/Modifiable): A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance. NOTE: AWCACHE[1] must be set if any of the AWCACHE[3:2] configuration bits are set or if the transaction is marked as sharable (coherent). • AWCACHE[2] (Check for already cached): A setting of 1 indicates the transaction must be checked to see whether any data located at the target address range is cached so that the correct storage location can be updated. NOTE: Setting AWCACHE[2] to 1 requires that AWCACHE[1] also be set to 1 (AXI requirement to enable caching). • AWCACHE[3] (Write data allocate recommended): A setting of 1 recommends to allocate the associated data in a downstream cache. NOTE: Setting AWCACHE[3] to 1 requires that AWCACHE[1] also be set to 1 (AXI requirement to enable caching) <p>NOTE: The AWCACHE[3:0] configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation.</p>
7-4 AXIPIPE	<p>AXI Pipeline Depth.</p> <p>The AXIPIPE field is a debug field used to adjust the maximum number of outstanding DMA transactions that SEC is able to queue. Optimal performance will be achieved by retaining the default value of this field.</p>
3 —	Reserved
2 LARGE_BURST	<p>Enable Large Bursts. When LARGE_BURST=1, Job Descriptor reads, Shared Descriptor reads and reads of data for the Data FIFO can use transactions as large as the maximum AXI interface transaction size, which equals 16 times the width of the AXI data buses (i.e. 256 bytes for 128-bit data buses). When LARGE_BURST=0, all master bus transactions use the normal burst size. Changes to LARGE_BURST should be made only when SEC is not processing jobs.</p>
1 —	Reserved
0 NORMAL_BURST	<p>Normal Burst Size</p> <p>This field indicates the normal burst (e.g. transaction) size limits for SEC's read and write accesses to memory. (Note that this field was writable in earlier versions of SEC, but this capability is now obsolete.)</p> <p>0b - 32 byte burst size 1b - 64 byte burst size</p>

14.3 Security Configuration Register (SCFGR)

14.3.1 Offset

Register	Offset
SCFGR	Ch

14.3.2 Function

The Security Configuration Register is used to set security-related mode bits. These bits are used to switch from special boot-time operating modes to normal operating modes. At POR, all bits in SCFGR reset to 0.

When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in deterministic mode. This allows the secure boot software to run deterministic tests on the RNG and its State Handle 0 logic. Once the tests have been completed, the secure boot software can write a 1 to RNGSH0 to prevent State Handle 0 from being instantiated in deterministic mode. This ensures that random data, rather than deterministic data, is used for the Zeroizable Master Key Register in the low-power section of Security Monitor, the Differential Power Analysis Resistance Mask in the AESA, the Job Descriptor Key Encryption Register, the Trusted Descriptor Key Encryption Register, the Trusted Descriptor Signing Key Register and the random padding used by built-in protocols.

At POR the AESA will seed its Differential Power Analysis Resistance Mask using a default deterministic value. Writing a 1 to the RANDDDPAR bit will cause the AESA to reseed its Differential Power Analysis Resistance Mask using data from RNG DRNG State Handle 0. Typically the secure boot software will write a 1 to RANDDDPAR after RNG DRNG State Handle 0 has been instantiated in a non-deterministic mode. Once a 1 has been written to RANDDDPAR, this bit will remain 1 until the next POR. Software can read this bit to determine whether the Differential Power Analysis Resistance Mask was seeded from the RNG.

The PRIBLOB field is used to select a private blob type during Trusted Mode. When a General Memory Blob or Secure Memory Blob is encapsulated or decapsulated during Trusted Mode, the PRIBLOB bits are used to modify the derivation of the Blob Key Encryption Key (see [Blob encapsulation](#)). This is used to enforce cryptographic separation of private blob types during the boot process (and thereafter). These bits reset to 0 at POR, but when a PRIBLOB bit is written to a 1, it remains a 1 until the next POR.

The PRIBLOB=00 setting allows secure boot software to have its own private blobs that cannot be decapsulated or encapsulated by other software, even software that later runs in Trusted Mode. This feature can be used to safeguard boot reference metrics (e.g. hash values over software). In this use case, the reference metrics might be initially verified

via a public key signature and then encapsulated in a private secure boot blob. On subsequent boot cycles the protected reference metrics would be obtained by decapsulating the private secure boot blob, obviating the time-consuming public key signature verification process.

The PRIBLOB=01 and PRIBLOB=10 settings allow trusted provisioning software (e.g. software that handles DRM keys) to have private blobs that cannot be decapsulated or encapsulated by software that runs later in the boot process, even if that software runs in Trusted Mode.

As illustrated in [Figure 14-1](#), typically the secure boot software would enter Trusted Mode, then encapsulate or decapsulate all of its private blobs, and then would write either a 01, 10 or 11 to PRIBLOB. For the remainder of the current power-on session, private secure boot blobs could no longer be encapsulated or decapsulated. The secure boot software would then either run provisioning software with the 10 or 01 setting, or would skip the provisioning software and run the normal boot software with the 11 setting. If the provisioning software runs, it would encapsulate or decapsulate its own private blobs and then write 11 to PRIBLOB. At this point PRIBLOB=11 for the remainder of the current power on session, and no software can encapsulate or decapsulate private secure boot software blobs or either type of private provisioning blobs.

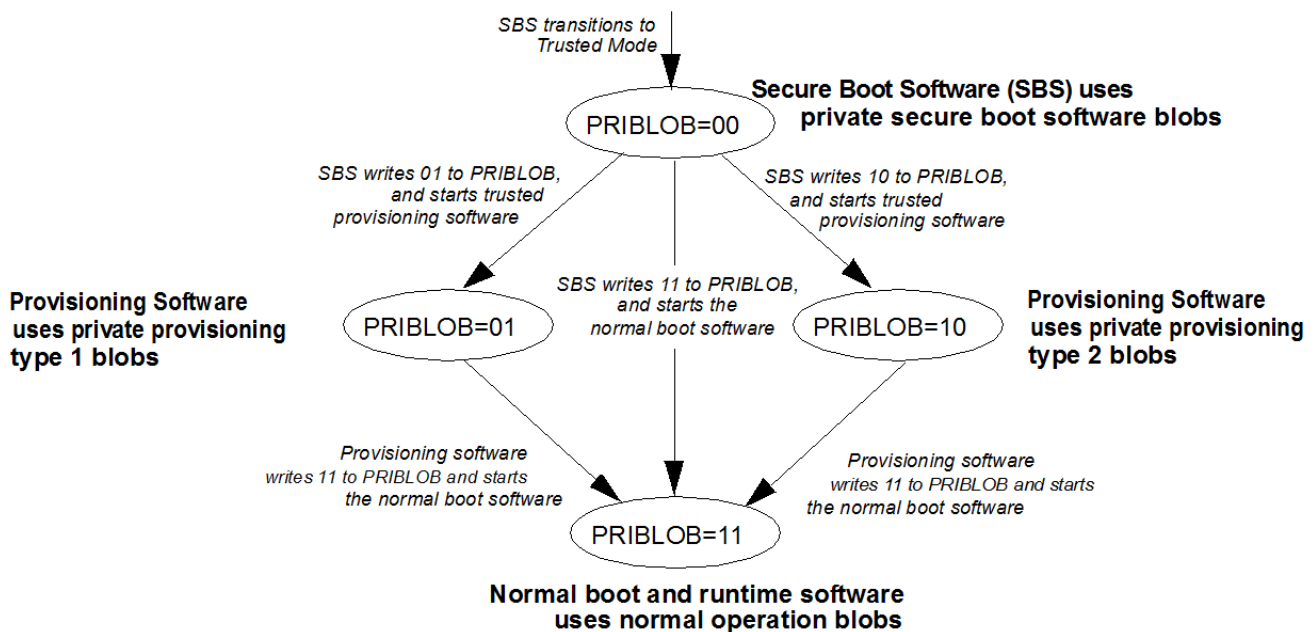
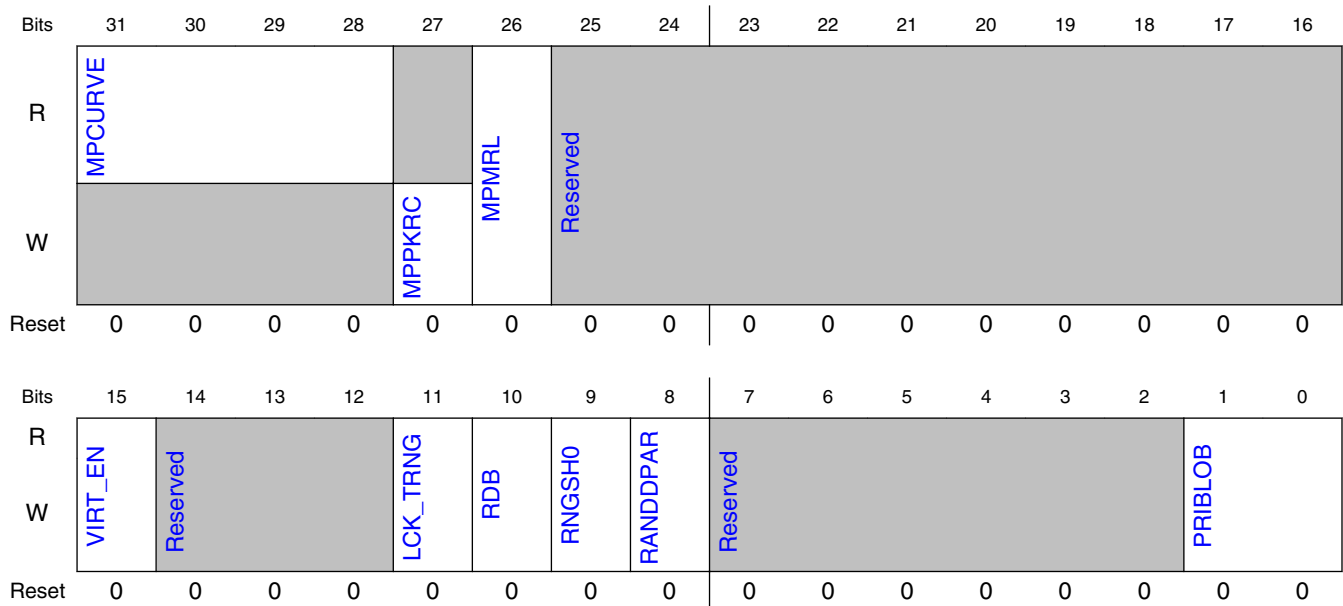


Figure 14-1. Process for Managing Private Blobs

14.3.3 Diagram



14.3.4 Fields

Field	Function
31-28 MPCURVE	Manufacturing Protection Curve. This shows the elliptic curve that was selected when the MPPrivK Generation protocol was run.
27 MPPKRC	Manufacturing Protection Private Key Register Clear. Writing a 1 to this bit clears the Manufacturing Protection Private Key Register.
26 MPMRL	Manufacturing Protection Message Register Lock. Writing a 1 to this bit locks the Manufacturing Protection Message Register for writing. The register remains locked until the next POR.
25-16 —	Reserved
15 VIRT_EN	Virtualization enable. Virtualization is disabled by default. Writing a 1 to this bit enables Job Ring virtualization. When Job Ring virtualization is enabled, the Start_JRa bits in the JRSTART register must be used to switch between writing the Job Ring registers in register page 0 and writing the Job Ring registers in register pages 1....4. Note that the LAMTD bit in the JRaCID register cannot be written unless VIRT_EN is 1.
14-12 —	Reserved
11 LCK_TRNG	Lock TRNG Program Mode. Writing a 1 to this bit locks the TRNG. That is, when this bit is set TRNG can't go into program mode. If it is in program mode when this bit is set, the TRNG will immediately leave program mode. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.

Table continues on the next page...

Field	Function
10 RDB	Enable random data buffer. Set this bit to 1 to enable a buffer for data obtained from the RNG. This is a 196-byte buffer that is intended to improve performance when built-in protocols request random data to be used for initialization vectors or padding. When the buffer contains 64 or fewer bytes of random data, another 128 bytes is obtained from RNG State Handle 0. This bit would typically be set to 1 after any boot-time RNG testing has been completed and State Handle 0 has been instantiated in non-deterministic mode. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.
9 RNGSH0	Random Number Generator State Handle 0. 0b - When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in any mode. RNGSH0 is set to 0 only for testing. 1b - When RNGSH0 is 1, RNG DRNG State Handle 0 cannot be instantiated in deterministic (test) mode. RNGSH0 should be set to 1 before the RNG is instantiated. If it is currently instantiated in a deterministic mode, it will be un-instantiated. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.
8 RANDDPAR	Random Differential Power Analysis Resistance (DPAR) Mask. After RNGSH0 has been set and the RNG has been instantiated, RANDDPAR should be set to 1. 0b - The AESA DPAR Mask was seeded using the default deterministic seed. This mode is used for testing. 1b - When RANDDPAR is written with a 1, the AESA DPAR Mask is reseeded from RNG DRNG State Handle 0. This is the normal runtime mode. Once RANDDPAR has been written with a 1, it cannot be changed to a 0 until the next power on reset.
7-2 —	Reserved
1-0 PRIBLOB	Private Blob. This field selects one of four different types of private blobs during Trusted Mode. All blobs encapsulated or decapsulated during Trusted Mode will be of the type specified in this field, until a 1 is written to any or the bits, or until the next POR. The bits of this field are "sticky", i.e. once a bit has been written to a 1, it cannot be changed to a 0 until the next power on reset. 00b - private secure boot software blobs 01b - private provisioning type 1 blobs 10b - private provisioning type 2 blobs 11b - normal operation blobs

14.4 Job Ring a ICID Register - most significant half (JR01 CID_MS - JR3ICID_MS)

14.4.1 Offset

For a = 0 to 3:

Register	Offset	Description
JRaICID_MS	10h + (a × 8h)	used with Job Ring a

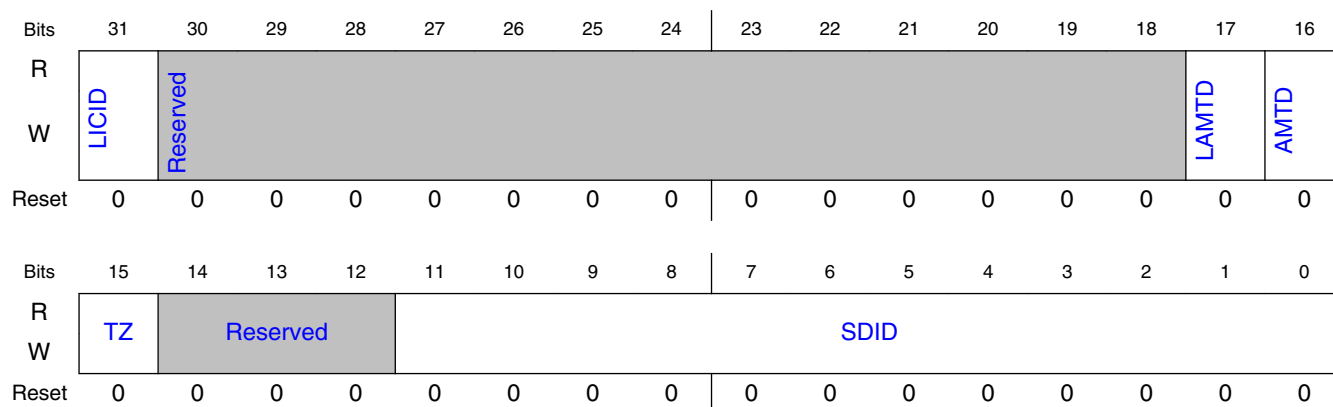
14.4.2 Function

There is one JRaICID register per Job Ringan it is used to indicate the Security Domain that currently owns the Job Ring and to specify the ICID value that the SEC DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular Job Ring. The ICID field is typically written at boot time and then locked. TrustZone SecureWorld can reserve a Job Ring for itself by setting the TZ bit to 1. Note that TZ can be set to 1 only if the register is written using a SecureWorld bus transaction. If the Job Ring is reserved by SecureWorld the Job Ring registers associated with this ring can be written only via SecureWorld bus transactions. NonSecureWorld writes to Job Ring registers reserved by SecureWorld will be ignored.

This register also contains a bit that grants permission for Trusted Descriptors to be created in this Job Ring. If the Job Ring is owned by TrustZone SecureWorld (TZ=1), any Trusted Descriptor created in this job ring is marked as a TrustZone Trusted Descriptor (see TDES field in [HEADER command](#). Once a TrustZone Trusted Descriptor has been created, it can be executed in any Job Ring. If the job ring is not owned by TrustZone SecureWorld, any Trusted Descriptor created in this Job Ring is marked as a nonTrustZone Trusted Descriptor.

If virtualization mode is disabled in the Security Configuration register, the SDID field is not writable and will remain at the default all-0 value.

14.4.3 Diagram



14.4.4 Fields

Field	Function
31 LICID	Lock ICID. Once LICID has been set, until reset no further changes can be made to the LICID in this register or the ICID, PL, or BMT fields in the JRaICID_LS register. The SDID field is not locked, and the AMTD is locked only if LAMTD=1.
30-18 —	Reserved
17 LAMTD	Lock AMTD. Once LAMTD has been set, no further changes can be made to the AMTD field. Note that the LAMTD bit cannot be written unless virtualization mode is enabled (SCFGR[VIRT_EN]=1).
16 AMTD	Allow Make Trusted Descriptor. If AMTD is set, the Job Ring associated with this register is permitted to issue jobs that create Trusted Descriptors. When DECO encounters a descriptor header with the MTD bit set, the options specified in the SIGNATURE command at the end of the descriptor determine whether DECO will execute the commands in the descriptor, or append a signature to the descriptor, or both. If the Job Ring is owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone Trusted Descriptor, otherwise the descriptor will be treated as a non-SecureWorld Trusted Descriptor. If AMTD is not set, then executing a descriptor with the MTD bit set in the descriptor's header will result in an error, and no signature will be generated.
15 TZ	TrustZone SecureWorld. This bit can be written only by TrustZone SecureWorld (i.e. a bus transaction with ns=0). If TZ=1, this Job Ring is owned by TrustZone SecureWorld and the SDID field is forced to all 0s. If TZ=0 this Job Ring is owned by non-SecureWorld and the SDID field is writable.
14-12 —	Reserved
11-0 SDID	Security Domain Identifier. If TZ=0, SDID indicates the Security Domain Identifier that owns this Job Ring. The 12-bit SDID value is used to tag Black Keys, Blobs and Trusted Descriptors so they can be used only by this Security Domain. If TZ=1, the SDID value is forced to all 0s and Black Keys, Blobs and Trusted Descriptors are tagged with the special TrustZone SecureWorld tag. If virtualization mode is disabled, the SDID is forced to all 0s.

14.5 Job Ring a ICID Register - least significant half (JR0aICID_LS - JR3aICID_LS)

14.5.1 Offset

For a = 0 to 3:

Register	Offset	Description
JRaICID_LS	14h + (a × 8h)	used with Job Ring a

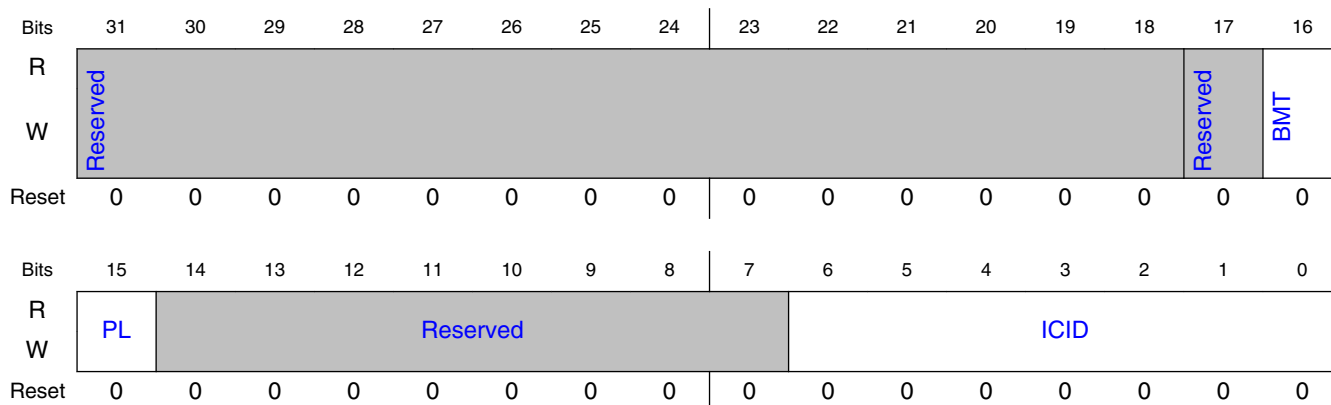
14.5.2 Function

There is one JRaICID register per Job Ring and it is used to indicate the Security Domain that currently owns the Job Ring and to specify the ICID value that the SEC DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular Job Ring. TrustZone SecureWorld can reserve a Job Ring for itself by setting the TZ bit to 1. Note that TZ can be set to 1 only if the register is written using a SecureWorld bus transaction. If the Job Ring is reserved by SecureWorld the Job Ring registers associated with this ring can be written only via SecureWorld bus transactions. NonSecureWorld writes to Job Ring registers reserved by SecureWorld will be ignored.

This register also contains a bit that grants permission for Trusted Descriptors to be created in this Job Ring. If the job ring is owned by TrustZone SecureWorld (TZ=1), any Trusted Descriptor created in this Job Ring is marked as a TrustZone Trusted Descriptor (see TDES field in [HEADER command](#). Once a TrustZone Trusted Descriptor has been created, it can be executed in any Job Ring. If the Job Ring is not owned by TrustZone SecureWorld, any Trusted Descriptor created in this Job Ring is marked as a nonTrustZone Trusted Descriptor. The ICID field is typically written at boot time and then locked.

If virtualization mode is disabled in the Security Configuration register, the SDID and LAMTD fields are not writable, and the SDID will remain at the default all 0 value.

14.5.3 Diagram



14.5.4 Fields

Field	Function
31-18 —	Reserved
17 —	Reserved
16 BMT	Bypass Memory Translation. If BMT=1, bus transactions associated with this Job Ring will assert the BMT signal as 1, indicating that the bus transaction should bypass memory translation (if PL = 1). If PL=0, the value of the BMT signal is ignored, and memory translation is performed by the system MMU. Once the LICID bit in the JRaICID_MS register has been set to 1, until reset no further changes can be made to the BMT field.
15 PL	Privilege Level. If PL=1, bus transactions associated with this Job Ring will assert the PL signal as 1, indicating that BMT=1 is permitted for this bus transaction. Once the LICID bit in the JRaICID_MS register has been set to 1, until reset no further changes can be made to the PL field.
14-7 —	Reserved
6-0 ICID	ICID. This field defines the ICID value asserted for DMA transactions associated with job descriptors executed from this Job Ring. Once the LICID bit in the JRaICID_MS register has been set to 1, until reset no further changes can be made to the ICID field.

14.6 Queue Manager Interface SDID Register (QISDID)

14.6.1 Offset

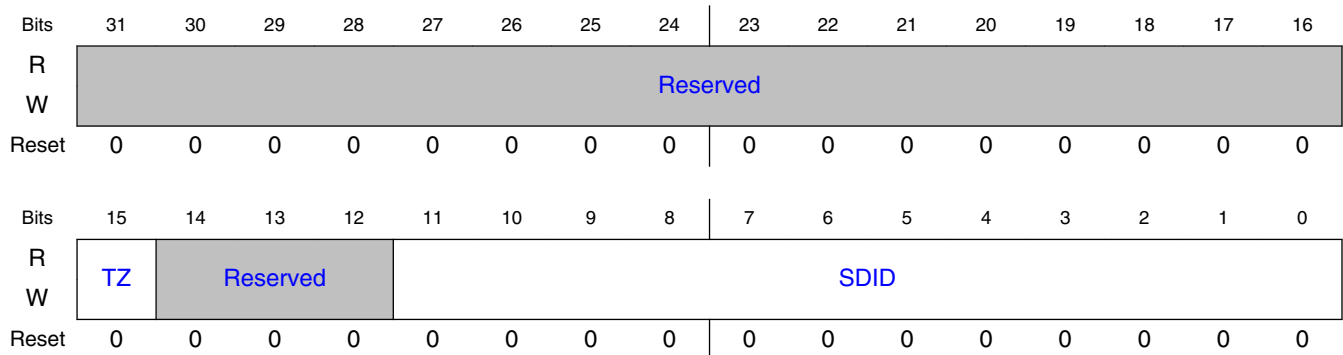
Register	Offset
QISDID	50h

14.6.2 Function

The QISDID Register is used to indicate the Security Domain that is associated with the Queue Manager Interface. This SDID value is used to tag Black Keys and blobs that are referenced by descriptors executed via the Queue Manager Interface.

If virtualization mode is disabled in the Security Configuration register, the SDID field is not writable and will remain at the default all 0 value.

14.6.3 Diagram



14.6.4 Fields

Field	Function
31-16 —	Reserved
15 TZ	TrustZone SecureWorld. This bit can be written only by TrustZone SecureWorld (i.e. a bus transaction with ns=0). If TZ=1, the Queue Manager Interface is associated with TrustZone SecureWorld and the SDID field is forced to all 0s. If TZ=0 the Queue Manager Interface is associated with non-SecureWorld and the SDID field is writable.
14-12 —	Reserved
11-0 SDID	Security Domain Identifier. If TZ=0, SDID indicates the Security Domain associated with the Queue Manager Interface. The SDID value is used to tag Black Keys, Blobs and Trusted Descriptors so they can be used only by this Security Domain. If TZ=1, the SDID value is forced to all 0s and Black Keys, Blobs and Trusted Descriptors are tagged with the special TrustZone SecureWorld tag. If virtualization mode is disabled, the SDID is forced to all 0s.

14.7 Debug Control Register (DEBUGCTL)

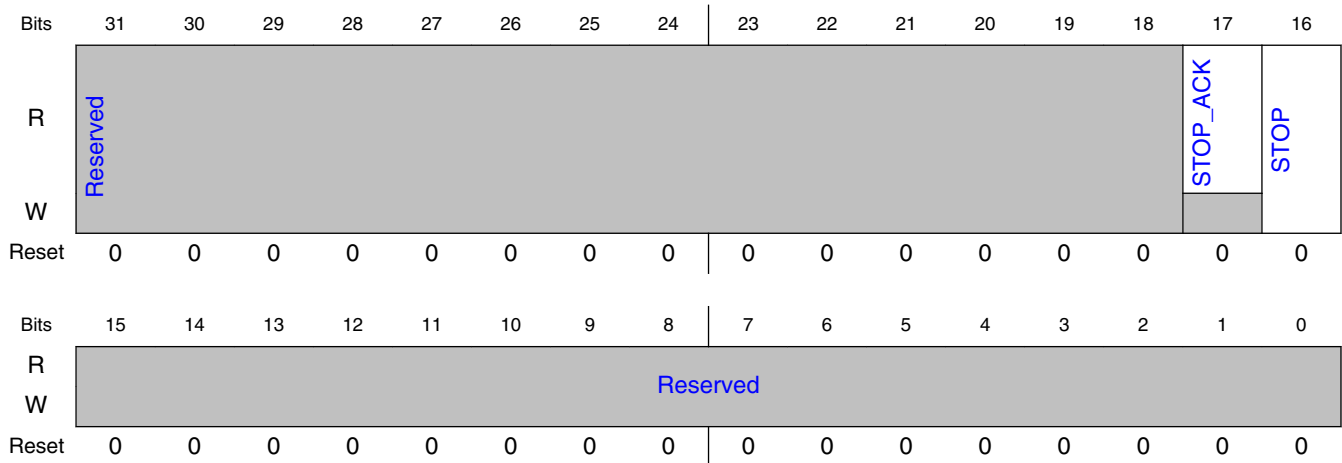
14.7.1 Offset

Register	Offset
DEBUGCTL	58h

14.7.2 Function

The DEBUGCTL Register is used to stop SEC from processing jobs so that a consistent read of the debug registers can be performed.

14.7.3 Diagram



14.7.4 Fields

Field	Function
31-18 —	Reserved
17 STOP_ACK	STOP_ACK will assert when the job queue controller, QI, and AI acknowledge that they are stopped.
16 STOP	<p>STOP is written to 1 to request that SEC stop processing jobs. This is intended to be a graceful halt. SEC will shut down in such a way that it will be able to resume processing where it left off once software has finished reading the debug registers. Note that the RTIC watchdog timer will continue to run during the halt. It is recommended that the DECO watchdog timer be turned off (see Master Config Register) prior to stopping SEC in order to prevent a possible watchdog error from a job in a stopped DECO. When STOP is asserted, the DECO Availability register can be used to monitor which DECOs are stopped or are available.</p> <p>If the REI registers are programmed to halt SEC after a recoverable error and that recoverable error occurs, this will cause the STOP bit to assert. SEC will acknowledge that the stop is complete by setting the STOP_ACK bit. The STOP bit must be cleared in order to restart SEC.</p>
15-0 —	Reserved.

14.8 Job Ring Start Register (JRSTARTR)

14.8.1 Offset

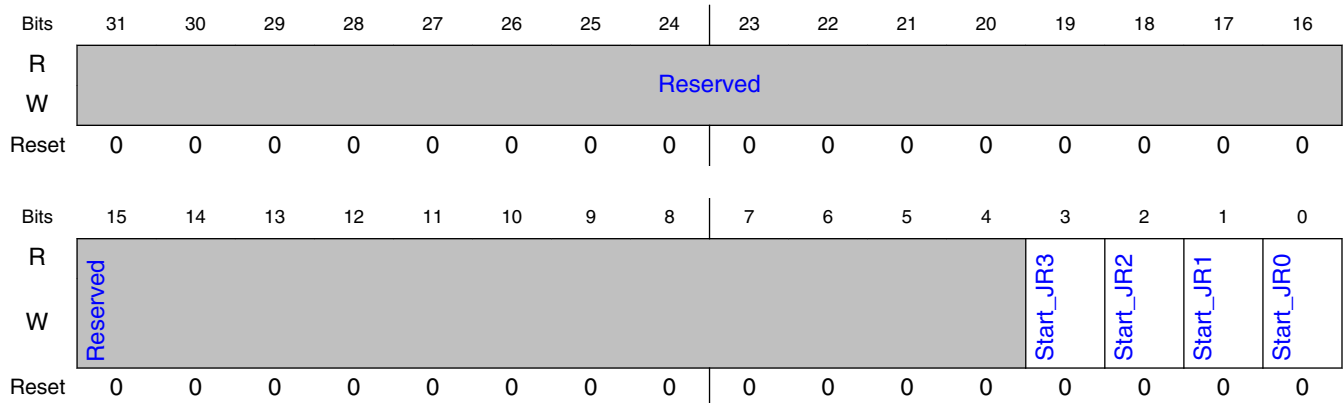
Register	Offset
JRSTARTR	5Ch

14.8.2 Function

The Job Ring Start register is used by the system software or TrustZone SecureWorld when configuring a Job Ring for a new user. Before the Job Ring is configured for a new user, the Job Ring must be in stop mode. Before the new user can set up the Job Ring and begin using it, the Job Ring must be in start mode.

The Job Ring Start register is not used and is not writable when virtualization mode is disabled in the Security Configuration register. All bits in this register will remain at the default 0 value when virtualization mode is disabled.

14.8.3 Diagram



14.8.4 Fields

Field	Function
31-4 —	Reserved
3 Start_JR3	<p>Start Job Ring 3. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0). If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), Start_JR3 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.</p> <p>0b - Stop Mode. The JR3ICID register for Job Ring 3 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 3 are NOT accessible. If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), the JR3ICID register can be written only via a bus transaction that has ns=0.</p> <p>1b - Start Mode. The JR3ICID register for Job Ring 3 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 3 ARE accessible. If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 3 can be written only via a bus transaction that has ns=0.</p>
2 Start_JR2	<p>Start Job Ring 2. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0). If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), Start_JR2 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.</p> <p>0b - Stop Mode. The JR2ICID register for Job Ring 2 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 are NOT accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), the JR2ICID register can be written only via a bus transaction that has ns=0.</p> <p>1b - Start Mode. The JR2ICID register for Job Ring 2 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 ARE accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 2 can be written only via a bus transaction that has ns=0.</p>
1 Start_JR1	<p>Start Job Ring 1. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0). If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), Start_JR1 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.</p> <p>0b - Stop Mode. The JR1ICID register for Job Ring 1 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 are NOT accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), the JR1ICID register can be written only via a bus transaction that has ns=0.</p> <p>1b - Start Mode. The JR1ICID register for Job Ring 1 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 ARE accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 1 can be written only via a bus transaction that has ns=0.</p>
0 Start_JR0	<p>Start Job Ring 0. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0). If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), Start_JR0 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.</p> <p>0b - Stop Mode. The JR0ICID register for Job Ring 0 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 are NOT accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), the JR0ICID register can be written only via a bus transaction that has ns=0.</p> <p>1b - Start Mode. The JR0ICID register for Job Ring 0 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 ARE accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 0 can be written only via a bus transaction that has ns=0.</p>

14.9 RTIC ICID Register for Block a - most significant half (RTICAICID_MS - RTICDICID_MS)

14.9.1 Offset

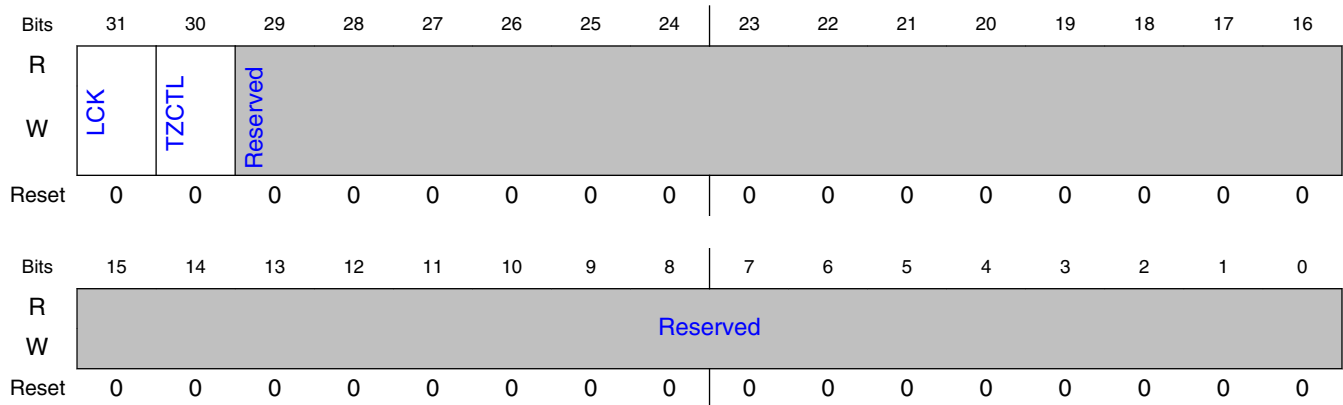
For a = A to D (0 to 3):

Register	Offset	Description
RTICaICID_MS	60h + (a × 8h)	used with Block a

14.9.2 Function

There is one RTICaICID register per RTIC hash block. The RTIC ICID Register is used to specify the AXI bus ICID value and BMT value that the SEC DMA asserts when reading a particular RTIC hash block from memory external to SEC. This register is typically written at boot time and then locked.

14.9.3 Diagram



14.9.4 Fields

Field	Function
31 LCK	RTIC ICID Lock. Once LCK has been set, no further changes can be made to RTICICID (including changes to LCK) until the next POR.
30 TZCTL	TrustZone Control. The TZCTL field exists only in the RTICAICID register. This bit is reserved in RTICBICID, RTICCCICID, and RTICDICID. The TZCTL bit can be written only if ips_nonsecure_access=0 (i.e. only via a TrustZone SecureWorld bus transaction). Software sets the TZCTL bit to 1 to indicate that TrustZone SecureWorld owns RTIC. If TZCTL=1 the RTIC registers (offsets 60h ... 7Ch and offsets 60000h .. 6FFFFh) can be written only via a TrustZone SecureWorld bus transaction. If TZCTL=1 the RTICICID[TZ] bits can be written with either a 1 or a 0 (i.e. TrustZone SecureWorld software can decide whether SecureWorld or NonSecureWorld bus transactions are asserted for a particular RTIC memory block). If TZCTL=0 all RTICICID[TZ] bits are forced to 0 and all RTIC bus transactions are NonSecureWorld.
29-0 —	Reserved

14.10 RTIC ICID Register for Block a - least significant half (RTICAICID_LS - RTICDICID_LS)

14.10.1 Offset

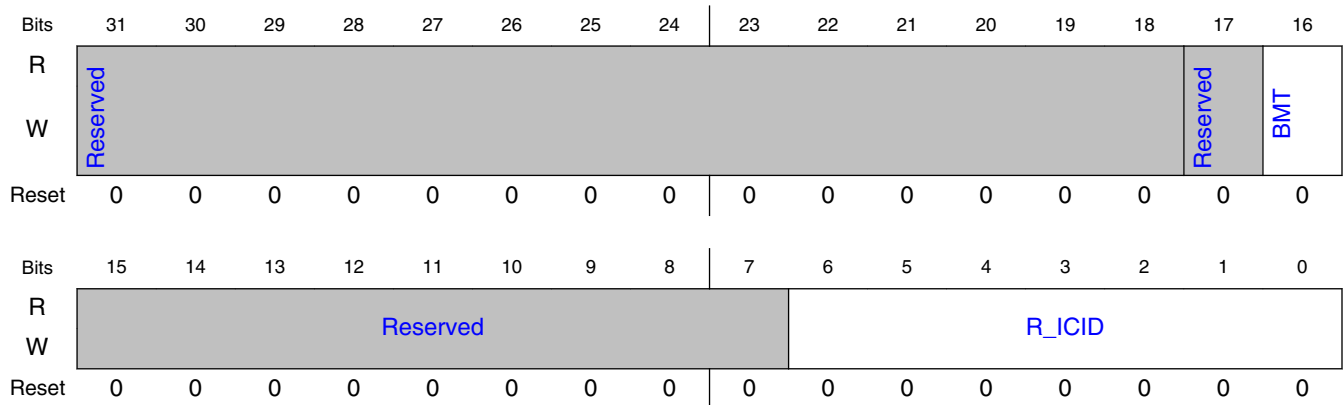
For a = A to D (0 to 3):

Register	Offset	Description
RTICaICID_LS	64h + (a × 8h)	used with Block a

14.10.2 Function

There is one RTICaICID register per RTIC hash block. The RTIC ICID Register is used to specify the AXI bus ICID value and BMT value that the SEC DMA asserts when reading a particular RTIC hash block from memory external to SEC. This register is typically written at boot time and then locked.

14.10.3 Diagram



14.10.4 Fields

Field	Function
31-18 —	Reserved
17 —	Reserved
16 BMT	RTIC Bypass Memory Translation. This field defines the BMT value asserted when RTIC accesses memory external to SEC.
15-7 —	Reserved
6-0 R_ICID	RTIC ICID. This field defines the ICID value asserted when RTIC accesses the addresses in the hash block a regions of memory.

14.11 Protocol Configuration Register (PROTCFG)

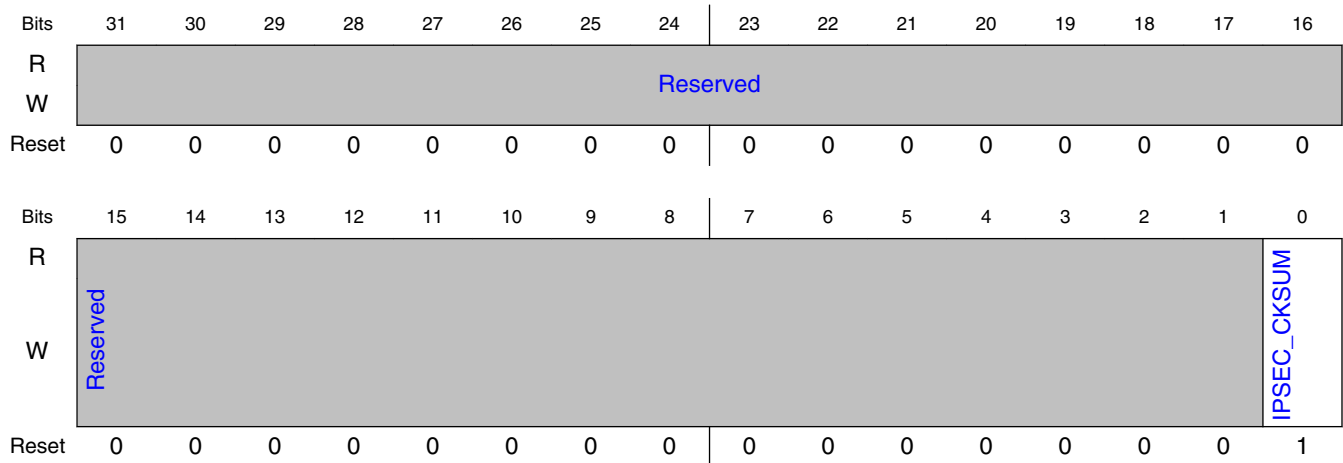
14.11.1 Offset

Register	Offset
PROTCFG	84h

14.11.2 Function

The Protocol Configuration Register is used to configure the behavior of SEC's built-in protocol operations.

14.11.3 Diagram



14.11.4 Fields

Field	Function
31-1 —	Reserved
0 IPSEC_CKSUM	Configure the IPSEC checksum calculation. 0 - Padding is removed but still included in the checksum calculation. 1 - Padding is removed but not included in the checksum calculation.

14.12 DECO Request Source Register (DECORSR)

14.12.1 Offset

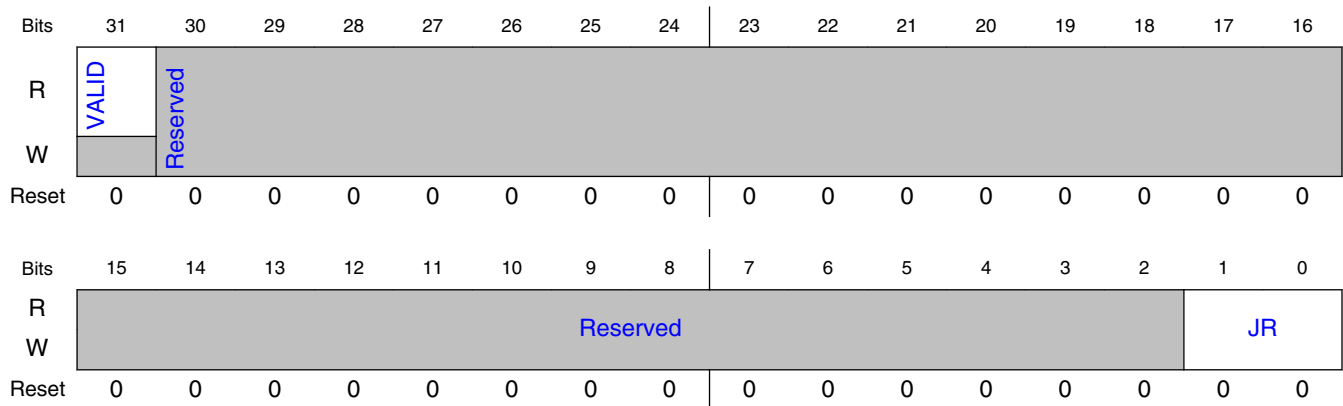
Register	Offset
DECORSR	94h

14.12.2 Function

The DECO Request Source Register is used to indicate a particular Job Ring whose JRaCID register will be used to supply the ICID, TZ and SDID values when descriptor commands are executed under direct software control. The selected Job Ring will be used for all DECOs that are presently under direct software control regardless of whether those DECOs were requested via a single write or via multiple writes to the DECO Request Register. The DECO Request Source Register is writable only when all bits in the DECO Request Register are 0 (i.e. no DECOs are requested or already under direct software control). If the Job Ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

Note that the DECO Request Source register is not used and is not writable when virtualization is disabled.

14.12.3 Diagram



14.12.4 Fields

Field	Function
31 VALID	Valid. This bit will be set to 1 to indicate that the JR field contains a valid Job Ring number. If an invalid Job Ring number is written to JR, the VALID bit will be 0. The VALID bit will always remain 0 if virtualization mode is disabled.
30-2 —	Reserved

Table continues on the next page...

Field	Function
1-0 JR	Job Ring number. This Job Ring's JRaICID register will be used to supply the ICID, TZ and SDID values when descriptor commands are executed under direct software control. If the specified Job Ring is not implemented or if virtualization is enabled and the specified Job Ring is not in start mode, the JR field will not be changed and the VALID bit will remain 0. If the DECORSR is written via a bus transaction that has ns=0, any implemented Job Ring can be selected. But if the bus transaction has ns=1, then only Job Rings with TZ=0 can be selected. If a TrustZone Job Ring is selected via an ns=1 bus write to DECORSR the VALID bit will remain 0. This field is not writable if virtualization mode is disabled.

14.13 DECO Request Register (DECORR)

14.13.1 Offset

Register	Offset
DECORR	9Ch

14.13.2 Function

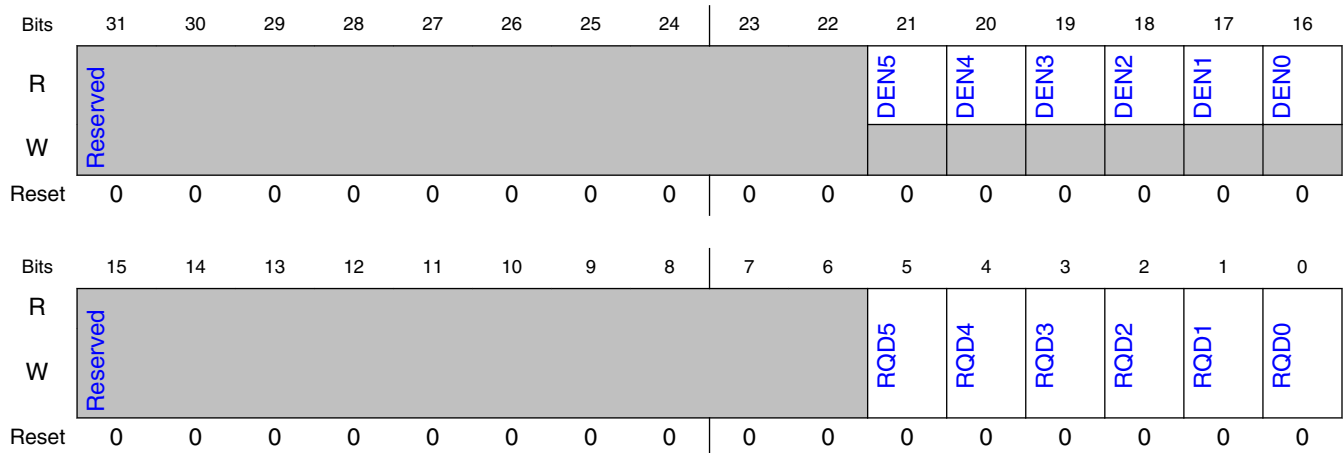
This register is used when software wants to bypass the normal job queue controller mechanism and directly access one of DECO/CCB blocks. This interface would normally be used only for debugging and testing purposes since it is not as efficient as the Queue Manager Interface or Job Ring Interface. Note that multiple DECOs can be requested via a single write to DECORR, and additional DECOs can be requested by writing to DECORR multiple times. DECORR may be written only when AI adaptive resource reservation is disabled (ARRR.AR_EN=0) and when all DECOs that have been requested have had that access granted (DENn=RQDn). The procedure for directly accessing a DECO/CCB block is described in detail in [Register-based service interface](#).

Note that, unless virtualization mode is disabled in the Security Configuration register, a Job Ring that is in the start mode must be selected via the DECO Request Source Register prior to requesting a DECO via DECORR. If the DECO Request Source Register's VALID bit is not set, or if the selected Job Ring is not in start mode, the DECO Request Register cannot be written. The ICID, TZ and SDID value from the selected job ring's JRaICID register will be used when descriptor commands are executed under direct software control. If the Job Ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

DECO Request Register (DECORR)

If virtualization mode is disabled the DECO ICID registers supply the ICID values used when descriptor commands are executed under direct software control, and an all-zero SDID is used. The DECO Request Source register is not used when virtualization is disabled.

14.13.3 Diagram



14.13.4 Fields

Field	Function
31-22 —	Reserved. Always 0.
21 DEN5	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 5/CCB 5.
20 DEN4	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 4/CCB 4.
19 DEN3	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 3/CCB 3.
18 DEN2	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 2/CCB 2.
17 DEN1	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 1/CCB 1.
16 DEN0	The job queue controller asserts this bit when permission is granted for the software to directly access DECO 0/CCB 0.
15-6 —	Reserved. Always 0.

Table continues on the next page...

Field	Function
5 RQD5	This bit is set by software to request direct access to DECO 5/CCB 5. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.
4 RQD4	This bit is set by software to request direct access to DECO 4/CCB 4. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.
3 RQD3	This bit is set by software to request direct access to DECO 3/CCB 3. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.
2 RQD2	This bit is set by software to request direct access to DECO 2/CCB 2. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.
1 RQD1	This bit is set by software to request direct access to DECO 1/CCB 1. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.
0 RQD0	This bit is set by software to request direct access to DECO 0/CCB 0. It cannot be cleared until the direct access operation is complete or SEC gets a software reset.

14.14 DECOa ICID Register - most significant half (DECO0ICID_MS - DECO5ICID_MS)

14.14.1 Offset

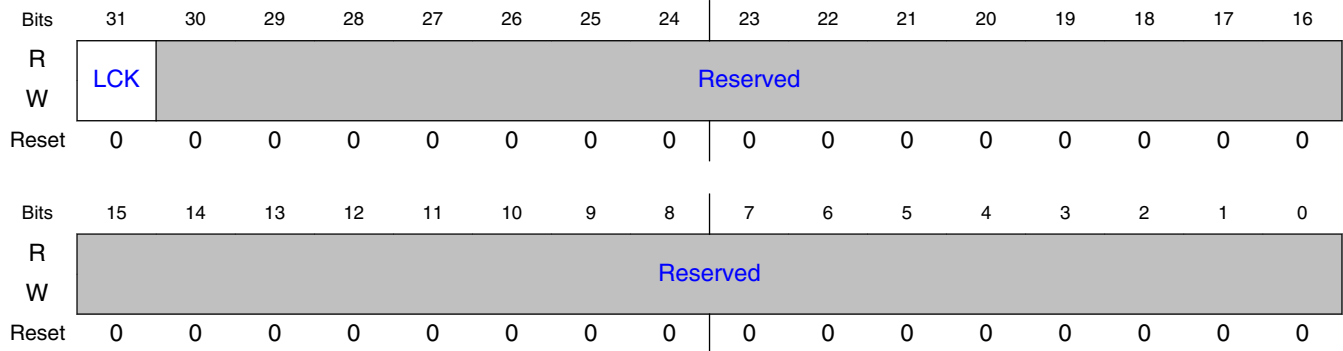
For a = 0 to 5:

Register	Offset	Description
DECOaICID_MS	A0h + (a × 8h)	used with DECOa

14.14.2 Function

This register is used when virtualization is disabled via the VIRT_EN bit in the Security Configuration Register. In that case this register specifies various bus signal values that SEC's DMA will assert when a DECO is under direct software control. When virtualization is enabled, this register is not used and instead the DECO Request Source Register is used to select one of the Job Rings whose JRaICID register will supply these bus signal values. The DECO ICID register is used to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of a DECO that is under the direct control of software. This register is intended to be written by the same processor that writes to the DECORR.

14.14.3 Diagram



14.14.4 Fields

Field	Function
31 LCK	Lock. Once LCK is set, no further changes can be made to DECOaICID (including any changes to LCK). LCK is not writable when virtualization mode is enabled.
30-0 —	Reserved

14.15 DECOa ICID Register - least significant half (DECO0ICID_LS - DECO5ICID_LS)

14.15.1 Offset

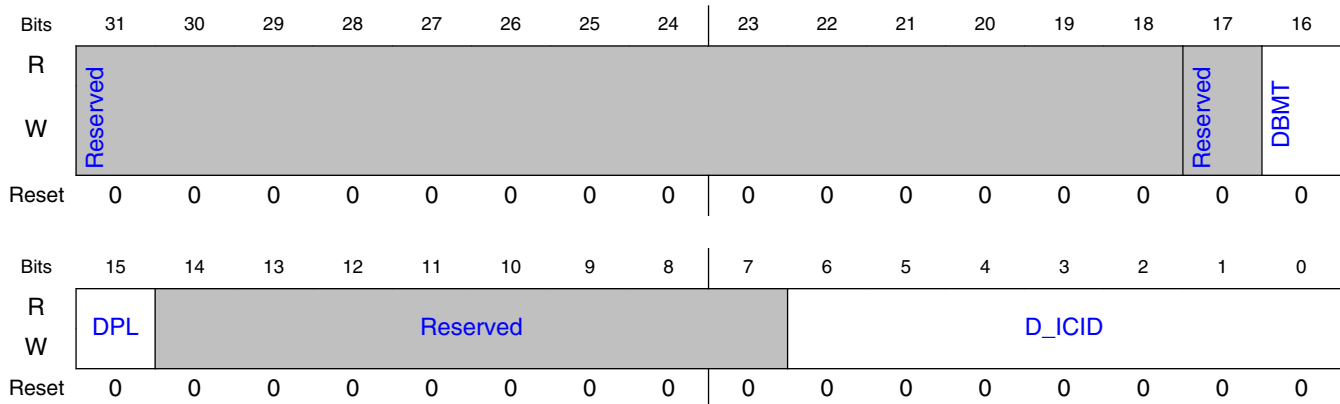
For a = 0 to 5:

Register	Offset	Description
DECOaICID_LS	A4h + (a × 8h)	used with DECOa

14.15.2 Function

This register is used when virtualization is disabled via the VIRT_EN bit in the Security Configuration Register. In that case this register specifies various bus signal values that SEC's DMA will assert when a DECO is under direct software control. When virtualization is enabled, this register is not used and instead the DECO Request Source Register is used to select one of the Job Rings whose JRaICID register will supply these bus signal values. The DECO ICID register is used to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of a DECO that is under the direct control of software. This register is intended to be written by the same processor that writes to the DECORR.

14.15.3 Diagram



14.15.4 Fields

Field	Function
31-18 —	Reserved
17 —	Reserved
16 DBMT	DECO Bypass Memory Translation. If DBMT=1, when this DECO is under the direct control of software bus transactions will assert the BMT signal as 1, indicating that the bus transaction should bypass memory translation (if PL = 1). If PL=0, the value of the BMT signal is ignored, and memory translation is performed by the system MMU.
15 DPL	DECO Privilege Level. If DPL=1, when this DECO is under the direct control of software bus transactions will assert the PL signal as 1, indicating that BMT=1 is permitted for this bus transaction.
14-7	Reserved

Table continues on the next page...

DECO Availability Register (DAR)

Field	Function
—	
6-0 D_ICID	ICID. This field defines the ICID value asserted for DMA transactions when this DECO is under the direct control of software. This field is not writable when virtualization mode is enabled.

14.16 DECO Availability Register (DAR)

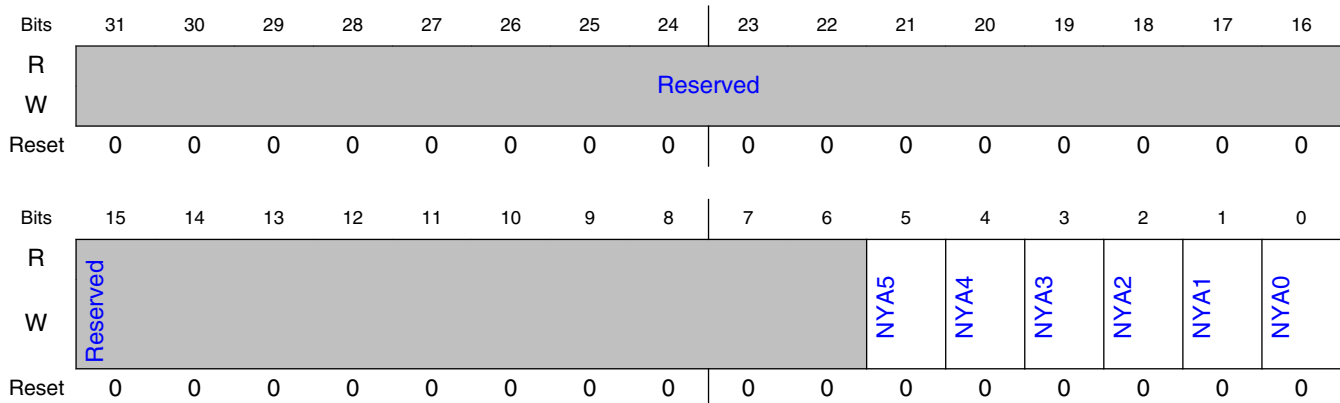
14.16.1 Offset

Register	Offset
DAR	120h

14.16.2 Function

The DECO Availability Register can be used to determine whether DECOs are hung. If software writes a 1 to the DECO's NYA field, the corresponding DECO will clear that bit whenever the DECO is, or becomes, available. The bit can be polled to determine if the DECO is completing jobs. If STOP is asserted in the DEBUG Control register, the DECO Availability Register cannot be written. While STOP is asserted DECO Availability provides a status for whether each DECO is stopped or available. Any bit that is zero in this case indicates a DECO that is still running and needs to stop before the Debug Control Register can assert STOP_ACK.

14.16.3 Diagram



14.16.4 Fields

Field	Function
31-6 —	Reserved. Always 0.
5 NYA5	This bit is set by software to start polling for the availability of DECO 5. This bit will be reset when DECO 5 is or becomes, available.
4 NYA4	This bit is set by software to start polling for the availability of DECO 4. This bit will be reset when DECO 4 is or becomes, available.
3 NYA3	This bit is set by software to start polling for the availability of DECO 3. This bit will be reset when DECO 3 is or becomes, available.
2 NYA2	This bit is set by software to start polling for the availability of DECO 2. This bit will be reset when DECO 2 is or becomes, available.
1 NYA1	This bit is set by software to start polling for the availability of DECO 1. This bit will be reset when DECO 1 is or becomes, available.
0 NYA0	This bit is set by software to start polling for the availability of DECO 0. This bit will be reset when DECO 0 is or becomes, available.

14.17 DECO Reset Register (DRR)

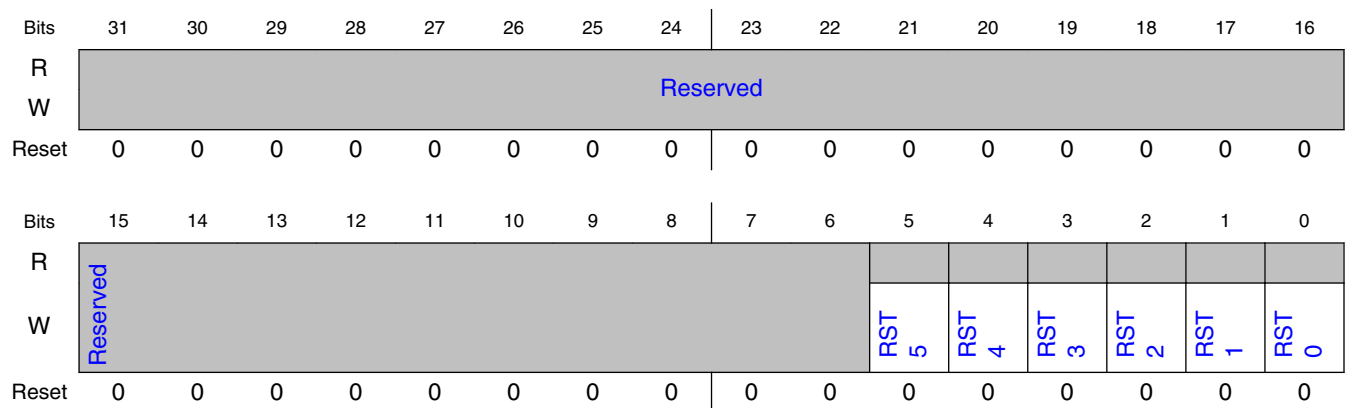
14.17.1 Offset

Register	Offset
DRR	124h

14.17.2 Function

The DECO Reset Register can be used to force a soft reset of one or more DECOs with appropriate status write back (error code 20h). Note that using this can result in lost DMA transactions and/or memory leaks. In some cases a soft reset of a DECO will not result in a status write back, or may not free a hung DECO. If a hung DECO cannot be freed via a soft DECO reset, then a software SEC reset or a POR will be required.

14.17.3 Diagram



14.17.4 Fields

Field	Function
31-6 —	Reserved. Always 0.
5 RST5	Software writes a 1 to this bit to initiate a soft reset of DECO 5. This bit is self-clearing after one clock cycle.
4 RST4	Software writes a 1 to this bit to initiate a soft reset of DECO 4. This bit is self-clearing after one clock cycle.

Table continues on the next page...

Field	Function
3 RST3	Software writes a 1 to this bit to initiate a soft reset of DECO 3. This bit is self-clearing after one Clock cycle.
2 RST2	Software writes a 1 to this bit to initiate a soft reset of DECO 2. This bit is self-clearing after one Clock cycle.
1 RST1	Software writes a 1 to this bit to initiate a soft reset of DECO 1. This bit is self-clearing after one clock cycle.
0 RST0	Software writes a 1 to this bit to initiate a soft reset of DECO 0. This bit is self-clearing after one clock cycle.

14.18 Adaptive Resource Reservation Register (ARRR)

14.18.1 Offset

Register	Offset
ARRR	1F0h

14.18.2 Function

The ARR register is used to specify whether DECOs are reserved for use by the AIOP Interface, and if so, how many DECOs are reserved. Note that this register is not writable if any DECOs are owned by the register interface (i.e. non-zero value in the DECO Request Register). The number of DECOs reserved for processing jobs from the AIOP Interface reduces the number of DECOs available for processing jobs submitted via the Job Rings or the QI. When AR_EN=1 and there are no uncompleted AI jobs in SEC, AR_MIN DECOs are reserved for AI jobs. Whenever SEC dequeues an AI job for processing, the job counter (JBCOUNT) is incremented. Whenever a DECO writes back status for a completed AI job, JBCOUNT is decremented. If $(JBCOUNT + AR_DELTA) < AR_MIN$, AR_MIN DECOs are reserved for AI jobs. If $AR_MIN < (JBCOUNT + AR_DELTA) \leq AR_MAX$, $JBCOUNT + AR_DELTA$ DECOs are reserved for AI jobs. If $(JBCOUNT + AR_DELTA) > AR_MAX$, AR_MAX DECOs are reserved for AI jobs. It is possible to configure the system with AR_MIN = AR_MAX and AR_DELTA = 0. This would reserve AR_MAX DECOs at all times for AI jobs. It is also possible to

Adaptive Resource Reservation Register (ARRR)

configure the system with $AR_MIN = 0$ and $AR_DELTA = AR_MAX$. This would reserve no DECOs for AI when there are no AI jobs running, but always reserve AR_MAX DECOs when one or more AI jobs are running.

The same algorithm is used to reserve holding tanks.

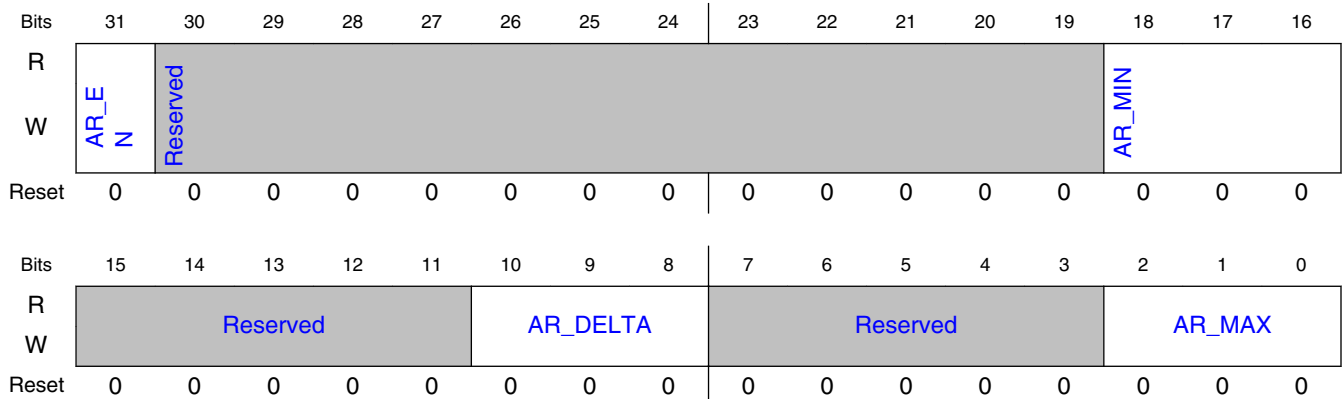
When adaptive resource reservation is disabled, scheduling jobs to holding tanks and DECOs operates in round robin fashion among AI jobs, QI jobs, job ring jobs, and RTIC jobs. When adaptive resource reservation is enabled, scheduling jobs to holding tanks and DECOs operates as follows. Assume HCOUNT is the number of HTs with AI jobs, DCOUNT is the number of DECOs with AI jobs, and RCOUNT is the number of HTs and DECOs to reserve for AI jobs. To allow resource sharing between AI and the other job sources, the assignment of jobs to holding tanks follows this scheme:

- If not enough HTs or DECOs are running AI jobs ($HCOUNT < RCOUNT$ or $DCOUNT < RCOUNT$) and there is an AI job ready, the next HT gets that AI job.
- Else if HCOUNT plus the number of empty HTs exceeds RCOUNT, and if DCOUNT plus the number of empty DECOs exceeds RCOUNT, and if there is a JQ or QI or RTIC job ready, the next HT gets a job from the JQ/QI/RTIC round-robin.
- Else if there is an AI job, the next HT gets that AI job.

When holding tanks are assigned to DECOs:

- If the available DECO has a pending job, it gets its oldest pending job from any job source.
- Else if DCOUNT plus the number of empty DECOs is less than RCOUNT, the oldest non-pending AI job is assigned to the available DECO (or the DECO is left empty if there is no AI job).
- Else if DCOUNT plus the number of empty DECOs is greater than RCOUNT, the oldest non-pending job from any source is assigned to the available DECO.

14.18.3 Diagram



14.18.4 Fields

Field	Function
31 AR_EN	If AR_EN = 0 : Adaptive resource reservation disabled. Other fields of ARRR are ignored. If AR_EN = 1 : Adaptive resource reservation enabled.
30-19 —	Reserved. Always 0.
18-16 AR_MIN	If AR_EN = 1, this field specifies the minimum number of DECOs to be reserved for use by AI, in accordance with the algorithm given in the register description. Software must take care not to write a value greater than AR_MAX or greater than the number of implemented DECOs. Note: If AR_MIN is set to the number of implemented DECOs, no Job Ring, QI or RTIC jobs can run. If AR_DELTA or AR_MAX is set to the number of implemented DECOs and AR_MIN is set to some smaller value and there are any uncompleted AI jobs, no Job Ring, QI or RTIC jobs can run.
15-11 —	Reserved. Always 0.
10-8 AR_DELTA	If AR_EN = 1, this field specifies the number of additional DECOs to be reserved for use by AI, in accordance with the algorithm given in the register description. Software must take care not to write a value greater than AR_MAX or greater than the number of implemented DECOs.
7-3 —	Reserved. Always 0.
2-0 AR_MAX	If AR_EN = 1, this field specifies the maximum number of DECOs to be reserved for use by AI, in accordance with the algorithm given in the register description. Software must take care not to write a value greater than the number of implemented DECOs.

14.19 DMA Control Register (DMAC - DMA_CTRL)

14.19.1 Offset

Register	Offset	Description
DMAC	204h	DMA_CTRL is an alias of DMAC. DMAC is provided only for backward compatibility. Its usage is deprecated.
DMA_CTRL	504h	For new software the usage of DMA_CTRL is preferred over DMAC.

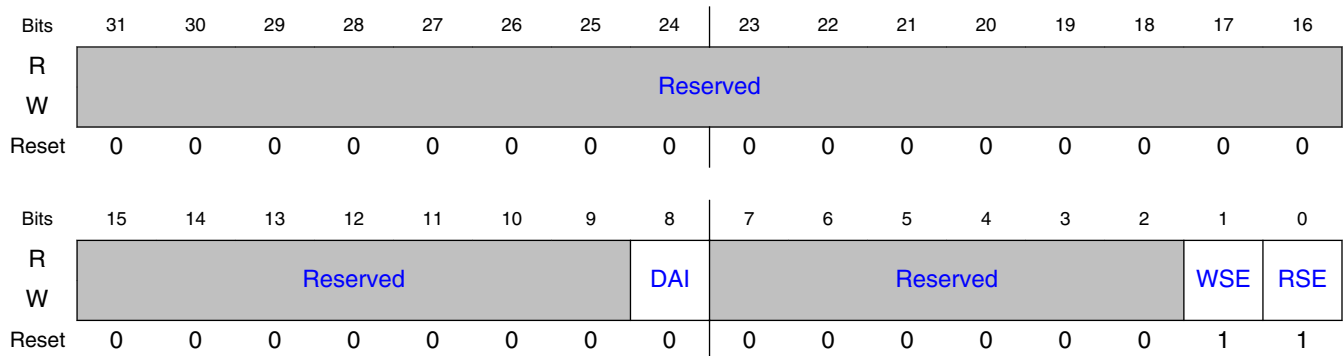
14.19.2 Function

The DMA Control register is used to configure the behavior of the DMA engines. Register settings apply to all DMAs.

NOTE

Note that for backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. Usage of the legacy addresses in the range 00240..002CF is deprecated.

14.19.3 Diagram



14.19.4 Fields

Field	Function						
31-9 —	Reserved						
8 DAI	<p>DMA Access Index. Configuration and status information for all of the DMA engines are accessed through the registers listed below. Writing a DMA number to DAI selects the DMA engine whose information is accessed through these registers. Note that these registers are aliased to the legacy registers located at 240h ... 2FBh. That is, when DAI=0 the register at 510h (DMAx_AID_7_4_MAP) is an alias of the register at 240h (DMA0_AIDL_MAP) and the register at 514h (DMAx_AID_3_0_MAP) is an alias of the register at 240h (DMA0_AIDH_MAP), etc. But when DAI=1 the register at 510h (DMAx_AID_7_4_MAP) is an alias of the register at 280h (DMA1_AIDL_MAP) and the register at 514h (DMAx_AID_3_0_MAP) is an alias of the register at 288h (DMA1_AIDH_MAP), etc. The usage of the registers in the range 240h ... 2FBh is deprecated in favor of the registers listed in the table below.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Address</th> <th>Register</th> </tr> </thead> <tbody> <tr> <td>510h</td> <td>DMAx_AID_7_4_MAP</td> </tr> <tr> <td>514h</td> <td>DMAx_AID_3_0_MAP</td> </tr> </tbody> </table>	Address	Register	510h	DMAx_AID_7_4_MAP	514h	DMAx_AID_3_0_MAP
Address	Register						
510h	DMAx_AID_7_4_MAP						
514h	DMAx_AID_3_0_MAP						

Table continues on the next page...

Field	Function	
	Address	Register
	518h	DMAx_AID_15_12_MAP
	51Ch	DMAx_AID_11_8_MAP
	524h	DMAx_AID_15_0_EN
	530h	DMAx_ARTC_CTRL
	534h	DMAx_ARTC_LC
	538h	DMAx_ARTC_SC
	53Ch	DMAx_ARTC_LAT
	540h	DMAx_AWTC_CTL
	544h	DMAx_AWTC_LC
	548h	DMAx_AWTC_SC
	54Ch	DMAx_AWTC_LAT
	7-2 —	Reserved
1 WSE	Write Safe Enable. When WSE=1, the Write Safe feature is enabled for the DMA engine(s). When WSE=0, the Write Safe feature is disabled for the DMA engine(s). (see DMA interface write-safe transactions)	
0 RSE	Read Safe Enable. When RSE=1, the Read Safe feature is enabled for the DMA engine(s). When RSE=0, the Read Safe feature is disabled for the DMA engine(s). (see DMA read-safe transactions)	

14.20 Peak Bandwidth Smoothing Limit Register (PBSL)

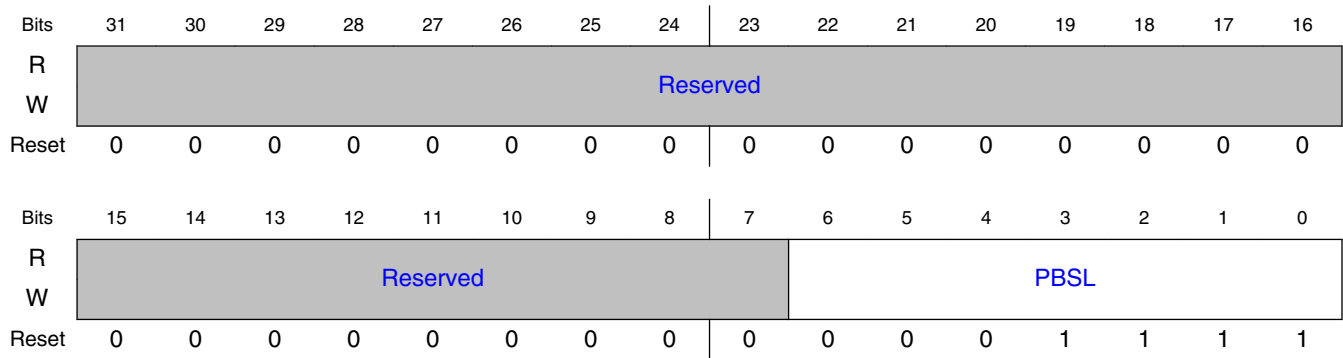
14.20.1 Offset

Register	Offset
PBSL	220h

14.20.2 Function

The Peak Bandwidth Smoothing Limit Register is used to limit the maximum bus bandwidth consumed by SEC.

14.20.3 Diagram



14.20.4 Fields

Field	Function
31-7 —	Reserved
6-0 PBSL	Whenever the number of outstanding AXI read bursts exceeds the value programmed in this field, the AI and QI and Job Rings will be prevented from issuing additional AXI reads. While the number of outstanding AXI read burst exceeds the PBSL, DECOs may continue to issue additional AXI read requests. The AI and QI and Job Rings will be allowed to issue additional AXI reads only when the number of outstanding AXI read bursts drops to, or below, the PBSL. Throttling the AXI reads reduces the SEC peak bandwidth on the AXI bus, and giving priority to DECOs improves SEC performance when SEC is heavily loaded with jobs. A limit of PBSL=0 indicates that no AXI read smoothing will be performed.

14.21 DMAa_AIDL_MAP_MS (DMA0_AIDL_MAP_MS - DMA1_AIDL_MAP_MS)

14.21.1 Offset

For a = 0 to 1:

Register	Offset	Description
DMAa_AIDL_MAP_MS	240h + (a × 40h)	Mapping for DMA AXI IDs 7 ... 4

14.21.2 Function

The four registers `DMAn_AIDL_MAP_MS`, `DMAn_AIDL_MAP_LS`, `DMAn_AIDM_MAP_MS` and `DMAn_AIDM_MAP_LS` show the mapping of AXI transaction IDs to SEC internal blocks. These assignments are made via hardwired signals and are SoC-specific. The value of each 8-bit field indicates the internal ID of the SEC block that will use the AXI ID corresponding to the field. For example, `AID2BID=00001000` means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMA_n AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMAs. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

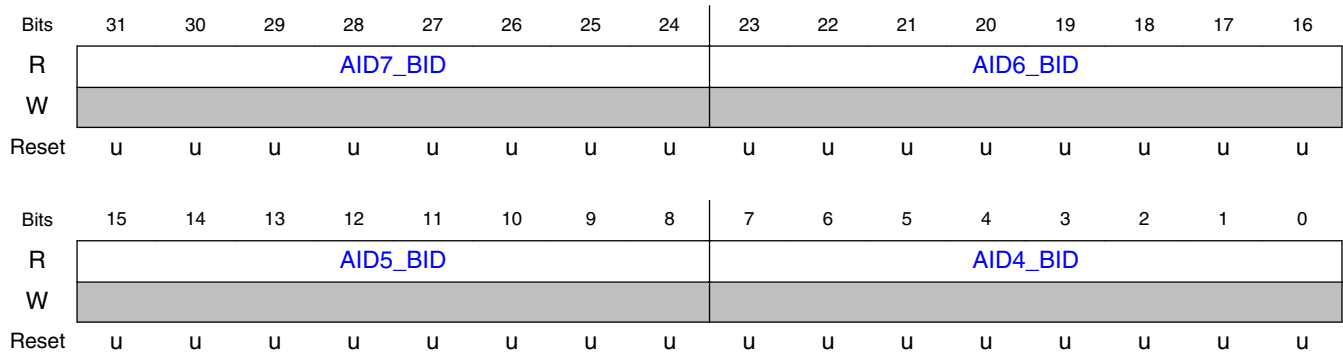
The SEC internal block IDs are encoded as follows:

Internal Block ID	Internal Logic Block
00000001b	Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.)
00000100b	Burst Buffer
00000101b	AIOP Interface
00000111b	Queue Interface
00001000b	DECO0
00001001b	DECO1
00001010b	DECO2
00001011b	DECO3
00001100b	DECO4
00001101b	DECO5
All other values are reserved.	

NOTE

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

14.21.3 Diagram



14.21.4 Fields

Field	Function
31-24 AID7_BID	This field shows the SEC Block ID that uses AXI ID 7.
23-16 AID6_BID	This field shows the SEC Block ID that uses AXI ID 6.
15-8 AID5_BID	This field shows the SEC Block ID that uses AXI ID 5.
7-0 AID4_BID	This field shows the SEC Block ID that uses AXI ID 4.

14.22 DMAa_AIDL_MAP_LS (DMA0_AIDL_MAP_LS - DMA1_AIDL_MAP_LS)

14.22.1 Offset

For a = 0 to 1:

Register	Offset	Description
DMAa_AIDL_MAP_LS	244h + (a × 40h)	Mapping for DMA AXI IDs 3 ... 0

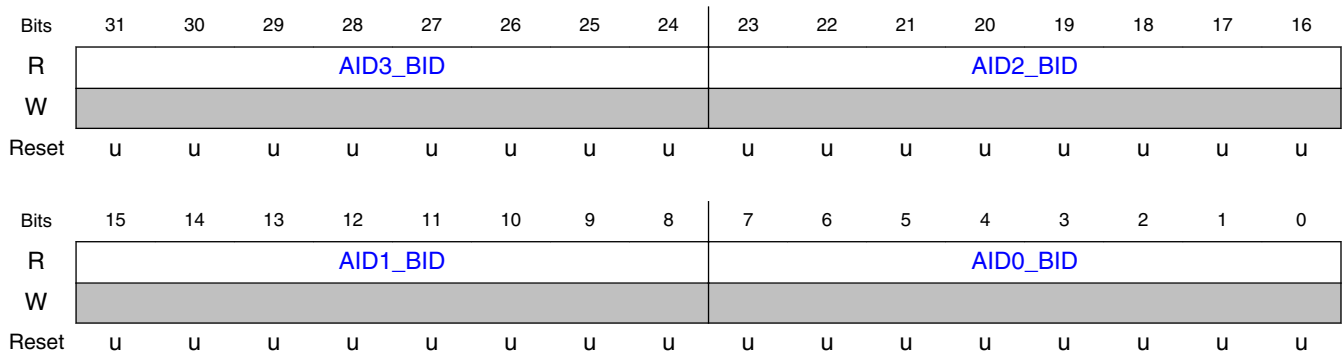
14.22.2 Function

The four registers `DMAn_AIDL_MAP_MS`, `DMAn_AIDL_MAP_LS`, `DMAn_AIDM_MAP_MS` and `DMAn_AIDM_MAP_LS` show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register `DMAn_AID_7_4_MAP` for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.22.3 Diagram



14.22.4 Fields

Field	Function
31-24 <code>AID3_BID</code>	This field shows the SEC Block ID that uses AXI ID 3.
23-16 <code>AID2_BID</code>	This field shows the SEC Block ID that uses AXI ID 2.
15-8 <code>AID1_BID</code>	This field shows the SEC Block ID that uses AXI ID 1.
7-0 <code>AID0_BID</code>	This field shows the SEC Block ID that uses AXI ID 0.

14.23 DMAa_AIDM_MAP_MS (DMA0_AIDM_MAP_MS - DMA1_AIDM_MAP_MS)

14.23.1 Offset

For a = 0 to 1:

Register	Offset	Description
DMAa_AIDM_MAP_MS	248h + (a × 40h)	Mapping for DMA AXI IDs 15 ... 12

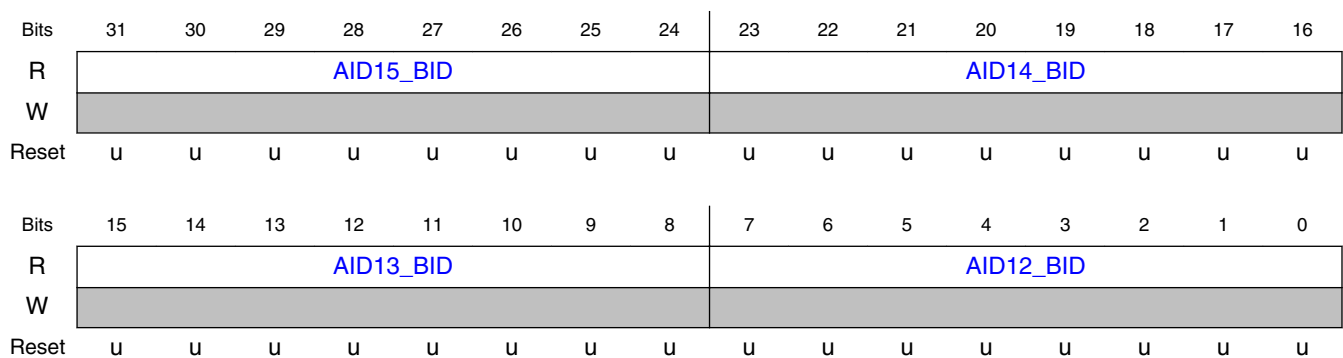
14.23.2 Function

The four registers DMA_n_AIDL_MAP_MS, DMA_n_AIDL_MAP_LS, DMA_n_AIDM_MAP_MS and DMA_n_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_n_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.23.3 Diagram



14.23.4 Fields

Field	Function
31-24 AID15_BID	This field shows the SEC Block ID that uses AXI ID 15.
23-16 AID14_BID	This field shows the SEC Block ID that uses AXI ID 14.
15-8 AID13_BID	This field shows the SEC Block ID that uses AXI ID 13.
7-0 AID12_BID	This field shows the SEC Block ID that uses AXI ID 12.

14.24 DMAa_AIDM_MAP_LS (DMA0_AIDM_MAP_LS - DMA1_AIDM_MAP_LS)

14.24.1 Offset

For a = 0 to 1:

Register	Offset	Description
DMAa_AIDM_MAP_LS	24Ch + (a × 40h)	Mapping for DMA AXI IDs 11 ... 8

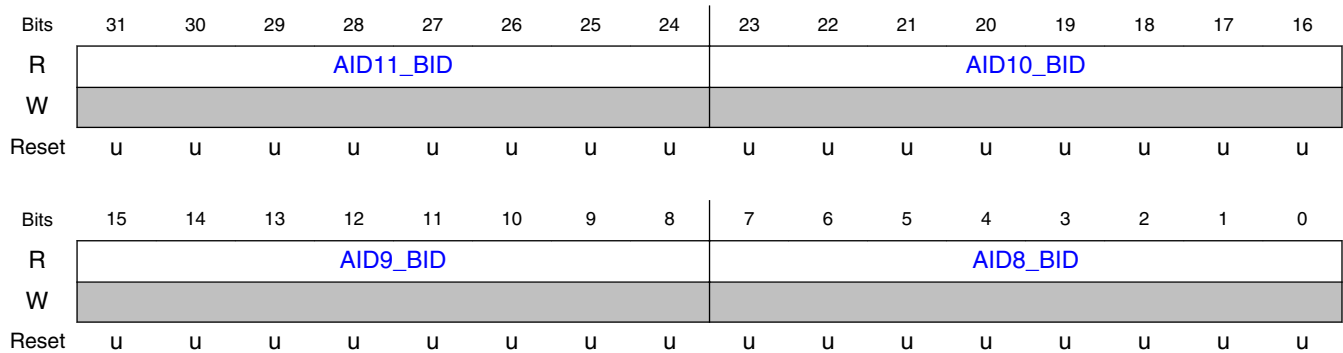
14.24.2 Function

The four registers DMA_n_AIDL_MAP_MS, DMA_n_AIDL_MAP_LS, DMA_n_AIDM_MAP_MS and DMA_n_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_n_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.24.3 Diagram



14.24.4 Fields

Field	Function
31-24 AID11_BID	This field shows the SEC Block ID that uses AXI ID 11.
23-16 AID10_BID	This field shows the SEC Block ID that uses AXI ID 10.
15-8 AID9_BID	This field shows the SEC Block ID that uses AXI ID 9.
7-0 AID8_BID	This field shows the SEC Block ID that uses AXI ID 8.

14.25 DMAf AXI ID Enable Register (DMA0_AID_ENB - DMA1_AID_ENB)

14.25.1 Offset

For f = 0 to 1:

Register	Offset	Description
DMAf_AID_ENB	250h + (f × 40h)	Use of this register alias is deprecated. Instead, use the register alias DMA_X_AID_15_0_EN

14.25.2 Function

The DMA AXI ID Enable register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each SEC internal connected to it. The assignments are made using the lowest-numbered, available IDs.

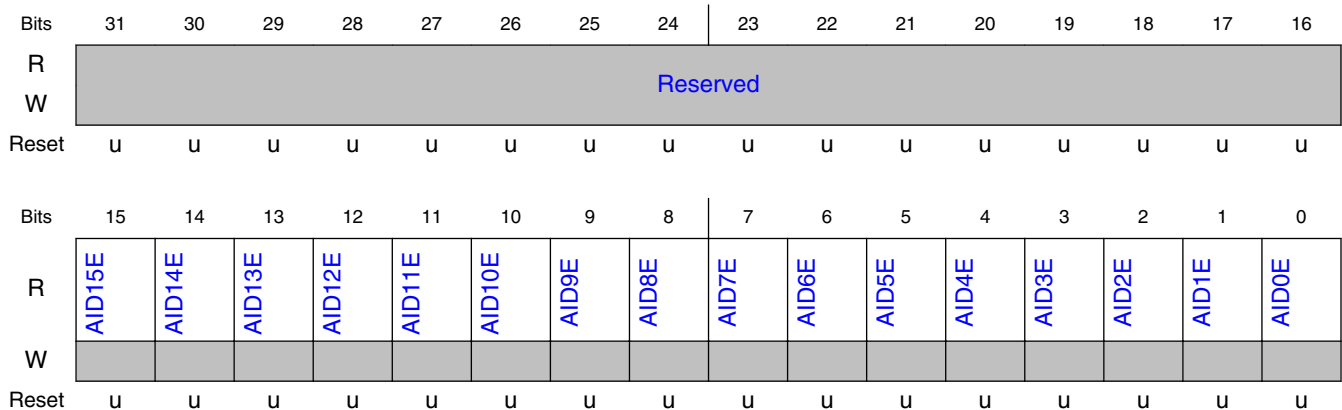
NOTE

Note that for backward compatibility the same register is readable at two different addresses.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.25.3 Diagram



14.25.4 Fields

Field	Function
31-16 —	Reserved.
15 AID15E	If AID15E=1 then AXI ID 15 is enabled for this DMA engine.
14 AID14E	If AID14E=1 then AXI ID 14 is enabled for this DMA engine.

Table continues on the next page...

DMAa AXI Read Timing Check Register (DMA0_ARD_TC - DMA1_ARD_TC)

Field	Function
13 AID13E	If AID13E=1 then AXI ID 13 is enabled for this DMA engine.
12 AID12E	If AID12E=1 then AXI ID 12 is enabled for this DMA engine.
11 AID11E	If AID11E=1 then AXI ID 11 is enabled for this DMA engine.
10 AID10E	If AID10E=1 then AXI ID 10 is enabled for this DMA engine.
9 AID9E	If AID9E=1 then AXI ID 9 is enabled for this DMA engine.
8 AID8E	If AID8E=1 then AXI ID 8 is enabled for this DMA engine.
7 AID7E	If AID7E=1 then AXI ID 7 is enabled for this DMA engine.
6 AID6E	If AID6E=1 then AXI ID 6 is enabled for this DMA engine.
5 AID5E	If AID5E=1 then AXI ID 5 is enabled for this DMA engine.
4 AID4E	If AID4E=1 then AXI ID 4 is enabled for this DMA engine.
3 AID3E	If AID3E=1 then AXI ID 3 is enabled for this DMA engine.
2 AID2E	If AID2E=1 then AXI ID 2 is enabled for this DMA engine.
1 AID1E	If AID1E=1 then AXI ID 1 is enabled for this DMA engine.
0 AID0E	If AID0E=1 then AXI ID 0 is enabled for this DMA engine.

14.26 DMAa AXI Read Timing Check Register (DMA0_ARD_TC - DMA1_ARD_TC)

14.26.1 Offset

For a = 0 to 1:

Register	Offset
DMAa_ARD_TC	260h + (a × 40h)

14.26.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

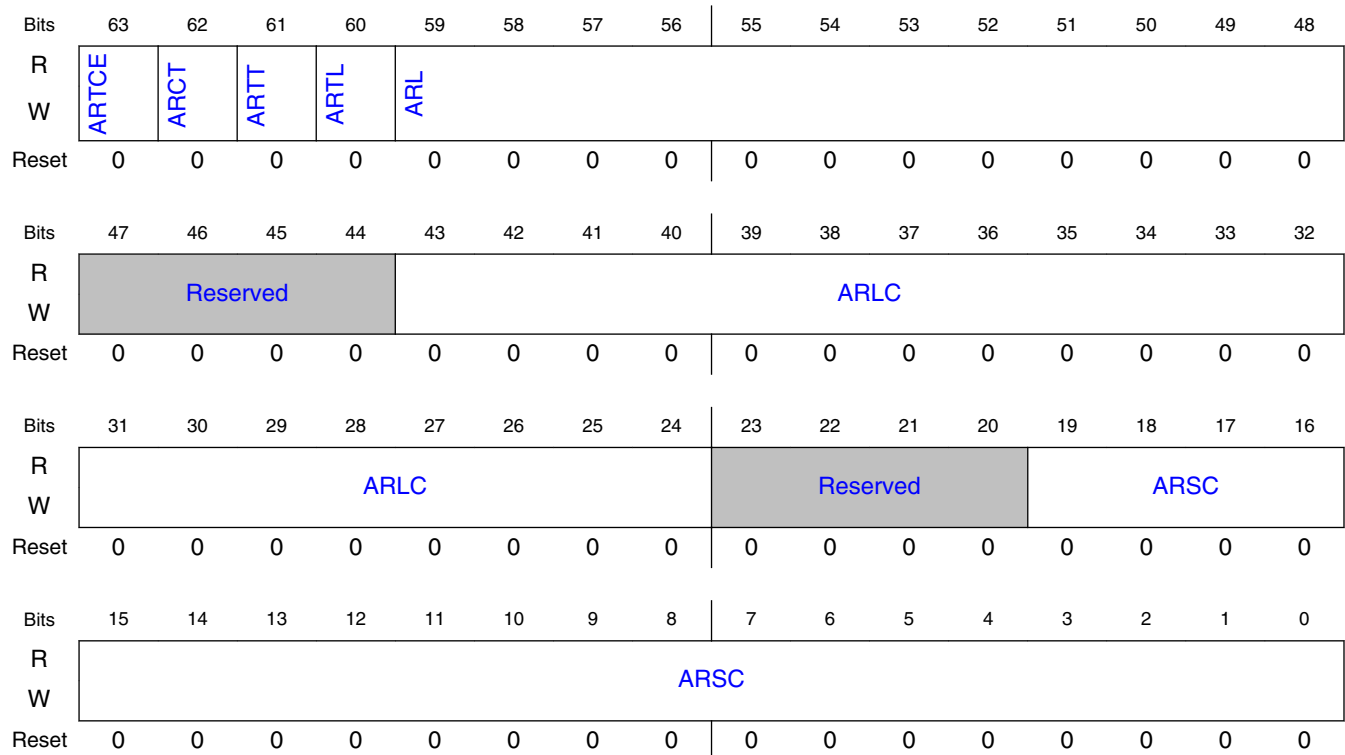
- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMA_n_ARD_TC register located in the address range 00260..002EF. Writing to either register affects the corresponding fields in the other register. But note that some fields in the DMA_n_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or DMA_X_ARTC_SC register. The preferred registers are located in the address range 00530..005DF. The use of the DMA_n_ARD_TC register located in the address range 00260..002EF is deprecated.

14.26.3 Diagram



14.26.4 Fields

Field	Function
63 ARTCE	AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMA _n _ARD_TC and SARL in DMA _n _SARL and ARL in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC and SARL in DMA_X_ARTC_LC are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only. Note that writing ARTCE in either DMA _n _ARD_TC or DMA_X_ARTC_CTL has the same effect.
62 ARCT	AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMA _n _ARD_TC, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMA _n _SARL and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMA _n _ARD_TC and DMA _n _SARL or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMA_X_SARL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
61 ARTT	AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The count can optionally be modified to count until the last beat of data instead of the first by setting the ARTL (AXI Read Timer Last) bit. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMA _n _ARD_TC register is aliased to bit field ARTT in the DMA_X_ARTC_CTL register, i.e. writing to either ARTT bit field alters the ARTT value in the other register.

Table continues on the next page...

Field	Function
60 ARTL	AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement. 0b - A read transaction counter measurement is stopped when the first beat of data arrives 1b - A read transaction counter measurement is stopped when the last beat of data arrives
59-48 ARL	AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMA _n _SARL /DMA_X_ARTC_LAT. This field is writeable only when ARTCE=0. Note that bit field ARL in the DMA _n _ARD_TC register is aliased to bit field ARL in the DMA_X_ARTC_CTL register, i.e. writing to either ARL bit field alters the ARL value in the other register.
47-44 —	Reserved
43-24 ARLC	AXI Read Late Count. This field is incremented whenever the AXI Read Timer equals or exceeds the AXI Read Limit. AXI read timing checks are suspended when ARLC=FFFFh. This field is writeable only when ARTCE=0.
23-20 —	Reserved
19-0 ARSC	AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFh. This field is writeable only when ARTCE=0.

14.27 DMA_f Read Timing Check Latency Register (DMA0_ARD_LAT - DMA1_ARD_LAT)

14.27.1 Offset

For f = 0 to 1:

Register	Offset
DMA _f _ARD_LAT	26Ch + (f × 40h)

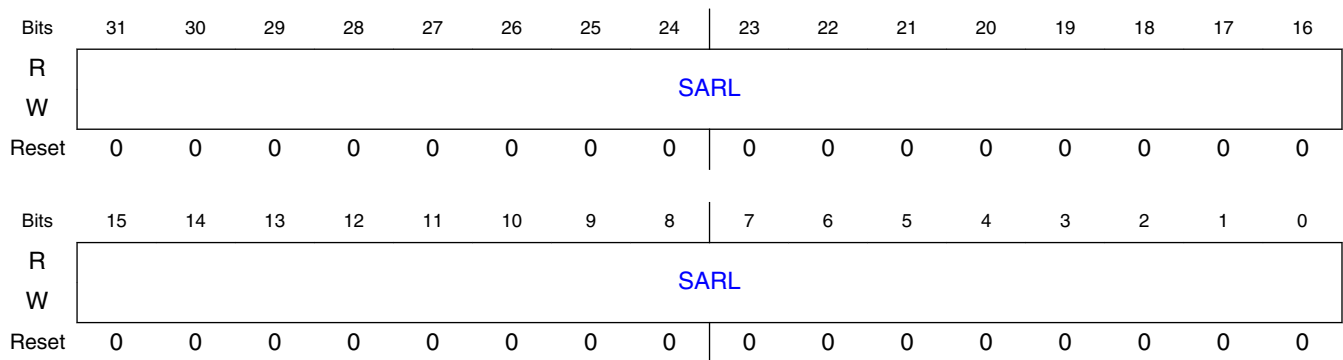
14.27.2 Function

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

NOTE

Note that the DMA_n_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMA_n_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

14.27.3 Diagram



14.27.4 Fields

Field	Function
31-0 SARL	Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMA _n _SARL. This field is writeable only when ARTCE=0.

14.28 DMAa AXI Write Timing Check Register (DMA0_AWR_TC - DMA1_AWR_TC)

14.28.1 Offset

For a = 0 to 1:

Register	Offset
DMAa_AWR_TC	270h + (a × 40h)

14.28.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the write response. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

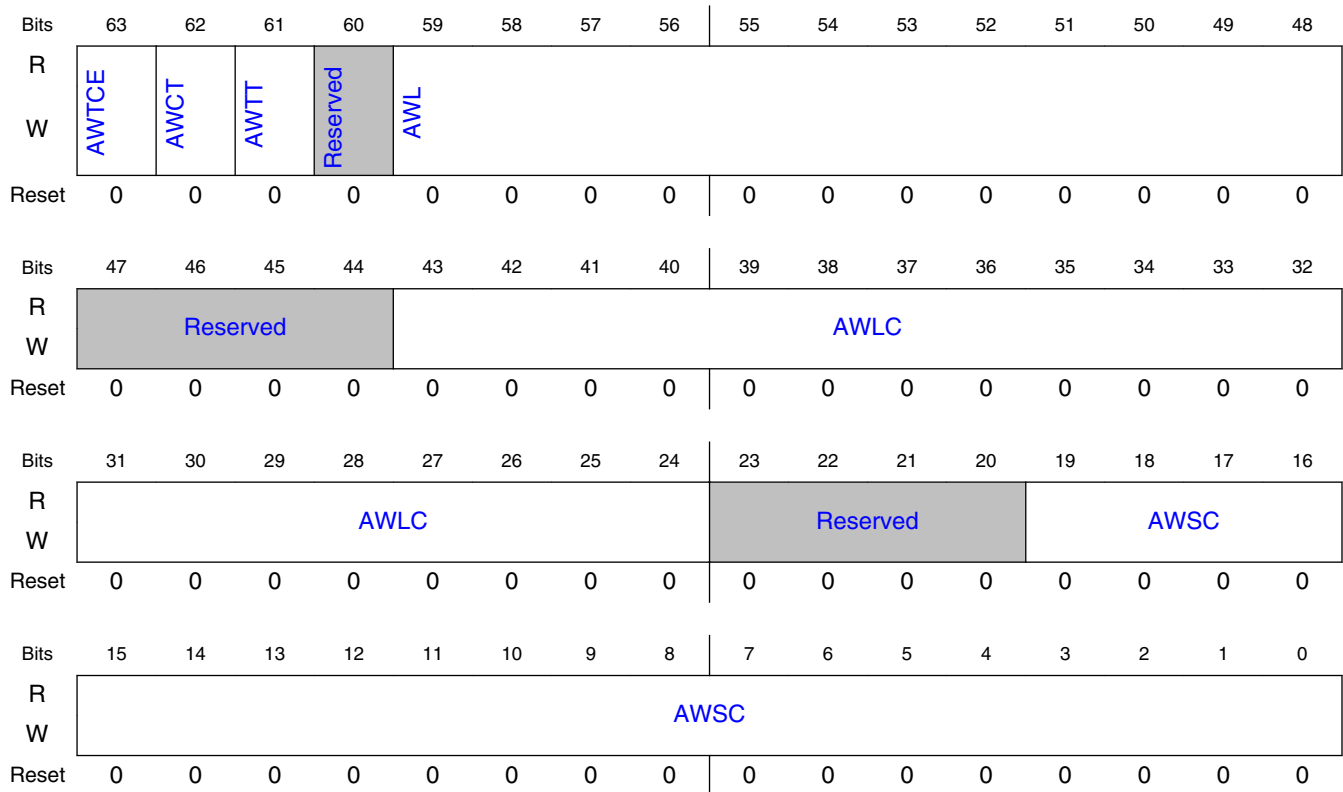
- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

NOTE

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMAa_AWR_TC register located in the address range 00270..002EF. Writing to either register affects the corresponding fields in the other register. But note that some fields in the DMAa_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_TC_SAWL register or DMA_X_AWTC_SC register. The preferred registers are located in the address range 00540..005DF. The use of the DMAa_AWR_TC register located in the address range 00270..002EF is deprecated.

14.28.3 Diagram



14.28.4 Fields

Field	Function
63 AWTCE	AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMA _n _AWR_TC and SAWL in DMA _n _AWL and AWL in DMA_X_AWTC_CTL, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC and SAWL in DMA_X_TC_SAWL are writeable. When AWTCE=1, AXI write timing checks are enabled and these fields are read-only. Note that writing AWTCE in either DMA _n _AWR_TC or DMA_X_AWTC_CTL has the same effect.
62 AWCT	AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMA _n _AWR_TC, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC, and SAWL in DMA _n _AWL and DMA_X_TC_SAWL, are not cleared when timing checks are enabled and when timing checks resume after writing DMA _n _AWR_TC and DMA _n _AWL or DMA_X_TC_SAWL, DMA_X_AWTC_SC and DMA_X_AWL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
61 AWTT	AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field AWTT in the DMA _n _AWR_TC register is aliased to bit field AWTT in the DMA_X_AWTC_CTL register, i.e. writing to either AWTT bit field alters the AWTT value in the other register.

Table continues on the next page...

Field	Function
60 —	Reserved
59-48 AWL	AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. If the latency equals or exceeds the AXI Write Limit, the Write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMA _n _AWL /DMA_X_AWTC_LAT. This field is writeable only when ARTCE=0. Note that bit field AWL in the DMA _n _AWR_TC register is aliased to bit field AWL in the DMA_X_AWTC_CTL register, i.e. writing to either AWL bit field alters the AWL value in the other register.
47-44 —	Reserved
43-24 AWLC	AXI Write Late Count. This field is incremented whenever the AXI Write Timer equals or exceeds the AXI Write Limit. AXI write timing checks are suspended when AWLC=FFFFFh. This field is writeable only when AWTCE=0.
23-20 —	Reserved
19-0 AWSC	AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFFh. This field is writeable only when AWTCE=0.

14.29 DMA_f Write Timing Check Latency Register (DMA0_AWR_LAT - DMA1_AWR_LAT)

14.29.1 Offset

For f = 0 to 1:

Register	Offset
DMA _f _AWR_LAT	27Ch + (f × 40h)

14.29.2 Function

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

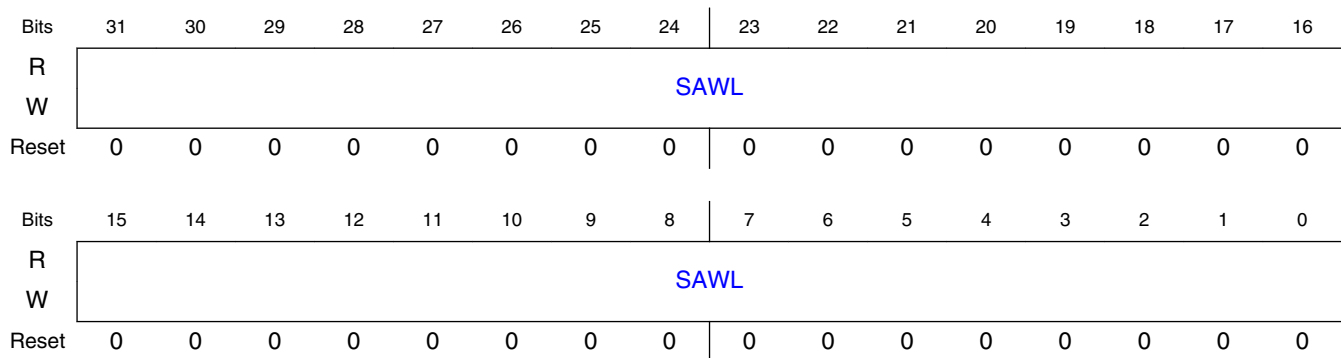
NOTE

Note that the DMA_n_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMA_n_AWL

Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)

register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

14.29.3 Diagram



14.29.4 Fields

Field	Function
31-0 SAWL	Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMA _n _AWL. This field is writeable only when AWTCE=0.

14.30 Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)

14.30.1 Offset

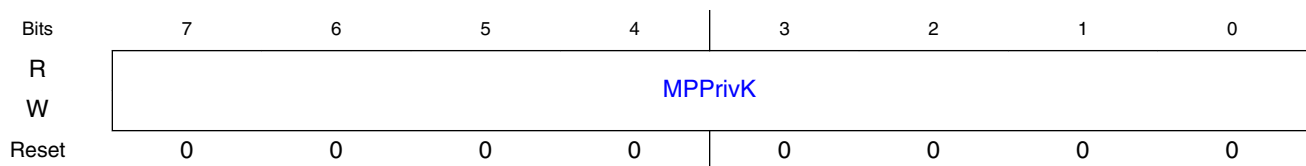
For a = 0 to 63:

Register	Offset
MPPKR _a	300h + (a × 1h)

14.30.2 Function

The Manufacturing Protection Private Key register is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to all this by signing a message using the private key stored in the MPPKR. Software running on the SOC then sends this attestation message to the OEM's server. The OEM's server can verify that all this information is correct by verifying the signature over the signed message. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

14.30.3 Diagram



14.30.4 Fields

Field	Function
7-0 MPPrivK	MPPrivK. The 512-bit Manufacturing Protection Private Key.

14.31 Manufacturing Protection Message Register (MPMR0 - MPMR31)

14.31.1 Offset

For a = 0 to 31:

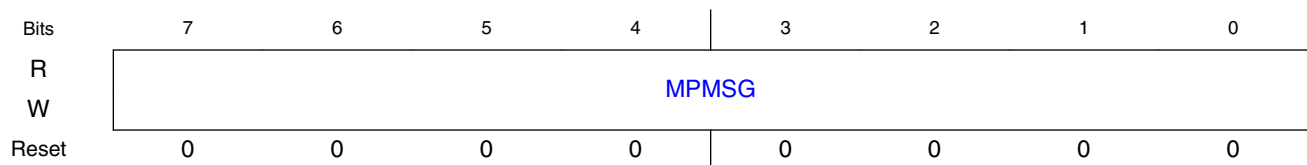
Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)

Register	Offset
MPMRa	380h + (a × 1h)

14.31.2 Function

The Manufacturing Protection Message register is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to this by signing a message using the private key stored in the MPPKR. The message is composed, in part, of the content of the MPMR. The value in the MPMR is written by ROM-resident boot software. The value normally includes the hash of the public key used to verify the signature over the signed code image. Software running on the SOC then sends this signed message to the OEM's server. The OEM's server can confirm that all this information is correct by verifying the signature over the signed message. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

14.31.3 Diagram



14.31.4 Fields

Field	Function
7-0 MPMSG	Holds 256 bits of message data that will be prepended to the input data to the MPSIGN operation. When accessed via the register bus this should be treated as a byte array (although it must be accessed as eight 32-bit words).

14.32 Manufacturing Protection Test Register (MPTESTR0 - MPTSTR31)

14.32.1 Offset

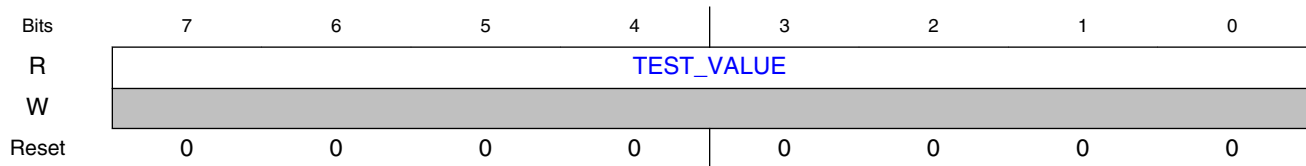
For a = 0 to 31:

Register	Offset
MPTSTRa	3C0h + (a × 1h)

14.32.2 Function

The Manufacturing Protection TEST register is used only for hardware verification.

14.32.3 Diagram



14.32.4 Fields

Field	Function
7-0 TEST_VALUE	TEST_VALUE. When accessed via the register bus this should be treated as a byte array with the first byte in offset 3C0h (although it must be accessed as eight 32-bit words).

14.33 Job Descriptor Key Encryption Key Register (JDKE KR0 - JDKEKR7)

14.33.1 Offset

For a = 0 to 7:

Register	Offset
JDKEKR _a	400h + (a × 4h)

14.33.2 Function

The Job Descriptor Key Encryption Key Register contains the Job Descriptor Key Encryption Key (JDKEK), which can be used when encrypting or decrypting Black Keys (see [Black keys](#)). Since Black Keys are not intended for storage of keys across SOC power cycles (SEC's Blob mechanism is intended for this purpose), the value in the JDKEKR is not preserved at SOC power-down. Instead, a new 256-bit secret value is loaded into the JDKEKR from the RNG for use during the new power-on session. The JDKEK is loaded by executing a special descriptor, which can be run in any security mode. (see [RNG functional description](#))

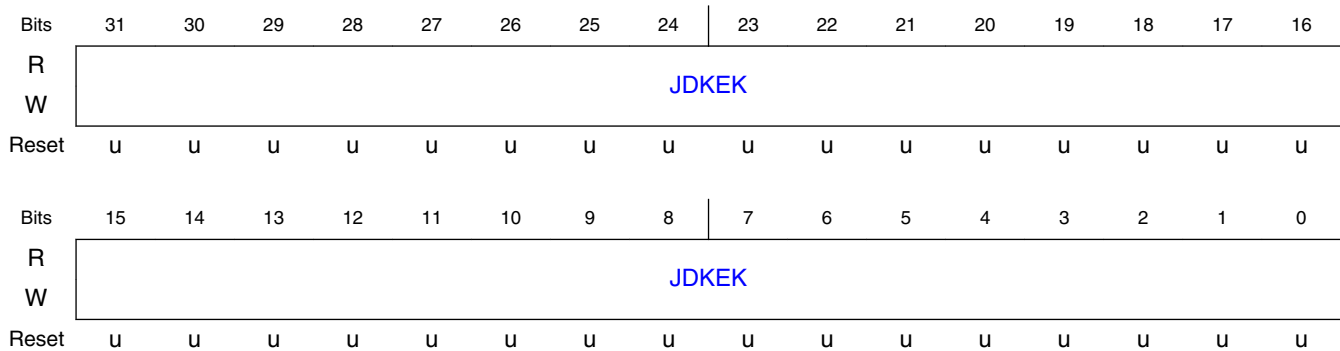
Note that the Secure Mode/Trusted Mode value in JDKEKR is not available when SEC is in Non-secure Mode because the only possible transitions between Trusted Mode or Secure Mode that lead to Non-secure Mode cause SEC to pass through Fail Mode, and JDKEKR is cleared whenever SEC enters Fail Mode.

The Job Descriptor Key Encryption Key is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 400h.

NOTE

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG. The JDKEKR cannot be read (i.e. reading returns all zeros) or written from the register bus while SEC is in Secure Mode or Trusted Mode, but JDKEKR can be read and written while SEC is in Non-secure Mode.

14.33.3 Diagram



14.33.4 Fields

Field	Function
31-0 JDKEK	The 256-bit Job Descriptor Key Encryption Key used to encrypt and decrypt Black Keys.

14.34 Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)

14.34.1 Offset

For a = 0 to 7:

Register	Offset
TDKEKR _a	420h + (a × 4h)

14.34.2 Function

The Trusted Descriptor Key Encryption Key Register contains the Trusted Descriptor Key Encryption Key (TDKEK), which can be used when encrypting or decrypting Black Keys (see [Black keys](#)). The TDKEKR operates exactly like the JDKEKR, except that the TDKEKR is usable only by Trusted Descriptors. This allows Trusted Descriptors to

Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)

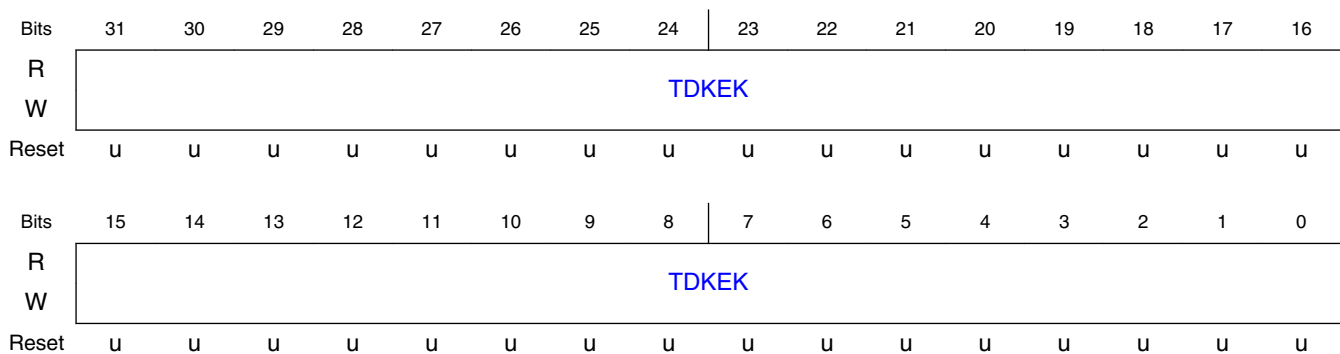
protect particularly sensitive keys from access by Job Descriptors. Trusted Descriptors can use either the JDKEKR or the TDKEKR, so Trusted Descriptors can be used to derive non-Trusted Black keys for use by Job Descriptors from Trusted Black Keys that contain master secrets. The Trusted Descriptor Key Encryption Key is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 420h.

A new 256-bit secret value is loaded into the TDKEKR from the RNG for use during the new power-on session. The TDKEK is loaded by executing a special descriptor, which can be run in any security mode. (see [RNG functional description](#))

NOTE

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG. The TDKEKR cannot be read (i.e. reading returns all zeros) or written from the register bus while SEC is in Secure Mode or Trusted Mode, but TDKEKR can be read and written while SEC is in Non-secure Mode.

14.34.3 Diagram



14.34.4 Fields

Field	Function
31-0 TDKEK	The 256-bit Trusted Descriptor Key Encryption Key used to encrypt and decrypt Black Keys.

14.35 Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)

14.35.1 Offset

For a = 0 to 7:

Register	Offset
TDSKR _a	440h + (a × 4h)

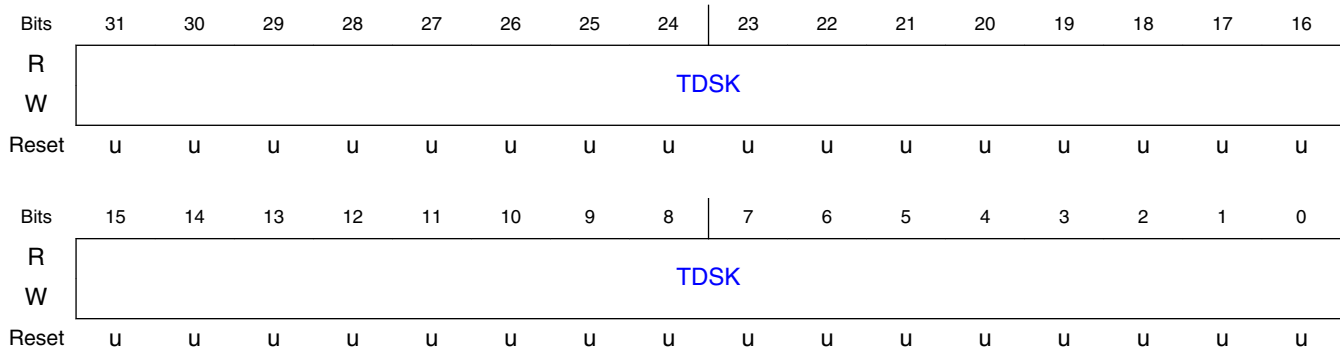
14.35.2 Function

The Trusted Descriptor Signing Key Register contains the TDSK, which is used to generate and verify signatures on Trusted Descriptors. The TDSKR is loaded in the same fashion as the JDKEK. The TDSK is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 440h.

NOTE

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG. The TDSKR cannot be read (i.e. reading returns all zeros) or written from the register bus while SEC is in Secure Mode or Trusted Mode, but TDSKR can be read and written while SEC is in Non-secure Mode.

14.35.3 Diagram



14.35.4 Fields

Field	Function
31-0 TDSK	The 256-bit Trusted Descriptor Signing Key used to sign and verify Trusted Descriptors.

14.36 Secure Key Nonce Register (SKNR)

14.36.1 Offset

Register	Offset	Description
SKNR	4E0h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR) .

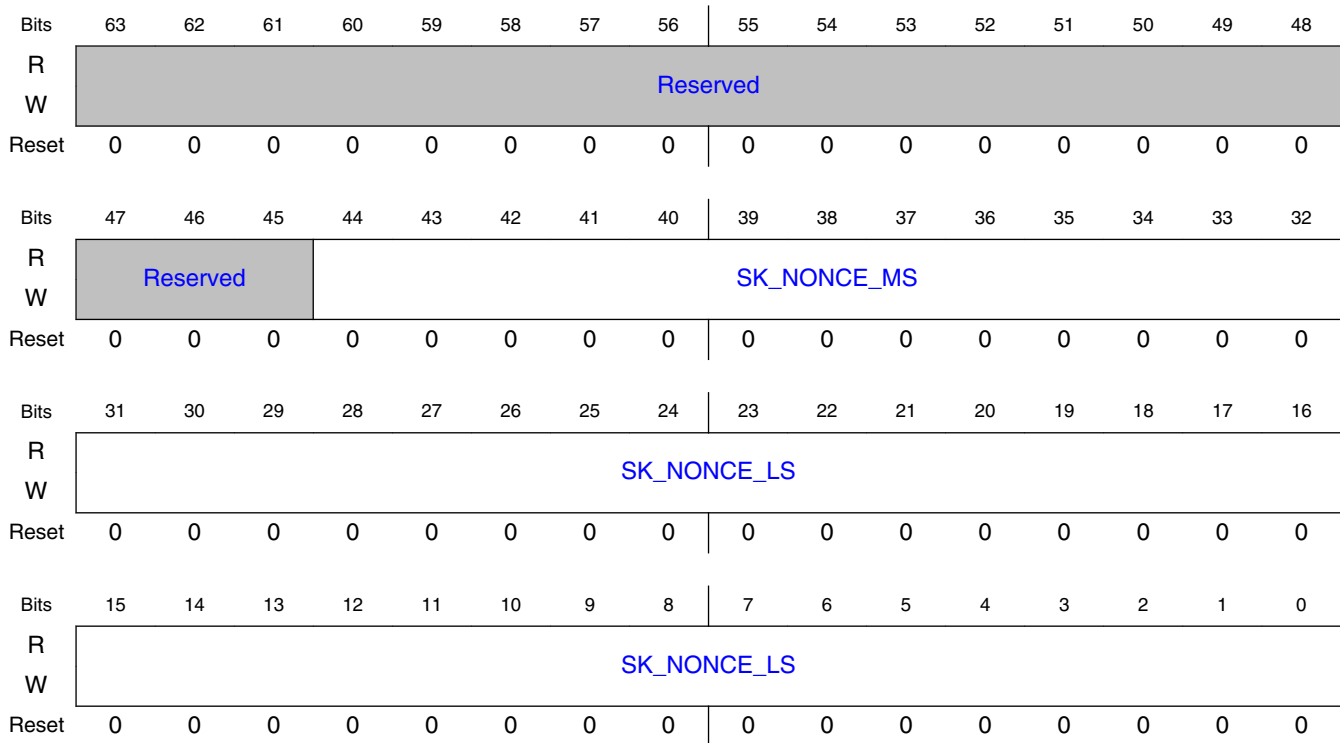
14.36.2 Function

The Secure Key Nonce Register holds a nonce value that is used for Black Key encryption. To ensure that a nonce is never reused during a power-on session, the nonce is used and incremented whenever a Black Key is encrypted using AES-CCM encryption (i.e. a FIFO STORE with EKT=1, of the PKHA E Memory, the Class 1 Key Register or the Class 2 Key Register.) The SKNR is reset to all 0 at power on reset or when SEC enters Fail mode, but it is not reset at software-initiated SEC reset. Since the SKNR holds more than 32 bits, it is accessed over the IP bus as two 32-bit words. Note that two or more DECOs could encrypt a Black Key at the same time and since all DECOs share the same SKNR, the DECO identification number is concatenated with the value in the SKNR to ensure that each Black Key is encrypted using a unique nonce.

NOTE

This register is writable only when SEC is in NonSecure mode.

14.36.3 Diagram



14.36.4 Fields

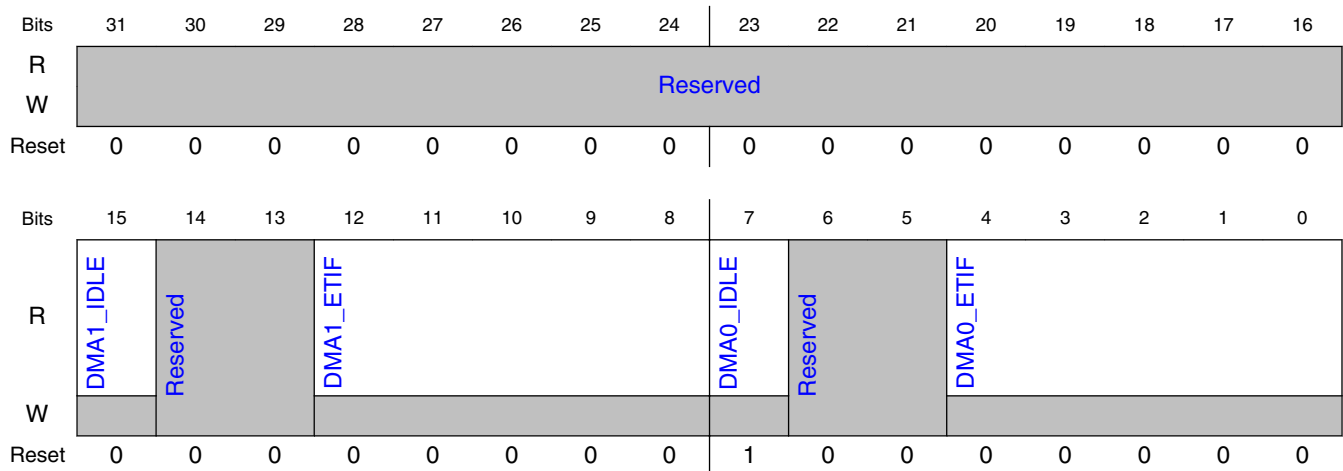
Field	Function
63-45 —	Reserved
44-32 SK_NONCE_MS	Secure Key Nonce - Most Significant Bits. This field holds the 13 most-significant bits of the auto-incrementing secure key nonce field. See the description of the SK_NONCE_LS field for more information.
31-0 SK_NONCE_LS	Secure Key Nonce - Least Significant Bits. This field holds the 32 least-significant bits of the auto-incrementing secure key nonce field. The actual nonce value that is used during AES-CCM encryption of Black Keys consists of the SK_NONCE_MS, the SK_NONCE_LS and the DECO_ID. The DECO_ID indicates which DECO is requesting a nonce. If two or more DECOs happen to request a nonce during the same clock cycle, this ensures that each DECO receives a different nonce.

14.37 DMA Status Register (DMA_STA)

14.37.1 Offset

Register	Offset
DMA_STA	50Ch

14.37.2 Diagram



14.37.3 Fields

Field	Function
31-16 —	Reserved
15 DMA1_IDLE	DMA1 is idle. DMA1’s command queue is empty.
14-13 —	Reserved
12-8 DMA1_ETIF	DMA1 External Transactions in Flight. DMA1_ETIF indicates the number of transactions DMA1 currently has in flight on SEC’s external AXI bus.
7 DMA0_IDLE	DMA0 is idle. DMA0’s command queue is empty.
6-5 —	Reserved
4-0 DMA0_ETIF	DMA0 External Transactions in Flight. DMA0_ETIF indicates the number of transactions the DMA0 engine currently has in flight on SEC’s external AXI bus.

14.38 DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)

14.38.1 Offset

Register	Offset	Description
DMA_X_AID_7_4_MAP	510h	Mapping for DMA AXI IDs 7 ... 4

14.38.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. These assignments are made via hardwired signals and are SOC-specific. The value of each 8-bit field indicates the internal ID of the SEC block that will use the AXI ID corresponding to the field. For example, AID2BID=00001000 means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMA_X_AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMAs. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

The SEC internal block IDs are encoded as follows:

Internal Block ID	Internal Logic Block
00000001b	Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.)
00000100b	Burst Buffer
00000101b	AIOP Interface
00000111b	Queue Interface
00001000b	DECO0
00001001b	DECO1
00001010b	DECO2
00001011b	DECO3
00001100b	DECO4
00001101b	DECO5
All other values are reserved.	

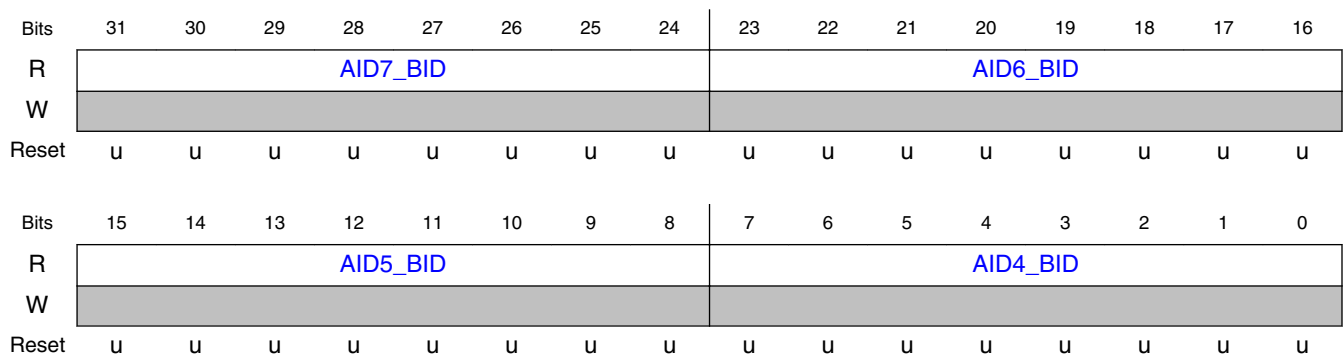
NOTE

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.38.3 Diagram



14.38.4 Fields

Field	Function
31-24 AID7_BID	This field shows the SEC Block ID that uses AXI ID 7.
23-16 AID6_BID	This field shows the SEC Block ID that uses AXI ID 6.
15-8 AID5_BID	This field shows the SEC Block ID that uses AXI ID 5.
7-0 AID4_BID	This field shows the SEC Block ID that uses AXI ID 4.

14.39 DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)

14.39.1 Offset

Register	Offset	Description
DMA_X_AID_3_0_MAP	514h	Mapping for DMA AXI IDs 3 ... 0

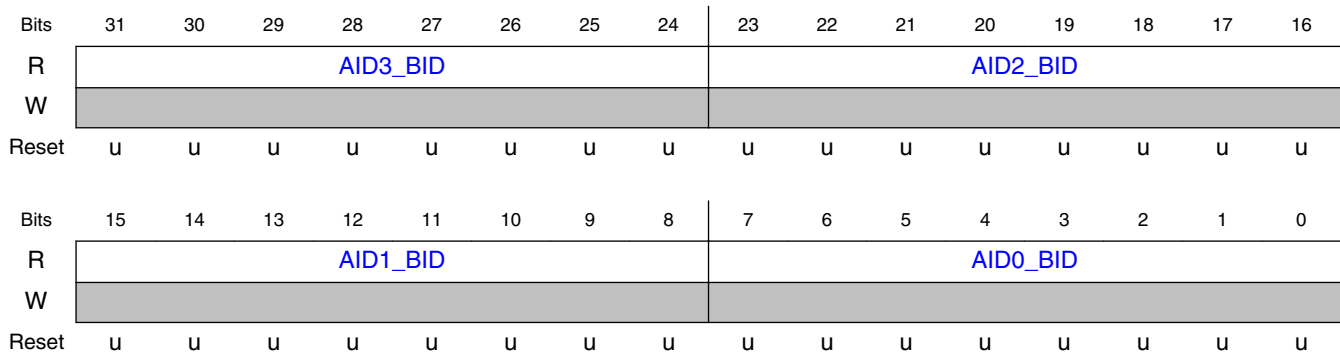
14.39.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.39.3 Diagram



14.39.4 Fields

Field	Function
31-24 AID3_BID	This field shows the SEC Block ID that uses AXI ID 3.
23-16 AID2_BID	This field shows the SEC Block ID that uses AXI ID 2.
15-8	This field shows the SEC Block ID that uses AXI ID 1.

Table continues on the next page...

DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)

Field	Function
AID1_BID	
7-0 AID0_BID	This field shows the SEC Block ID that uses AXI ID 0.

14.40 DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)

14.40.1 Offset

Register	Offset	Description
DMA_X_AID_15_12_MAP	518h	Mapping for DMA AXI IDs 15 ... 12

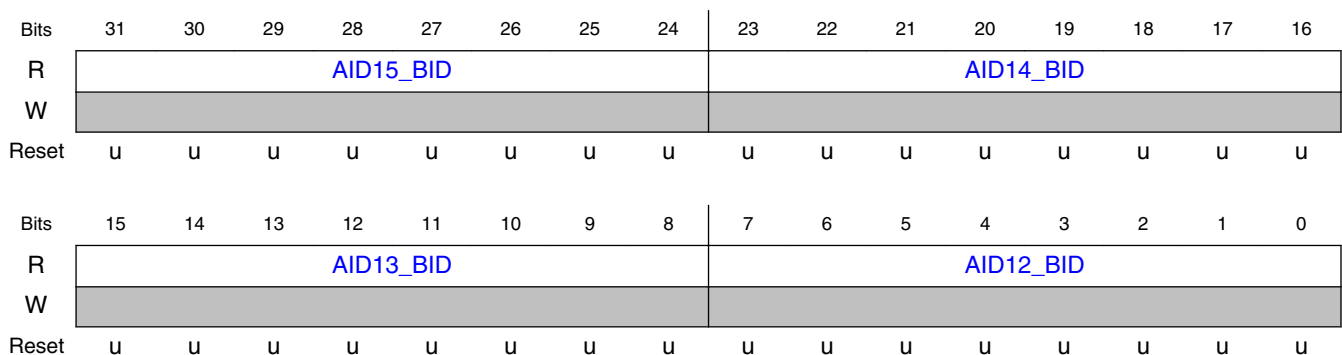
14.40.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.40.3 Diagram



14.40.4 Fields

Field	Function
31-24 AID15_BID	This field shows the SEC Block ID that uses AXI ID 15.
23-16 AID14_BID	This field shows the SEC Block ID that uses AXI ID 14.
15-8 AID13_BID	This field shows the SEC Block ID that uses AXI ID 13.
7-0 AID12_BID	This field shows the SEC Block ID that uses AXI ID 12.

14.41 DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)

14.41.1 Offset

Register	Offset	Description
DMA_X_AID_11_8_MAP	51Ch	Mapping for DMA AXI IDs 11 ... 8

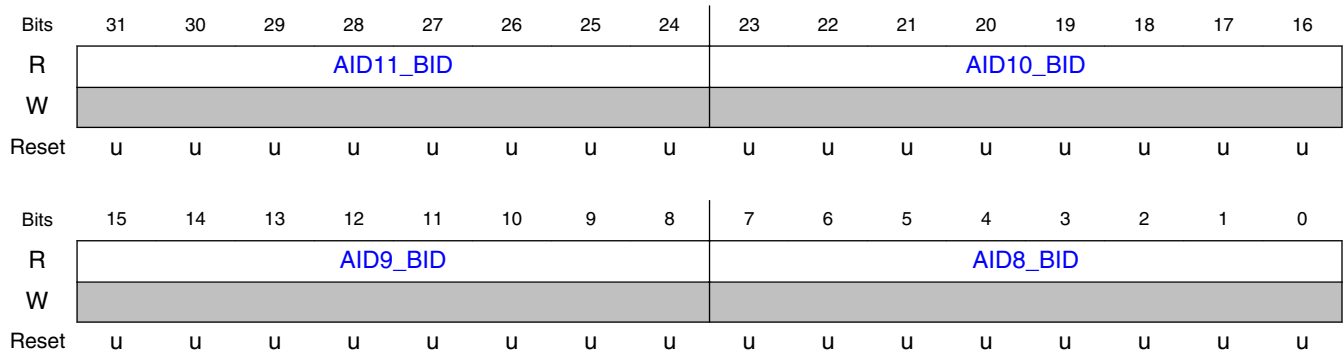
14.41.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

NOTE

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

14.41.3 Diagram



14.41.4 Fields

Field	Function
31-24 AID11_BID	This field shows the SEC Block ID that uses AXI ID 11.
23-16 AID10_BID	This field shows the SEC Block ID that uses AXI ID 10.
15-8 AID9_BID	This field shows the SEC Block ID that uses AXI ID 9.
7-0 AID8_BID	This field shows the SEC Block ID that uses AXI ID 8.

14.42 DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN)

14.42.1 Offset

Register	Offset	Description
DMA_X_AID_15_0_EN	524h	<p>For new software DMA_X_AID_15_0_EN (address 510h) should be used rather than DMA_0..1_AID_ENB (address 250h).</p> <p>NOTE: The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.</p>

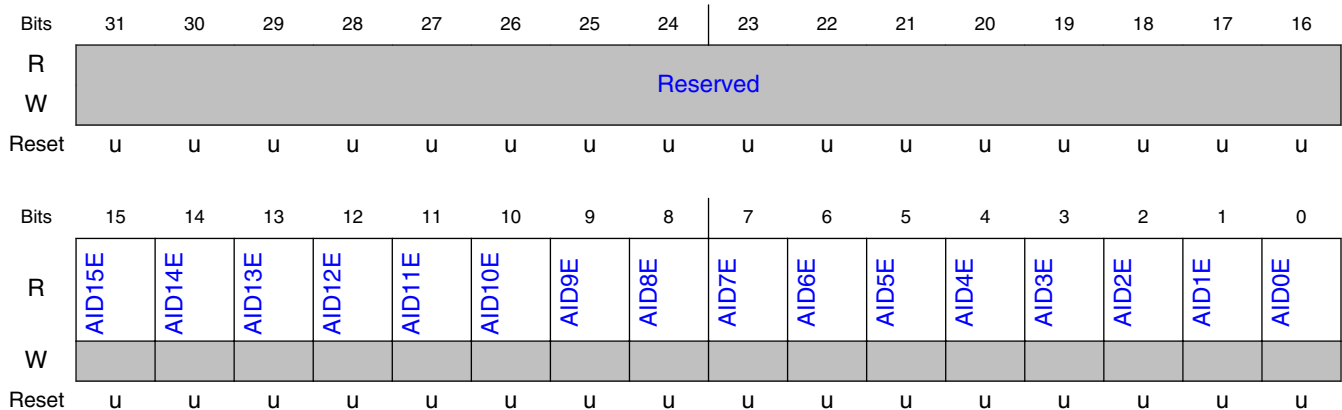
14.42.2 Function

The DMA_X_AID_15_0_EN register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each SEC internal connected to it. The assignments are made using the lowest-numbered, available IDs. The AXI transaction IDs for both are read via the DMA_X_AID_15_0_EN register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose AXI IDs are currently displayed in DMA_X_AID_15_0_EN.

NOTE

Note that for backward compatibility the same information is readable at two different addresses, 250h and 524h. The 250h address is deprecated.

14.42.3 Diagram



14.42.4 Fields

Field	Function
31-16 —	Reserved.
15 AID15E	If AID15E=1 then AXI ID 15 is enabled for this DMA engine.

Table continues on the next page...

DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)

Field	Function
14 AID14E	If AID14E=1 then AXI ID 14 is enabled for this DMA engine.
13 AID13E	If AID13E=1 then AXI ID 13 is enabled for this DMA engine.
12 AID12E	If AID12E=1 then AXI ID 12 is enabled for this DMA engine.
11 AID11E	If AID11E=1 then AXI ID 11 is enabled for this DMA engine.
10 AID10E	If AID10E=1 then AXI ID 10 is enabled for this DMA engine.
9 AID9E	If AID9E=1 then AXI ID 9 is enabled for this DMA engine.
8 AID8E	If AID8E=1 then AXI ID 8 is enabled for this DMA engine.
7 AID7E	If AID7E=1 then AXI ID 7 is enabled for this DMA engine.
6 AID6E	If AID6E=1 then AXI ID 6 is enabled for this DMA engine.
5 AID5E	If AID5E=1 then AXI ID 5 is enabled for this DMA engine.
4 AID4E	If AID4E=1 then AXI ID 4 is enabled for this DMA engine.
3 AID3E	If AID3E=1 then AXI ID 3 is enabled for this DMA engine.
2 AID2E	If AID2E=1 then AXI ID 2 is enabled for this DMA engine.
1 AID1E	If AID1E=1 then AXI ID 1 is enabled for this DMA engine.
0 AID0E	If AID0E=1 then AXI ID 0 is enabled for this DMA engine.

14.43 DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)

14.43.1 Offset

Register	Offset
DMA_X_ARTC_CTL	530h

14.43.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

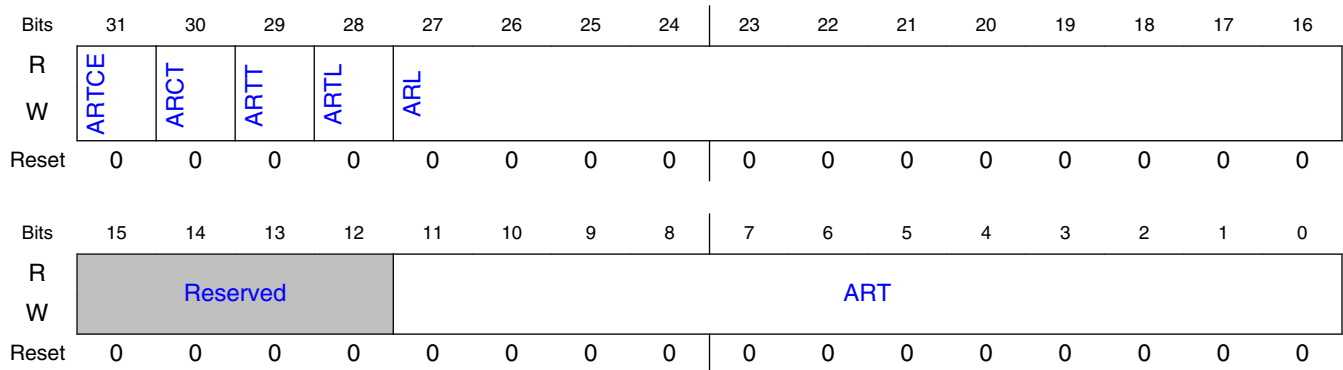
- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the DMA_X AXI Read Latency Register or DMA_X Read Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMA_n_ARD_TC register located in the address range 00260..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMA_n_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or the DMA_X_ARTC_SC register. The read timing checks for both are controlled via the DMA_X_ARTC_CTL register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose read timing checks are currently controlled via DMA_X_ARTC_CTL.

14.43.3 Diagram



14.43.4 Fields

Field	Function
31 ARTCE	AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMA_X_ARTC and SARL in DMA _n _ARD_LAT and ARL in DMA_X_ARTC_CTL, ARLC in DMA0_ARTC_LC, ARSC in DMA0ARTC_SC and SARL in DMA0ARL_LAT are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only. NOTE: Note that writing ARTCE in either DMA_X_ARTC_B or DMA_X_ARTC_CTL has the same effect.
30 ARCT	AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMA _n _ARD_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMA _n _ARD_TC and DMA _n _ARD_LAT or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMA _n _ARD_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
29 ARTT	AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMA_X_ARTC_CTL register is aliased to bit field ARTT in the DMA _n _ARD_TC register, i.e. writing to either ARTT bit field alters the ARTT value in the other register.
28 ARTL	AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement. 0b - A read transaction counter measurement is stopped when the first beat of data arrives 1b - A read transaction counter measurement is stopped when the last beat of data arrives
27-16 ARL	AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMA _n _ARD_LAT/DMA_X_ARTC_LAT. This field is

Table continues on the next page...

Field	Function
	writeable only when ARTCE=0. Note that bit field ARL in the DMA_X_ARTC_CTL register is aliased to bit field ARL in the DMA _n _ARD_TC register, i.e. writing to either ARL bit field alters the ARL value in the other register.
15-12 —	Reserved
11-0 ART	AXI Read Timer. The number of AXI clock cycles from the latest external AXI read address transaction initiated by this DMA to the beginning of the corresponding read data transaction. This field is writeable only when ARTCE=0.

14.44 DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)

14.44.1 Offset

Register	Offset
DMA_X_ARTC_LC	534h

14.44.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the ARTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

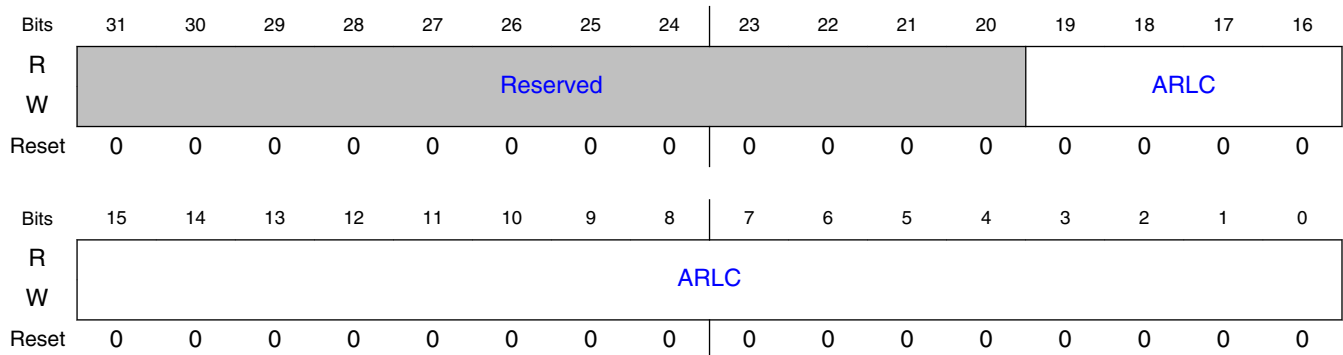
NOTE

Note that the DMA_X_ARTC_LC register provides functionality similar to the AXI Read Timing Late Check fields in the DMA_n_ARD_TC register located in the address range 00260..002EF, but the fields have been rearranged. Usage of

DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)

the DMA_n_ARD_TC register is deprecated. The ARLC values for both are read via the DMA_X_ARTC_LC register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose ARLC value is currently displayed via DMA_X_ARTC_LC.

14.44.3 Diagram



14.44.4 Fields

Field	Function
31-20 —	Reserved
19-0 ARLC	AXI Read Late Count. This field is incremented each time that the ART field exceeds the ARL field in the DMA_X AXI Read Timing Check Control Register. AXI read timing checks are suspended when ARLC=FFFFFh. Note that this field is an alias of the ARLC field in the DMA _n _ARD_TC Register. Reading or writing the ARLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMA _n _ARD_TC/DMA_X_ARTC_TC[ARTCE]=0, the ARLC bit field in DMA _n _ARD_TC and the DMA_X_ARTC_LC register are writeable. When ARTCE=1, AXI read timing checks are enabled and the ARLC bit fields are read-only.

14.45 DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)

14.45.1 Offset

Register	Offset
DMA_X_ARTC_SC	538h

14.45.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

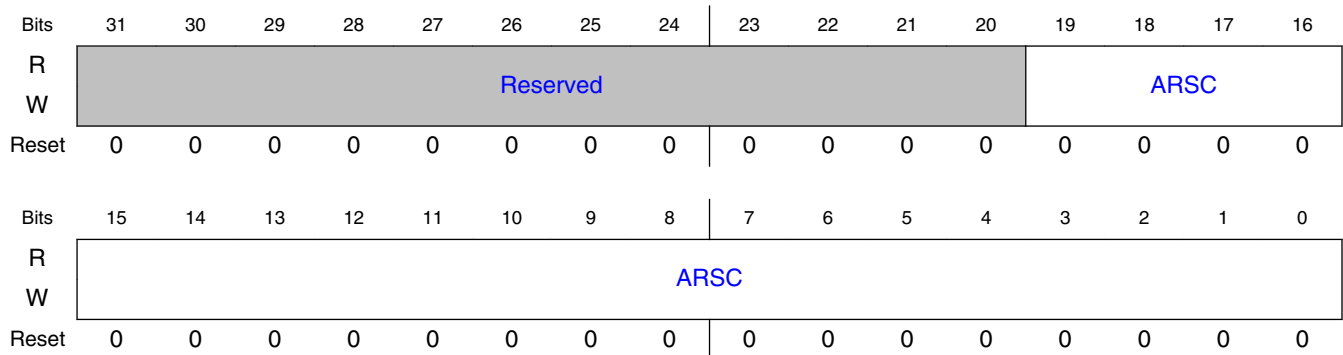
- the AXI read sample count value reaches FFFFh, or
- the AXI read latency total reaches FFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

NOTE

Note that the ARSC field in the DMA_X_ARTC_SC register located in the address range 00530..005DF provides functionality equivalent to the ARSC field in the DMA_n_ARD_TC register located in the address range 00260..002EF. Writing to the ARSC bit field in either register affects the ARSC bit field in the other register. The ARSC values for both are read via the DMA_X_ARTC_SC register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose ARSC value is currently displayed via DMA_X_ARTC_SC.

14.45.3 Diagram



14.45.4 Fields

Field	Function
31-20 —	Reserved.
19-0 ARSC	AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFh. This field is writeable only when ARTCE=0.

14.46 DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)

14.46.1 Offset

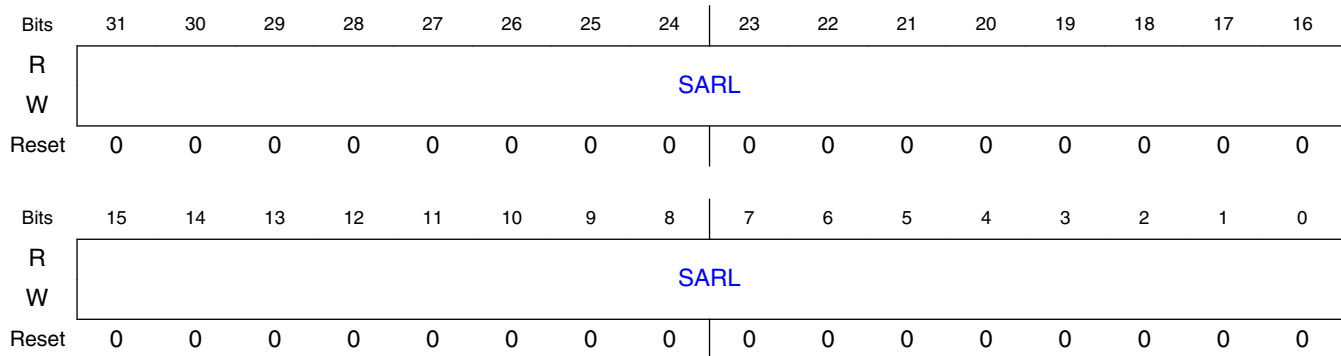
Register	Offset
DMA_X_ARTC_LAT	53Ch

14.46.2 Function

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

NOTE

Note that the DMA_X_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMA_n_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs. The SARL values for both are read via the DMA_X_ARTC_LAT register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose SARL value is currently displayed via DMA_X_ARTC_LAT.

14.46.3 Diagram**14.46.4 Fields**

Field	Function
31-0 SARL	Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMA _n _SARL. This field is writeable only when ARTCE=0.

14.47 DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)

14.47.1 Offset

Register	Offset
DMA_X_AWTC_CTL	540h

14.47.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

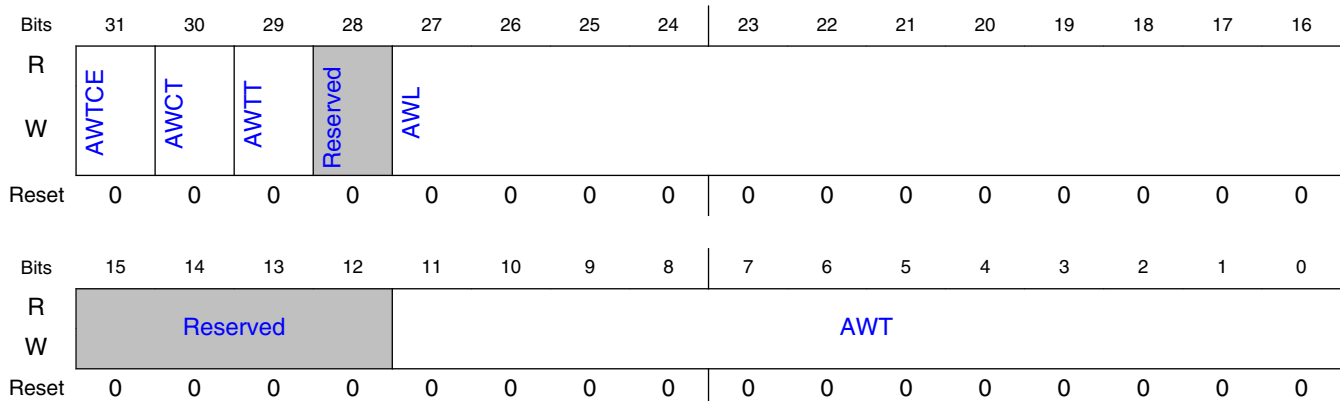
- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFh, or
- the AXI Write Timing Check Register is read

After the DMA_X AXI Write Latency Register or DMA_X Write Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

NOTE

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMA_n_AWR_TC register located in the address range 00270..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMA_n_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_AWTC_LC register or the DMA_X_AWTC_SC register. The write timing checks for both are controlled via the DMA_X_AWTC_CTL register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose write timing checks are currently controlled via DMA_X_AWTC_CTL.

14.47.3 Diagram



14.47.4 Fields

Field	Function
31 AWTCE	AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMA_X_AWTC and SAWL in DMA _n _AWR_LAT and AWL in DMA_X_AWTC_CTL, AWLC in DMA0_AWTC_LC, AWSC in DMA0AWTC_SC and SARL in DMA0AWL_LAT are writeable. When AWTCE=1, AXI Write timing checks are enabled and these fields are Write-only. NOTE: Note that writing AWTCE in either DMA _n _AWR_TC or DMA_X_AWTC_CTL has the same effect.
30 AWCT	AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMA_X_AWTC_CTL, AWLC in DMA_X_AWTC_LC, ARSC in DMA_X_WRTC_SC, and SAWL in DMA _n _AWR_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMA _n _AWR_TC and DMA _n _AWR_LAT or DMA_X_AWTC_LC, DMA_X_AWTC_SC and DMA_X_AWTC_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested.
29 AWTT	AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field AWTT in the DMA_X_AWTC_CTL register is aliased to bit field AWTT in the DMA _n _AWR_TC register, i.e. writing to either AWTT bit field alters the AWTT value in the other register.
28 —	Reserved
27-16 AWL	AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding write data transaction. If the latency equals or exceeds the AXI Write Limit, the write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMA _n _AWR_LAT/DMA_X_ARTC_LAT. This field is writeable only when AWTCE=0. Note that bit field AWL in the DMA_X_AWTC_CTL register is aliased to bit field AWL in the DMA _n _AWR_TC register, i.e. writing to either AWL bit field alters the AWL value in the other register.

Table continues on the next page...

DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)

Field	Function
15-12 —	Reserved
11-0 AWT	AXI Write Timer. The number of AXI clock cycles from the latest external AXI write address transaction initiated by this DMA to the beginning of the corresponding write data transaction. This field is writeable only when AWTCE=0.

14.48 DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)

14.48.1 Offset

Register	Offset
DMA_X_AWTC_LC	544h

14.48.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the AWTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

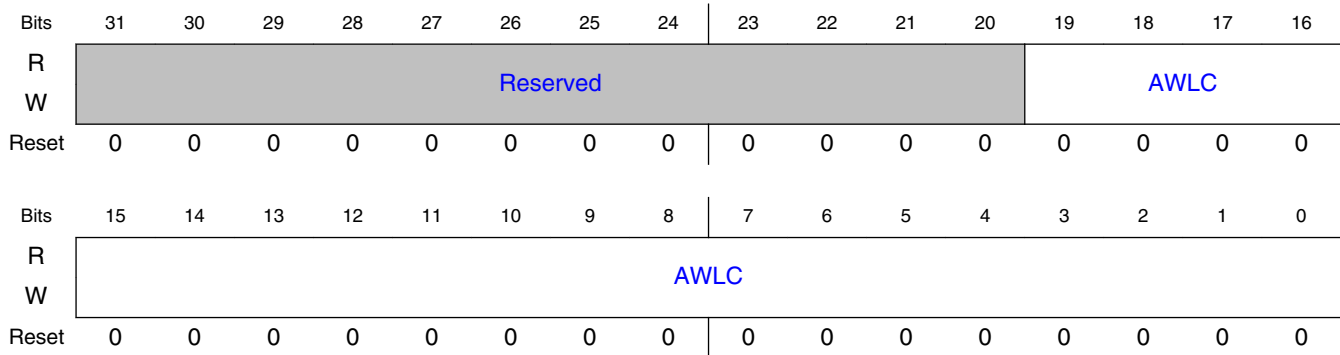
- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is Write

NOTE

Note that the DMA_X_AWTC_LC register provides functionality similar to the AXI Write Timing Late Check fields in the DMA_n_AWR_TC register located in the address range 00270..002EF, but the fields have been rearranged. Usage of the DMA_n_AWR_TC register is deprecated. The AWLC values for both are read via the DMA_X_AWTC_LC register.

The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose AWLC value is currently displayed via DMA_X_AWTC_LC.

14.48.3 Diagram



14.48.4 Fields

Field	Function
31-20 —	Reserved
19-0 AWLC	AXI Write Late Count. This field is incremented each time that the ART field exceeds the ARL field in the DMA_X AXI Write Timing Check Control Register. AXI Write timing checks are suspended when AWLC=FFFFFFh. Note that this field is an alias of the AWLC field in the DMA _n _AWR_TC Register. Reading or writing the AWLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMA _n _AWR_TC/DMA_X_AWTC_TC[AWTCE]=0, the AWLC bit field in DMA _n _AWR_TC and the DMA_X_AWTC_LC register are writeable. When AWTCE=1, AXI write timing checks are enabled and the AWLC bit fields are read-only.

14.49 DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)

14.49.1 Offset

Register	Offset
DMA_X_AWTC_SC	548h

14.49.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

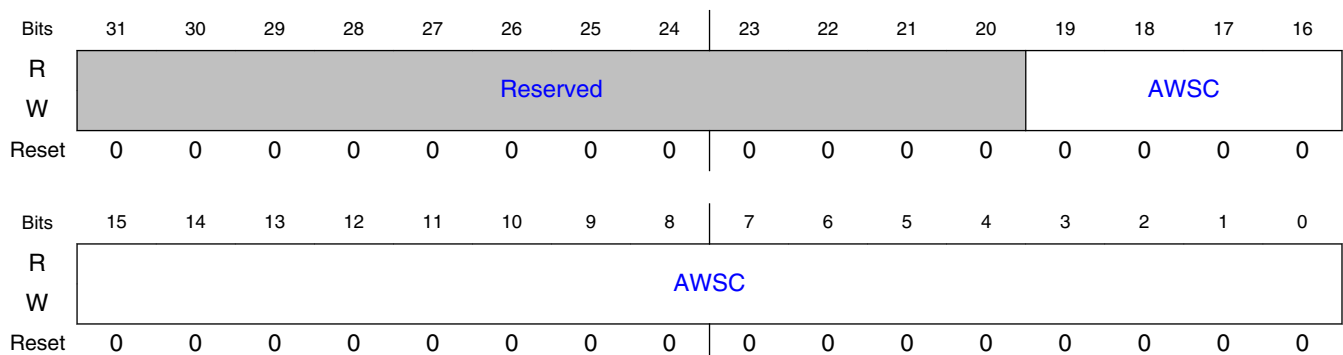
- the AXI write sample count value reaches FFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

NOTE

Note that the AWSC field in the DMA_X_AWTC_SC register located in the address range 00530..005DF provides functionality equivalent to the AWSC field in the DMA_n_AWR_TC register located in the address range 00260..002EF. Writing to the AWSC bit field in either register affects the AWSC bit field in the other register. The AWSC values for both are read via the DMA_X_AWTC_SC register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to select the particular DMA engine whose AWSC value is currently displayed via DMA_X_AWTC_SC.

14.49.3 Diagram



14.49.4 Fields

Field	Function
31-20 —	Reserved.
19-0 AWSC	AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFh. This field is writeable only when AWTCE=0.

14.50 DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)

14.50.1 Offset

Register	Offset
DMA_X_AWTC_LAT	54Ch

14.50.2 Function

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

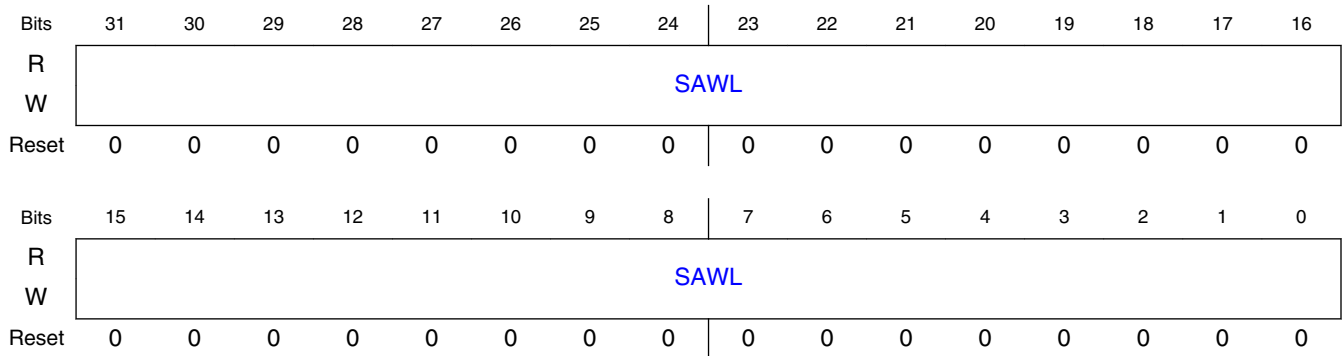
NOTE

Note that the DMA_X_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMA_n_AWL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs. The SAWL values for both are read via the DMA_X_AWTC_LAT register. The DMA Access Index (DAI) bit field in the DMA Control Register is used to

RNG TRNG Miscellaneous Control Register (RTMCTL)

select the particular DMA engine whose SAWL value is currently displayed via DMA_X_AWTC_LAT.

14.50.3 Diagram



14.50.4 Fields

Field	Function
31-0 SAWL	Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMA _n _AWL. This field is writeable only when AWTCE=0.

14.51 RNG TRNG Miscellaneous Control Register (RTMCTL)

14.51.1 Offset

Register	Offset
RTMCTL	600h

14.51.2 Function

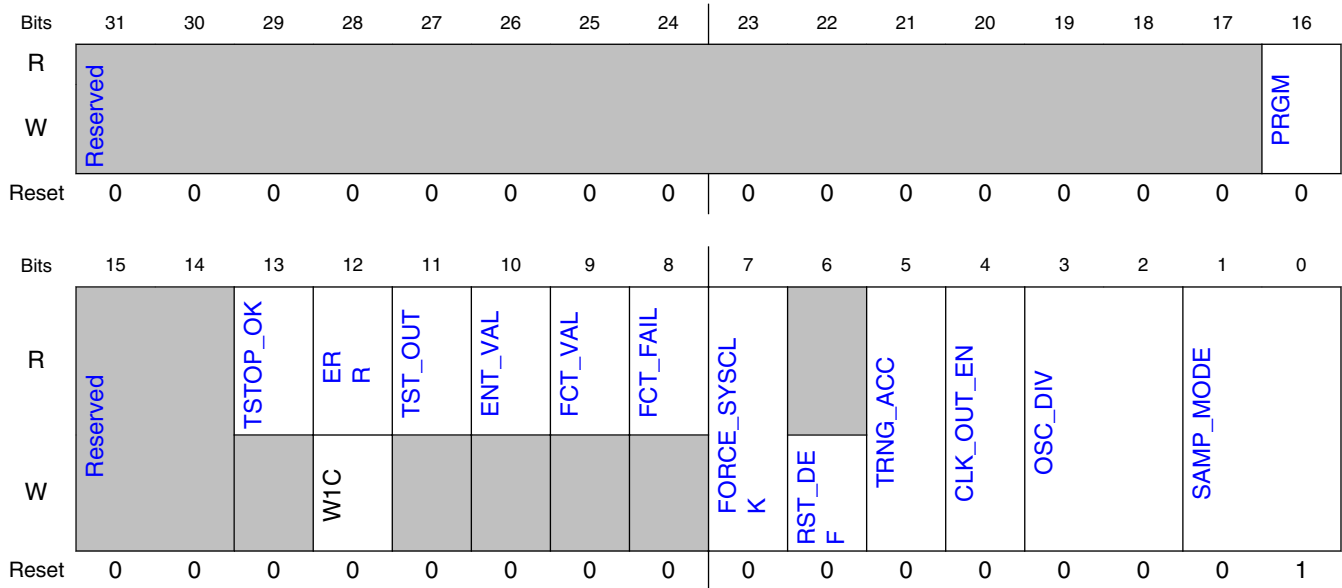
These registers are intended to be used when testing the RNG. They would not be used during normal operation. During normal operation the RNG is configured and data is obtained from the RNG via Job Descriptors.

The RNG TRNG Miscellaneous Control Register is a read/write register used to control the RNG's True Random Number Generator (TRNG) access, operation and test.

NOTE

Note that in many cases two RNG registers share the same address, and a particular register at the shared address is selected based upon the value in the PRGM field of the RTMCTL register.

14.51.3 Diagram



14.51.4 Fields

Field	Function
31-17 —	Reserved.
16 PRGM	Programming Mode Select. When this bit is 1, the TRNG is in Program Mode, otherwise it is in Run Mode. No Entropy value will be generated while the TRNG is in Program Mode. Note that different RNG registers are accessible at the same address depending on whether PRGM is set to 1 or 0. This is noted in the RNG register descriptions.
15-14 —	Reserved.
13 TSTOP_OK	TRNG_OK_TO_STOP. Software should check that this bit is a 1 before transitioning SEC to low power mode (SEC clock stopped). SEC turns on the TRNG free-running ring oscillator whenever new entropy is being generated and turns off the ring oscillator when entropy generation is complete. If the SEC clock is

Table continues on the next page...

RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)

Field	Function
	stopped while the TRNG ring oscillator is running, the oscillator will continue running even though the SEC clock is stopped. TSTOP_OK is asserted when the TRNG ring oscillator is not running. and therefore it is OK to stop the SEC clock.
12 ERR	Read: Error status. 1 = error detected. 0 = no error. Write: Write 1 to clear errors. Writing 0 has no effect.
11 TST_OUT	Read only: Test point inside ring oscillator.
10 ENT_VAL	Read only: Entropy Valid. Will assert only if TRNG ACC bit is set, and then after an entropy value is generated. Will be cleared when RTENT15 is read. (RTENT0 through RTENT14 should be read before reading RTENT15).
9 FCT_VAL	Read only: Frequency Count Valid. Indicates that a valid frequency count may be read from RTFRQCNT.
8 FCT_FAIL	Read only: Frequency Count Fail. The frequency counter has detected a failure. This may be due to improper programming of the RTFRQMAX and/or RTFRQMIN registers, or a hardware failure in the ring oscillator. This error may be cleared by writing a 1 to the ERR bit.
7 FORCE_SYSCLK	Force System Clock. If set, the system clock is used to operate the TRNG, instead of the ring oscillator. This is for test use only, and indeterminate results may occur. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. This bit is cleared by writing the RST_DEF bit to 1.
6 RST_DEF	Reset Defaults. Writing a 1 to this bit clears various TRNG registers, and bits within registers, to their default state. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. Reading this bit always produces a 0.
5 TRNG_ACC	TRNG Access Mode. If this bit is set to 1, the TRNG will generate an Entropy value that can be read via the RTENT registers. The Entropy value may be read once the ENT VAL bit is asserted. This Entropy value will never be used by the RNG. IMPORTANT: If this bit is set, no Entropy value can be generated for the RNG, which can prevent the RNG from generating data for the SEC system.
4 CLK_OUT_EN	Clock Output Enable. If set, the ring oscillator output is gated to an output pad. If this bit is set and PRGM mode is selected, this allows external viewing of the ring oscillator.
3-2 OSC_DIV	Oscillator Divide. Determines the amount of dividing done to the ring oscillator before it is used by the TRNG. This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this field. This field is cleared to 00 by writing the RST_DEF bit to 1. 00b - use ring oscillator with no divide 01b - use ring oscillator divided-by-2 10b - use ring oscillator divided-by-4 11b - use ring oscillator divided-by-8
1-0 SAMP_MODE	Sample Mode. Determines the method of sampling the ring oscillator while generating the Entropy value: This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously with writing this field. This field is cleared to 01 by writing the RST_DEF bit to 1. 00b - use Von Neumann data into both Entropy shifter and Statistical Checker 01b - use raw data into both Entropy shifter and Statistical Checker 10b - use Von Neumann data into Entropy shifter. Use raw data into Statistical Checker 11b - undefined/reserved.

14.52 RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)

14.52.1 Offset

Register	Offset
RTSCMISC	604h

14.52.2 Function

The RNG TRNG Statistical Check Miscellaneous Register contains the Long Run Maximum Limit value and the Retry Count value. This register is accessible only when the RTMCTL[PRGM] bit is 1, otherwise this register will read zeroes, and cannot be written.

NOTE

Reset occurs at POR, and when RTMCTL[RST_DEF] is written to 1.

14.52.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved												RTY_CNT			
W	Reserved												RTY_CNT			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								LRUN_MAX							
W	Reserved								LRUN_MAX							
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

14.52.4 Fields

Field	Function
31-20	Reserved

Table continues on the next page...

RNG TRNG Poker Range Register (RTPKRRNG)

Field	Function
—	
19-16 RTY_CNT	RETRY COUNT. If a statistical check fails during the TRNG Entropy Generation, the RTY_CNT value indicates the number of times a retry should occur before generating an error. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 1h by writing the RTMCTL[RST_DEF] bit to 1.
15-8 —	Reserved.
7-0 LRUN_MAX	LONG RUN MAX LIMIT. This value is the largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 34 by writing the RTMCTL[RST_DEF] bit to 1.

14.53 RNG TRNG Poker Range Register (RTPKRRNG)

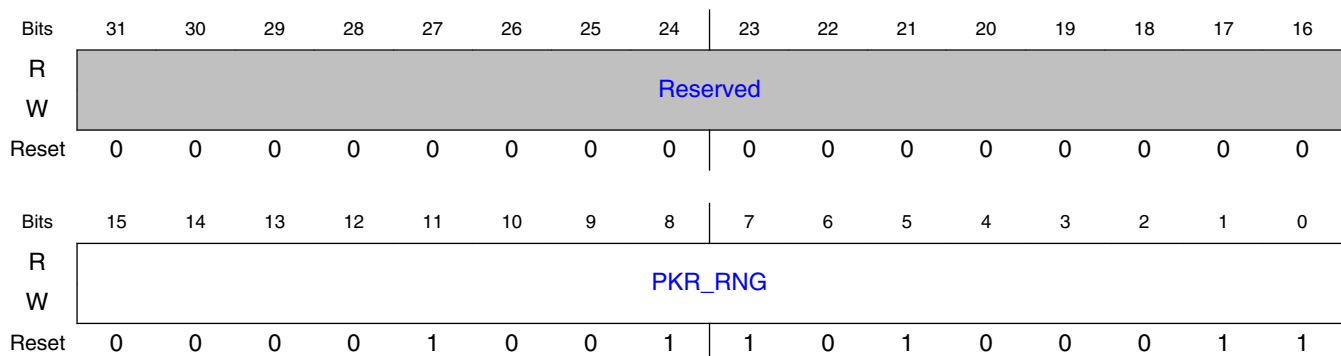
14.53.1 Offset

Register	Offset
RTPKRRNG	608h

14.53.2 Function

The RNG TRNG Poker Range Register defines the difference between the TRNG Poker Maximum Limit and the minimum limit. These limits are used during the TRNG Statistical Check Poker Test.

14.53.3 Diagram



14.53.4 Fields

Field	Function
31-16 —	Reserved. Always 0.
15-0 PKR_RNG	Poker Range. During the TRNG Statistical Checks, a "Poker Test" is run which requires a maximum and minimum limit. The maximum is programmed in the RTPKRMAX[PKR_MAX] register, and the minimum is derived by subtracting the PKR_RNG value from the programmed maximum value. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09A3h (decimal 2467) by writing the RTMCTL[RST_DEF] bit to 1. Note that the minimum allowable Poker result is PKR_MAX - PKR_RNG + 1.

14.54 RNG TRNG Poker Maximum Limit Register (RTPKRMAX)

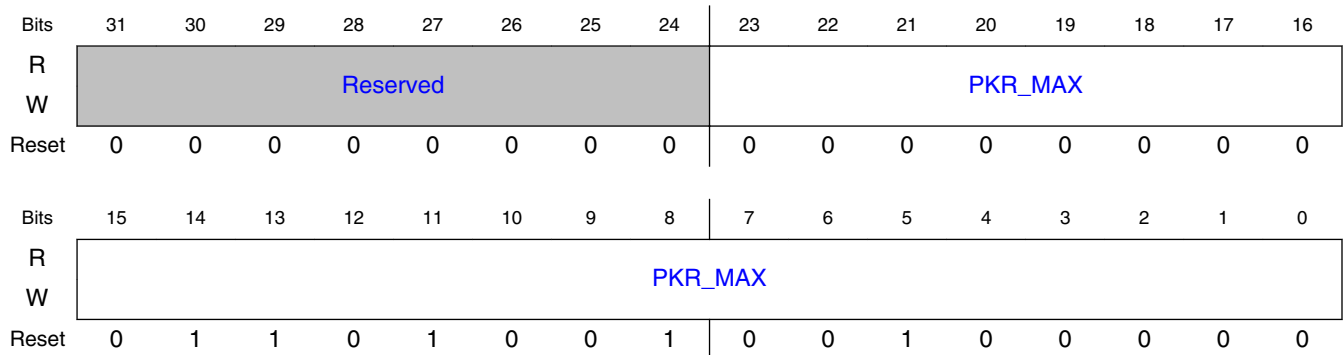
14.54.1 Offset

Register	Offset	Description
RTPKRMAX	60Ch	Accessible at this address when RTMCTL[PRGM] = 1]

14.54.2 Function

The RNG TRNG Poker Maximum Limit Register defines Maximum Limit allowable during the TRNG Statistical Check Poker Test. Note that this offset (060Ch) is used as RTPKRMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register.

14.54.3 Diagram



14.54.4 Fields

Field	Function
31-24 —	Reserved. Always 0.
23-0 PKR_MAX	Poker Maximum Limit. During the TRNG Statistical Checks, a "Poker Test" is run which requires a maximum and minimum limit. The maximum allowable result is programmed in the RTPKRMAX[PKR_MAX] register. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 006920h (decimal 26912) by writing the RTMCTL[RST_DEF] bit to 1. Note that the RTPKRMAX and RTPKRRNG registers combined are used to define the minimum allowable Poker result, which is PKR_MAX - PKR_RNG + 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Poker Test Square Calculation result in register RTPKRSQ, as defined in the following section.

14.55 RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)

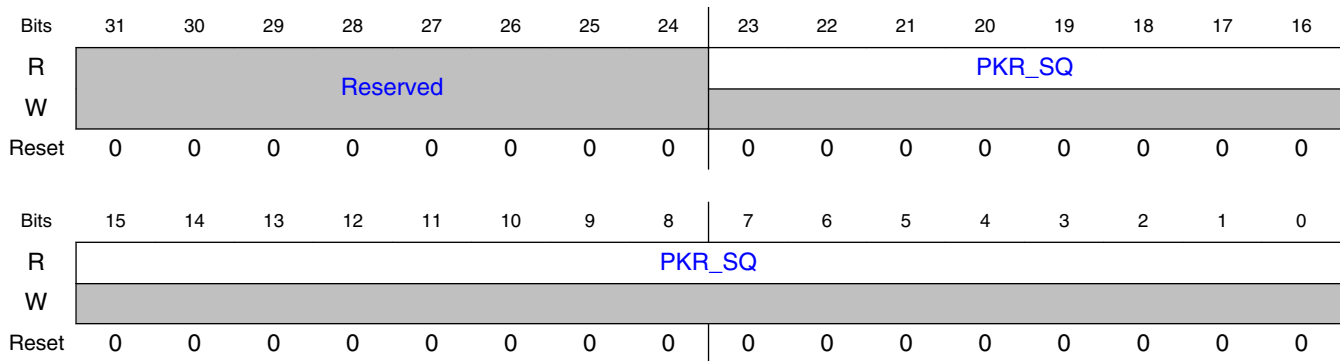
14.55.1 Offset

Register	Offset	Description
RTPKRSQ	60Ch	Accessible at this address when RTMCTL[PRGM] = 0]

14.55.2 Function

The RNG TRNG Poker Square Calculation Result Register is a read-only register used to read the result of the TRNG Statistical Check Poker Test's Square Calculation. This test starts with the RTPKRMAX value and decreases towards a final result, which is read here. For the Poker Test to pass, this final result must be less than the programmed RTPKRRNG value. Note that this offset (060Ch) is used as RTPKRMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register, as described here.

14.55.3 Diagram



14.55.4 Fields

Field	Function
31-24 —	Reserved. Always 0.
23-0 PKR_SQ	Poker Square Calculation Result. During the TRNG Statistical Checks, a "Poker Test" is run which starts with the value RTPKRMAX[PKR_MAX]. This value decreases according to a "sum of squares" algorithm, and must remain greater than zero, but less than the RTPKRRNG[PKR_RNG] limit. The resulting value may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Poker Test Maximum Limit in register RTPKRMAX, as defined in the previous section.

14.56 RNG TRNG Seed Control Register (RTSDCTL)

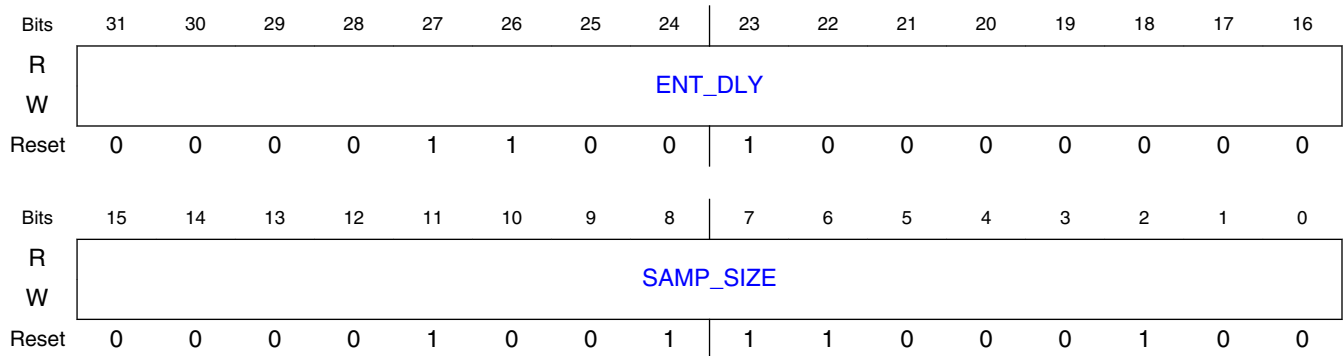
14.56.1 Offset

Register	Offset
RTSDCTL	610h

14.56.2 Function

The RNG TRNG Seed Control Register contains two fields. One field defines the length (in system clocks) of each Entropy sample (ENT_DLY), and the other field indicates the number of samples that will be taken during each TRNG Entropy generation (SAMP_SIZE).

14.56.3 Diagram



14.56.4 Fields

Field	Function
31-16 ENT_DLY	Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 00C80h (decimal 3200) by writing the RTMCTL[RST_DEF] bit to 1.
15-0 SAMP_SIZE	Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09C4h (decimal 2500) by writing the RTMCTL[RST_DEF] bit to 1.

14.57 RNG TRNG Sparse Bit Limit Register (RTSBLIM)

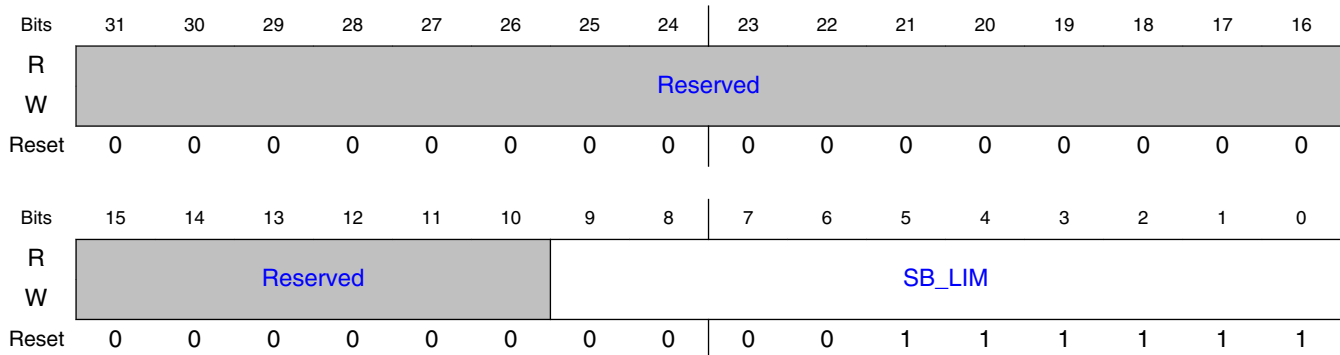
14.57.1 Offset

Register	Offset	Description
RTSBLIM	614h	Accessible at this address when RTMCTL[PRGM] = 1]

14.57.2 Function

The RNG TRNG Sparse Bit Limit Register is used when Von Neumann sampling is selected during Entropy Generation. It defines the maximum number of consecutive Von Neumann samples which may be discarded before an error is generated. Note that this address (0614h) is used as RTSBLIM only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTTOTSAM readback register.

14.57.3 Diagram



14.57.4 Fields

Field	Function
31-10 —	Reserved. Always 0.
9-0 SB_LIM	Sparse Bit Limit. During Von Neumann sampling (if enabled by RTMCTL[SAMP_MODE]), samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy. The Sparse Bit Limit defines the maximum number of

RNG TRNG Total Samples Register (RTTOTSAM)

Field	Function
	consecutive samples that may be discarded before an error is generated. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 03hF by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Total Samples count in register RTTOTSAM, as defined in the following section.

14.58 RNG TRNG Total Samples Register (RTTOTSAM)

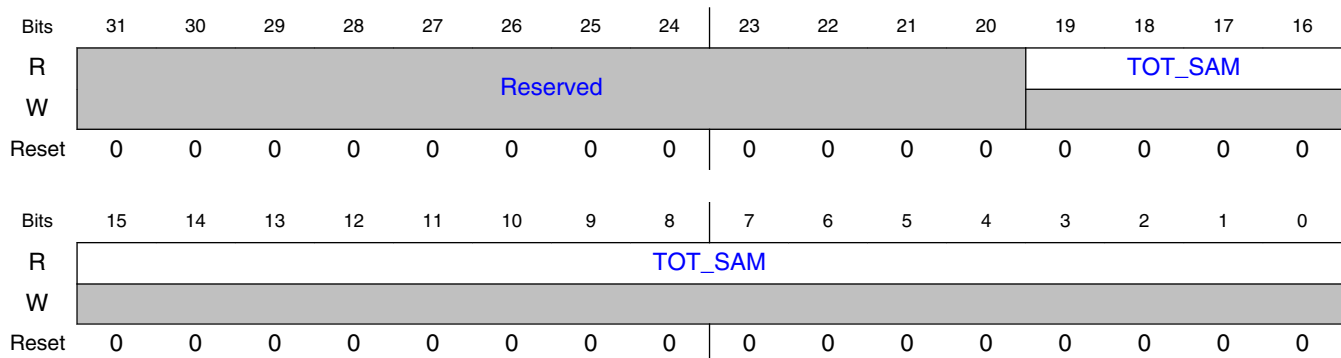
14.58.1 Offset

Register	Offset	Description
RTTOTSAM	614h	Accessible at this address when RTMCTL[PRGM] = 0]

14.58.2 Function

The RNG TRNG Total Samples Register is a read-only register used to read the total number of samples taken during Entropy generation. It is used to give an indication of how often a sample is actually used during Von Neumann sampling. Note that this offset (0614h) is used as RTSBLIM if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTTOTSAM readback register, as described here.

14.58.3 Diagram



14.58.4 Fields

Field	Function
31-20 —	Reserved. Always 0.
19-0 TOT_SAM	Total Samples. During Entropy generation, the total number of raw samples is counted. This count is useful in determining how often a sample is used during Von Neumann sampling. The count may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Sparse Bit Limit in register RTSBLIM, as defined in the previous section.

14.59 RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)

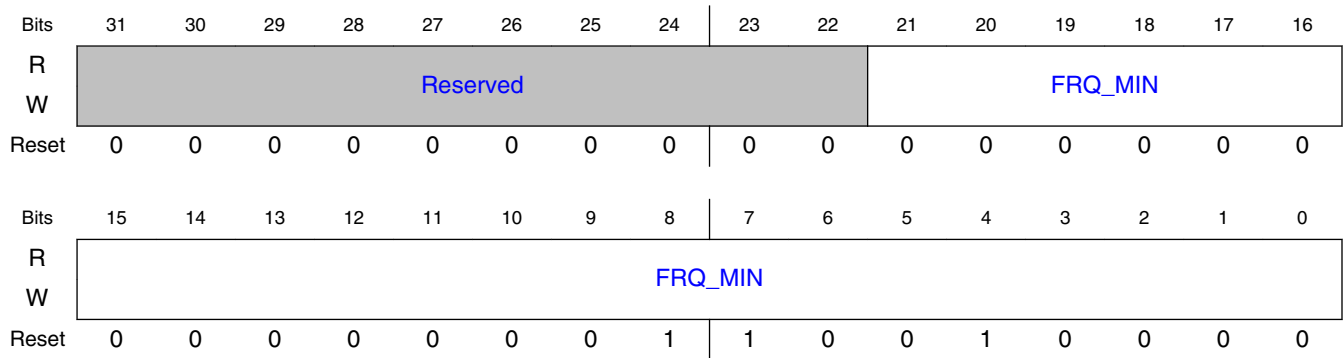
14.59.1 Offset

Register	Offset
RTFRQMIN	618h

14.59.2 Function

The RNG TRNG Frequency Count Minimum Limit Register defines the minimum allowable count taken by the Entropy sample counter during each Entropy sample. During any sample period, if the count is less than this programmed minimum, a Frequency Count Fail is flagged in RTMCTL[FCT_FAIL] and an error is generated.

14.59.3 Diagram



14.59.4 Fields

Field	Function
31-22 —	Reserved. Always 0.
21-0 FRQ_MIN	Frequency Count Minimum Limit. Defines the minimum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 000190h by writing the RTMCTL[RST_DEF] bit to 1.

14.60 RNG TRNG Frequency Count Register (RTFRQCNT)

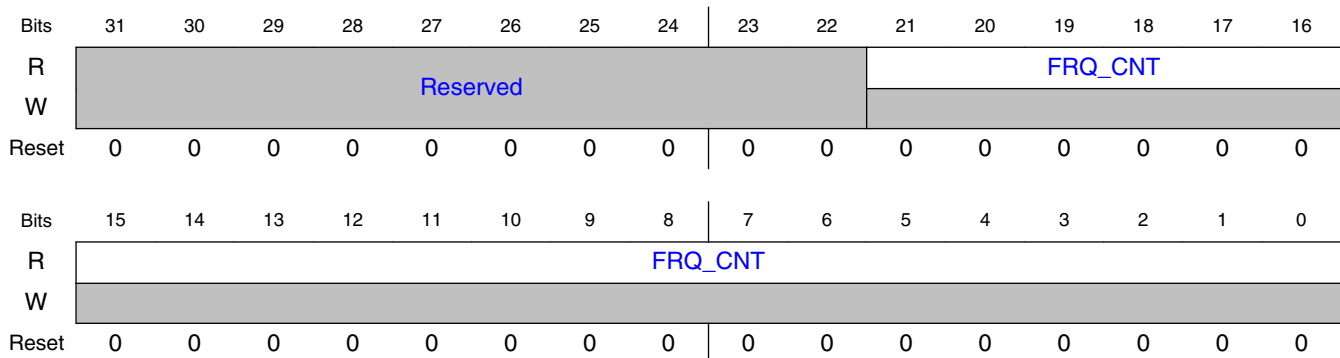
14.60.1 Offset

Register	Offset	Description
RTFRQCNT	61Ch	RNG TRNG Frequency Count accessible at this address when RTMCTL[PRGM] = 0]

14.60.2 Function

The RNG TRNG Frequency Count Register is a read-only register used to read the frequency counter within the TRNG entropy generator. It will read all zeroes unless $RTMCTL[TRNG_ACC] = 1$. Note that this offset (061Ch) is used as $RTFRQMAX$ if $RTMCTL[PRGM] = 1$. If $RTMCTL[PRGM]$ is 0, this offset is used as $RTFRQCNT$ readback register, as described here.

14.60.3 Diagram



14.60.4 Fields

Field	Function
31-22 —	Reserved. Always 0.
21-0 FRQ_CNT	Frequency Count. If $RTMCTL[TRNG_ACC] = 1$, reads a sample frequency count taken during entropy generation. Requires $RTMCTL[PRGM] = 0$. The value read from FRQ_CNT is valid only if $RTMCTL[FCT_VAL] = 1$.

14.61 RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)

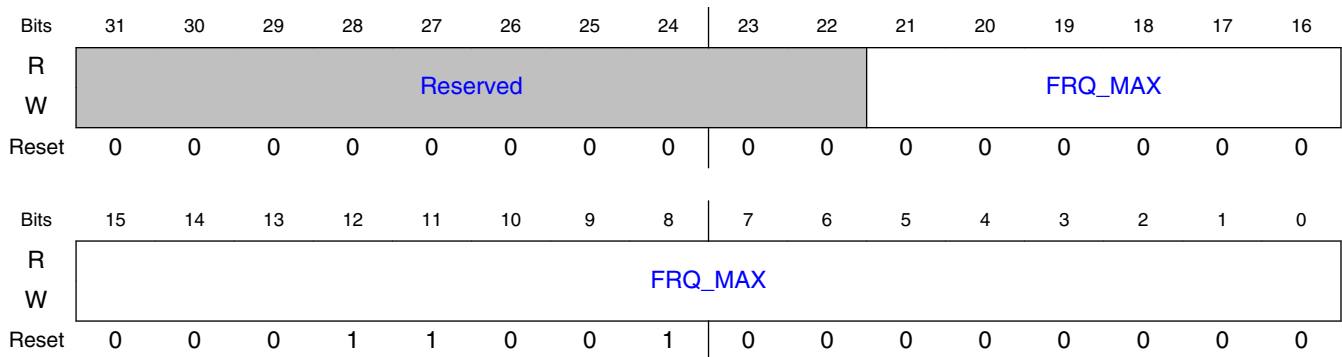
14.61.1 Offset

Register	Offset	Description
RTFRQMAX	61Ch	Accessible at this address when RTMCTL[PRGM] = 1]

14.61.2 Function

The RNG TRNG Frequency Count Maximum Limit Register defines the maximum allowable count taken by the Entropy sample counter during each Entropy sample. During any sample period, if the count is greater than this programmed maximum, a Frequency Count Fail is flagged in RTMCTL[FCT_FAIL] and an error is generated. Note that this address (061C) is used as RTFRQMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTFRQCNT readback register.

14.61.3 Diagram



14.61.4 Fields

Field	Function
31-22 —	Reserved. Always 0.
21-0 FRQ_MAX	Frequency Counter Maximum Limit. Defines the maximum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 00190h by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Frequency Count result in register RTFRQCNT, as defined in the following section.

14.62 RNG TRNG Statistical Check Monobit Count Register (RTSCMC)

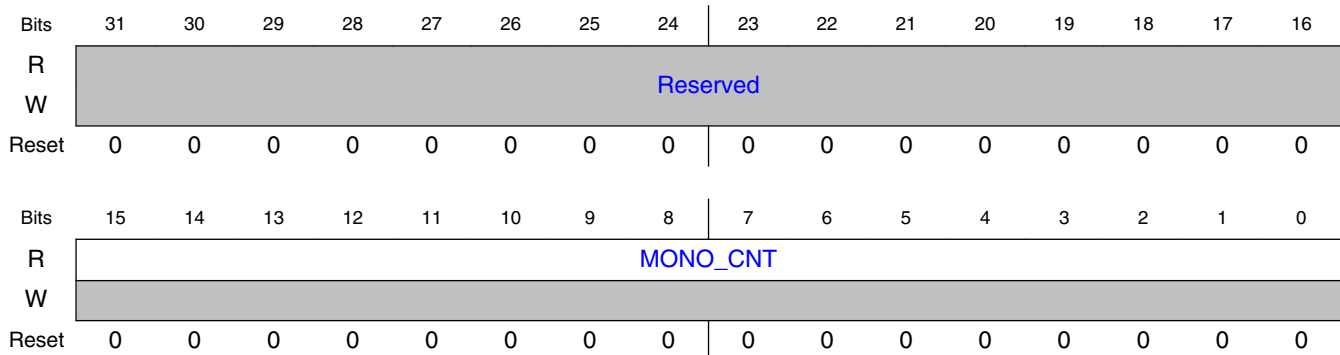
14.62.1 Offset

Register	Offset	Description
RTSCMC	620h	Accessible at this address when RTMCTL[PRGM] = 0]

14.62.2 Function

The RNG TRNG Statistical Check Monobit Count Register is a read-only register used to read the final monobit count after entropy generation. This counter starts with the value in RTSCML[MONO_MAX], and is decremented each time a one is sampled. Note that this offset (0620h) is used as RTSCML if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register, as described here.

14.62.3 Diagram



14.62.4 Fields

Field	Function
31-16	Reserved. Always 0.

Table continues on the next page...

RNG TRNG Statistical Check Monobit Limit Register (RTSCML)

Field	Function
—	
15-0 MONO_CNT	Monobit Count. Reads the final Monobit count after entropy generation. Requires RTMCTL[PRGM] = 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Statistical Check Monobit Limit in register RTSCML, as defined in the previous section.

14.63 RNG TRNG Statistical Check Monobit Limit Register (RTSCML)

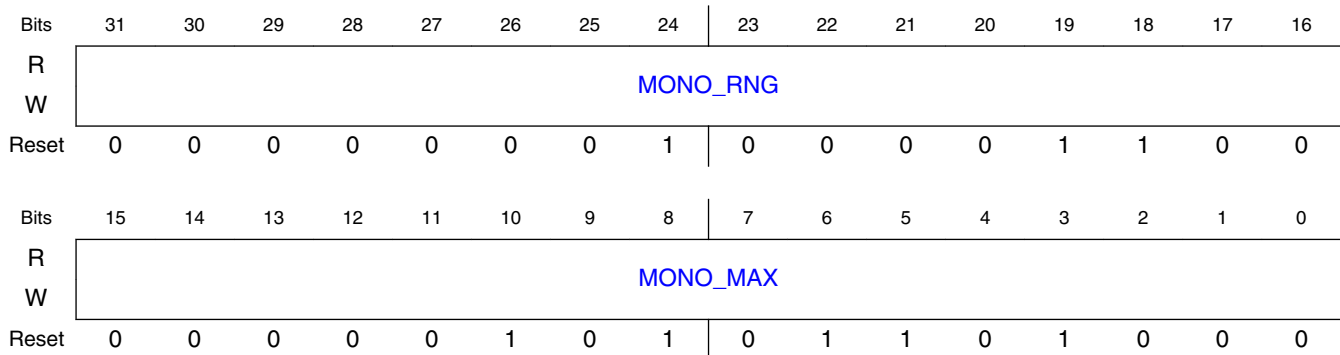
14.63.1 Offset

Register	Offset	Description
RTSCML	620h	Accessible at this address when RTMCTL[PRGM] = 1]

14.63.2 Function

The RNG TRNG Statistical Check Monobit Limit Register defines the allowable maximum and minimum number of ones/zero detected during entropy generation. To pass the test, the number of ones/zeroes generated must be less than the programmed maximum value, and the number of ones/zeroes generated must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0620h) is used as RTSCML only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register.

14.63.3 Diagram



14.63.4 Fields

Field	Function
31-16 MONO_RNG	Monobit Range. The number of ones/zeros detected during entropy generation must be greater than MONO_MAX - MONO_RNG, else a retry or error will occur. This register is cleared to 000112h (decimal 274) by writing the RTMCTL[RST_DEF] bit to 1.
15-0 MONO_MAX	Monobit Maximum Limit. Defines the maximum allowable count taken during entropy generation. The number of ones/zeros detected during entropy generation must be less than MONO_MAX, else a retry or error will occur. This register is cleared to 00056Bh (decimal 1387) by writing the RTMCTL[RST_DEF] bit to 1.

14.64 RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)

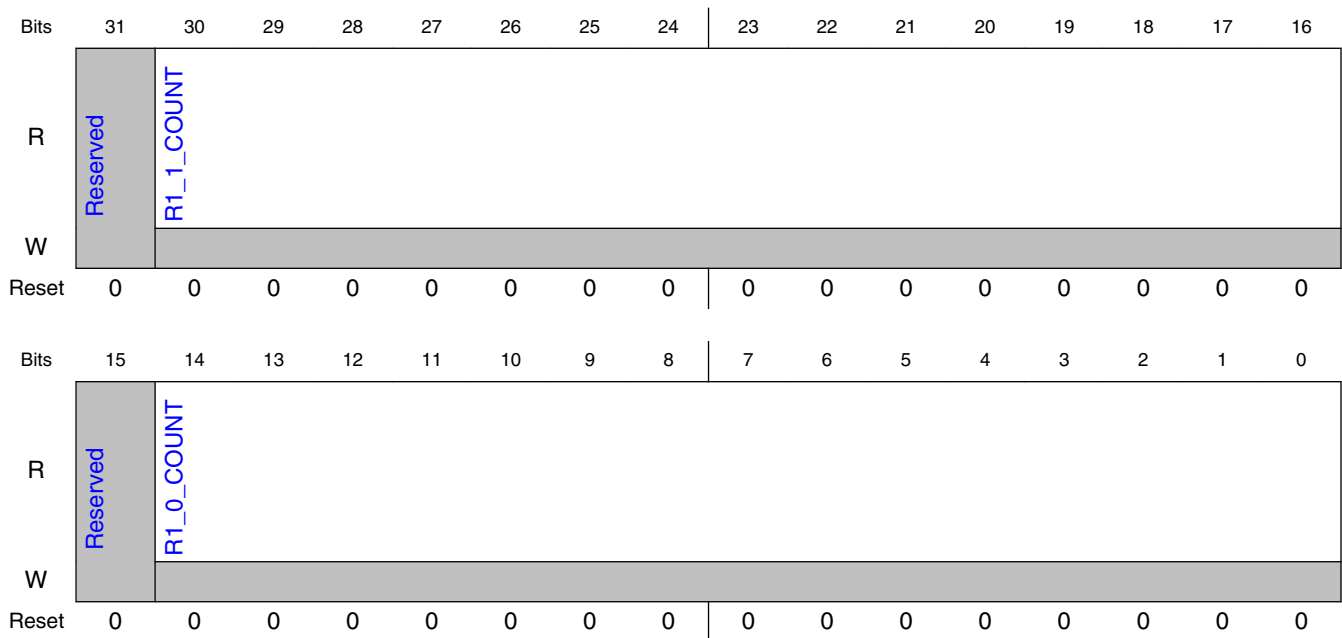
14.64.1 Offset

Register	Offset	Description
RTSCR1C	624h	Accessible at this address when RTMCTL[PRGM] = 0]

14.64.2 Function

The RNG TRNG Statistical Check Run Length 1 Counters Register is a read-only register used to read the final Run Length 1 counts after entropy generation. These counters start with the value in RTSCR1L[RUN1_MAX]. The R1_1_COUNT decrements each time a single one is sampled (preceded by a zero and followed by a zero). The R1_0_COUNT decrements each time a single zero is sampled (preceded by a one and followed by a one). Note that this offset (0624h) is used as RTSCR1L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR1C readback register, as described here.

14.64.3 Diagram



14.64.4 Fields

Field	Function
31	Reserved. Always 0.
—	
30-16 R1_1_COUNT	Runs of One, Length 1 Count. Reads the final Runs of Ones, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15	Reserved. Always 0.

Table continues on the next page...

Field	Function
—	
14-0 R1_0_COUNT	Runs of Zero, Length 1 Count. Reads the final Runs of Zeroes, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.65 RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)

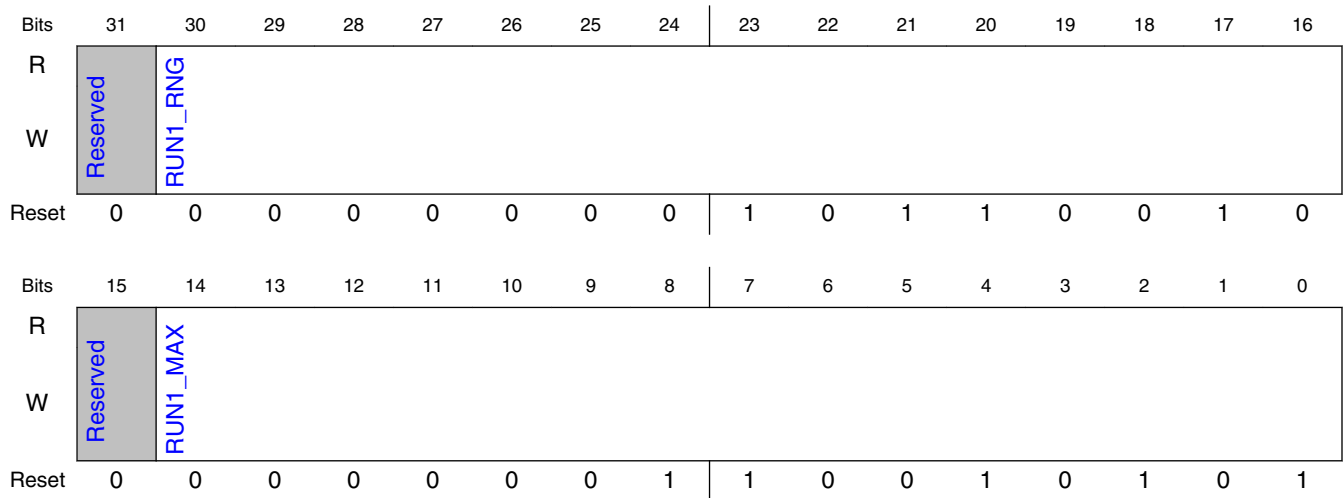
14.65.1 Offset

Register	Offset	Description
RTSCR1L	624h	Accessible at this address when RTMCTL[PRGM] = 1]

14.65.2 Function

The RNG TRNG Statistical Check Run Length 1 Limit Register defines the allowable maximum and minimum number of runs of length 1 detected during entropy generation. To pass the test, the number of runs of length 1 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 1 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0624h) is used as RTSCR1L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR1C readback register.

14.65.3 Diagram



14.65.4 Fields

Field	Function
31 —	Reserved. Always 0.
30-16 RUN1_RNG	Run Length 1 Range. The number of runs of length 1 (for both 0 and 1) detected during entropy generation must be greater than RUN1_MAX - RUN1_RNG, else a retry or error will occur. This register is cleared to 0102h (decimal 258) by writing the RTMCTL[RST_DEF] bit to 1.
15 —	Reserved. Always 0.
14-0 RUN1_MAX	Run Length 1 Maximum Limit. Defines the maximum allowable runs of length 1 (for both 0 and 1) detected during entropy generation. The number of runs of length 1 detected during entropy generation must be less than RUN1_MAX, else a retry or error will occur. This register is cleared to 01E5h (decimal 485) by writing the RTMCTL[RST_DEF] bit to 1.

14.66 RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)

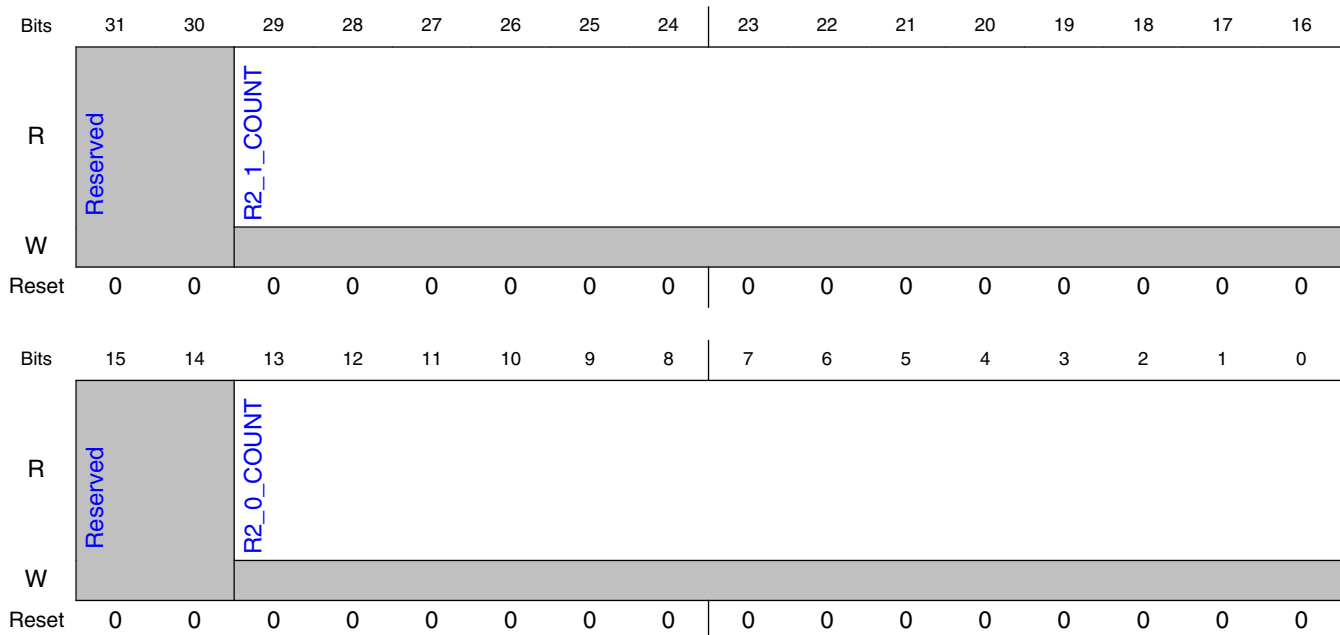
14.66.1 Offset

Register	Offset	Description
RTSCR2C	628h	Accessible at this address when RTMCTL[PRGM] = 0]

14.66.2 Function

The RNG TRNG Statistical Check Run Length 2 Counters Register is a read-only register used to read the final Run Length 2 counts after entropy generation. These counters start with the value in RTSCR2L[RUN2_MAX]. The R2_1_COUNT decrements each time two consecutive ones are sampled (preceded by a zero and followed by a zero). The R2_0_COUNT decrements each time two consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0628h) is used as RTSCR2L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR2C readback register, as described here.

14.66.3 Diagram



14.66.4 Fields

Field	Function
31-30 —	Reserved. Always 0.
29-16 R2_1_COUNT	Runs of One, Length 2 Count. Reads the final Runs of Ones, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-14 —	Reserved. Always 0.
13-0 R2_0_COUNT	Runs of Zero, Length 2 Count. Reads the final Runs of Zeroes, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.67 RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)

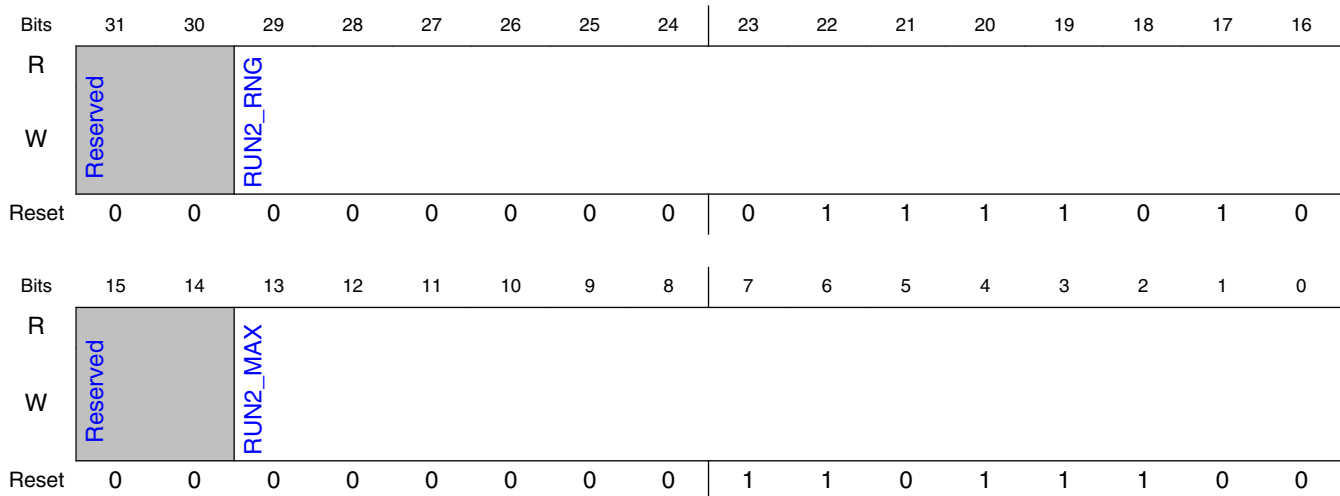
14.67.1 Offset

Register	Offset	Description
RTSCR2L	628h	Accessible at this address when RTMCTL[PRGM] = 1]

14.67.2 Function

The RNG TRNG Statistical Check Run Length 2 Limit Register defines the allowable maximum and minimum number of runs of length 2 detected during entropy generation. To pass the test, the number of runs of length 2 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 2 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0628h) is used as RTSCR2L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR2C readback register.

14.67.3 Diagram



14.67.4 Fields

Field	Function
31-30 —	Reserved. Always 0.
29-16 RUN2_RNG	Run Length 2 Range. The number of runs of length 2 (for both 0 and 1) detected during entropy generation must be greater than $RUN2_MAX - RUN2_RNG$, else a retry or error will occur. This register is cleared to 007Ah (decimal 122) by writing the <code>RTMCTL[RST_DEF]</code> bit to 1.
15-14 —	Reserved. Always 0.
13-0 RUN2_MAX	Run Length 2 Maximum Limit. Defines the maximum allowable runs of length 2 (for both 0 and 1) detected during entropy generation. The number of runs of length 2 detected during entropy generation must be less than $RUN2_MAX$, else a retry or error will occur. This register is cleared to 00DCh (decimal 220) by writing the <code>RTMCTL[RST_DEF]</code> bit to 1.

14.68 RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)

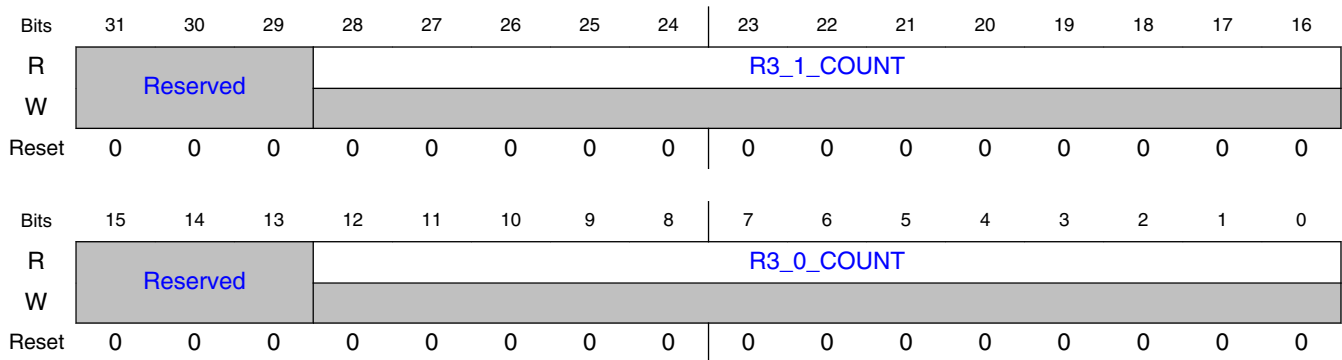
14.68.1 Offset

Register	Offset	Description
RTSCR3C	62Ch	Accessible at this address when RTMCTL[PRGM] = 0]

14.68.2 Function

The RNG TRNG Statistical Check Run Length 3 Counters Register is a read-only register used to read the final Run Length 3 counts after entropy generation. These counters start with the value in RTSCR3L[RUN3_MAX]. The R3_1_COUNT decrements each time three consecutive ones are sampled (preceded by a zero and followed by a zero). The R3_0_COUNT decrements each time three consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (062Ch) is used as RTSCR3L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR3C readback register, as described here.

14.68.3 Diagram



14.68.4 Fields

Field	Function
31-29 —	Reserved. Always 0.
28-16 R3_1_COUNT	Runs of Ones, Length 3 Count. Reads the final Runs of Ones, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0.

Table continues on the next page...

Field	Function
15-13 —	Reserved. Always 0.
12-0 R3_0_COUNT	Runs of Zeroes, Length 3 Count. Reads the final Runs of Zeroes, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.69 RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L)

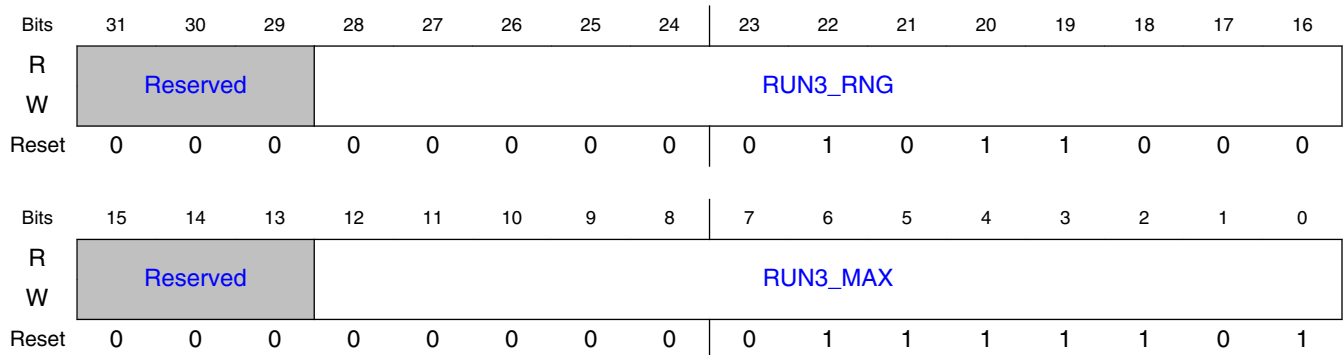
14.69.1 Offset

Register	Offset	Description
RTSCR3L	62Ch	Accessible at this address when RTMCTL[PRGM] = 1]

14.69.2 Function

The RNG TRNG Statistical Check Run Length 3 Limit Register defines the allowable maximum and minimum number of runs of length 3 detected during entropy generation. To pass the test, the number of runs of length 3 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 3 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (062Ch) is used as RTSCR3L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR3C readback register.

14.69.3 Diagram



14.69.4 Fields

Field	Function
31-29 —	Reserved. Always 0.
28-16 RUN3_RNG	Run Length 3 Range. The number of runs of length 3 (for both 0 and 1) detected during entropy generation must be greater than RUN3_MAX - RUN3_RNG, else a retry or error will occur. This register is cleared to 0058h (decimal 88) by writing the RTMCTL[RST_DEF] bit to 1.
15-13 —	Reserved. Always 0.
12-0 RUN3_MAX	Run Length 3 Maximum Limit. Defines the maximum allowable runs of length 3 (for both 0 and 1) detected during entropy generation. The number of runs of length 3 detected during entropy generation must be less than RUN3_MAX, else a retry or error will occur. This register is cleared to 007Dh (decimal 125) by writing the RTMCTL[RST_DEF] bit to 1.

14.70 RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C)

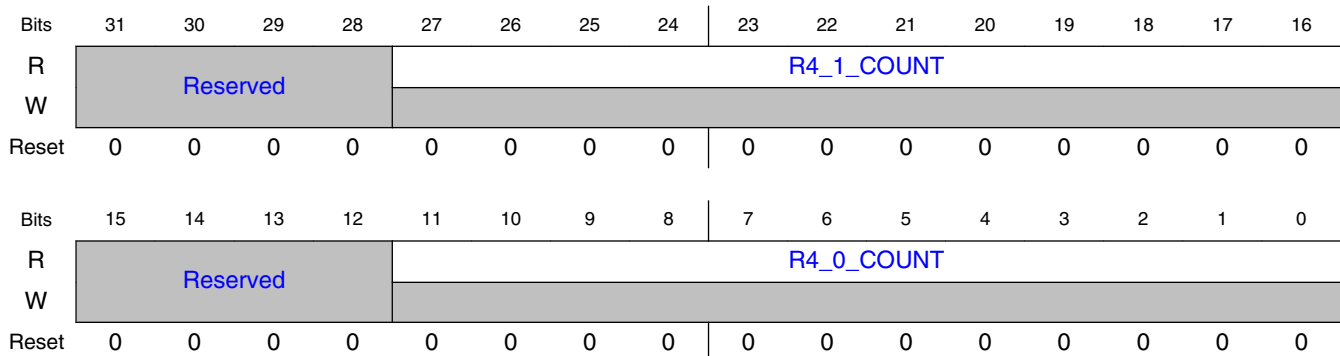
14.70.1 Offset

Register	Offset	Description
RTSCR4C	630h	Accessible at this address when RTMCTL[PRGM] = 0]

14.70.2 Function

The RNG TRNG Statistical Check Run Length 4 Counters Register is a read-only register used to read the final Run Length 4 counts after entropy generation. These counters start with the value in RTSCR4L[RUN4_MAX]. The R4_1_COUNT decrements each time four consecutive ones are sampled (preceded by a zero and followed by a zero). The R4_0_COUNT decrements each time four consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0630h) is used as RTSCR4L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR4C readback register, as described here.

14.70.3 Diagram



14.70.4 Fields

Field	Function
31-28 —	Reserved. Always 0.
27-16 R4_1_COUNT	Runs of One, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-12 —	Reserved. Always 0.
11-0 R4_0_COUNT	Runs of Zero, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.71 RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L)

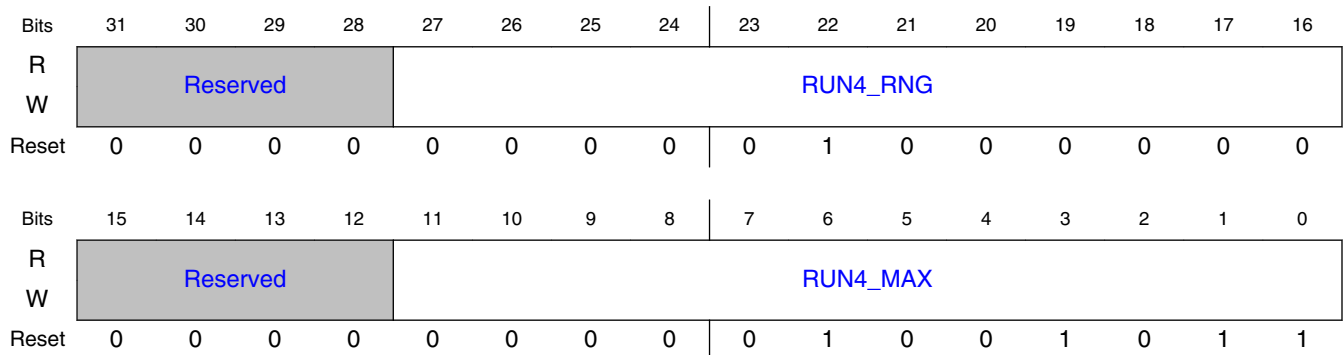
14.71.1 Offset

Register	Offset	Description
RTSCR4L	630h	Accessible at this address when RTMCTL[PRGM] = 1]

14.71.2 Function

The RNG TRNG Statistical Check Run Length 4 Limit Register defines the allowable maximum and minimum number of runs of length 4 detected during entropy generation. To pass the test, the number of runs of length 4 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 4 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0630h) is used as RTSCR4L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR4C readback register.

14.71.3 Diagram



14.71.4 Fields

Field	Function
31-28 —	Reserved. Always 0.
27-16 RUN4_RNG	Run Length 4 Range. The number of runs of length 4 (for both 0 and 1) detected during entropy generation must be greater than RUN4_MAX - RUN4_RNG, else a retry or error will occur. This register is cleared to 0040h (decimal 64) by writing the RTMCTL[RST_DEF] bit to 1.
15-12 —	Reserved. Always 0.
11-0 RUN4_MAX	Run Length 4 Maximum Limit. Defines the maximum allowable runs of length 4 (for both 0 and 1) detected during entropy generation. The number of runs of length 4 detected during entropy generation must be less than RUN4_MAX, else a retry or error will occur. This register is cleared to 004Bh (decimal 75) by writing the RTMCTL[RST_DEF] bit to 1.

14.72 RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C)

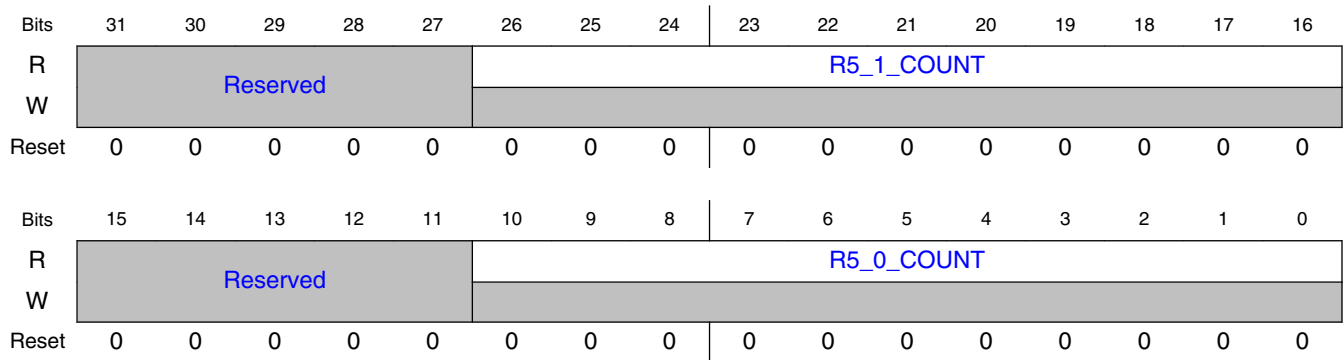
14.72.1 Offset

Register	Offset	Description
RTSCR5C	634h	Accessible at this address when RTMCTL[PRGM] = 0]

14.72.2 Function

The RNG TRNG Statistical Check Run Length 5 Counters Register is a read-only register used to read the final Run Length 5 counts after entropy generation. These counters start with the value in RTSCR5L[RUN5_MAX]. The R5_1_COUNT decrements each time five consecutive ones are sampled (preceded by a zero and followed by a zero). The R5_0_COUNT decrements each time five consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0634h) is used as RTSCR5L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR5C readback register, as described here.

14.72.3 Diagram



14.72.4 Fields

Field	Function
31-27 —	Reserved. Always 0.
26-16 R5_1_COUNT	Runs of One, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-11 —	Reserved. Always 0.
10-0 R5_0_COUNT	Runs of Zero, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.73 RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L)

14.73.1 Offset

Register	Offset	Description
RTSCR5L	634h	Accessible at this address when RTMCTL[PRGM] = 1]

14.73.2 Function

The RNG TRNG Statistical Check Run Length 5 Limit Register defines the allowable maximum and minimum number of runs of length 5 detected during entropy generation. To pass the test, the number of runs of length 5 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 5 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0634h) is used as RTSCR5L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR5C readback register.

14.73.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved								RUN5_RNG							
W	Reserved								RUN5_RNG							
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved								RUN5_MAX							
W	Reserved								RUN5_MAX							
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1

14.73.4 Fields

Field	Function
31-27 —	Reserved. Always 0.
26-16 RUN5_RNG	Run Length 5 Range. The number of runs of length 5 (for both 0 and 1) detected during entropy generation must be greater than RUN5_MAX - RUN5_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1.
15-11 —	Reserved. Always 0.
10-0 RUN5_MAX	Run Length 5 Maximum Limit. Defines the maximum allowable runs of length 5 (for both 0 and 1) detected during entropy generation. The number of runs of length 5 detected during entropy generation must be less than RUN5_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1.

14.74 RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC)

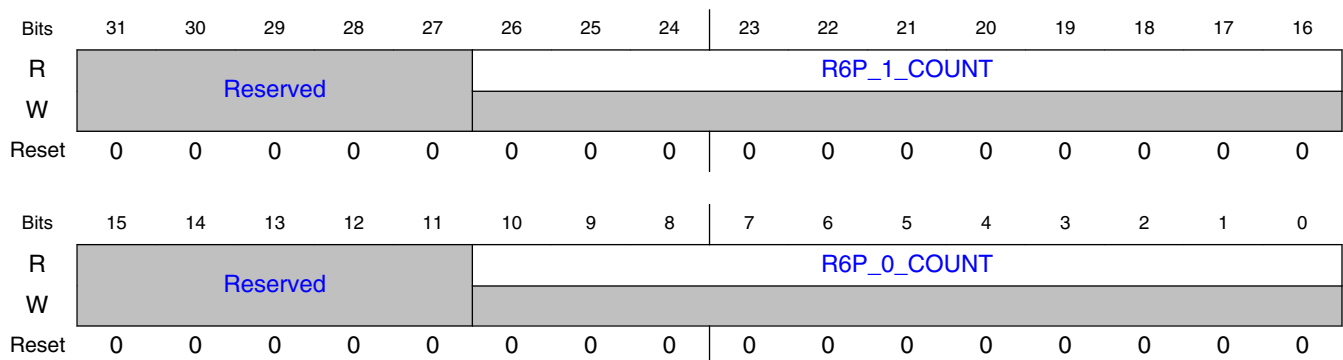
14.74.1 Offset

Register	Offset	Description
RTSCR6PC	638h	Accessible at this address when RTMCTL[PRGM] = 0]

14.74.2 Function

The RNG TRNG Statistical Check Run Length 6+ Counters Register is a read-only register used to read the final Run Length 6+ counts after entropy generation. These counters start with the value in RTSCR6PL[RUN6P_MAX]. The R6P_1_COUNT decrements each time six or more consecutive ones are sampled (preceded by a zero and followed by a zero). The R6P_0_COUNT decrements each time six or more consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0638h) is used as RTSCR6PL if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register, as described here.

14.74.3 Diagram



14.74.4 Fields

Field	Function
31-27 —	Reserved. Always 0.
26-16 R6P_1_COUNT	Runs of One, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0.
15-11 —	Reserved. Always 0.
10-0 R6P_0_COUNT	Runs of Zero, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0.

14.75 RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL)

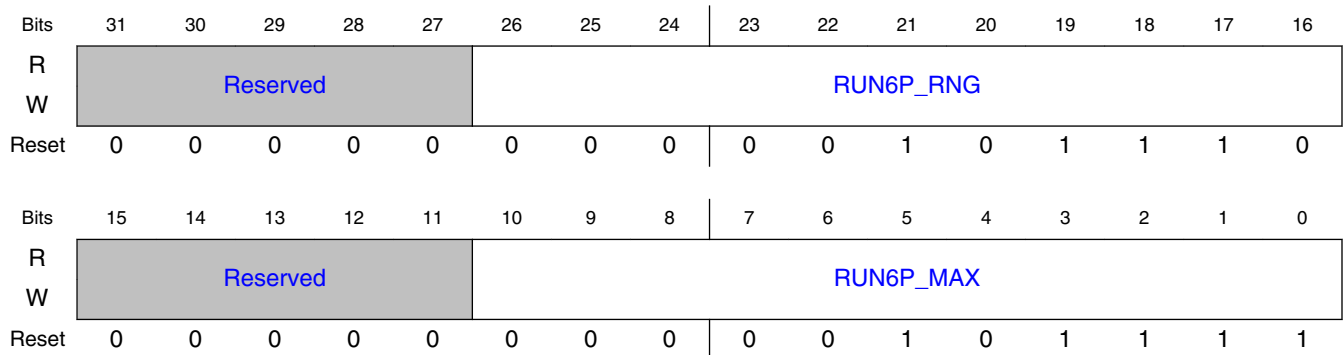
14.75.1 Offset

Register	Offset	Description
RTSCR6PL	638h	Accessible at this address when RTMCTL[PRGM] = 1]

14.75.2 Function

The RNG TRNG Statistical Check Run Length 6+ Limit Register defines the allowable maximum and minimum number of runs of length 6 or more detected during entropy generation. To pass the test, the number of runs of length 6 or more (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 6 or more must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0638h) is used as RTSCR6PL only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register.

14.75.3 Diagram



14.75.4 Fields

Field	Function
31-27 —	Reserved. Always 0.
26-16 RUN6P_RNG	Run Length 6+ Range. The number of runs of length 6 or more (for both 0 and 1) detected during entropy generation must be greater than RUN6P_MAX - RUN6P_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1.
15-11 —	Reserved. Always 0.
10-0 RUN6P_MAX	Run Length 6+ Maximum Limit. Defines the maximum allowable runs of length 6 or more (for both 0 and 1) detected during entropy generation. The number of runs of length 6 or more detected during entropy generation must be less than RUN6P_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1.

14.76 RNG TRNG Status Register (RTSTATUS)

14.76.1 Offset

Register	Offset
RTSTATUS	63Ch

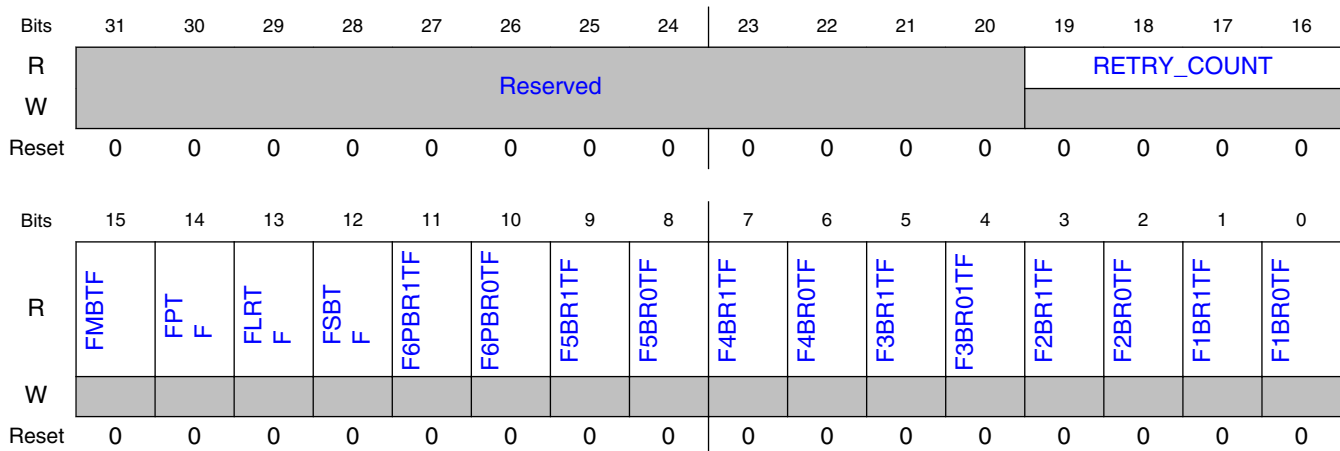
14.76.2 Function

Various statistical tests are run as a normal part of the TRNG's entropy generation process. If the RNG TRNG Miscellaneous Control Register (RTMCTL) ERR field indicates an error, the least-significant 16 bits of the RTSTATUS register will indicate which test(s) have failed. The status of these bits will be valid when the TRNG has finished its entropy generation process. Software can determine when this occurs by polling the ENT_VAL bit in RTMCTL. If RTMCTL[ERR] indicates no error, then RTSTATUS register does not contain valid test status data.

Note that there is a very small probability that a statistical test will fail even though the TRNG is operating properly. If this happens the TRNG will automatically retry the entire entropy generation process, including running all the statistical tests. The value in RETRY_COUNT is decremented each time an entropy generation retry occurs. If a statistical check fails when the retry count is nonzero, a retry is initiated. But if a statistical check fails when the retry count is zero, an error is generated by the RNG. By default RETRY_COUNT is initialized to 1, but software can increase the retry count by writing to the RTY_CNT field in the RTSCMISC register (see [RNG TRNG Statistical Check Miscellaneous Register \(RTSCMISC\)](#)).

All 0s will be returned if this register address is read while the RNG is in Program Mode (see PRGM field in RTMCTL register (see [RNG TRNG Miscellaneous Control Register \(RTMCTL\)](#))). If this register is read while the RNG is in Run Mode the value returned will be formatted as follows.

14.76.3 Diagram



14.76.4 Fields

Field	Function
31-20 —	Reserved. Always 0.
19-16 RETRY_COUNT	RETRY COUNT. This represents the current number of entropy generation retries left before a statistical text failure will cause the RNG to generate an error condition.
15 FMBTF	Mono Bit Test Fail. If MBTF=1, the Mono Bit Test has failed.
14 FPTF	Poker Test Fail. If PTF=1, the Poker Test has failed.
13 FLRTF	Long Run Test Fail. If LRTF=1, the Long Run Test has failed.
12 FSBTF	Sparse Bit Test Fail. If SBTF=1, the Sparse Bit Test has failed.
11 F6PBR1TF	6 Plus Bit Run, Sampling 1s, Test Fail. If 6PBR1TF=1, the 6 Plus Bit Run, Sampling 1s Test has failed.
10 F6PBR0TF	6 Plus Bit Run, Sampling 0s, Test Fail. If 6PBR0TF=1, the 6 Plus Bit Run, Sampling 0s Test has failed.
9 F5BR1TF	5-Bit Run, Sampling 1s, Test Fail. If 5BR1TF=1, the 5-Bit Run, Sampling 1s Test has failed.
8 F5BR0TF	5-Bit Run, Sampling 0s, Test Fail. If 5BR0TF=1, the 5-Bit Run, Sampling 0s Test has failed.
7 F4BR1TF	4-Bit Run, Sampling 1s, Test Fail. If 4BR1TF=1, the 4-Bit Run, Sampling 1s Test has failed.
6 F4BR0TF	4-Bit Run, Sampling 0s, Test Fail. If 4BR0TF=1, the 4-Bit Run, Sampling 0s Test has failed.
5 F3BR1TF	3-Bit Run, Sampling 1s, Test Fail. If 3BR1TF=1, the 3-Bit Run, Sampling 1s Test has failed.
4 F3BR01TF	3-Bit Run, Sampling 0s, Test Fail. If 3BR0TF=1, the 3-Bit Run, Sampling 0s Test has failed.
3 F2BR1TF	2-Bit Run, Sampling 1s, Test Fail. If 2BR1TF=1, the 2-Bit Run, Sampling 1s Test has failed.
2 F2BR0TF	2-Bit Run, Sampling 0s, Test Fail. If 2BR0TF=1, the 2-Bit Run, Sampling 0s Test has failed.
1 F1BR1TF	1-Bit Run, Sampling 1s, Test Fail. If 1BR1TF=1, the 1-Bit Run, Sampling 1s Test has failed.
0 F1BR0TF	1-Bit Run, Sampling 0s, Test Fail. If 1BR0TF=1, the 1-Bit Run, Sampling 0s Test has failed.

14.77 RNG TRNG Entropy Read Register (RTENT0 - RTENT15)

14.77.1 Offset

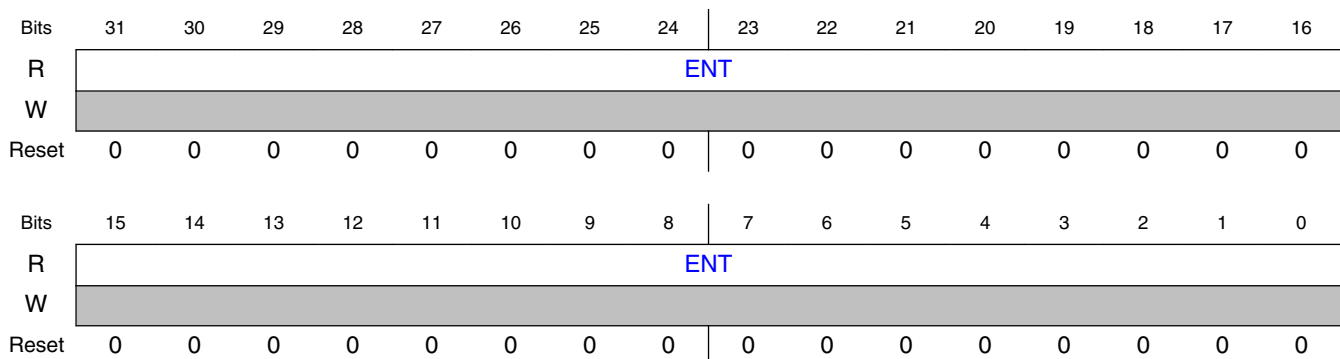
For a = 0 to 15:

Register	Offset
RTENTa	640h + (a × 4h)

14.77.2 Function

The RNG TRNG can be programmed to generate an entropy value that is readable via the SkyBlue bus. To do this, set the RTMCTL[TRNG_ACC] bit to 1. Once the entropy value has been generated, the RTMCTL[ENT_VAL] bit will be set to 1. At this point, RTENT0 through RTENT15 may be read to retrieve the 512-bit entropy value. Note that once RTENT15 is read, the entropy value will be cleared and a new value will begin generation, so it is important that RTENT15 be read last. Also note that the entropy value read from the RTENT0 - RTENT15 registers will never be used by the SEC for any purpose other than to be read via these registers. Any entropy value used for any security function cannot be read. These registers are readable only when RTMCTL[PRGM] = 0 (Run Mode), RTMCTL[TRNG_ACC] = 1 (TRNG access mode) and RTMCTL[ENT_VAL] = 1, otherwise zeroes will be read.

14.77.3 Diagram



14.77.4 Fields

Field	Function
31-0 ENT	Entropy Value. Will be non-zero only if RTMCTL[PRGM] = 0 (Run Mode) and RTMCTL[ENT_VAL] = 1 (Entropy Valid). The most significant bits of the entropy are read from the lowest offset, and the least significant bits are read from the highest offset. Note that reading the highest offset also clears the entire entropy value, and starts a new entropy generation.

14.78 RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)

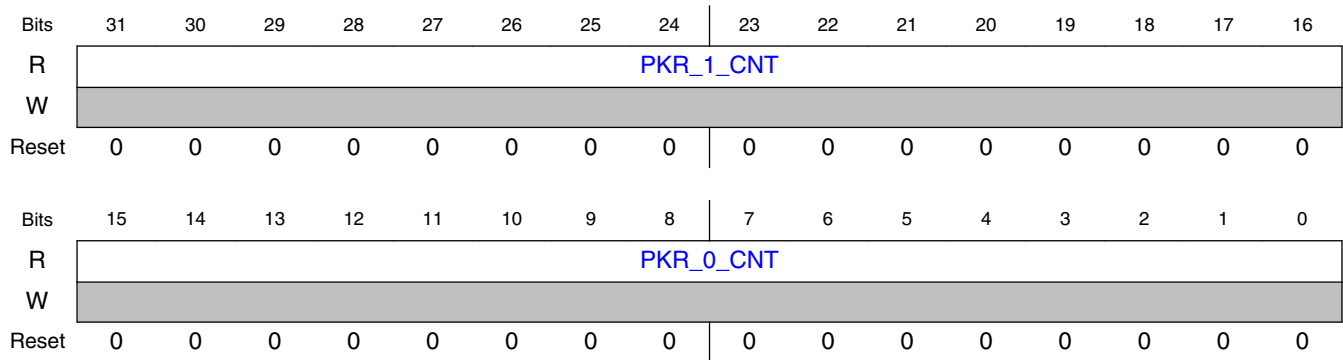
14.78.1 Offset

Register	Offset
RTPKRCNT10	680h

14.78.2 Function

The RNG TRNG Statistical Check Poker Count 1 and 0 Register is a read-only register used to read the final Poker test counts of 1h and 0h patterns. The Poker 0h Count increments each time a nibble of sample data is found to be 0h. The Poker 1h Count increments each time a nibble of sample data is found to be 1h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.78.3 Diagram



14.78.4 Fields

Field	Function
31-16 PKR_1_CNT	Poker 1h Count. Total number of nibbles of sample data which were found to be 1h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_0_CNT	Poker 0h Count. Total number of nibbles of sample data which were found to be 0h. Requires RTMCTL[PRGM] = 0.

14.79 RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)

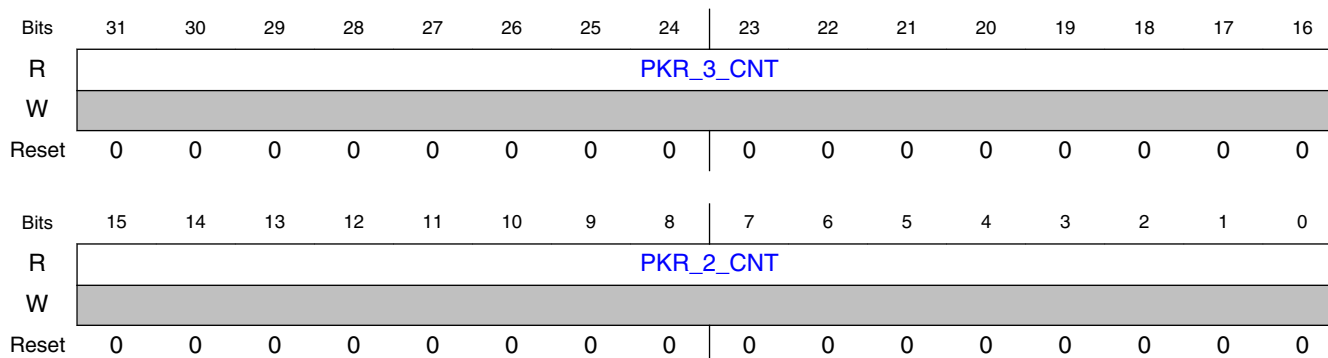
14.79.1 Offset

Register	Offset
RTPKRCNT32	684h

14.79.2 Function

The RNG TRNG Statistical Check Poker Count 3 and 2 Register is a read-only register used to read the final Poker test counts of 3h and 2h patterns. The Poker 2h Count increments each time a nibble of sample data is found to be 2h. The Poker 3h Count increments each time a nibble of sample data is found to be 3h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.79.3 Diagram



14.79.4 Fields

Field	Function
31-16 PKR_3_CNT	Poker 3h Count. Total number of nibbles of sample data which were found to be 3h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_2_CNT	Poker 2h Count. Total number of nibbles of sample data which were found to be 2h. Requires RTMCTL[PRGM] = 0.

14.80 RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)

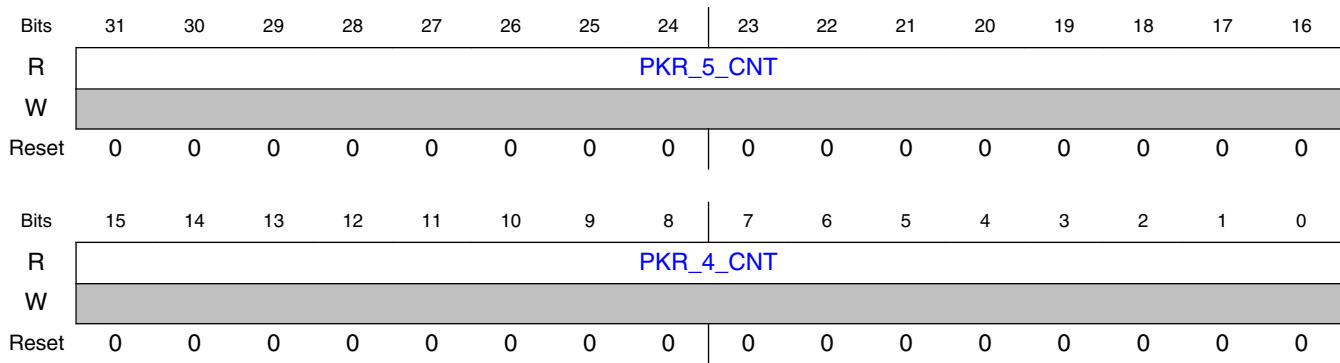
14.80.1 Offset

Register	Offset
RTPKRCNT54	688h

14.80.2 Function

The RNG TRNG Statistical Check Poker Count 5 and 4 Register is a read-only register used to read the final Poker test counts of 5h and 4h patterns. The Poker 4h Count increments each time a nibble of sample data is found to be 4h. The Poker 5h Count increments each time a nibble of sample data is found to be 5h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.80.3 Diagram



14.80.4 Fields

Field	Function
31-16 PKR_5_CNT	Poker 5h Count. Total number of nibbles of sample data which were found to be 5h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_4_CNT	Poker 4h Count. Total number of nibbles of sample data which were found to be 4h. Requires RTMCTL[PRGM] = 0.

14.81 RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)

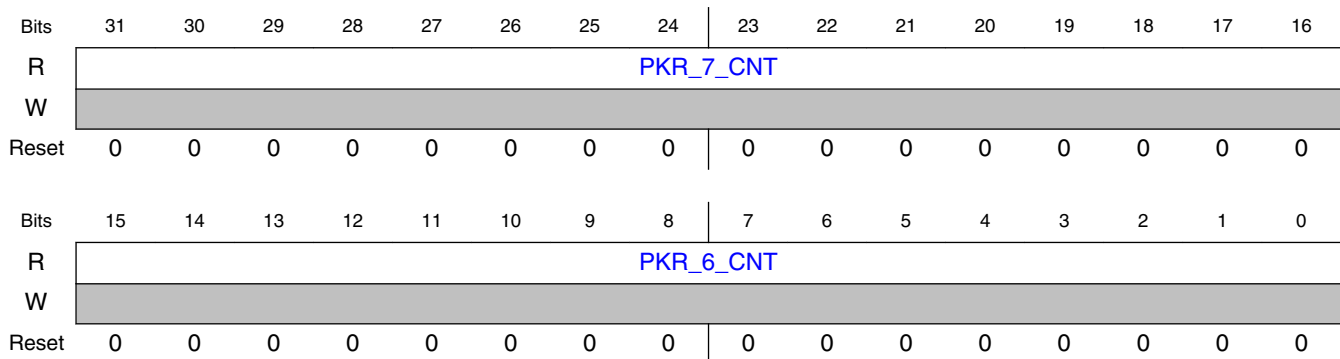
14.81.1 Offset

Register	Offset
RTPKRCNT76	68Ch

14.81.2 Function

The RNG TRNG Statistical Check Poker Count 7 and 6 Register is a read-only register used to read the final Poker test counts of 7h and 6h patterns. The Poker 6h Count increments each time a nibble of sample data is found to be 6h. The Poker 7h Count increments each time a nibble of sample data is found to be 7h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.81.3 Diagram



14.81.4 Fields

Field	Function
31-16 PKR_7_CNT	Poker 7h Count. Total number of nibbles of sample data which were found to be 7h. Requires RTMCTL[PRGM] = 0.
15-0 PKR_6_CNT	Poker 6h Count. Total number of nibbles of sample data which were found to be 6h. Requires RTMCTL[PRGM] = 0.

14.82 RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)

14.82.1 Offset

Register	Offset
RTPKRCNT98	690h

14.82.2 Function

The RNG TRNG Statistical Check Poker Count 9 and 8 Register is a read-only register used to read the final Poker test counts of 9h and 8h patterns. The Poker 8h Count increments each time a nibble of sample data is found to be 8h. The Poker 9h Count increments each time a nibble of sample data is found to be 9h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.82.3 Diagram

Bits	31	30	29	28	27	26	25	24		23	22	21	20	19	18	17	16
R	PKR_9_CNT																
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0
R	PKR_8_CNT																
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

14.82.4 Fields

Field	Function
31-16	Poker 9h Count. Total number of nibbles of sample data which were found to be 9h. Requires RTMCTL[PRGM] = 0.

Table continues on the next page...

RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)

Field	Function
PKR_9_CNT	
15-0 PKR_8_CNT	Poker 8h Count. Total number of nibbles of sample data which were found to be 8h. Requires RTMCTL[PRGM] = 0.

14.83 RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)

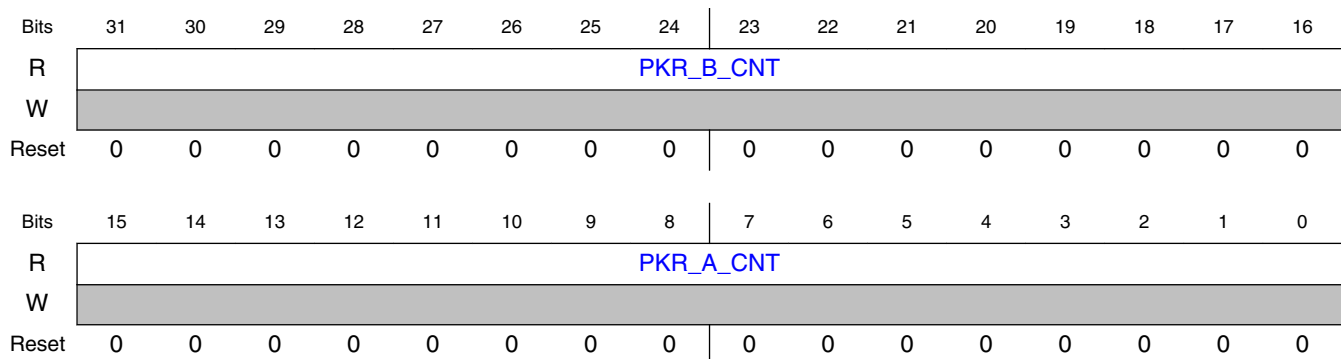
14.83.1 Offset

Register	Offset
RTPKRCNTBA	694h

14.83.2 Function

The RNG TRNG Statistical Check Poker Count B and A Register is a read-only register used to read the final Poker test counts of Bh and Ah patterns. The Poker Ah Count increments each time a nibble of sample data is found to be Ah. The Poker Bh Count increments each time a nibble of sample data is found to be Bh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.83.3 Diagram



14.83.4 Fields

Field	Function
31-16 PKR_B_CNT	Poker Bh Count. Total number of nibbles of sample data which were found to be Bh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_A_CNT	Poker Ah Count. Total number of nibbles of sample data which were found to be Ah. Requires RTMCTL[PRGM] = 0.

14.84 RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)

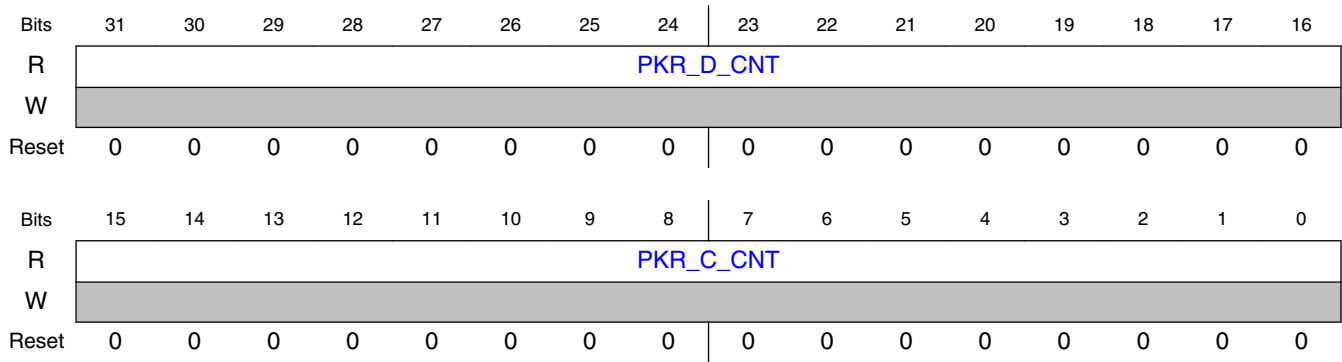
14.84.1 Offset

Register	Offset
RTPKRCNTDC	698h

14.84.2 Function

The RNG TRNG Statistical Check Poker Count D and C Register is a read-only register used to read the final Poker test counts of Dh and Ch patterns. The Poker Ch Count increments each time a nibble of sample data is found to be Ch. The Poker Dh Count increments each time a nibble of sample data is found to be Dh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.84.3 Diagram



14.84.4 Fields

Field	Function
31-16 PKR_D_CNT	Poker Dh Count. Total number of nibbles of sample data which were found to be Dh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_C_CNT	Poker Ch Count. Total number of nibbles of sample data which were found to be Ch. Requires RTMCTL[PRGM] = 0.

14.85 RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)

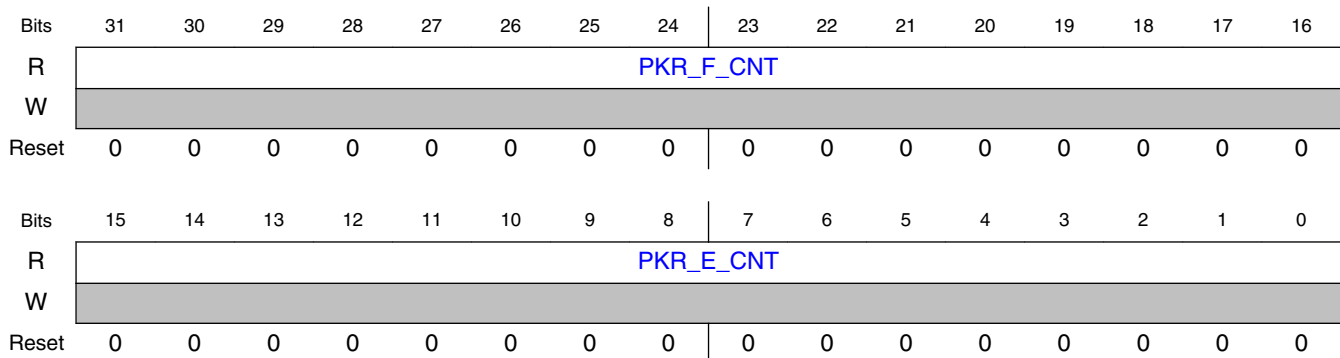
14.85.1 Offset

Register	Offset
RTPKRCNTFE	69Ch

14.85.2 Function

The RNG TRNG Statistical Check Poker Count F and E Register is a read-only register used to read the final Poker test counts of Fh and Eh patterns. The Poker Eh Count increments each time a nibble of sample data is found to be Eh. The Poker Fh Count increments each time a nibble of sample data is found to be Fh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

14.85.3 Diagram



14.85.4 Fields

Field	Function
31-16 PKR_F_CNT	Poker Fh Count. Total number of nibbles of sample data which were found to be Fh. Requires RTMCTL[PRGM] = 0.
15-0 PKR_E_CNT	Poker Eh Count. Total number of nibbles of sample data which were found to be Eh. Requires RTMCTL[PRGM] = 0.

14.86 RNG DRNG Status Register (RDSTA)

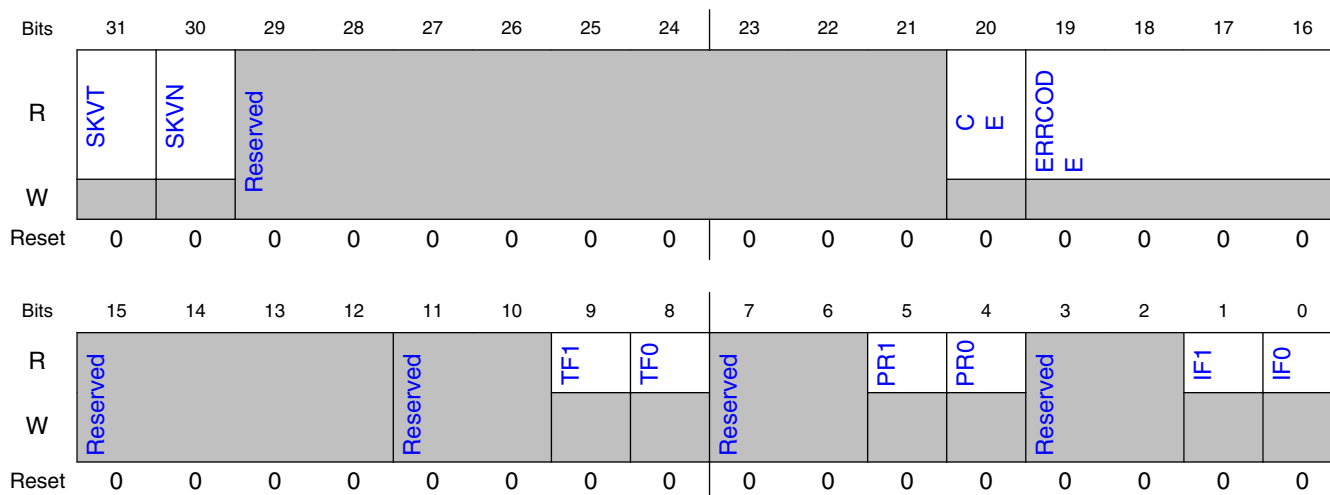
14.86.1 Offset

Register	Offset	Description
RDSTA	6C0h	Accessible at this address when RTMCTL[PRGM] = 0]

14.86.2 Function

The RNG DRNG Status Register shows the current status of the DRNG portion of the RNG.

14.86.3 Diagram



14.86.4 Fields

Field	Function
31 SKVT	Secure Key Valid Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a test (deterministic) instance.
30 SKVN	Secure Key Valid Non-Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a non-test (non-deterministic) instance.
29-21 —	Reserved
20 CE	Catastrophic Error. A catastrophic error will occur when the RNG gets a hardware error while requesting new entropy and the current State Handle is instantiated as a non-test (non-deterministic) instance.
19-16 ERRCODE	Error Code. These bits represent the current error in the RNG.
15-12 —	Reserved
11-10	Reserved

Table continues on the next page...

Field	Function
—	
9 TF1	Test Flag State Handle 1. State handle 1 has been instantiated as a test (deterministic) instance.
8 TF0	Test Flag State Handle 0. State handle 0 has been instantiated as a test (deterministic) instance.
7-6 —	Reserved
5 PR1	Prediction Resistance Flag State Handle 1. State Handle 1 has been instantiated to support prediction resistance.
4 PR0	Prediction Resistance Flag State Handle 0. State Handle 0 has been instantiated to support prediction resistance.
3-2 —	Reserved
1 IF1	Instantiated Flag State Handle 1. State Handle 1 has been instantiated.
0 IF0	Instantiated Flag State Handle 0. State Handle 0 has been instantiated.

14.87 RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)

14.87.1 Offset

Register	Offset
RDINT0	6D0h

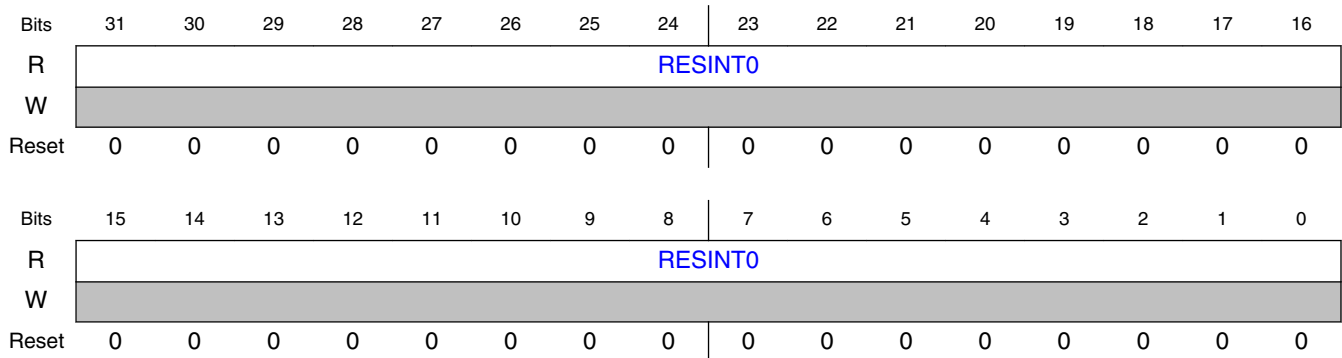
14.87.2 Function

The RNG DRNG State Handle 0 Reseed Interval Register shows the current value of the reseed interval for State Handle 0. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT0 will

RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)

be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT0. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see [RNG operations](#)).

14.87.3 Diagram



14.87.4 Fields

Field	Function
31-0 RESINT0	RESINT0. This read-only register holds the Reseed Interval for State Handle 0.

14.88 RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)

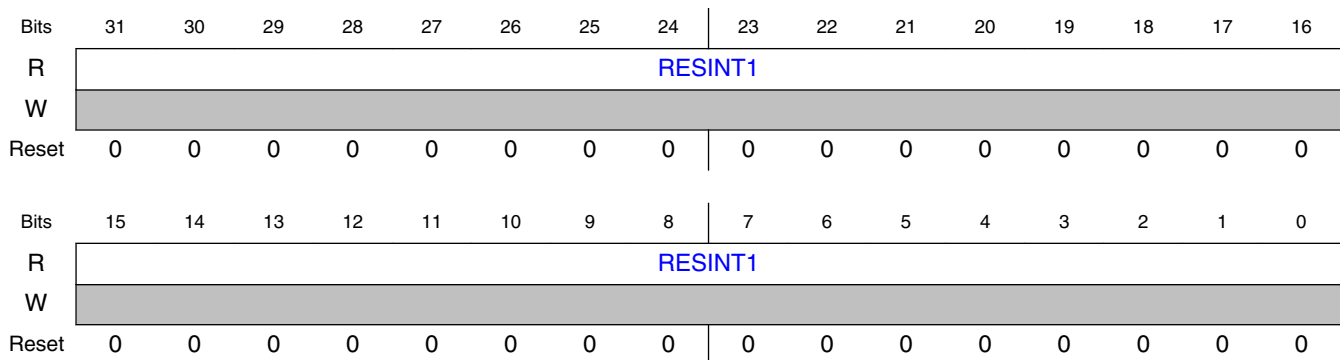
14.88.1 Offset

Register	Offset
RDINT1	6D4h

14.88.2 Function

The RNG DRNG State Handle 1 Reseed Interval Register shows the current value of the reseed interval for State Handle 1. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT1 will be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT1. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see [RNG operations](#)).

14.88.3 Diagram



14.88.4 Fields

Field	Function
31-0 RESINT1	RESINT1. This read-only register holds the Reseed Interval for State Handle 1.

14.89 RNG DRNG Hash Control Register (RDHCNTL)

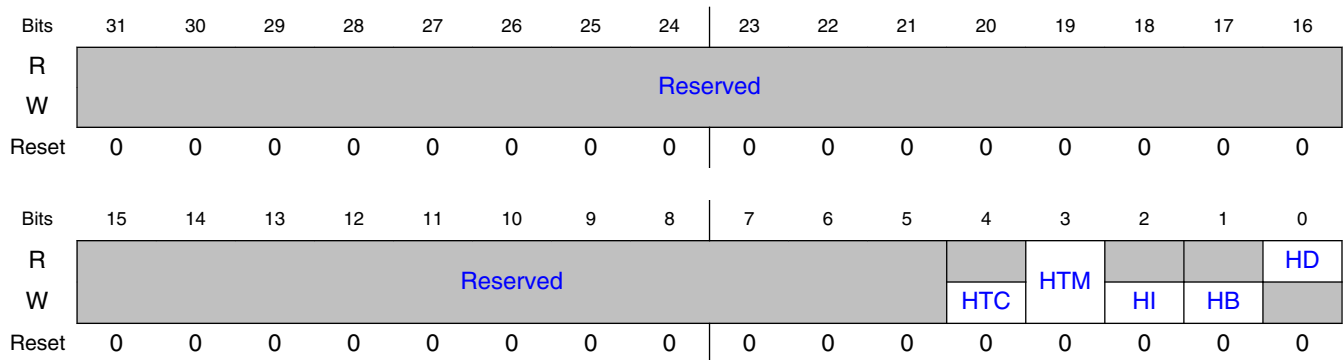
14.89.1 Offset

Register	Offset
RDHCNTL	6E0h

14.89.2 Function

The RNG DRNG Hash Control Register is used to gain control of the SHA-256 hashing engine that is internal to the RNG. Once Hashing test mode is initialized then the user can begin the hashing operation and poll for the done bit.

14.89.3 Diagram



14.89.4 Fields

Field	Function
31-5 —	Reserved
4 HTC	Hashing Test Mode Clear. Writing this bit will take the RNG out of hashing test mode.
3 HTM	Hashing Test Mode. Writing this bit will put RNG in Hashing Test Mode.
2 HI	Hashing Initialize. Writing to this bit will initialize the Hashing Engine.
1	Hashing Begin. Writing this bit will causing the Hashing Engine to begin hashing.

Table continues on the next page...

Field	Function
HB	
0	Hashing Done. This bit asserts when the hashing engine is done.
HD	

14.90 RNG DRNG Hash Digest Register (RDHDIG)

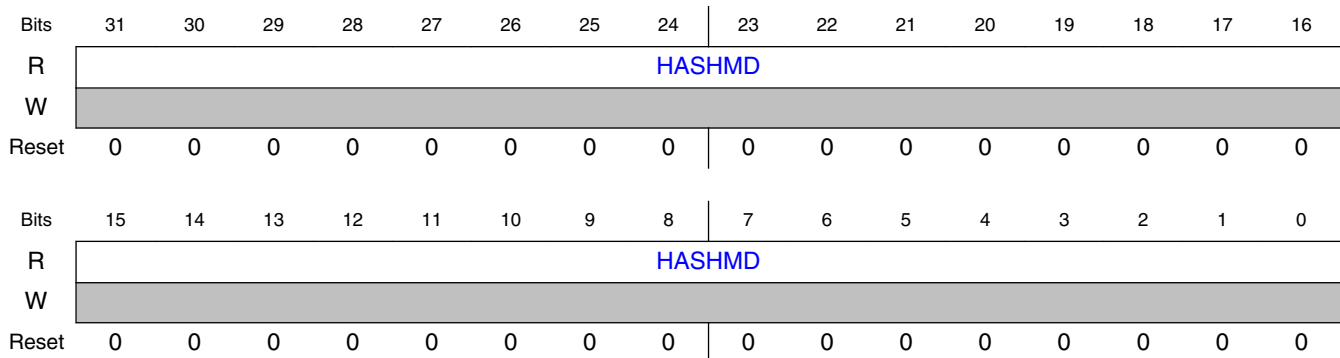
14.90.1 Offset

Register	Offset
RDHDIG	6E4h

14.90.2 Function

The RNG DRNG Hash Digest Register allows user access to the eight 32-bit message digest registers of the SHA-256 hashing engine that is internal to the RNG. All eight registers are read in order from most-significant bits to least-significant bits by reading this address eight times. These registers are only readable while in Hashing Test Mode and when the Hashing Engine is done.

14.90.3 Diagram



14.90.4 Fields

Field	Function
31-0 HASHMD	HASHMD. Hashing Message Digest Register. This register needs to be read 8 times to retrieve the entire message digest.

14.91 RNG DRNG Hash Buffer Register (RDHBUF)

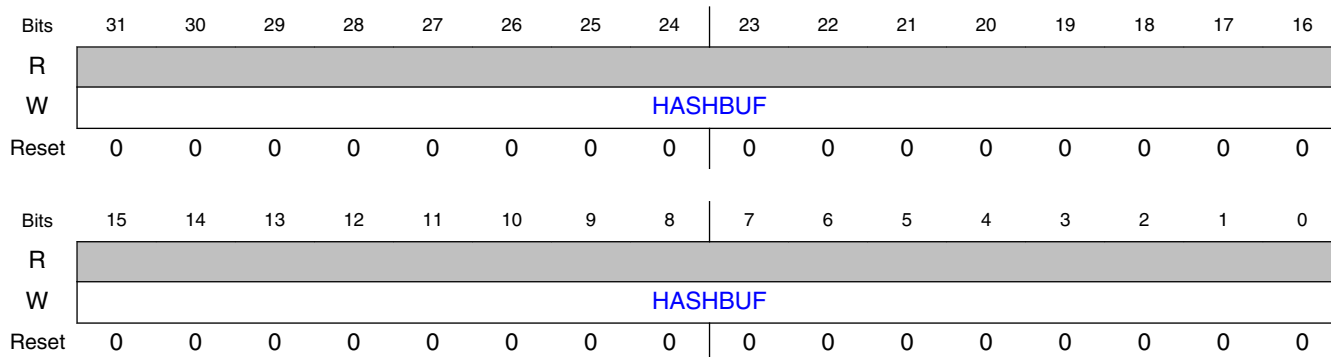
14.91.1 Offset

Register	Offset
RDHBUF	6E8h

14.91.2 Function

The RNG DRNG Hash buffer allows access to the SHA-256 hashing engine that is internal to the RNG for the purpose of conformance testing. To fill the buffer this register must be written 16 times at this address. This register is writable only while the RNG is in Hashing Test mode. This mode can be selected via the RNG DRNG Hash Control Register.

14.91.3 Diagram



14.91.4 Fields

Field	Function
31-0 HASHBUF	HASHBUF. This write-only register provides access to the internal SHA-256 hashing engine's 64-byte buffer. This register must be written 16 times to fill the buffer.

14.92 Recoverable Error Interrupt Status (REIS)

14.92.1 Offset

Register	Offset
REIS	B00h

14.92.2 Function

REIS indicates the assertion status of different SEC recoverable error interrupt sources (1 bit per source). A recoverable error interrupt is generated if any interrupt source bit is asserted and enabled via the REIE register. Software can clear a bit in REIS by writing a 1 to that bit.

14.92.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved				JBAE ₃	JBAE ₂	JBAE ₁	JBAE ₀	Reserved							RBAE
W	Reserved				W1C	W1C	W1C	W1C	Reserved							W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				QFDD	QIVE	QBAE	QHLT	Reserved							CWDE
W	Reserved					W1C	W1C	W1C	Reserved							W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.92.4 Fields

Field	Function
31-28 —	Reserved
27 JBAE3	A job descriptor executed from Job Ring 3 caused a bus access error.
26 JBAE2	A job descriptor executed from Job Ring 2 caused a bus access error.
25 JBAE1	A job descriptor executed from Job Ring 1 caused a bus access error.
24 JBAE0	A job descriptor executed from Job Ring 0 caused a bus access error.
23-17 —	Reserved
16 RBAE	A bus transaction initiated by SEC RTIC resulted in a bus access error.
15-12 —	Reserved
11 QFDD	A SEC Queue Manager Interface frame descriptor was dropped.
10 QIVE	A SEC Queue Manager Interface isolation violation error occurred.
9 QBAE	A job initiated by SEC's Queue Manager Interface resulted in a bus access error.
8 QHLT	SEC's Queue Manager Interface halted due to stop or stop on error.
7-1 —	Reserved
0 CWDE	The SEC watchdog timer expired.

14.93 Recoverable Error Interrupt Enable (REIE)

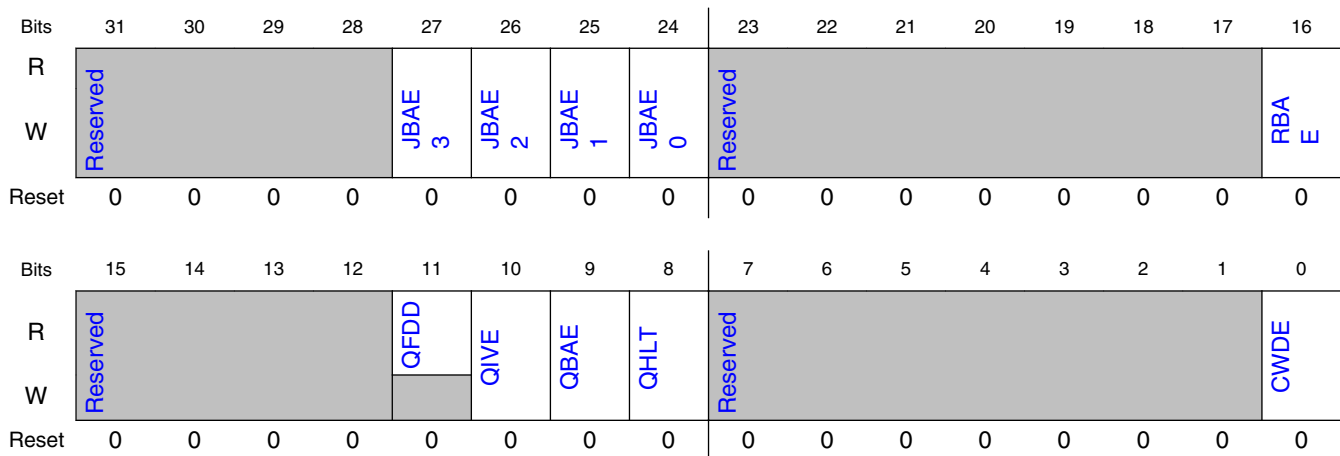
14.93.1 Offset

Register	Offset
REIE	B04h

14.93.2 Function

SEC can generate an interrupt when certain bus errors occur that are potentially recoverable via software. The Recoverable Error Interrupt registers provide information about the bus error that software can use for recovery purposes. The REIE register enables the generation of a recoverable error interrupt (1 bit per source). When a bit in the REIE register is enabled and the corresponding bit in the REIS register is set, SEC will assert the recoverable error interrupt.

14.93.3 Diagram



14.93.4 Fields

Field	Function
31-28 —	Reserved
27 JBAE3	JR3-initiated job execution caused bus access error

Table continues on the next page...

Recoverable Error Interrupt Force (REIF)

Field	Function
26 JBAE2	JR2-initiated job execution caused bus access error
25 JBAE1	JR1-initiated job execution caused bus access error
24 JBAE0	JR0-initiated job execution caused bus access error
23-17 —	Reserved
16 RBAE	RTIC-initiated job execution caused bus access error.
15-12 —	Reserved
11 QFDD	QI frame descriptor dropped
10 QIVE	QI isolation violation error
9 QBAE	QI-initiated job execution caused bus access error
8 QHLT	QI halted (due to stop or stop on error)
7-1 —	Reserved
0 CWDE	SEC watchdog timer expired

14.94 Recoverable Error Interrupt Force (REIF)

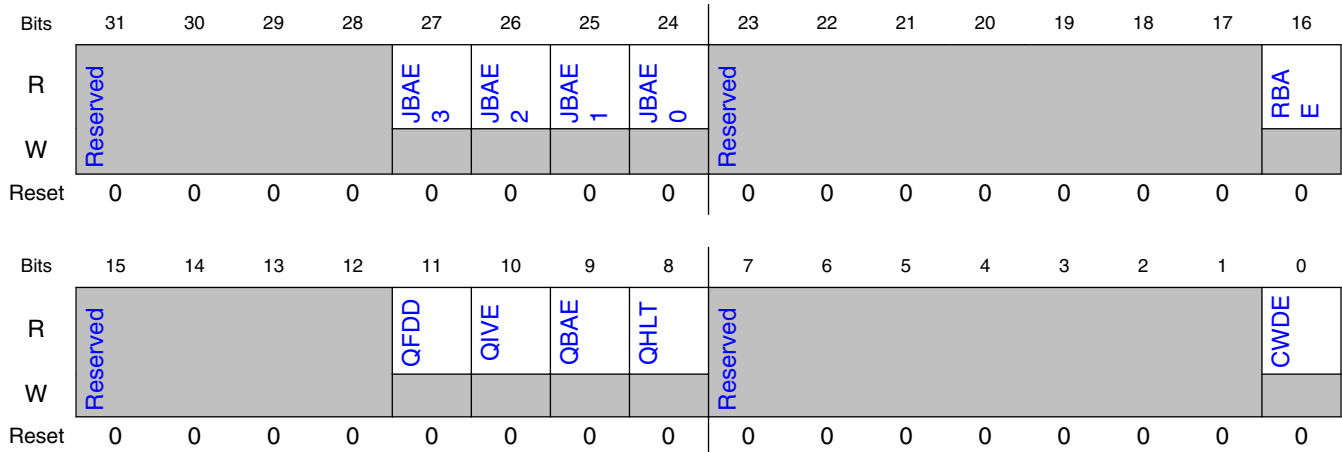
14.94.1 Offset

Register	Offset
REIF	B08h

14.94.2 Function

Writing to REIF sets the associated REIS bits. Reading from REIF returns all zeros.

14.94.3 Diagram



14.94.4 Fields

Field	Function
31-28 —	Reserved
27 JBAE3	JR3-initiated job execution caused bus access error
26 JBAE2	JR2-initiated job execution caused bus access error
25 JBAE1	JR1-initiated job execution caused bus access error
24 JBAE0	JR0-initiated job execution caused bus access error
23-17 —	Reserved
16 RBAE	RTIC-initiated job execution caused bus access error.
15-12 —	Reserved
11 QFDD	QI frame descriptor dropped
10 QIVE	QI isolation violation error

Table continues on the next page...

Recoverable Error Interrupt Halt (REIH)

Field	Function
9 QBAE	QI-initiated job execution caused bus access error
8 QHLT	QI halted (due to stop or stop on error)
7-1 —	Reserved
0 CWDE	SEC watchdog timer expired

14.95 Recoverable Error Interrupt Halt (REIH)

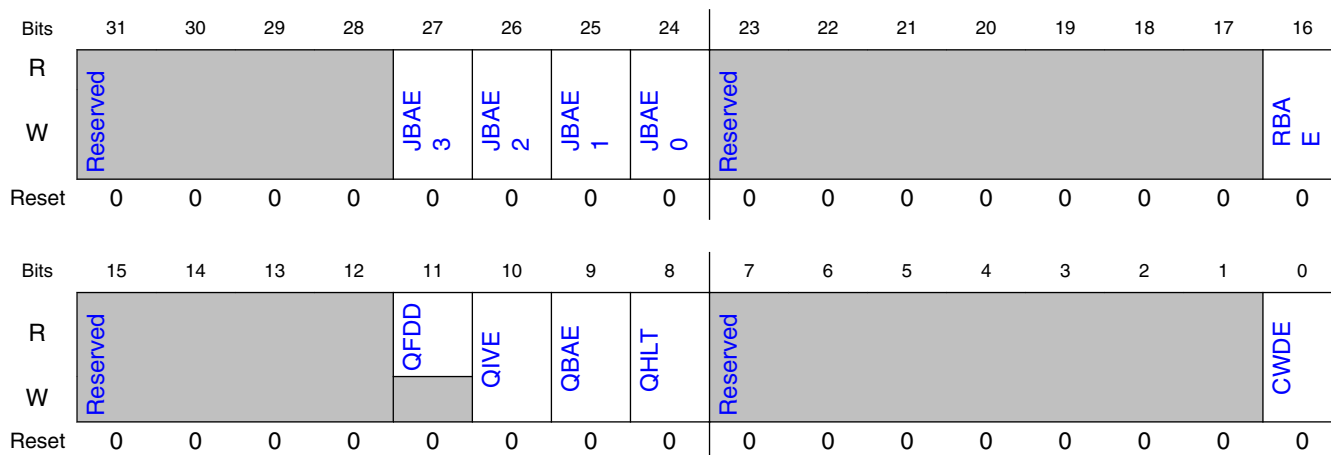
14.95.1 Offset

Register	Offset
REIH	B0Ch

14.95.2 Function

Writing a 1 to an REIH bit indicates that SEC should be halted if the associated recoverable error occurs.

14.95.3 Diagram



14.95.4 Fields

Field	Function
31-28 —	Reserved
27 JBAE3	Halt SEC if JR3-initiated job execution caused bus access error.
26 JBAE2	Halt SEC if JR2-initiated job execution caused bus access error.
25 JBAE1	Halt SEC if JR1-initiated job execution caused bus access error.
24 JBAE0	Halt SEC if JR0-initiated job execution caused bus access error.
23-17 —	Reserved
16 RBAE	Halt SEC if RTIC-initiated job execution caused bus access error.
15-12 —	Reserved
11 QFDD	Halt SEC if QI frame descriptor dropped.
10 QIVE	Halt SEC if QI isolation violation error.
9 QBAE	Halt SEC if QI-initiated job execution caused bus access error.
8 QHLT	Halt SEC if QI halted (due to stop or stop on error).
7-1 —	Reserved
0 CWDE	Halt SEC if SEC watchdog timer expires.

14.96 SEC Version ID Register, most-significant half (SECV ID_MS)

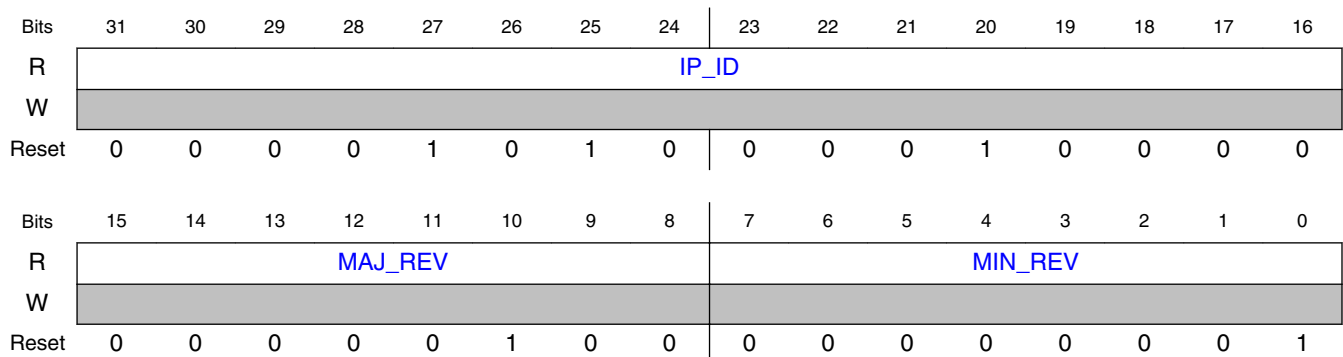
14.96.1 Offset

Register	Offset
SECVID_MS	BF8h (alias)
SECVID_MS	FF8h (alias)
SECVID_MS	1_0FF8h (alias)
SECVID_MS	2_0FF8h (alias)
SECVID_MS	3_0FF8h (alias)
SECVID_MS	4_0FF8h (alias)
SECVID_MS	6_0FF8h (alias)
SECVID_MS	7_0FF8h (alias)
SECVID_MS	8_0FF8h (alias)
SECVID_MS	9_0FF8h (alias)
SECVID_MS	A_0FF8h (alias)
SECVID_MS	B_0FF8h (alias)
SECVID_MS	C_0FF8h (alias)
SECVID_MS	D_0FF8h (alias)

14.96.2 Function

This register contains the ID for SEC and major and minor revision numbers. It also contains the integration options, ECO revision, and configuration options. Since this register holds more than 32 bits, it holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64KB address spaces. The register and its fields are described in the figure and table below.

14.96.3 Diagram



14.96.4 Fields

Field	Function
31-16 IP_ID	ID for SEC.
15-8 MAJ_REV	Major revision number for SEC.
7-0 MIN_REV	Minor revision number for SEC.

14.97 SEC Version ID Register, least-significant half (SECV ID_LS)

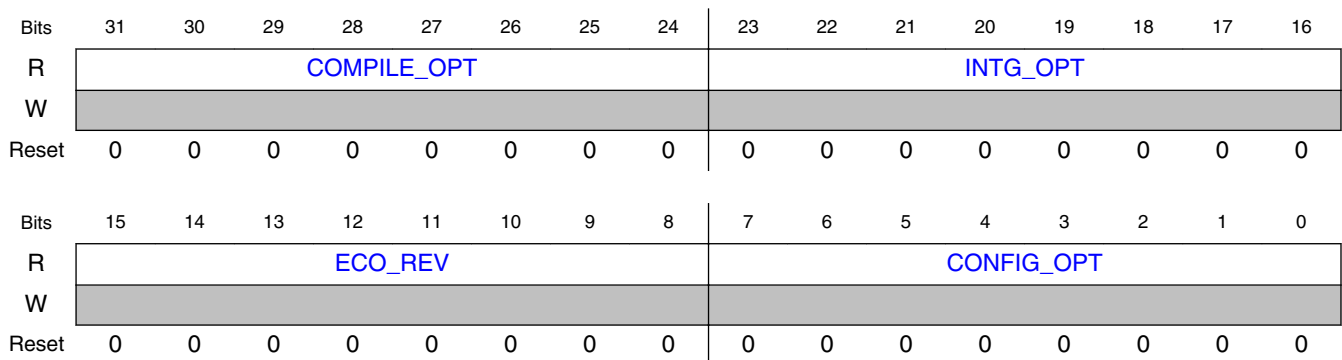
14.97.1 Offset

Register	Offset
SECVID_LS	BFCh (alias)
SECVID_LS	FFCh (alias)
SECVID_LS	1_0FFCh (alias)
SECVID_LS	2_0FFCh (alias)
SECVID_LS	3_0FFCh (alias)
SECVID_LS	4_0FFCh (alias)
SECVID_LS	6_0FFCh (alias)
SECVID_LS	7_0FFCh (alias)
SECVID_LS	8_0FFCh (alias)
SECVID_LS	9_0FFCh (alias)
SECVID_LS	A_0FFCh (alias)
SECVID_LS	B_0FFCh (alias)
SECVID_LS	C_0FFCh (alias)
SECVID_LS	D_0FFCh (alias)

14.97.2 Function

This register contains the ID for SEC and major and minor revision numbers. It also contains the integration options, ECO revision, and configuration options. Since this register holds more than 32 bits, it holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The register and its fields are described in the figure and table below.

14.97.3 Diagram



14.97.4 Fields

Field	Function
31-24 COMPILE_OPT	Compile options for SEC.
23-16 INTG_OPT	Integration options for SEC.
15-8 ECO_REV	ECO revision for SEC.
7-0 CONFIG_OPT	Configuration options for SEC.

14.98 Holding Tank 0 Job Descriptor Address (HT0_JD_A DDR)

14.98.1 Offset

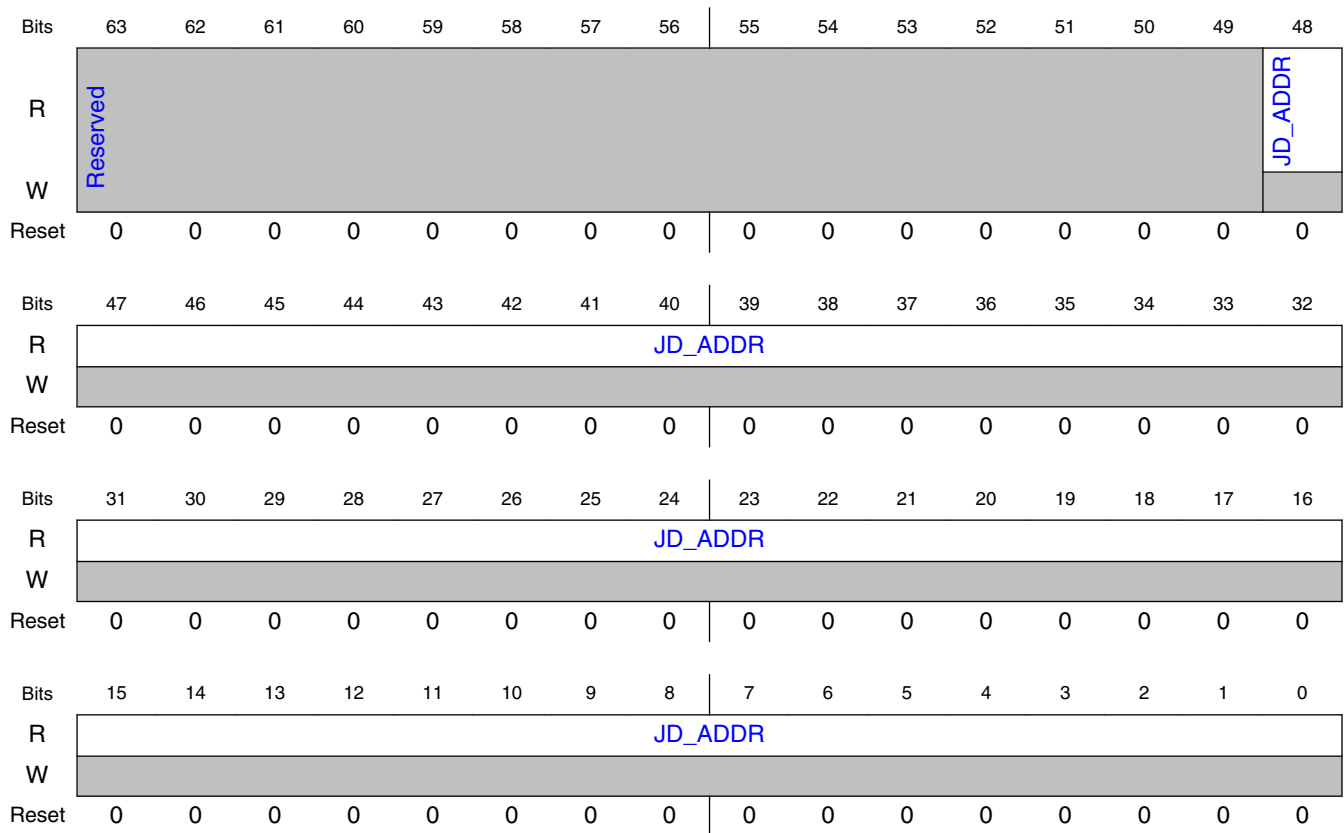
Register	Offset	Description
HT0_JD_ADDR	C00h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR) .

14.98.2 Function

The HTa_JD_ADDR register holds the address of a Job Descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution.

The [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#) HT_SEL field controls which holding tank supplies the Job Descriptor Address to the HTa_JD_ADDR.

14.98.3 Diagram



14.98.4 Fields

Field	Function
63-49 —	Reserved
48-0 JD_ADDR	Job Descriptor Address.

14.99 Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)

14.99.1 Offset

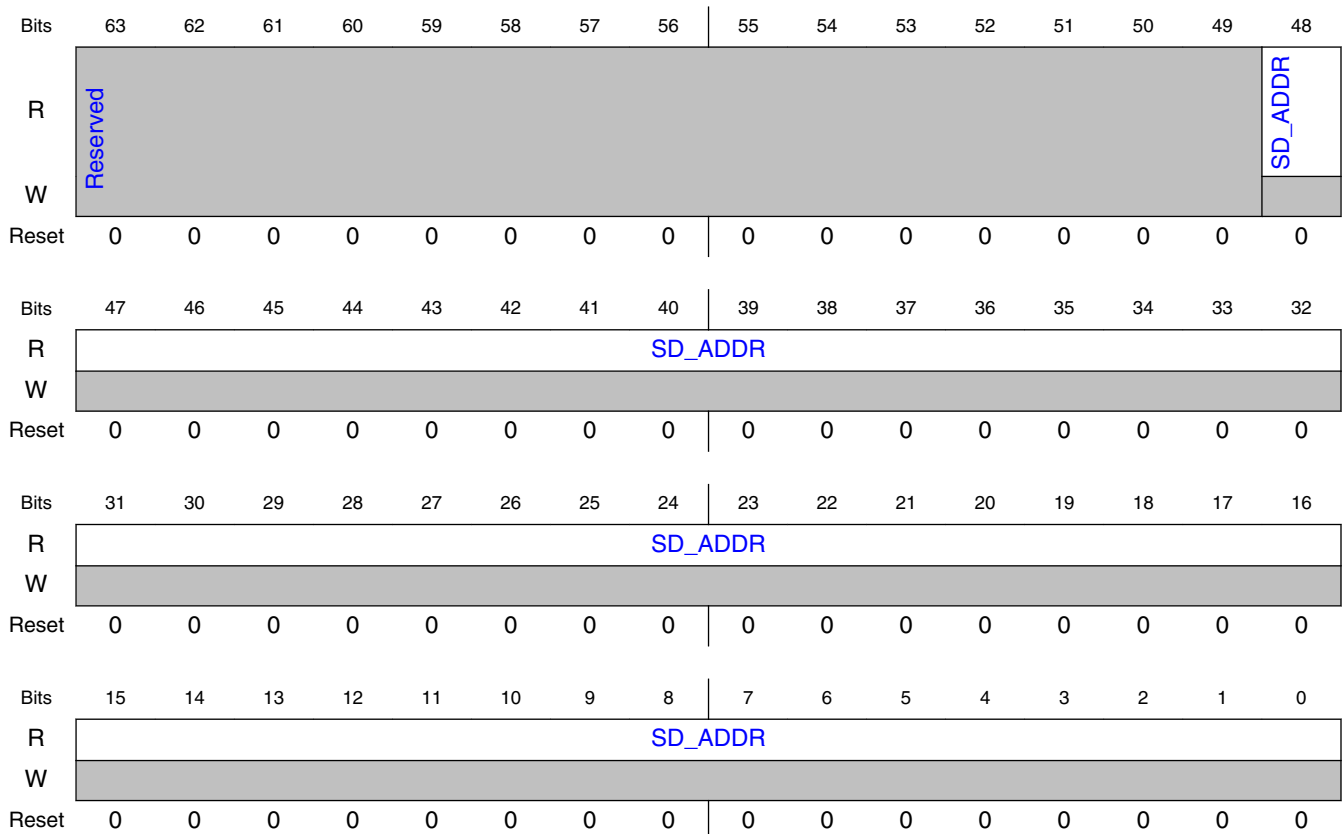
Register	Offset	Description
HT0_SD_ADDR	C08h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.99.2 Function

The HTa_SD_ADDR register holds the address of a Shared Descriptor that is in a Holding Tank waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution via a Job Ring.

The [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#) HT_SEL field controls which holding tank supplies the Shared Descriptor Address to the HTa_SD_ADDR.

14.99.3 Diagram



14.99.4 Fields

Field	Function
63-49 —	Reserved
48-0 SD_ADDR	Shared Descriptor Address.

14.100 Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS)

14.100.1 Offset

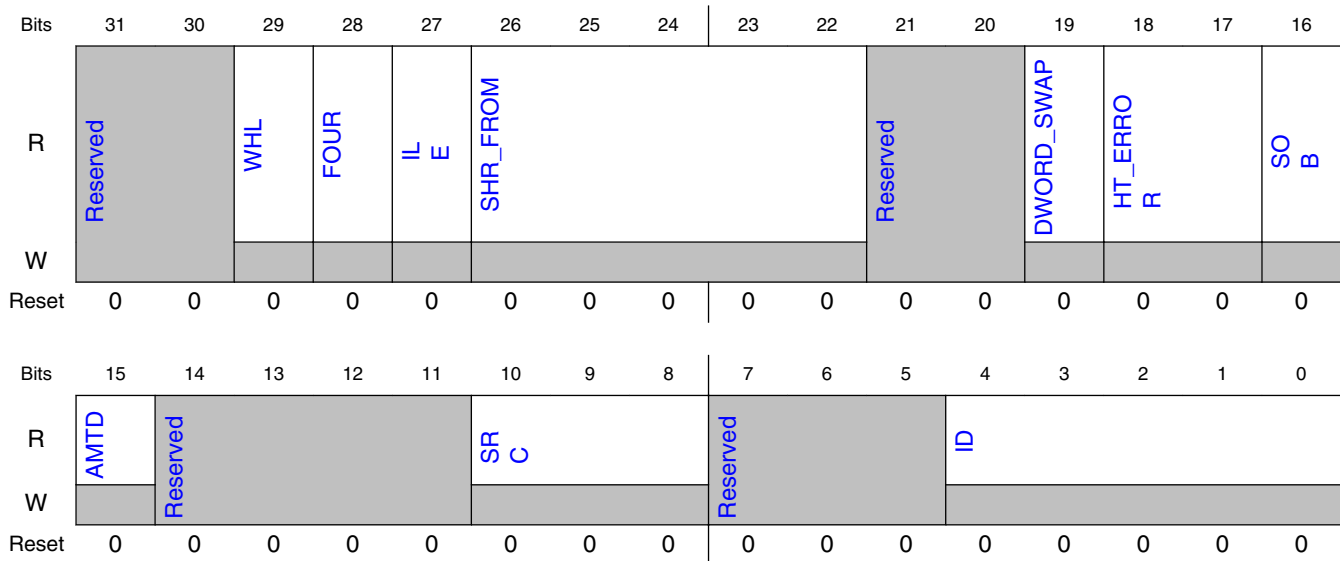
Register	Offset
HT0_JQ_CTRL_MS	C10h

14.100.2 Function

The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#) HT_SEL field controls which holding tank supplies the Shared Descriptor Address to the HTa_SD_ADDR.

14.100.3 Diagram



14.100.4 Fields

Field	Function
31-30 —	Reserved
29 WHL	Whole Descriptor. In versions of SEC that implement prefetching, the WHL field is interpreted in combination with the SOB field. In versions that don't implement prefetching, WHL=1 indicates that HT is passing the full job descriptor to DECO and therefore DECO does not need to fetch any additional Job Descriptor words from external memory.
28 FOUR	Four Words. Job Queue Controller will pass at least 4 words of the descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. ILE = 0: No byte-swapping is performed for data transferred to or from the Descriptor Buffer. ILE = 1: Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.
26-22 SHR_FROM	Share From. This is the DECO block from which the DECO block that runs this job will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software).
21-20 —	Reserved
19 DWORD_SWAP	Double Word Swap. 0b - DWords are in the order most-significant word, least-significant word.

Table continues on the next page...

Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)

Field	Function
	1b - DWords are in the order least-significant word, most-significant word.
18-17 HT_ERROR	Holding Tank Error. (This field is implemented only in versions of SEC that support prefetching.) 00b - No error 01b - Job Descriptor or Shared Descriptor length error 10b - AXI_error while reading a Job Ring or QI Shared Descriptor or the remainder of a Job Ring Job Descriptor 11b - AXI error while reading QI or AI input frame data
16 SOB	Shared or Burst. (This field is implemented only in versions of SEC that support prefetching.) The SOB field is interpreted along with the WHL field as follows: SOB=0 WHL=0 - No prefetch, not whole descriptor SOB=0 WHL=1 - Got whole Job Descriptor, no Shared Descriptor or input frame data SOB=1 WHL=0 - Got Shared Descriptor, no input frame data SOB=1 WHL=1 - Got whole Job Descriptor and input frame data
15 AMTD	Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRaCID Register.
14-11 —	Reserved
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGJRaMS) the DMA should use for bus transactions. It is illegal for the SRC field to have a value other than that of a Job Ring when running descriptors via the IP bus (i.e. under the direct control of software). 000b - Job Ring 0 001b - Job Ring 1 010b - Job Ring 2 011b - Job Ring 3 100b - RTIC 101b - QI 110b - AI 111b - Reserved
7-5 —	Reserved
4-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.

14.101 Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)

14.101.1 Offset

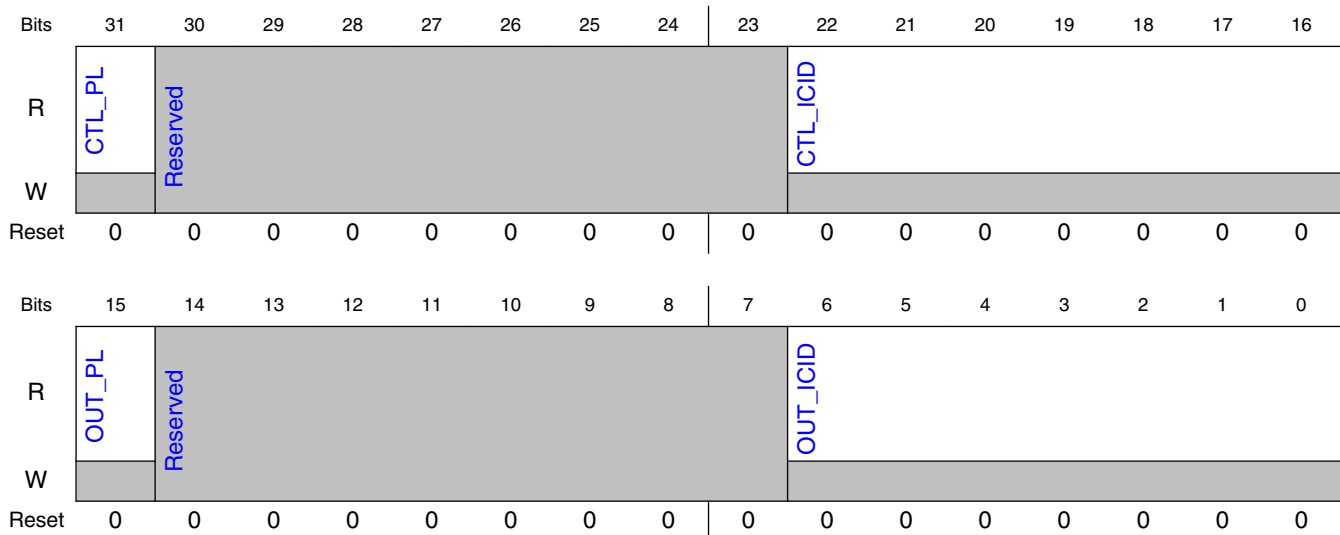
Register	Offset
HT0_JQ_CTRL_LS	C14h

14.101.2 Function

The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#) HT_SEL field controls which holding tank supplies the Job Queue control data to the HTa_JQ_CTRL_MS.

14.101.3 Diagram



14.101.4 Fields

Field	Function
31	Control Privilege Level. When this descriptor is fetched and executed, this value will be asserted on the AXI Master bus's PL signal along with the Control ICID.

Table continues on the next page...

Holding Tank Status (HT0_STATUS)

Field	Function
CTL_PL	
30-23 —	Reserved.
22-16 CTL_ICID	Control ICID This field defines the Control ICID value asserted for DMA transactions when the descriptor is fetched and executed.
15 OUT_PL	Output Privilege Level. When this descriptor is fetched and executed, this value will be asserted on the AXI Master bus's PL signal along with the Output ICID. Note that for Job Ring jobs the Output PL is always the same as the Control PL.
14-7 —	Reserved.
6-0 OUT_ICID	This field defines the Output ICID value asserted for DMA transactions when the descriptor is fetched and executed. Note that for Job Ring jobs the Output ICID is always the same as the Control ICID.

14.102 Holding Tank Status (HT0_STATUS)

14.102.1 Offset

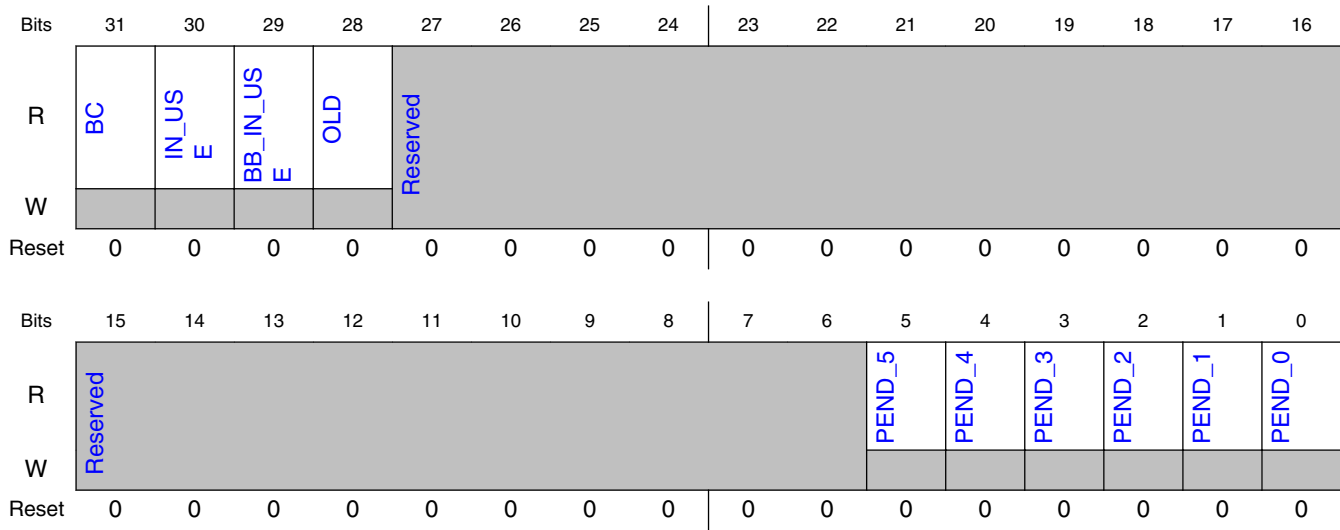
Register	Offset
HT0_STATUS	C1Ch

14.102.2 Function

The HT0_STATUS register holds the status information for a Job Descriptor "holding tank". The register is intended to be used when debugging descriptor execution.

The HT_SEL field in the [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#) controls which holding tank supplies the status information to the HT0_STATUS.

14.102.3 Diagram



14.102.4 Fields

Field	Function
31 BC	Been Changed. When using the Holding Tank debug registers, the Holding Tank Job Descriptor Address register should be the first register that is read. The BC ("Been Changed") bit is cleared when the Holding Tank Job Descriptor Address register is read. If data in the holding tanks changes after that time but before the HT Status register is read, the "Been Changed" bit is set. This indicates that the data read from some of the HT debug registers may be inconsistent with data read from other HT debug registers. In this case the HT debug registers should be reread, starting with the Holding Tank Job Descriptor Address register.
30 IN_USE	In Use. The "In use" bit is set when the HT contains some or all of the information for a job that has not yet been sent or not yet completely sent to a DECO.
29 BB_IN_USE	BB In Use. The "BB In use" bit is set when the burst buffer contains some or all of the input frame data for a job that has not yet been sent or not yet completely sent to a DECO.
28 OLD	Old BB data. The "OLD" bit is set when the burst buffer contains input frame data for the job that was previously in the associated HT. This input frame data has not yet been completely sent to a DECO, so the burst buffer is not available for the job currently in the HT.
27-6 —	Reserved.
5 PEND_5	Pending for DECO 5. The PEND_5 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 5. It is possible for more than one pending bit for the holding tank to be set at the same time.
4 PEND_4	Pending for DECO 4. The PEND_4 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 4. It is possible for more than one pending bit for the holding tank to be set at the same time.

Table continues on the next page...

Job Queue Debug Select Register (JQ_DEBUG_SEL)

Field	Function
3 PEND_3	Pending for DECO 3. The PEND_3 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 3. It is possible for more than one pending bit for the holding tank to be set at the same time.
2 PEND_2	Pending for DECO 2. The PEND_2 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 2. It is possible for more than one pending bit for the holding tank to be set at the same time.
1 PEND_1	Pending for DECO 1. The PEND_1 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 1. It is possible for more than one pending bit for the holding tank to be set at the same time.
0 PEND_0	Pending for DECO 0. The PEND_0 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 0. It is possible for more than one pending bit for the holding tank to be set at the same time.

14.103 Job Queue Debug Select Register (JQ_DEBUG_SEL)

14.103.1 Offset

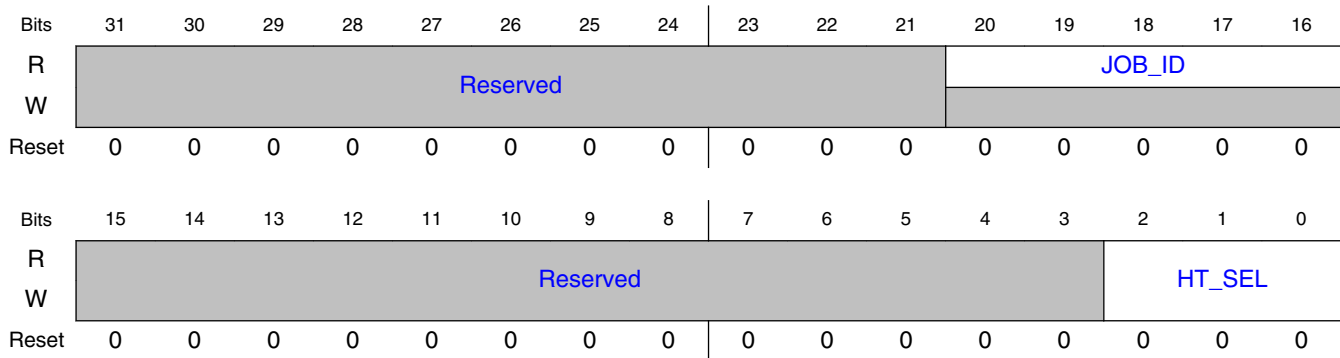
Register	Offset
JQ_DEBUG_SEL	C24h

14.103.2 Function

The Job Queue Debug Select register is used to select which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers). The Job Queue Debug Select register is also used to select the ID of the job that is being queried in the Job Ring Job-Done Source and Job Ring Job-Done Descriptor Address registers. Finally, it specifies which FIFO index to report in the Job Ring Job-Done Job ID FIFO register.

If the value written to the HT_SEL field is larger than the number of holding tanks in SEC, a value of 0 will be stored in the HT_SEL field and Holding Tank 0 will be used by the HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers.

14.103.3 Diagram



14.103.4 Fields

Field	Function
31-21 —	Reserved
20-16 JOB_ID	Job ID. Specifies a Job ID for which to return a Job Source in the Job Ring Job-Done Source FIFO register or Descriptor address in the Job Ring Job-Done Descriptor Address register. Specifies a FIFO index for the Job ID returned by the Job Ring Job-Done Job ID FIFO register, where a value of 0 indicates the oldest job in the FIFO.
15-3 —	Reserved
2-0 HT_SEL	Holding Tank Select. Selects which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers).

14.104 Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS)

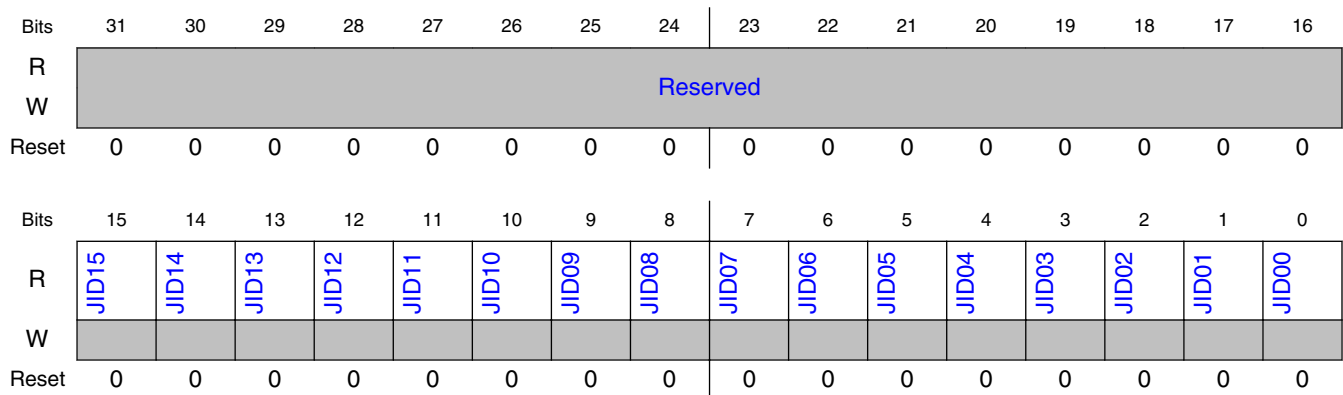
14.104.1 Offset

Register	Offset
JRJIDU_LS	DBCh

14.104.2 Function

The Job Ring Job IDs in Use register indicates which of the Job IDs tracked by the Job Controller are currently in use (i.e. identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring). The register is intended to be used when debugging descriptor execution via a Job Ring. The JRJIDU contains a bit for each of the Job IDs, indicating whether that Job ID is currently in use.

14.104.3 Diagram



14.104.4 Fields

Field	Function
31-16 —	Reserved.
15 JID15	Job ID 15. Job ID 15 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
14 JID14	Job ID 14. Job ID 14 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
13 JID13	Job ID 13. Job ID 13 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
12 JID12	Job ID 12. Job ID 12 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
11 JID11	Job ID 11. Job ID 11 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.

Table continues on the next page...

Field	Function
10 JID10	Job ID 10. Job ID 10 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
9 JID09	Job ID 09. Job ID 09 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
8 JID08	Job ID 08. Job ID 08 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
7 JID07	Job ID 07. Job ID 07 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
6 JID06	Job ID 06. Job ID 06 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
5 JID05	Job ID 05. Job ID 05 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
4 JID04	Job ID 04. Job ID 04 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
3 JID03	Job ID 03. Job ID 03 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
2 JID02	Job ID 02. Job ID 02 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
1 JID01	Job ID 01. Job ID 01 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.
0 JID00	Job ID 00. Job ID 00 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring.

14.105 Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)

14.105.1 Offset

Register	Offset
JRJDJIFBC	DC0h

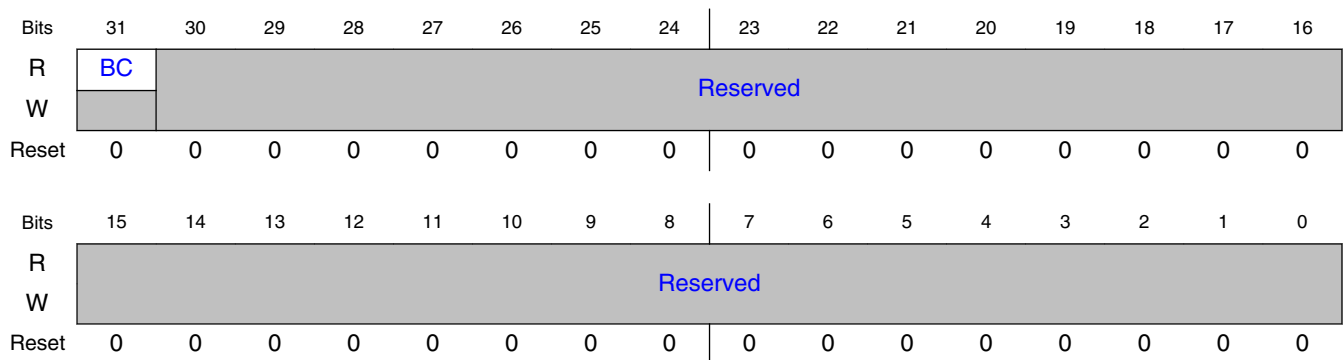
14.105.2 Function

This register indicates whether consistent data has been read from the JRJDI FIFO and JRJDS and JRJDDA registers.

Job Ring Job-Done Job ID FIFO (JRJDJIF)

The Job Queue maintains an ordered list of Job IDs for the completed jobs whose completion status is waiting to be written to a Job Ring output ring. The Job Ring Job-Done Job ID FIFO register returns the Job ID located at the index specified in the JOB_ID field of the [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#). When the JOB_ID field is set to 0, the oldest JOB_ID in the Job-Done FIFO is returned. See Section [Job Ring Output Status Register for Job Ring a \(JRSTAR_JR0 - JRSTAR_JR3\)](#). Note that these Job IDs are not reset as job status is written to output rings, so the completion status for some Job IDs that appear in these registers may already have been written to output rings.

14.105.3 Diagram



14.105.4 Fields

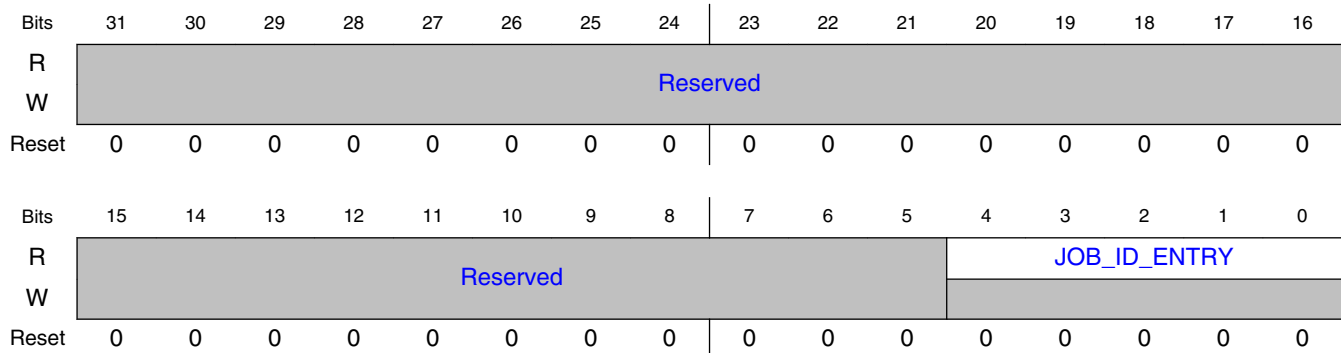
Field	Function
31 BC	Been changed. The hardware sets BC to 0 when the job descriptor address for Job ID 0 is read from Job Ring Job-Done Descriptor Address (JRJDDA). The hardware sets BC to 1 when any job is added or removed from the Job-Done Job ID FIFO. After software reads the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA registers software should read BC. If the BC bit is 1, the results read from the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA may be inconsistent with each other.
30-0 —	Reserved

14.106 Job Ring Job-Done Job ID FIFO (JRJDJIF)

14.106.1 Offset

Register	Offset
JRJDJIF	DC4h

14.106.2 Diagram



14.106.3 Fields

Field	Function
31-5 —	Reserved
4-0 JOB_ID_ENTRY	Job ID entry. This field contains the Job ID of a job whose completion status is located at the JQ_DEBUG_SEL[JOB-ID] index in the Job-Done FIFO.

14.107 Job Ring Job-Done Source 1 (JRJDS1)

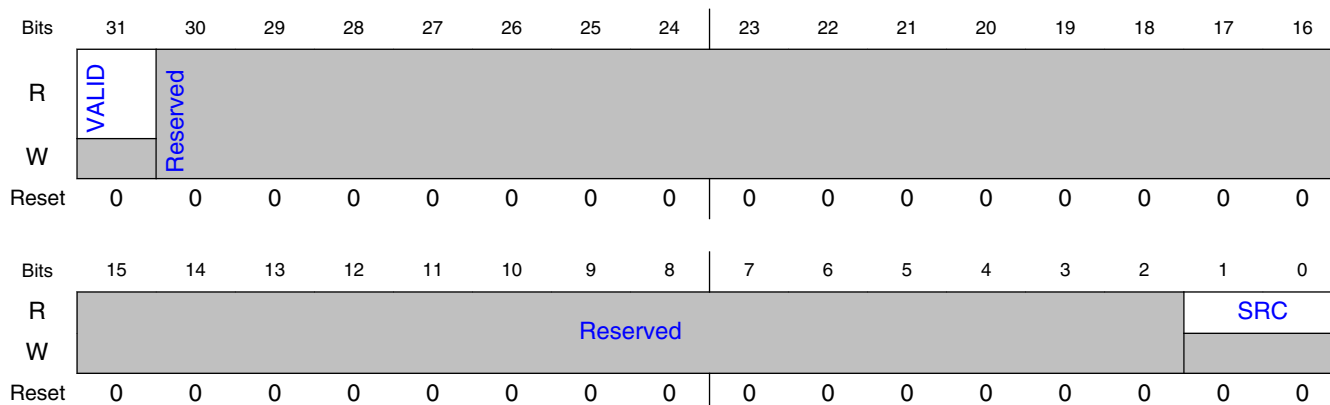
14.107.1 Offset

Register	Offset	Description
JRJDS1	DE4h	The source for the job with the Job ID specified in JQ_DEBUG_SEL[JOB-ID].

14.107.2 Function

The Job Queue keeps track of the job source (Job Ring numbers) for each Job ID, and values in this register are updated whenever a new Job Ring job starts in a holding tank. Each entry in this register is matched to corresponding entries in the JRJDV and JRDDAa registers.

14.107.3 Diagram



14.107.4 Fields

Field	Function
31 VALID	Valid. If this bit is 1, the job located at the index specified by the JOB_ID field in the Job Queue Debug Select register is complete, but its status has not yet been written to the output ring.
30-2 —	Reserved
1-0 SRC	Source. This field contains the number of the Job Ring that was the source of a job whose completion status is waiting to be written to an output ring. The job is located in the FIFO at the index specified by the JOB_ID field in the Job Queue Debug Select register.

14.108 Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)

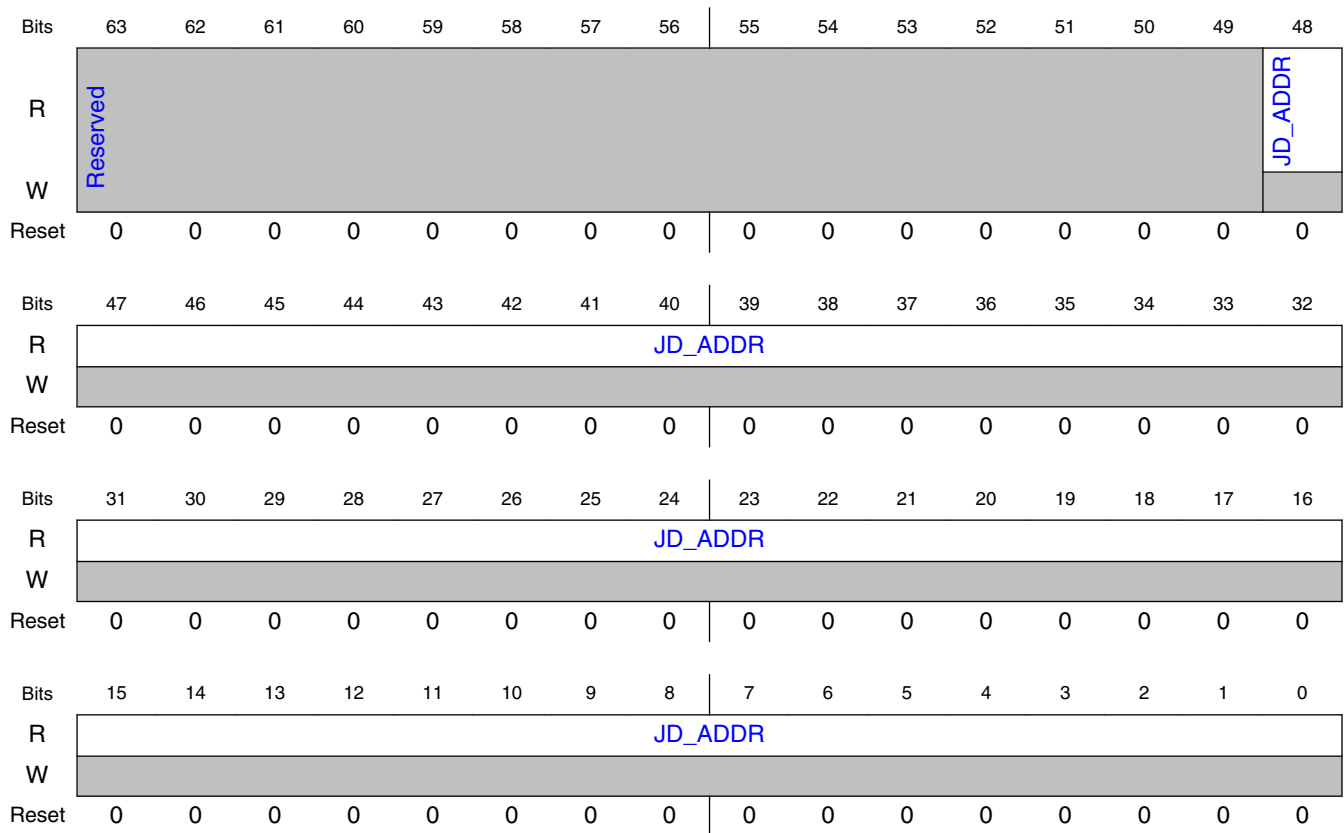
14.108.1 Offset

Register	Offset	Description
JRJDDA	E00h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.108.2 Function

The JRJDDA register is used to store the address of a job descriptor when the job is sent to a holding tank. The descriptor address read from JRJDDA is the one corresponding to the Job ID specified in the JOB_ID field of the [Job Queue Debug Select Register \(JQ_DEBUG_SEL\)](#). See Section [Job Ring Output Status Register for Job Ring a \(JRSTAR_JR0 - JRSTAR_JR3\)](#). Because these addresses are updated only when a new job starts in a holding tank, some addresses read from this register may be for completed jobs that have already been written to an output ring. This register is intended to be used when debugging descriptor execution via a Job Ring.

14.108.3 Diagram



14.108.4 Fields

Field	Function
63-49	Reserved
—	
48-0 JD_ADDR	Job Descriptor Address.

14.109 Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)

14.109.1 Offset

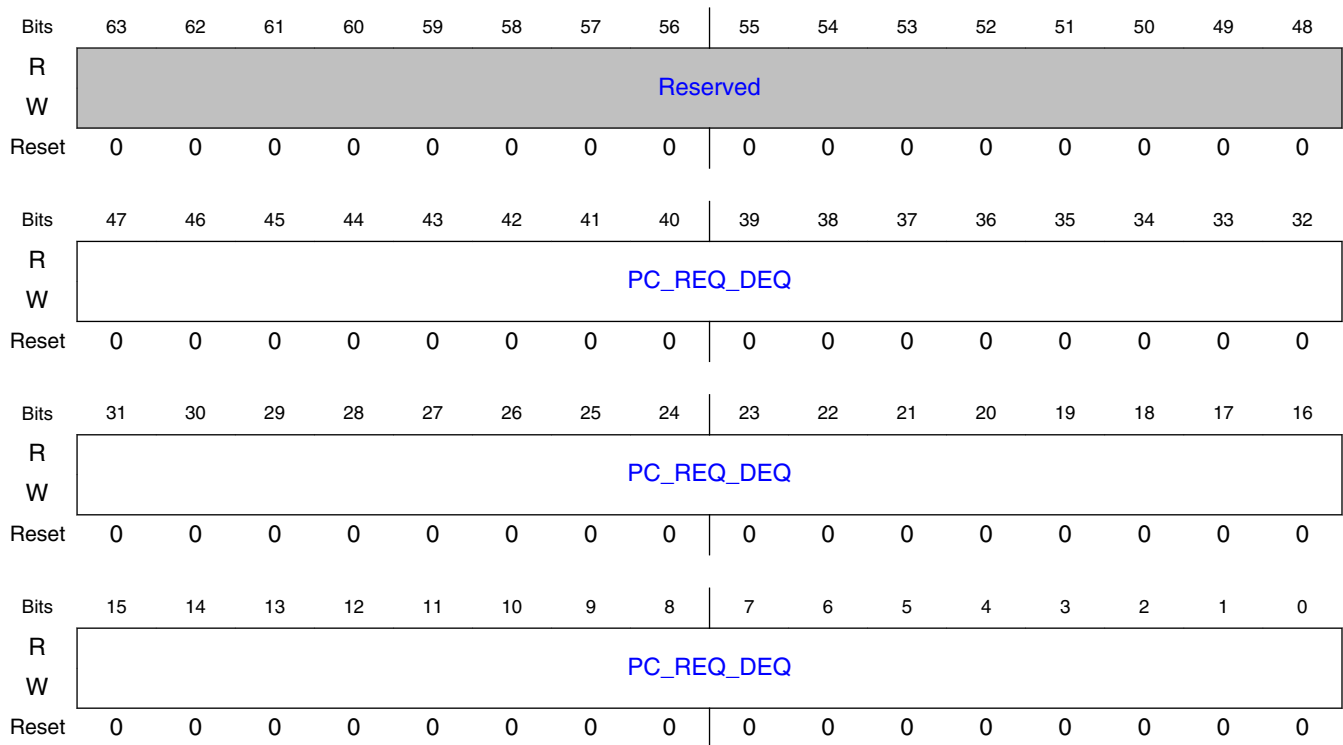
Register	Offset	Description
PC_REQ_DEQ	F00h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_REQ_DEQ	1_0F00h (alias)	-
PC_REQ_DEQ	2_0F00h (alias)	-
PC_REQ_DEQ	3_0F00h (alias)	-
PC_REQ_DEQ	4_0F00h (alias)	-
PC_REQ_DEQ	5_0F00h (alias)	-
PC_REQ_DEQ	6_0F00h (alias)	-
PC_REQ_DEQ	7_0F00h (alias)	-
PC_REQ_DEQ	8_0F00h (alias)	-
PC_REQ_DEQ	9_0F00h (alias)	-
PC_REQ_DEQ	A_0F00h (alias)	-
PC_REQ_DEQ	B_0F00h (alias)	-
PC_REQ_DEQ	C_0F00h (alias)	-
PC_REQ_DEQ	D_0F00h (alias)	-

14.109.2 Function

The PC_REQ_DEQ register counts the total number of DECO jobs that SEC has started. The counter is incremented whenever the job queue controller register in DECO is written to start a job. The job could originate from the Queue Manager Interface, one of the Job Rings, from RTIC or from the register interface when the DECO is under the direct control of software.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.109.3 Diagram



14.109.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_REQ_DEQ	Performance Counter Requests Dequeued.

14.110 Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)

14.110.1 Offset

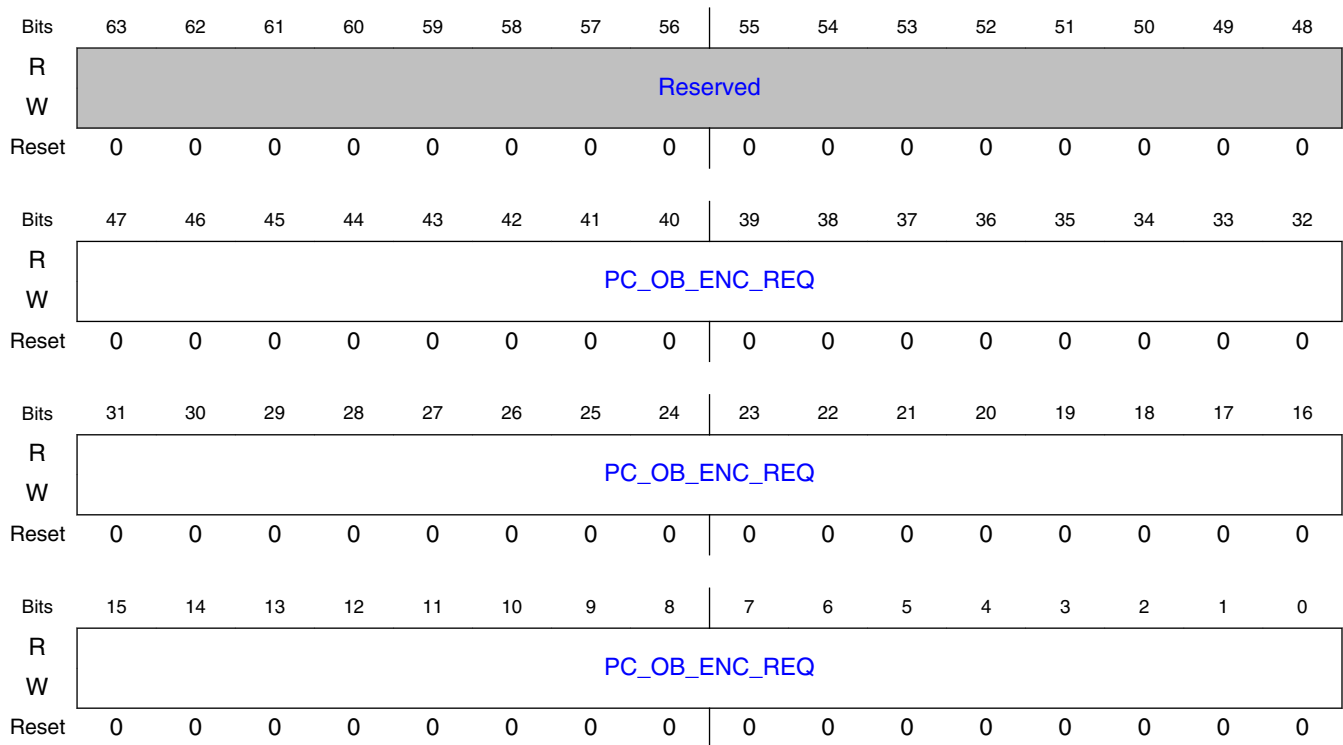
Register	Offset	Description
PC_OB_ENC_REQ	F08h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_OB_ENC_REQ	1_0F08h (alias)	-
PC_OB_ENC_REQ	2_0F08h (alias)	-
PC_OB_ENC_REQ	3_0F08h (alias)	-
PC_OB_ENC_REQ	4_0F08h (alias)	-
PC_OB_ENC_REQ	5_0F08h (alias)	-
PC_OB_ENC_REQ	6_0F08h (alias)	-
PC_OB_ENC_REQ	7_0F08h (alias)	-
PC_OB_ENC_REQ	8_0F08h (alias)	-
PC_OB_ENC_REQ	9_0F08h (alias)	-
PC_OB_ENC_REQ	A_0F08h (alias)	-
PC_OB_ENC_REQ	B_0F08h (alias)	-
PC_OB_ENC_REQ	C_0F08h (alias)	-
PC_OB_ENC_REQ	D_0F08h (alias)	-

14.110.2 Function

The PC_OB_ENC_REQ register counts the total number of requests for symmetric encryption (excluding requests associated with blob encapsulations and encryption of Black Keys). If a descriptor specifies an encryption operation, the register is incremented at the time that the encryption operation completes. Note that a single descriptor containing multiple encryption commands could cause this register to increment more than once. The LSB of the Class 1 Mode register determines if this register or PC_IB_DEC_REQ is incremented.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.110.3 Diagram



14.110.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_OB_ENC_R EQ	Performance Counter Outbound Encryption Requests.

14.111 Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)

14.111.1 Offset

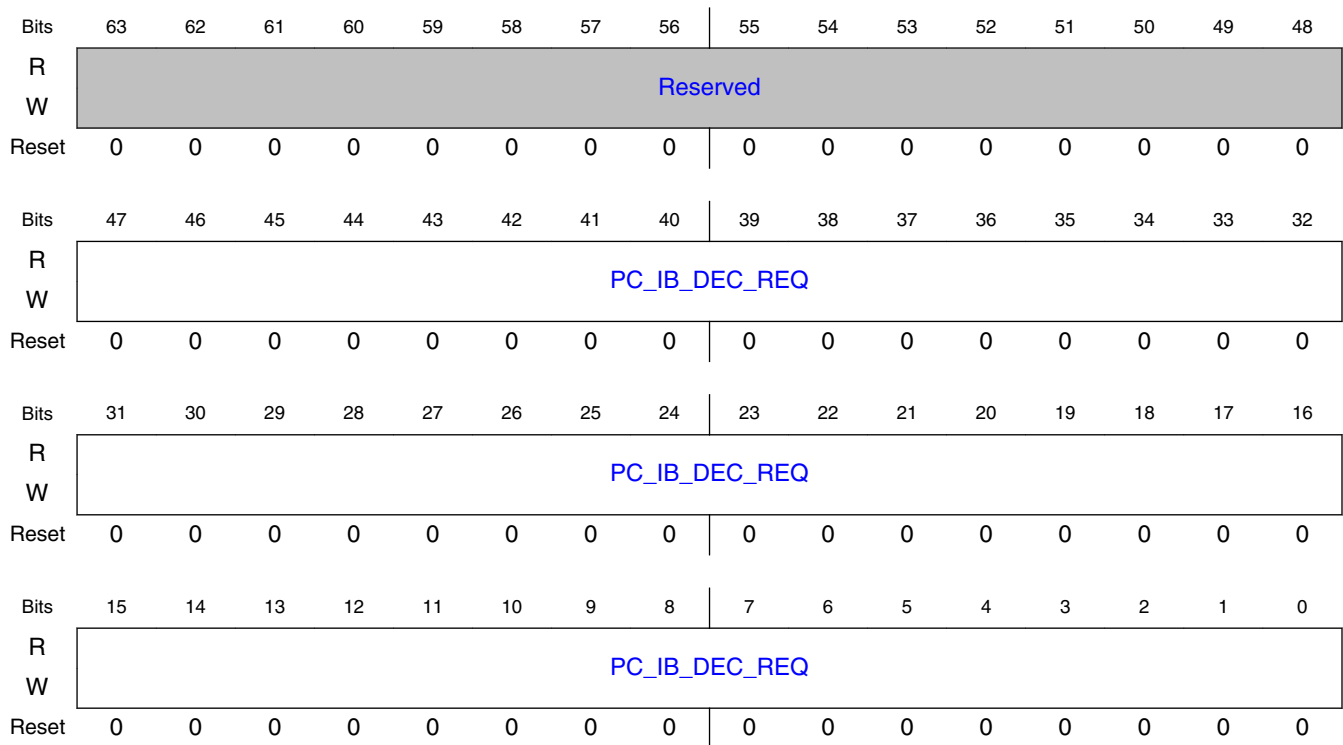
Register	Offset	Description
PC_IB_DEC_REQ	F10h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_IB_DEC_REQ	1_0F10h (alias)	-
PC_IB_DEC_REQ	2_0F10h (alias)	-
PC_IB_DEC_REQ	3_0F10h (alias)	-
PC_IB_DEC_REQ	4_0F10h (alias)	-
PC_IB_DEC_REQ	5_0F10h (alias)	-
PC_IB_DEC_REQ	6_0F10h (alias)	-
PC_IB_DEC_REQ	7_0F10h (alias)	-
PC_IB_DEC_REQ	8_0F10h (alias)	-
PC_IB_DEC_REQ	9_0F10h (alias)	-
PC_IB_DEC_REQ	A_0F10h (alias)	-
PC_IB_DEC_REQ	B_0F10h (alias)	-
PC_IB_DEC_REQ	C_0F10h (alias)	-
PC_IB_DEC_REQ	D_0F10h (alias)	-

14.111.2 Function

The PC_IB_DEC_REQ register counts the total number of requests for symmetric decryptions (excluding blob decapsulations and decryptions of Black Keys). If a descriptor specifies a decryption operation, the register is incremented at the time that the decryption operation completes. Note that a single descriptor containing multiple decryption commands could cause this register to increment more than once. The LSB of the Class 1 Mode register determines if this register or PC_OB_ENC_REQ is incremented.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.111.3 Diagram



14.111.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_IB_DEC_R EQ	Performance Counter Inbound Decryptions Requested

14.112 Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)

14.112.1 Offset

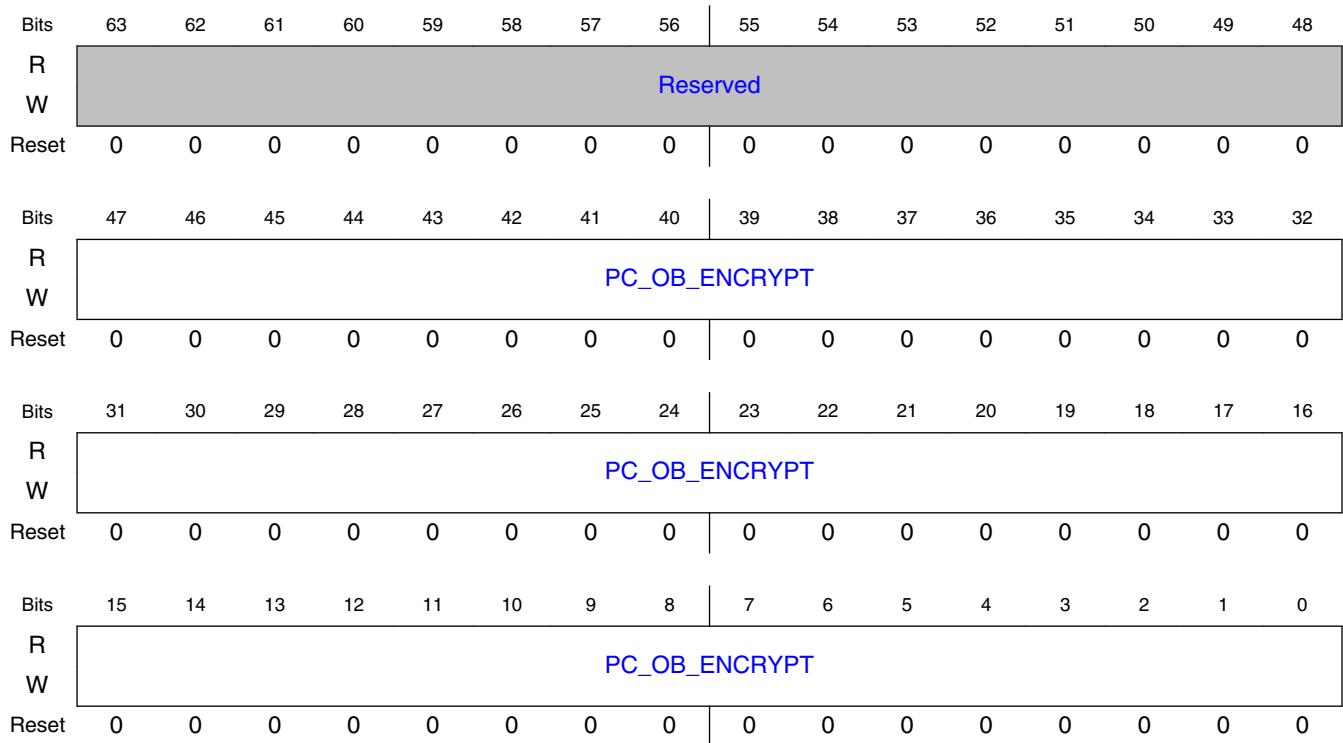
Register	Offset	Description
PC_OB_ENCRYPT	F18h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_OB_ENCRYPT	1_0F18h (alias)	-
PC_OB_ENCRYPT	2_0F18h (alias)	-
PC_OB_ENCRYPT	3_0F18h (alias)	-
PC_OB_ENCRYPT	4_0F18h (alias)	-
PC_OB_ENCRYPT	5_0F18h (alias)	-
PC_OB_ENCRYPT	6_0F18h (alias)	-
PC_OB_ENCRYPT	7_0F18h (alias)	-
PC_OB_ENCRYPT	8_0F18h (alias)	-
PC_OB_ENCRYPT	9_0F18h (alias)	-
PC_OB_ENCRYPT	A_0F18h (alias)	-
PC_OB_ENCRYPT	B_0F18h (alias)	-
PC_OB_ENCRYPT	C_0F18h (alias)	-
PC_OB_ENCRYPT	D_0F18h (alias)	-

14.112.2 Function

The PC_OB_ENCRYPT register counts the total number of bytes encrypted with a symmetric key algorithm (excluding blob encapsulations and encryptions of Black Keys). PC_OB_ENCRYPT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1, with the following exceptions. If the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9, the PC_OB_ENCRYPT register is not incremented but the PC_OB_PROTECT register is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1. PC_OB_PROTECT is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.112.3 Diagram



14.112.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_OB_ENCRYPT	Performance Counter Outbound Bytes Encrypted.

14.113 Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)

14.113.1 Offset

Register	Offset	Description
PC_OB_PROTECT	F20h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_OB_PROTECT	1_0F20h (alias)	-
PC_OB_PROTECT	2_0F20h (alias)	-
PC_OB_PROTECT	3_0F20h (alias)	-
PC_OB_PROTECT	4_0F20h (alias)	-
PC_OB_PROTECT	5_0F20h (alias)	-
PC_OB_PROTECT	6_0F20h (alias)	-
PC_OB_PROTECT	7_0F20h (alias)	-
PC_OB_PROTECT	8_0F20h (alias)	-
PC_OB_PROTECT	9_0F20h (alias)	-
PC_OB_PROTECT	A_0F20h (alias)	-
PC_OB_PROTECT	B_0F20h (alias)	-
PC_OB_PROTECT	C_0F20h (alias)	-
PC_OB_PROTECT	D_0F20h (alias)	-

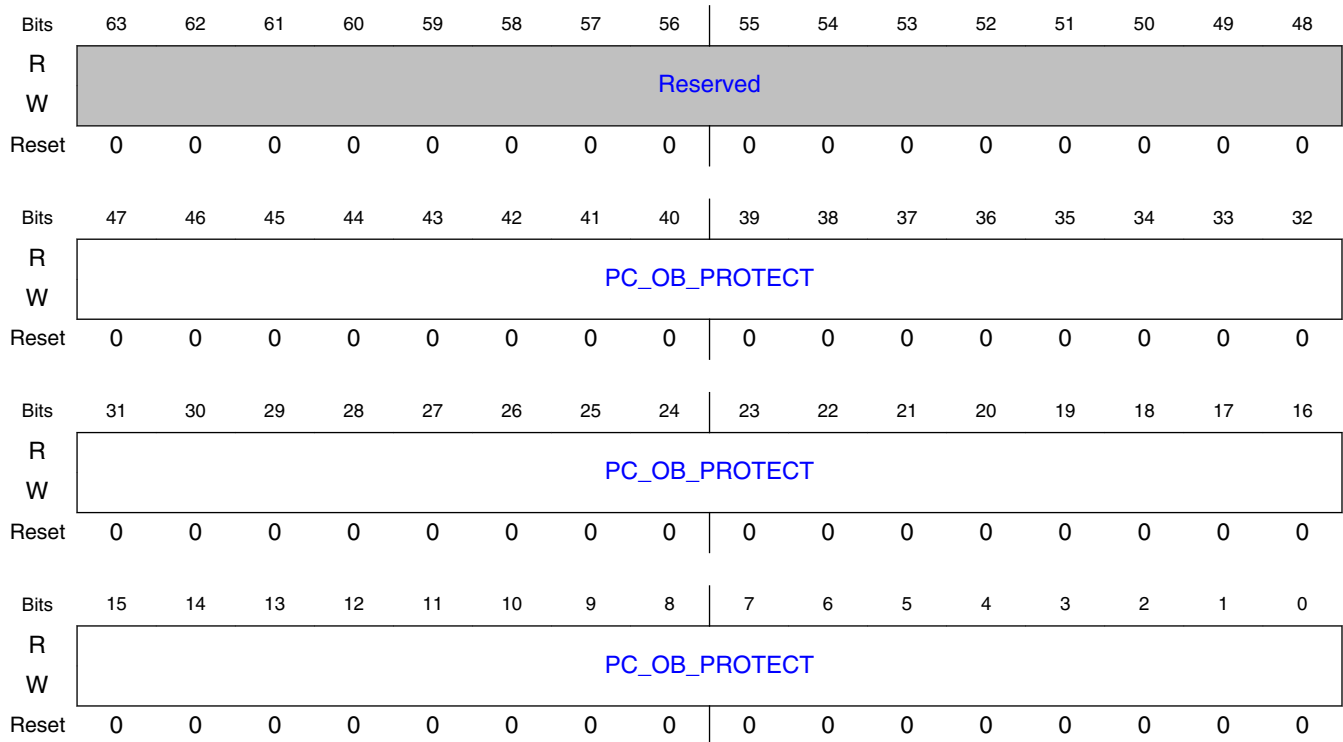
14.113.2 Function

The PC_OB_PROTECT register counts the total number of bytes protected—that is, the number of outbound bytes over which an integrity check value (ICV) was computed (for example, in HMAC and CMAC). (Note that this excludes blob encapsulations and CCM encryptions of Black Keys.) PC_OB_PROTECT is incremented by the value written to the Class 2 Data Size register if the AP bit in the Class 2 Mode register is set to 1. PC_OB_PROTECT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1 and the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9. PC_OB_PROTECT is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1. For AES-GCM, AES-CCM, AES-CBC-XCBC, AES-CTR-XCBC, AES-CBC-CMAC and AES-CTR-CMAC operations both the PC_OB_PROTECT register and the PC_OB_ENCRYPT register will be incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1.

Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.113.3 Diagram



14.113.4 Fields

Field	Function
63-48	Reserved
—	
47-0 PC_OB_PROTECT	Performance Counter Outbound Bytes Encrypted.

14.114 Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)

14.114.1 Offset

Register	Offset	Description
PC_IB_DECRYPT	F28h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_IB_DECRYPT	1_0F28h (alias)	-
PC_IB_DECRYPT	2_0F28h (alias)	-
PC_IB_DECRYPT	3_0F28h (alias)	-
PC_IB_DECRYPT	4_0F28h (alias)	-
PC_IB_DECRYPT	5_0F28h (alias)	-
PC_IB_DECRYPT	6_0F28h (alias)	-
PC_IB_DECRYPT	7_0F28h (alias)	-
PC_IB_DECRYPT	8_0F28h (alias)	-
PC_IB_DECRYPT	9_0F28h (alias)	-
PC_IB_DECRYPT	A_0F28h (alias)	-
PC_IB_DECRYPT	B_0F28h (alias)	-
PC_IB_DECRYPT	C_0F28h (alias)	-
PC_IB_DECRYPT	D_0F28h (alias)	-

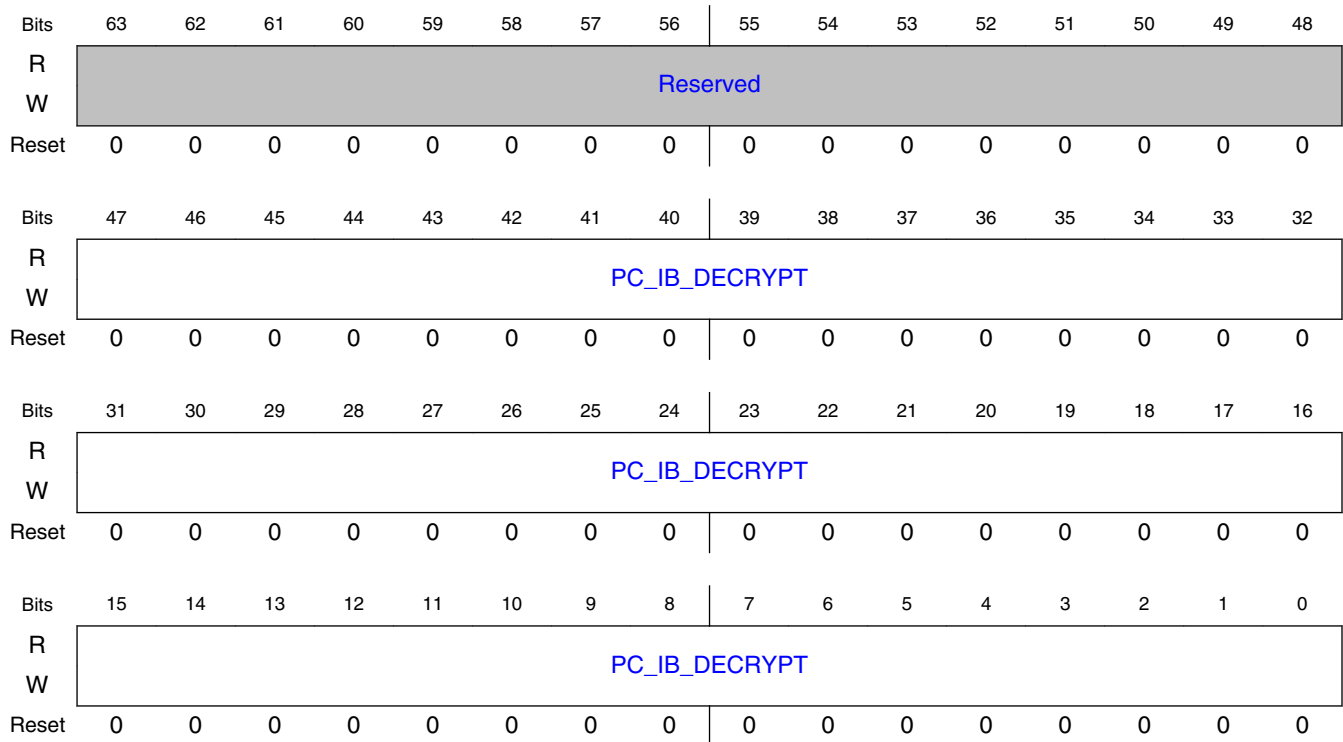
14.114.2 Function

The PC_IB_DECRYPT register counts the total number of bytes decrypted with a symmetric key algorithm (excluding blob decapsulations and decryptions of Black Keys). PC_IB_DECRYPT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0, with the following exceptions. If the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9, the PC_IB_DECRYPT register is not incremented but the PC_IB_VALIDATED register is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0. Note that writing to the “SAD Data Size” alias of the Class 1 Data Size register causes the PC_IB_VALIDATED register to be incremented rather than the PC_IB_DECRYPT register.

Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

14.114.3 Diagram



14.114.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_IB_DECRYPT	Performance Counter Inbound Bytes Decrypted.

14.115 Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)

14.115.1 Offset

Register	Offset	Description
PC_IB_VALIDATED	F30h (alias)	These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h.
PC_IB_VALIDATED	1_0F30h (alias)	-
PC_IB_VALIDATED	2_0F30h (alias)	-
PC_IB_VALIDATED	3_0F30h (alias)	-
PC_IB_VALIDATED	4_0F30h (alias)	-
PC_IB_VALIDATED	5_0F30h (alias)	-
PC_IB_VALIDATED	6_0F30h (alias)	-
PC_IB_VALIDATED	7_0F30h (alias)	-
PC_IB_VALIDATED	8_0F30h (alias)	-
PC_IB_VALIDATED	9_0F30h (alias)	-
PC_IB_VALIDATED	A_0F30h (alias)	-
PC_IB_VALIDATED	B_0F30h (alias)	-
PC_IB_VALIDATED	C_0F30h (alias)	-
PC_IB_VALIDATED	D_0F30h (alias)	-

14.115.2 Function

The PC_IB_VALIDATED register counts the total number of bytes validated (that is, the number of inbound bytes over which an ICV was computed for comparison purposes). PC_IB_VALIDATED is incremented by the value written to the Class 2 Data Size register if the AP bit in the Class 2 Mode register is set to 0. PC_IB_VALIDATED is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0 and the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9. PC_IB_VALIDATED is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register the ENC bit in the Class 1 Mode register is set to 0. For AES-GCM, AES-CCM, AES-CBC-XCBC, AES-CTR-XCBC, AES-CBC-CMAC and AES-CTR-CMAC operations both the PC_IB_VALIDATED register and the PC_IB_DECRYPT register will be incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0.

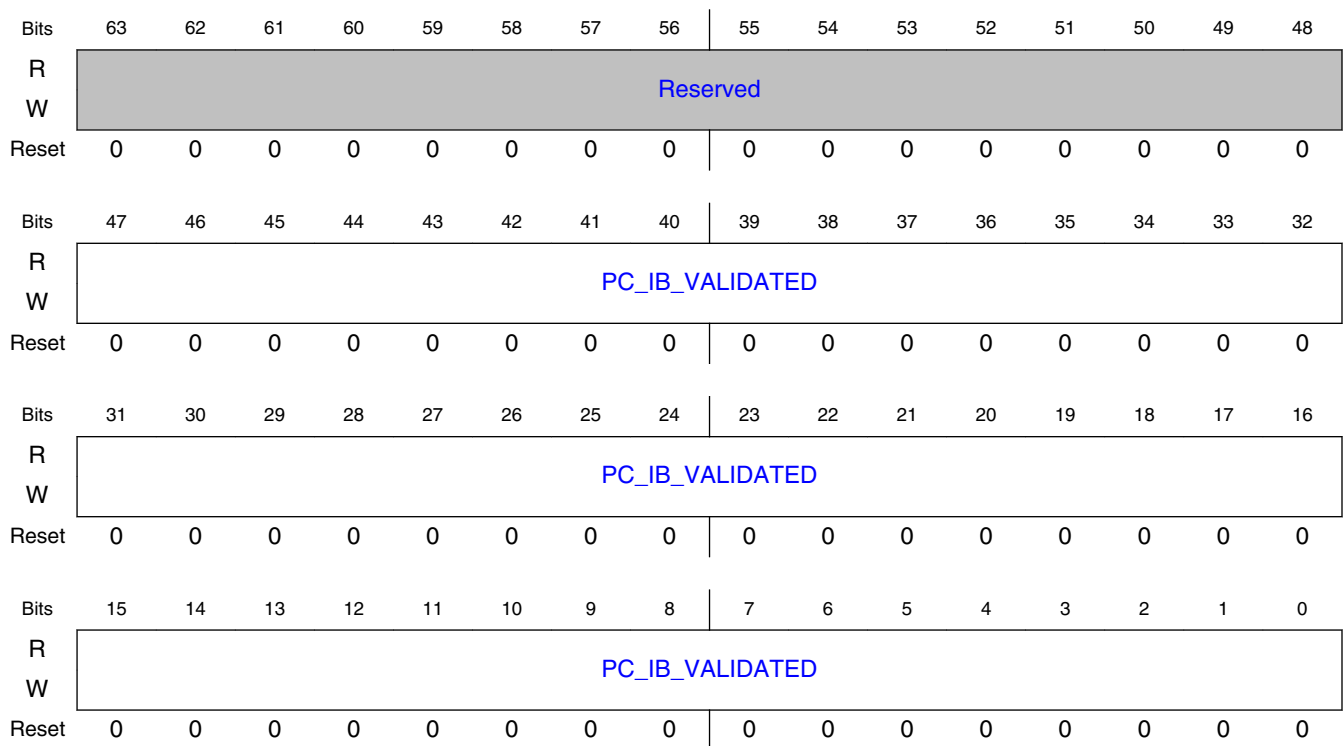
Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

NOTE

This counter does not include the number of bytes of the received ICV. Also, it increments whether the ICV comparison was successful or not.

14.115.3 Diagram



14.115.4 Fields

Field	Function
63-48 —	Reserved
47-0 PC_IB_VALIDATED	Performance Counter Inbound Bytes Validated.

14.116 CHA Revision Number Register, most-significant half (CRNR_MS)

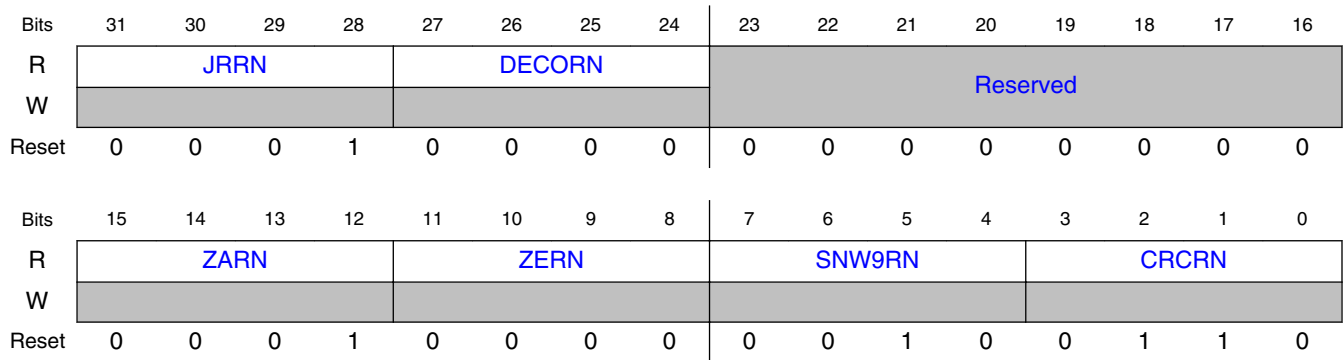
14.116.1 Offset

Register	Offset
CRNR_MS	FA0h (alias)
CRNR_MS	1_0FA0h (alias)
CRNR_MS	2_0FA0h (alias)
CRNR_MS	3_0FA0h (alias)
CRNR_MS	4_0FA0h (alias)
CRNR_MS	5_0FA0h (alias)
CRNR_MS	6_0FA0h (alias)
CRNR_MS	7_0FA0h (alias)
CRNR_MS	8_0FA0h (alias)
CRNR_MS	9_0FA0h (alias)
CRNR_MS	A_0FA0h (alias)
CRNR_MS	B_0FA0h (alias)
CRNR_MS	C_0FA0h (alias)
CRNR_MS	D_0FA0h (alias)

14.116.2 Function

The CHA Revision Number register indicates the revision number of each CHA. The revisions are numbered independently for each version of a particular CHA (see [CHA Version ID Register, most-significant half \(CHAVID_MS\)](#)). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

14.116.3 Diagram



14.116.4 Fields

Field	Function
31-28 JRRN	Job Ring Revision Number
27-24 DECORN	DECO Revision Number
23-16 —	Reserved
15-12 ZARN	ZUC Authentication Hardware Accelerator Revision Number
11-8 ZERN	ZUC Encryption Hardware Accelerator Revision Number
7-4 SNW9RN	SNOW-f9 Hardware Accelerator Revision Number
3-0 CRCRN	CRC Hardware Accelerator Revision Number

14.117 CHA Revision Number Register, least-significant half (CRNR_LS)

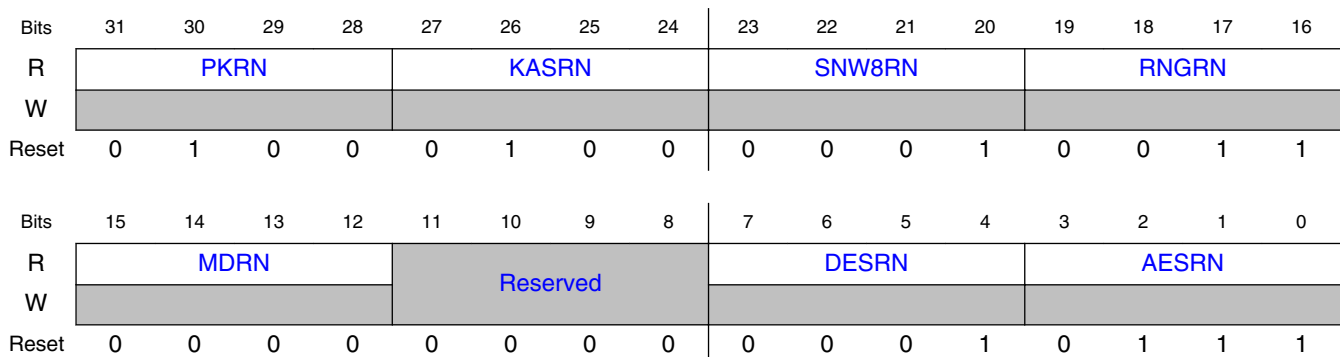
14.117.1 Offset

Register	Offset
CRNR_LS	FA4h (alias)
CRNR_LS	1_0FA4h (alias)
CRNR_LS	2_0FA4h (alias)
CRNR_LS	3_0FA4h (alias)
CRNR_LS	4_0FA4h (alias)
CRNR_LS	5_0FA4h (alias)
CRNR_LS	6_0FA4h (alias)
CRNR_LS	7_0FA4h (alias)
CRNR_LS	8_0FA4h (alias)
CRNR_LS	9_0FA4h (alias)
CRNR_LS	A_0FA4h (alias)
CRNR_LS	B_0FA4h (alias)
CRNR_LS	C_0FA4h (alias)
CRNR_LS	D_0FA4h (alias)

14.117.2 Function

The CHA Revision Number register indicates the revision number of each CHA. The revisions are numbered independently for each version of a particular CHA (see [CHA Version ID Register, most-significant half \(CHAVID_MS\)](#)). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

14.117.3 Diagram



14.117.4 Fields

Field	Function
31-28 PKRN	Public Key Hardware Accelerator Revision Number For PKHA-XT, PKRN=1. For PKHA-SD, see below. 0000b - PKHA-SDv1 0001b - PKHA-SDv2 0010b - PKHA-SDv3 0011b - PKHA-SDv4
27-24 KASRN	Kasumi f8/f9 Hardware Accelerator Revision Number
23-20 SNW8RN	SNOW-f8 Hardware Accelerator Revision Number
19-16 RNGRN	Random Number Generator Revision Number.
15-12 MDRN	Message Digest Hardware Accelerator module Revision Number.
11-8 —	Reserved
7-4 DESRN	DES Accelerator Revision Number.
3-0 AESRN	AES Accelerator Revision Number. 0000 No Differential Power Analysis resistance implemented 0001 Differential Power Analysis resistance implemented For all other values when AESVID = 4, Differential Power Analysis resistance is implemented.

14.118 Compile Time Parameters Register, most-significant half (CTPR_MS)

14.118.1 Offset

Register	Offset
CTPR_MS	FA8h (alias)
CTPR_MS	1_0FA8h (alias)
CTPR_MS	2_0FA8h (alias)

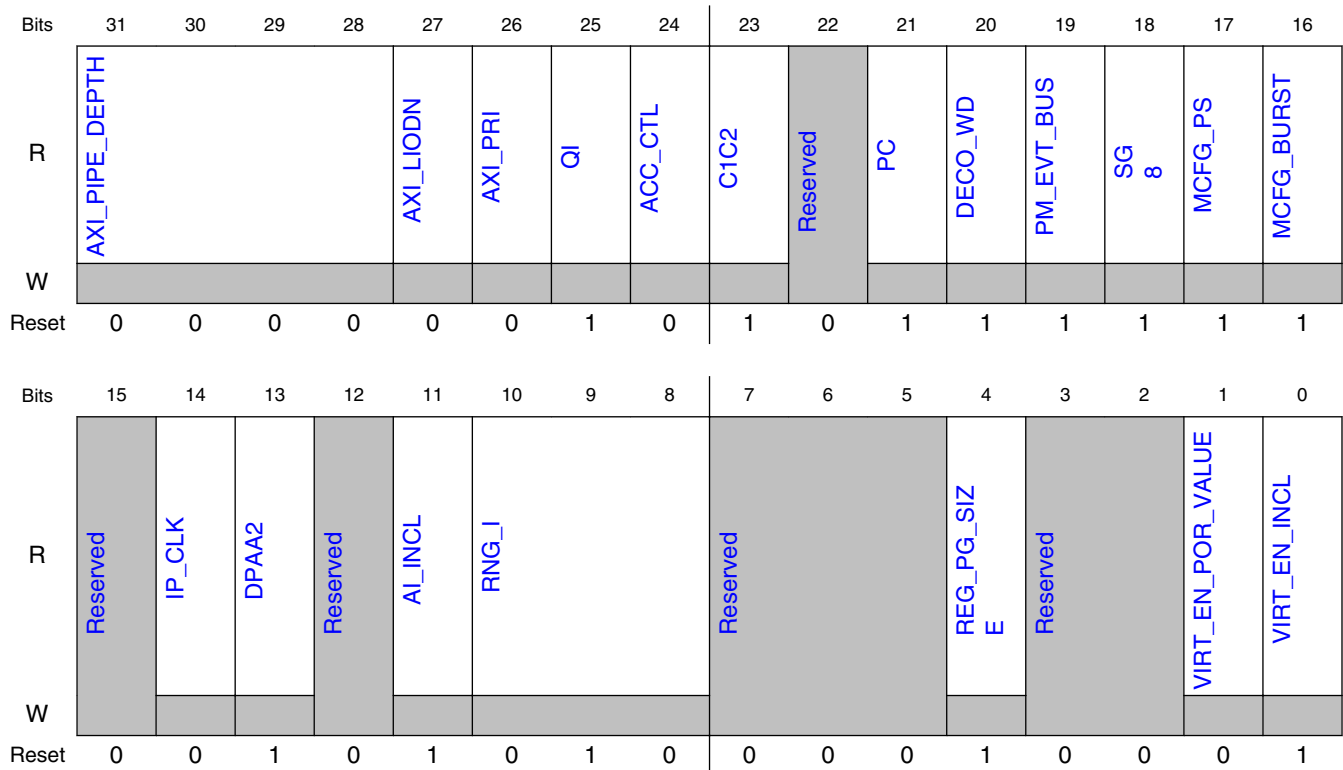
Table continues on the next page...

Register	Offset
CTPR_MS	3_0FA8h (alias)
CTPR_MS	4_0FA8h (alias)
CTPR_MS	5_0FA8h (alias)
CTPR_MS	6_0FA8h (alias)
CTPR_MS	7_0FA8h (alias)
CTPR_MS	8_0FA8h (alias)
CTPR_MS	9_0FA8h (alias)
CTPR_MS	A_0FA8h (alias)
CTPR_MS	B_0FA8h (alias)
CTPR_MS	C_0FA8h (alias)
CTPR_MS	D_0FA8h (alias)

14.118.2 Function

The Compile Time Parameters register indicates the parameter settings at the time SEC was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.118.3 Diagram



14.118.4 Fields

Field	Function
31-28 AXI_PIPE_DEPTH	AXI Pipeline Depth. 0000: SEC DMA implements maximum AXI bus pipeline depth. 0001 - 1111: SEC DMA implements an AXI bus pipeline depth as specified by AXI_PIPE_DEPTH.
27 AXI_LIODN	LIODN logic included. 0 : This version of SEC does not implement LIODN logic. 1 : This version of SEC implements logic to select LIODNs.
26 AXI_PRI	AXI Master Priority implemented. 0 : This version of SEC does not implement AXI Master Priority. 1 : This version of SEC implements logic for the AXI Master Priority signals.
25 QI	Queue Manager interface (QI) implemented. 0 : This version of SEC does not implement a QI. 1 : This version of SEC implements a QI.
24 ACC_CTL	ICID-based access control 0: SEC does not implement ICID-based access control for the IP Bus registers.

Table continues on the next page...

Field	Function
	1: SEC implements ICID-based or ICID-based access control for the IP Bus registers.
23 C1C2	Separate C1 and C2 registers 0: In this implementation of SEC the Class 2 Key and Context registers are shared with the Class 1 Key and Context registers. 1: SEC implements Class 2 Key and Context registers that are separate from the Class 1 Key and Context registers.
22 —	Reserved
21 PC	Performance Counter registers implemented 0: SEC does not implement Performance Counter registers. 1: SEC does implement Performance Counter registers.
20 DECO_WD	DECO Watchdog Counter implemented 0: SEC does not implement a DECO Watchdog Counter. 1: SEC does implement a DECO Watchdog Counter.
19 PM_EVT_BUS	Performance Monitor Event Bus implemented 0: SEC does not implement a Performance Monitor Event Bus. 1: SEC does implement a Performance Monitor Event Bus.
18 SG8	Eight Scatter-Gather Tables implemented 0: SEC implements one Scatter-Gather Table register. 1: SEC implements eight Scatter-Gather Table registers.
17 MCFG_PS	Pointer Size field implemented 0: The Master Configuration Register does not contain a Pointer Size field. 1: The Master Configuration Register does contain a Pointer Size field.
16 MCFG_BURST	If indicated, AXI burst support is implemented with a default normal burst size exceeding 32 bytes and/or with the ability to enable extended bursting by setting the LARGE_BURST bit in the Master Configuration Register. 0: Normal bursts are aligned 32-byte transfers and extended bursting cannot be enabled. 1: Normal bursts are aligned 64-byte transfers or extended bursting exceeding the normal burst size can be enabled with the LARGE_BURST enable bit in the Master Configuration register.
15 —	Reserved
14 IP_CLK	IP Bus Slave Clock. 0: The frequency of SEC's IP Bus Slave Clock is the same as the frequency of SEC's AXI bus clock. 1: The frequency of SEC's IP Bus Slave Clock is one-half of the frequency of SEC's AXI bus clock.
13 DPAA2	DPAA2 architecture supported. 0: This version of SEC does not support version 2 of the Data Path Acceleration Architecture (DPAA2). 1: This version of SEC does support version 2 of the Data Path Acceleration Architecture (DPAA2).
12 —	Reserved
11 AI_INCL	AIOP interface implemented. 0: This version of SEC does not implement an AIOP interface.

Table continues on the next page...

Compile Time Parameters Register, least-significant half (CTPR_LS)

Field	Function
	1 : This version of SEC implements one or more AIOP interfaces.
10-8 RNG_I	RNG Instantiations. RNG_I indicates the number of RNG instantiations that are implemented in the RNG hardware. Note that each instantiation is the data context for an independent RNG stream. The number of hardware RNGs is indicated in the RNGNUM field of the CHANUM register.
7-5 —	Reserved.
4 REG_PG_SIZE	SEC register page size. 0: SEC uses 4Kbyte register pages. 1: SEC uses 64Kbyte register pages.
3-2 —	Reserved
1 VIRT_EN_POR _VALUE	Job Ring Virtualization POR state. 0: Job Ring virtualization is not enabled at power up. 1: Job Ring virtualization is enabled at power up.
0 VIRT_EN_INCL	Job Ring Virtualization programmable. 0: Job Ring virtualization is always enabled and the Security Configuration register does not contain a VIRT_EN bit. 1: Job Ring virtualization can be programmed to be enabled or disabled by writing to the VIRT_EN bit in the Security Configuration register.

14.119 Compile Time Parameters Register, least-significant half (CTPR_LS)

14.119.1 Offset

Register	Offset
CTPR_LS	FACh (alias)
CTPR_LS	1_0FACh (alias)
CTPR_LS	2_0FACh (alias)
CTPR_LS	3_0FACh (alias)
CTPR_LS	4_0FACh (alias)
CTPR_LS	5_0FACh (alias)
CTPR_LS	6_0FACh (alias)
CTPR_LS	7_0FACh (alias)
CTPR_LS	8_0FACh (alias)
CTPR_LS	9_0FACh (alias)
CTPR_LS	A_0FACh (alias)

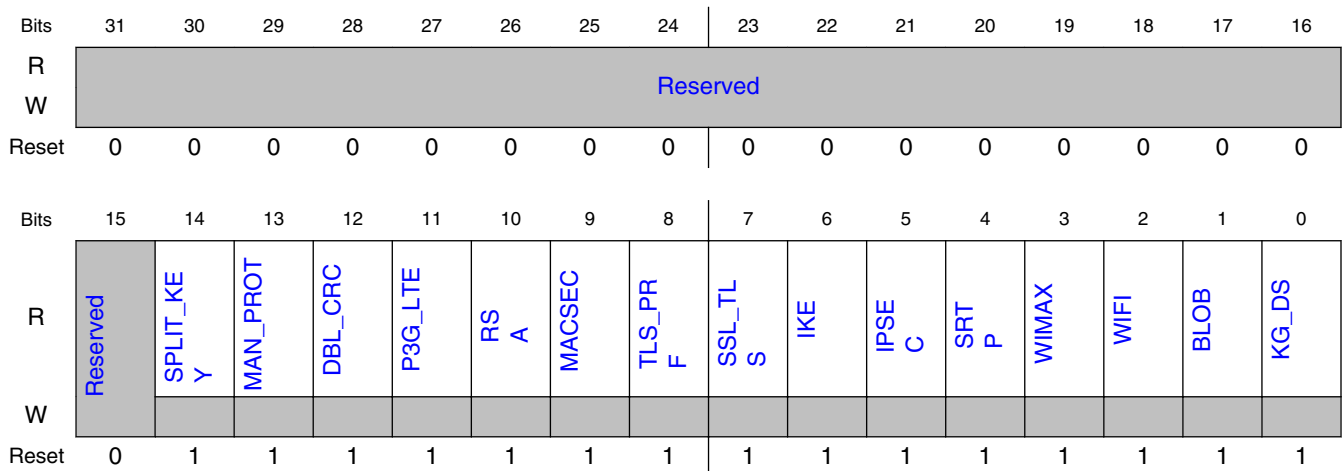
Table continues on the next page...

Register	Offset
CTPR_LS	B_0FACh (alias)
CTPR_LS	C_0FACh (alias)
CTPR_LS	D_0FACh (alias)

14.119.2 Function

The Compile Time Parameters register indicates the parameter settings at the time SEC was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.119.3 Diagram



14.119.4 Fields

Field	Function
31-15 —	Reserved.
14 SPLIT_KEY	Split key protocol 1b - SEC implements the split-key protocol.
13 MAN_PROT	Manufacturing Protection protocol 1b - SEC implements the two Manufacturing Protection functions.

Table continues on the next page...

Fault Address Register (FAR)

Field	Function
12 DBL_CRC	DOuble CRC protocol 1b - SEC implements specialized support for 3G Double CRC.
11 P3G_LTE	3GPP/LTE protocol 1b - SEC implements specialized support for 3G and LTE protocols.
10 RSA	RSA protocol 1b - SEC implements specialized support for RSA encrypt and decrypt operations.
9 MACSEC	MACSEC protocol 1b - SEC implements specialized support for the MACSEC protocol.
8 TLS_PRF	TLS PRF protocol 1b - SEC implements specialized support for the TLS protocol pseudo-random function."
7 SSL_TLS	SSL/TLS protocol 1b - SEC implements specialized support for the SSL and TLS protocols.
6 IKE	IKE protocols 1b - SEC implements specialized support for the IKE protocol.
5 IPSEC	IPSEC protocols 1b - SEC implements specialized support for the IPSEC protocol.
4 SRTP	SRTP protocol 1b - SEC implements specialized support for the SRTP protocol.
3 WIMAX	WiMax protocol 1b - SEC implements specialized support for the WIMAX protocol.
2 WIFI	WiFi protocol 1b - SEC implements specialized support for the WIFI protocol.
1 BLOB	Blob protocol 1b - SEC implements specialized support for encapsulating and decapsulating cryptographic blobs.
0 KG_DS	PK generation and digital signature protcols 1b - SEC implements specialized support for Public Key Generation and Digital Signatures.

14.120 Fault Address Register (FAR)

14.120.1 Offset

Register	Offset	Description
FAR	FC0h (alias)	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

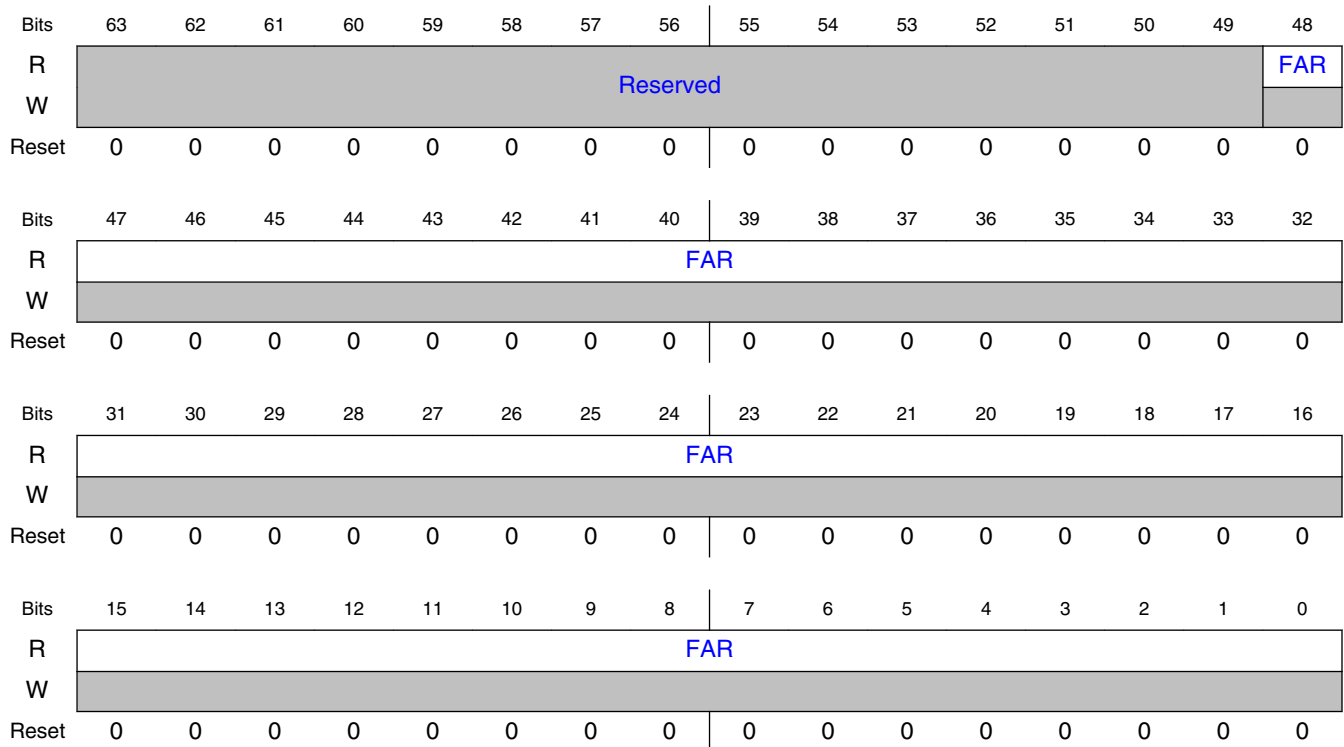
Table continues on the next page...

Register	Offset	Description
FAR	1_0FC0h (alias)	-
FAR	2_0FC0h (alias)	-
FAR	3_0FC0h (alias)	-
FAR	4_0FC0h (alias)	-
FAR	5_0FC0h (alias)	-
FAR	6_0FC0h (alias)	-
FAR	7_0FC0h (alias)	-
FAR	8_0FC0h (alias)	-
FAR	9_0FC0h (alias)	-
FAR	A_0FC0h (alias)	-
FAR	B_0FC0h (alias)	-
FAR	C_0FC0h (alias)	-
FAR	D_0FC0h (alias)	-

14.120.2 Function

The Fault Address Register is used for software debugging of external memory access errors. This register will hold the value of the AXI address where a read or write error occurred. The read error address is aligned to the data bus address boundary of the data sample where the error occurred. The write error address is the starting address of the transaction, aligned to the data bus address boundary. Additional details concerning the bus transaction appear in the FADR (see [Fault Address Detail Register \(FADR\)](#)). The associated ICID is in the Fault Address ICID Register (see Section [Fault Address ICID Register \(FAICID\)](#)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault Address ICID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

14.120.3 Diagram



14.120.4 Fields

Field	Function
63-49 —	Reserved
48-0 FAR	Fault Address. This is the AXI address at which the error occurred. If multiple errors occur, this is the AXI address at which the first error occurred. This address will remain in the register until software has read both the upper and lower halves of the register.

14.121 Fault Address ICID Register (FAICID)

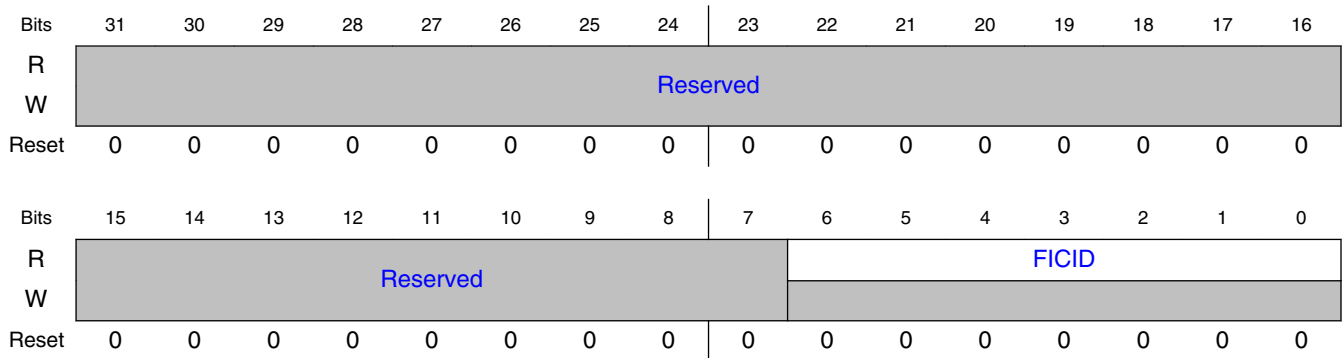
14.121.1 Offset

Register	Offset
FAICID	FC8h (alias)
FAICID	1_0FC8h (alias)
FAICID	2_0FC8h (alias)
FAICID	3_0FC8h (alias)
FAICID	4_0FC8h (alias)
FAICID	5_0FC8h (alias)
FAICID	6_0FC8h (alias)
FAICID	7_0FC8h (alias)
FAICID	8_0FC8h (alias)
FAICID	9_0FC8h (alias)
FAICID	A_0FC8h (alias)
FAICID	B_0FC8h (alias)
FAICID	C_0FC8h (alias)
FAICID	D_0FC8h (alias)

14.121.2 Function

The Fault Address ICID Register is used by software for debugging external memory access errors. This register indicates the ICID associated with the AXI transaction where the error occurred. The associated AXI address is in the Fault Address Register (see [Fault Address Register \(FAR\)](#)) and additional details appear in the Fault Address Detail Register (see [Fault Address Detail Register \(FADR\)](#)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault Address ICID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

14.121.3 Diagram



14.121.4 Fields

Field	Function
31-7 —	Reserved.
6-0 FICID	DMA transaction ICID. This was the ICID associated with the DMA transaction that failed.

14.122 Fault Address Detail Register (FADR)

14.122.1 Offset

Register	Offset
FADR	FCCh (alias)
FADR	1_0FCCh (alias)
FADR	2_0FCCh (alias)
FADR	3_0FCCh (alias)
FADR	4_0FCCh (alias)
FADR	5_0FCCh (alias)
FADR	6_0FCCh (alias)
FADR	7_0FCCh (alias)
FADR	8_0FCCh (alias)
FADR	9_0FCCh (alias)

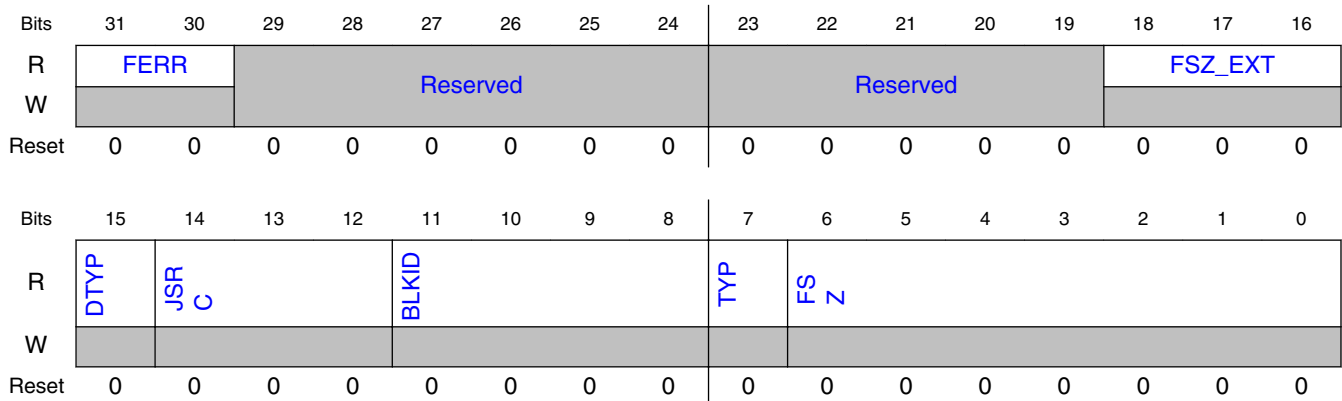
Table continues on the next page...

Register	Offset
FADR	A_0FCCh (alias)
FADR	B_0FCCh (alias)
FADR	C_0FCCh (alias)
FADR	D_0FCCh (alias)

14.122.2 Function

The Fault Address Detail Register is used by software for debugging external memory access errors. This register will hold details about the AXI transaction where the error occurred. The associated AXI address is in the [Fault Address Register \(FAR\)](#). The associated ICID is in the [Fault Address ICID Register \(FAICID\)](#). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault Address ICID Register, and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

14.122.3 Diagram



14.122.4 Fields

Field	Function
31-30 FERR	Fault Error Code. This is the AXI Error Response Code. 00b - OKAY - Normal Access

Table continues on the next page...

SEC Status Register (SSTA)

Field	Function
	01b - Reserved 10b - SLVERR - Slave Error 11b - DECERR - Decode Error
29-24 —	Reserved. Always 0.
23-19 —	Reserved. Always 0.
18-16 FSZ_EXT	AXI Transaction Transfer Size - extended. This field holds the most significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error.
15 DTYP	Data Type. The type of data being processed when the AXI transfer error occurred. 0b - message data 1b - control data
14-12 JSRC	Job Source. The source of the job whose AXI transfer ended with an error: 000b - Job Ring 0 001b - Job Ring 1 010b - Job Ring 2 011b - Job Ring 3 100b - RTIC 101b - Queue Interface 110b - reserved 111b - reserved
11-8 BLKID	Block ID. The Block ID is the identifier of the block internal to SEC that initiated the DMA transfer that resulted in an error. BLKID is interpreted as follows: 0100b - job queue controller Burst Buffer 0101b - One of the Job Rings (see JSRC field) 0111b - Queue Interface 1000b - DECO0 1001b - DECO1 1010b - DECO2 1011b - DECO3 1100b - DECO4 1101b - DECO5
7 TYP	AXI Transaction Type. This is the type, read or write, of the DMA transaction that resulted in an error. 0b - Read. 1b - Write.
6-0 FSZ	AXI Transaction Transfer Size. This field holds the least-significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error. For large transfers the most-significant bits are held in field FSZ_EXT.

14.123 SEC Status Register (SSTA)

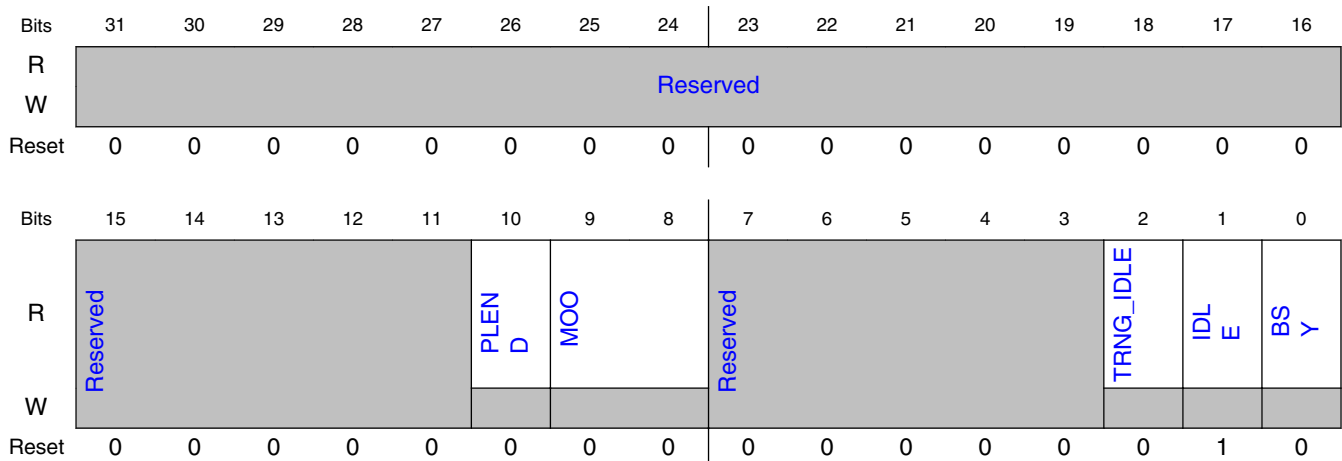
14.123.1 Offset

Register	Offset
SSTA	FD4h (alias)
SSTA	1_0FD4h (alias)
SSTA	2_0FD4h (alias)
SSTA	3_0FD4h (alias)
SSTA	4_0FD4h (alias)
SSTA	5_0FD4h (alias)
SSTA	6_0FD4h (alias)
SSTA	7_0FD4h (alias)
SSTA	8_0FD4h (alias)
SSTA	9_0FD4h (alias)
SSTA	A_0FD4h (alias)
SSTA	B_0FD4h (alias)
SSTA	C_0FD4h (alias)
SSTA	D_0FD4h (alias)

14.123.2 Function

The SEC Status Register indicates some status information that is relevant to the entire SEC block. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.123.3 Diagram



14.123.4 Fields

Field	Function
31-11 —	Reserved.
10 PLEND	Platform Endianness. This is a hardwired SOC-specific configuration. PLEND indicates whether the SEC bus master views memory by default as big endian or little endian. Software can override the default for particular data by setting bits in the Job Ring Configuration Register, the RTIC Endian Register, the Queue Interface Control Register and the DECO Job Queue Control Register. 0b - Platform default is Little Endian 1b - Platform default is Big Endian
9-8 MOO	Mode of Operation. These bits indicate the Security Mode that SEC is currently working in. The Security Mode is determined by the Security State Machine (see Security Monitor (SecMon) security states located in the Security Monitor. The modes are defined in SEC modes of operation . 00b - Non-Secure 01b - Secure 10b - Trusted 11b - Fail
7-3 —	Reserved
2 TRNG_IDLE	If TRNG_IDLE=1, the TRNG portion of the RNG is idle. The free-running oscillator is stopped, so no entropy is being generated.
1 IDLE	SEC is idle. IDLE=1 indicates that SEC is not currently processing any jobs from QI, from Job Rings, or from RTIC and there are no pending interrupts (or the interrupts are masked) and the output job-ring timers are not counting. There may still be results in the output rings that have not been removed. If the RIDLE field in the RTIC Control Register=1, IDLE will be 0 if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits is 1). If RIDLE=0 and SEC is otherwise idle, the IDLE will still occasionally be 0 while RTIC is actually hashing a chunk of memory. That is, if RTIC is in Run-Time Mode and one or more memory blocks is enabled, RTIC's Programmable DMA Throttle Timer may time out periodically and RTIC will launch a hashing job, which will cause IDLE to briefly go to 0. Software should always check the output rings and Output FQs for results prior to checking for IDLE.
0 BSY	SEC Busy. BSY=1 indicates that SEC is processing at least one Descriptor.

14.124 RTIC Version ID Register (RVID)

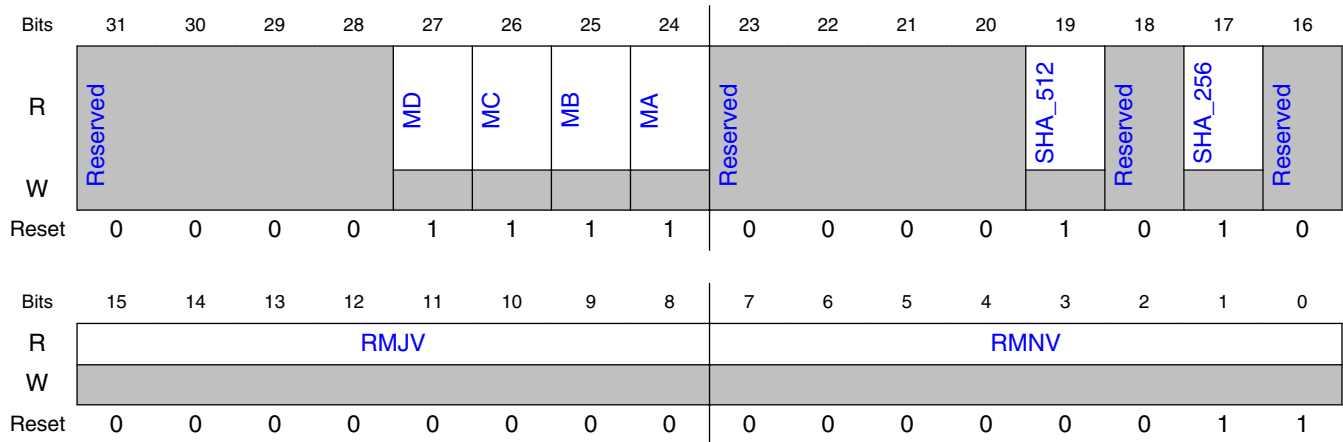
14.124.1 Offset

Register	Offset
RVID	FE0h (alias)
RVID	1_0FE0h (alias)
RVID	2_0FE0h (alias)
RVID	3_0FE0h (alias)
RVID	4_0FE0h (alias)
RVID	5_0FE0h (alias)
RVID	6_0FE0h (alias)
RVID	7_0FE0h (alias)
RVID	8_0FE0h (alias)
RVID	9_0FE0h (alias)
RVID	A_0FE0h (alias)
RVID	B_0FE0h (alias)
RVID	C_0FE0h (alias)
RVID	D_0FE0h (alias)

14.124.2 Function

The Run Time Integrity Checking Version ID register can be used by software to differentiate between different versions of the RTIC. Field RMJV is used for major revisions, field RMNV is used for minor revisions and the remaining fields are used for other revision information about the hardware. The bit assignments of this register appear below. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.124.3 Diagram



14.124.4 Fields

Field	Function
31-28 —	Reserved
27 MD	Memory Block D Available. This bit indicates that Memory Block D is available for Hash Once and Run Time Checking.
26 MC	Memory Block C Available. This bit indicates that Memory Block C is available for Hash Once and Run Time Checking.
25 MB	Memory Block B Available. This bit indicates that Memory Block B is available for Hash Once and Run Time Checking.
24 MA	Memory Block A Available. This bit indicates that Memory Block A is available for Hash Once and Run Time Checking.
23-20 —	Reserved
19 SHA_512	SHA-512. 0b - RTIC cannot use the SHA-512 hashing algorithm. 1b - RTIC can use the SHA-512 hashing algorithm.
18 —	Reserved
17 SHA_256	SHA-256. 0b - RTIC cannot use the SHA-256 hashing algorithm. 1b - RTIC can use the SHA-256 hashing algorithm.
16 —	Reserved

Table continues on the next page...

Field	Function
15-8 RMJV	RTIC Major Version. Represents major revision number of RTIC. This value is incremented when major functional changes are introduced or the programming model has changed.
7-0 RMNV	RTIC Minor Version. Represents minor revision number of RTIC. This value is incremented when minor functional changes are made that do not change the programming model. Corrections that require changes to the design are the typical reason for incrementing these bits.

14.125 CHA Cluster Block Version ID Register (CCBVID)

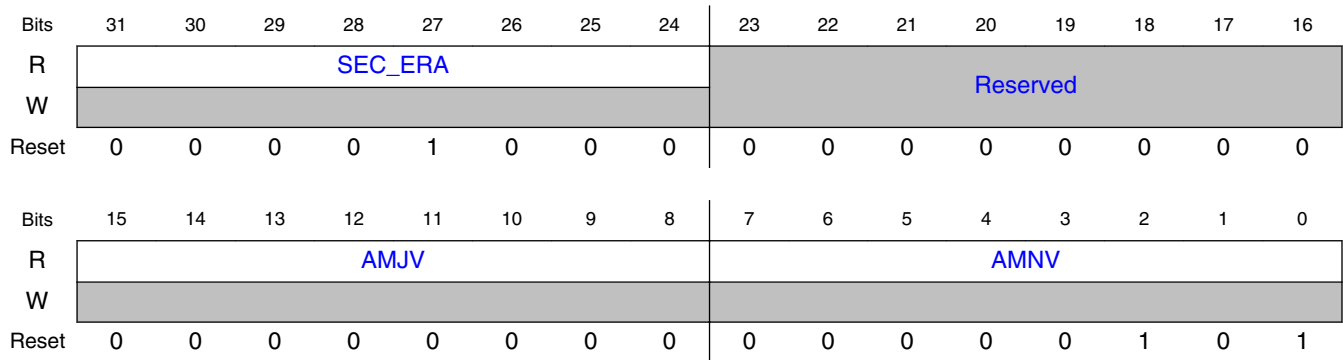
14.125.1 Offset

Register	Offset
CCBVID	FE4h (alias)
CCBVID	1_0FE4h (alias)
CCBVID	2_0FE4h (alias)
CCBVID	3_0FE4h (alias)
CCBVID	4_0FE4h (alias)
CCBVID	5_0FE4h (alias)
CCBVID	6_0FE4h (alias)
CCBVID	7_0FE4h (alias)
CCBVID	8_0FE4h (alias)
CCBVID	9_0FE4h (alias)
CCBVID	A_0FE4h (alias)
CCBVID	B_0FE4h (alias)
CCBVID	C_0FE4h (alias)
CCBVID	D_0FE4h (alias)

14.125.2 Function

The CHA Cluster Block Version ID register can be used by software to differentiate between different versions of the CCB. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.125.3 Diagram



14.125.4 Fields

Field	Function
31-24 SEC_ERA	SEC Era. 00000000b - This version of SEC is based on Era 5 or earlier RTL. 00000110b - This version of SEC is based on Era 6 RTL. 00000111b - This version of SEC is based on Era 7 RTL. 00001000b - This version of SEC is based on Era 8 RTL.
23-16 —	Reserved
15-8 AMJV	Accelerator Major Revision Number. This value will be incremented every time there is a major architectural change to the CCB design. Incrementing this results in the AMNV being set back to 0.
7-0 AMNV	Accelerator Minor Revision Number. This value will be incremented every time an RTL change has been made to the CCB module.

14.126 CHA Version ID Register, most-significant half (CHAV ID_MS)

14.126.1 Offset

Register	Offset
CHAVID_MS	FE8h (alias)
CHAVID_MS	1_0FE8h (alias)

Table continues on the next page...

Register	Offset
CHAVID_MS	2_0FE8h (alias)
CHAVID_MS	3_0FE8h (alias)
CHAVID_MS	4_0FE8h (alias)
CHAVID_MS	6_0FE8h (alias)
CHAVID_MS	7_0FE8h (alias)
CHAVID_MS	8_0FE8h (alias)
CHAVID_MS	9_0FE8h (alias)
CHAVID_MS	A_0FE8h (alias)
CHAVID_MS	B_0FE8h (alias)
CHAVID_MS	C_0FE8h (alias)
CHAVID_MS	D_0FE8h (alias)

14.126.2 Function

The CHA Version ID register can be used, along with the CCB Version ID, by software to differentiate between different versions of the cryptographic hardware accelerators. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.126.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JRVID				DECOVID				Reserved							
W																
Reset	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ZAVID				ZEVID				SNW9VID				CRCVID			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

14.126.4 Fields

Field	Function
31-28	Job Ring Version ID

Table continues on the next page...

CHA Version ID Register, least-significant half (CHAVID_LS)

Field	Function
JRVID	
27-24 DECOVID	DECO Version ID
23-16 —	Reserved
15-12 ZAVID	ZUC Authentication Hardware Accelerator Version ID
11-8 ZEVID	ZUC Encryption Hardware Accelerator Version ID
7-4 SNW9VID	SNOW-f9 Hardware Accelerator Version ID
3-0 CRCVID	CRC Hardware Accelerator Version ID

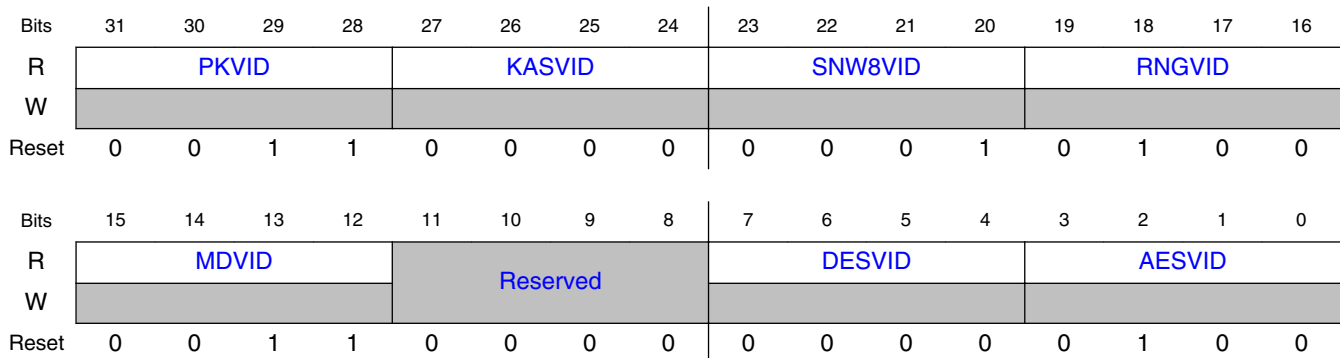
14.127 CHA Version ID Register, least-significant half (CHAVID_LS)**14.127.1 Offset**

Register	Offset
CHAVID_LS	FECh (alias)
CHAVID_LS	1_0FECh (alias)
CHAVID_LS	2_0FECh (alias)
CHAVID_LS	3_0FECh (alias)
CHAVID_LS	4_0FECh (alias)
CHAVID_LS	6_0FECh (alias)
CHAVID_LS	7_0FECh (alias)
CHAVID_LS	8_0FECh (alias)
CHAVID_LS	9_0FECh (alias)
CHAVID_LS	A_0FECh (alias)
CHAVID_LS	B_0FECh (alias)
CHAVID_LS	C_0FECh (alias)
CHAVID_LS	D_0FECh (alias)

14.127.2 Function

The CHA Version ID register can be used, along with the CCB Version ID, by software to differentiate between different versions of the cryptographic hardware accelerators. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.127.3 Diagram



14.127.4 Fields

Field	Function
31-28 PKVID	Public Key Hardware Accelerator Version ID The bit count is the size of the digit used during computation. The single-digit ("SD") versions allow a minimum modulus size of one byte. 0000b - PKHA-XT (32-bit); minimum modulus five bytes 0001b - PKHA-SD (32-bit) 0010b - PKHA-SD (64-bit) 0011b - PKHA-SD (128-bit)
27-24 KASVID	Kasumi f8/f9 Hardware Accelerator Version ID
23-20 SNW8VID	SNOW-f8 Hardware Accelerator Version ID
19-16 RNGVID	Random Number Generator Version ID. 0010b - RNGB 0100b - RNG4
15-12 MDVID	Message Digest Hardware Accelerator Version ID. 0000b - low-power MDHA, with SHA-1, SHA-256, SHA 224, MD5 and HMAC

Table continues on the next page...

CHA Number Register, most-significant half (CHANUM_MS)

Field	Function
	0001b - low-power MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5 and HMAC 0010b - medium-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC 0011b - high-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC
11-8 —	Reserved
7-4 DESVID	DES Accelerator Version ID.
3-0 AESVID	AES Accelerator Version ID 0011b - low-power AESA, implementing ECB, CBC, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, and GCM modes 0100b - high-performance AESA, implementing ECB, CBC, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, CBCXCBC, CTRXCBC, XTS, and GCM modes

14.128 CHA Number Register, most-significant half (CHANUM_MS)

14.128.1 Offset

Register	Offset
CHANUM_MS	FF0h (alias)
CHANUM_MS	1_0FF0h (alias)
CHANUM_MS	2_0FF0h (alias)
CHANUM_MS	3_0FF0h (alias)
CHANUM_MS	4_0FF0h (alias)
CHANUM_MS	6_0FF0h (alias)
CHANUM_MS	7_0FF0h (alias)
CHANUM_MS	8_0FF0h (alias)
CHANUM_MS	9_0FF0h (alias)
CHANUM_MS	A_0FF0h (alias)
CHANUM_MS	B_0FF0h (alias)
CHANUM_MS	C_0FF0h (alias)
CHANUM_MS	D_0FF0h (alias)

14.128.2 Function

The CHA Number register can be used by software to determine how many copies of each type of cryptographic hardware accelerator are implemented in this version of SEC. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.128.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	JRNUM				DECONUM				Reserved							
W																
Reset	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ZANUM				ZENUM				SNW9NUM				CRCNUM			
W																
Reset	0	0	0	1	0	1	1	0	0	0	0	1	0	1	1	0

14.128.4 Fields

Field	Function
31-28 JRNUM	The number of copies of the Job Ring that are implemented in this version of SEC
27-24 DECONUM	The number of copies of the DECO that are implemented in this version of SEC
23-16 —	Reserved
15-12 ZANUM	The number of copies of ZUCA that are implemented in this version of SEC
11-8 ZENUM	The number of copies of ZUCE that are implemented in this version of SEC
7-4 SNW9NUM	The number of copies of the SNOW-f9 module that are implemented in this version of SEC
3-0 CRCNUM	The number of copies of the CRC module that are implemented in this version of SEC

14.129 CHA Number Register, least-significant half (CHANUM_LS)

14.129.1 Offset

Register	Offset
CHANUM_LS	FF4h (alias)
CHANUM_LS	1_0FF4h (alias)
CHANUM_LS	2_0FF4h (alias)
CHANUM_LS	3_0FF4h (alias)
CHANUM_LS	4_0FF4h (alias)
CHANUM_LS	6_0FF4h (alias)
CHANUM_LS	7_0FF4h (alias)
CHANUM_LS	8_0FF4h (alias)
CHANUM_LS	9_0FF4h (alias)
CHANUM_LS	A_0FF4h (alias)
CHANUM_LS	B_0FF4h (alias)
CHANUM_LS	C_0FF4h (alias)
CHANUM_LS	D_0FF4h (alias)

14.129.2 Function

The CHA Number register can be used by software to determine how many copies of each type of cryptographic hardware accelerator are implemented in this version of SEC. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

14.129.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PKNUM				KASNUM				SNW8NUM				RNGNUM			
W																
Reset	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	1
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MDNUM				ARC4NUM				DESNUM				AESNUM			
W																
Reset	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0

14.129.4 Fields

Field	Function
31-28 PKNUM	The number of copies of the Public Key module that are implemented in this version of SEC
27-24 KASNUM	The number of copies of the Kasumi module that are implemented in this version of SEC
23-20 SNW8NUM	The number of copies of the SNOW-f8 module that are implemented in this version of SEC
19-16 RNGNUM	The number of copies of the Random Number Generator that are implemented in this version of SEC.
15-12 MDNUM	The number of copies of the MDHA (Hashing module) that are implemented in this version of SEC.
11-8 ARC4NUM	The number of copies of the ARC4 module that are implemented in this version of SEC.
7-4 DESNUM	The number of copies of the DES module that are implemented in this version of SEC.
3-0 AESNUM	The number of copies of the AES module that are implemented in this version of SEC.

14.130 Input Ring Base Address Register for Job Ring a (IRBAR_JR0 - IRRAR_JR3)

14.130.1 Offset

For a = 0 to 3:

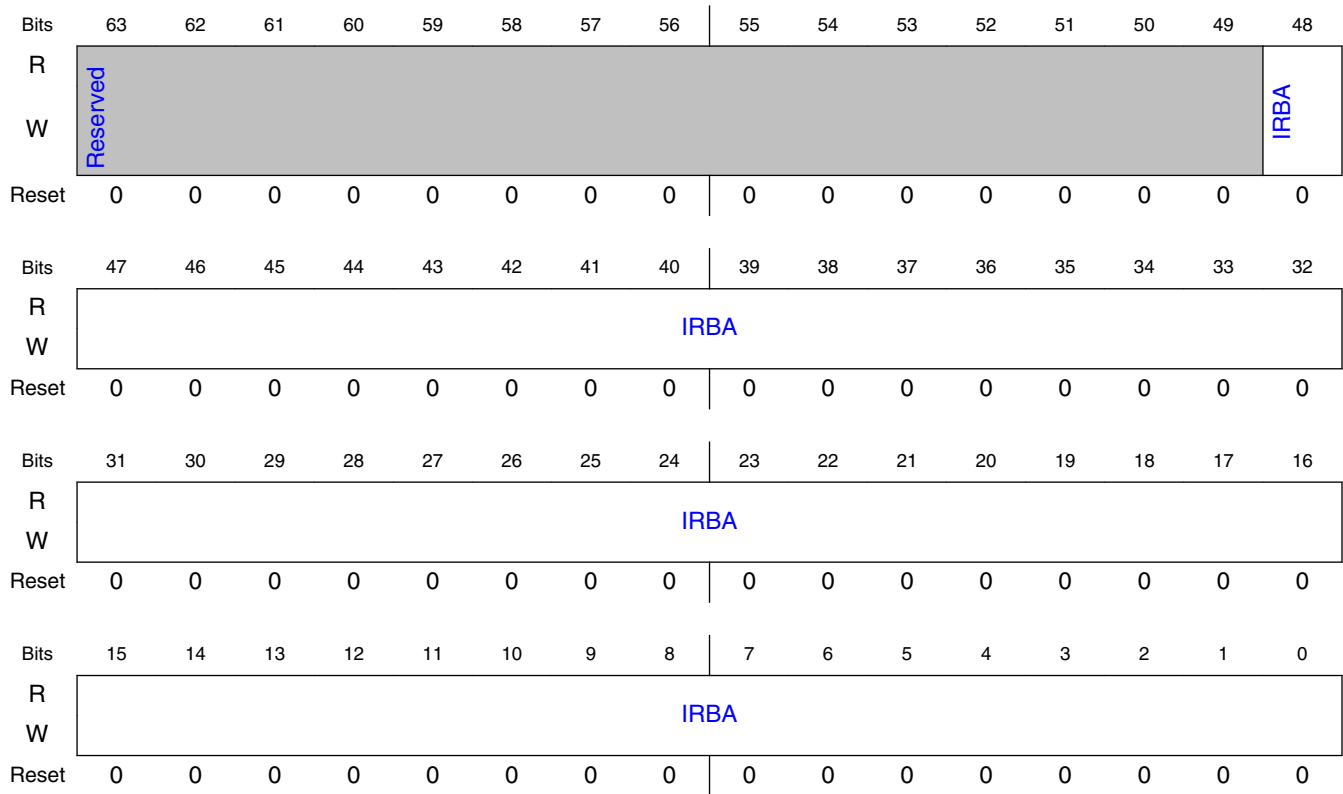
Register	Offset	Description
IRBAR_JRa	1_0000h + (a × 1_0000h)	Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.130.2 Function

The Input Ring Base Address register holds the physical address of the input ring in memory (see [Address pointers](#). SEC will use the number of address bits configured by the Pointer Size bit of the [Master Configuration Register \(MCFGR\)](#). Because there are 4 Job Rings, there are 4 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRBAR may only be written with a transaction with ns=0. Also, the Job Ring must be started, if virtualization is enabled. The IRBAR register can be written only when there are no jobs in the input ring or when the Job Ring is halted, else an input ring base address or size invalid write error will result and a Job Ring reset or a power on reset will be required. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRBAR the new head of the queue within the input ring will be located at the value just written to the IRBAR. Note that if the input ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost. The address written to the Input Ring Base Address register must be 4-byte aligned, else an error will result and the Job Ring will not process jobs until a valid address is written and the error is cleared. More information on job management can be found in [Job Ring interface](#).

14.130.3 Diagram



14.130.4 Fields

Field	Function
63-49 —	Reserved
48-0 IRBA	Input Ring Base Address.

14.131 Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR3)

14.131.1 Offset

For a = 0 to 3:

Register	Offset	Description
IRSR_JRa	1_000Ch + (a × 1_0000h)	(used by JR a)

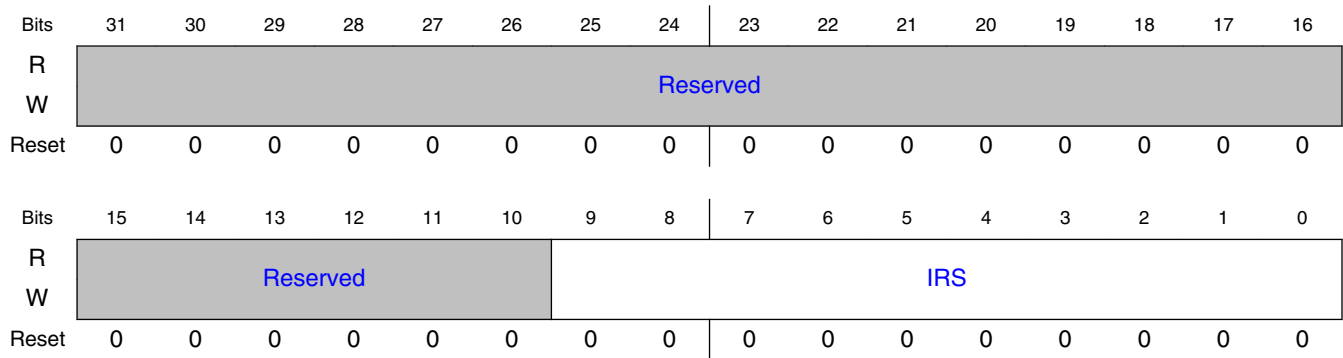
14.131.2 Function

The Input Ring Size register holds the current size of the input ring, measured in number of entries. Note that each entry will be one word if 32-bit pointers are in use, but will be two words if pointers larger than 32-bits are in use. Because there are 4 Job Rings, there are 4 copies of this register.

When the Job Ring is allocated to TrustZone SecureWorld, IRSR may only be written with a transaction with ns=0. Also, the Job Ring must be started, if virtualization is enabled, in order to write the register. See Section [Job Ring Registers](#). This register can be written only when there are no jobs in the input ring or when the Job Ring is halted, else an input ring base address or size invalid write error (type 5h) will result and a Job Ring reset or a power on reset will be required. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRSR the new head of the queue within the input ring will be located at the value stored in the IRBAR. Note that if the input ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost.

The size of the pointer entries in the ring is defined by the Pointer Size field of the (see [Master Configuration Register \(MCFGR\)](#)). See [Address pointers](#) for a discussion of address pointers. More information on job management can be found in [Job Ring interface](#).

14.131.3 Diagram



14.131.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 IRS	Input Ring Size. (measured in number of entries)

14.132 Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR3)

14.132.1 Offset

For a = 0 to 3:

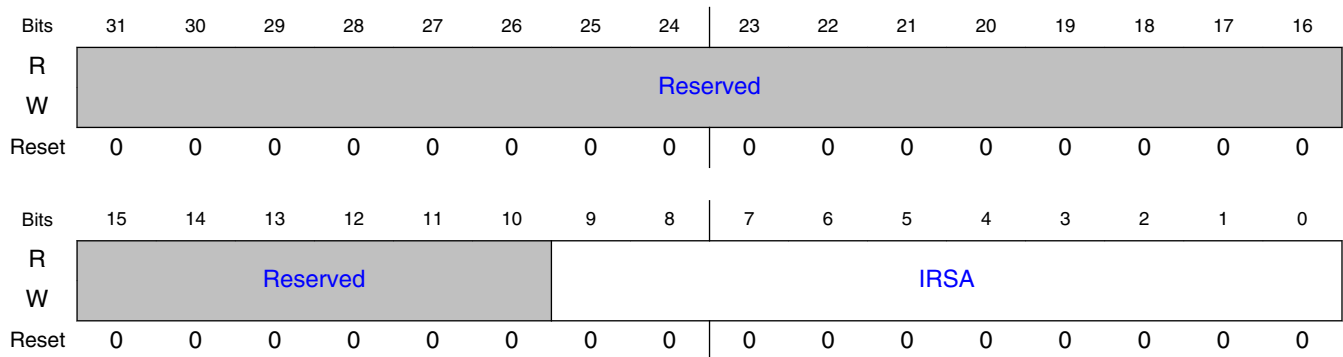
Register	Offset	Description
IRSAR_JRa	1_0014h + (a × 1_0000h)	(used by JR a)

14.132.2 Function

The Input Ring Slots Available Register gives the number of empty slots for jobs in the input ring. Each slot is one word long if 32-bit pointers are in use, but is two words long if pointers larger than 32 bits are in use. Because there are 4 Job Rings, there are 4 copies of this register. This tells software how many more jobs it can submit to SEC before the input ring would be full. SEC increments this register when it removes a job from the input ring for processing. SEC decrements this register by the value in the Input Ring Jobs Added Register (see Section [Input Ring Jobs Added Register for Job Ring a \(IRJA_R_JR0 - IRJAR_JR3\)](#)) when that register is updated. The value of the Input Ring Slots Available Register will never be larger than the Input Ring Size Register (see Section [Input Ring Size Register for Job Ring a \(IRSR_JR0 - IRSR_JR3\)](#)). More information on job management can be found in [Job Ring interface](#).

The Job Ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, IRSAR may only be written with a transaction with ns=0. See Section [Job Ring Registers](#).

14.132.3 Diagram



14.132.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 IRSA	Input Ring Slots Available. (measured in number of available job slots)

14.133 Input Ring Jobs Added Register for Job Ring a (IRJAR_JR0 - IRJAR_JR3)

14.133.1 Offset

For a = 0 to 3:

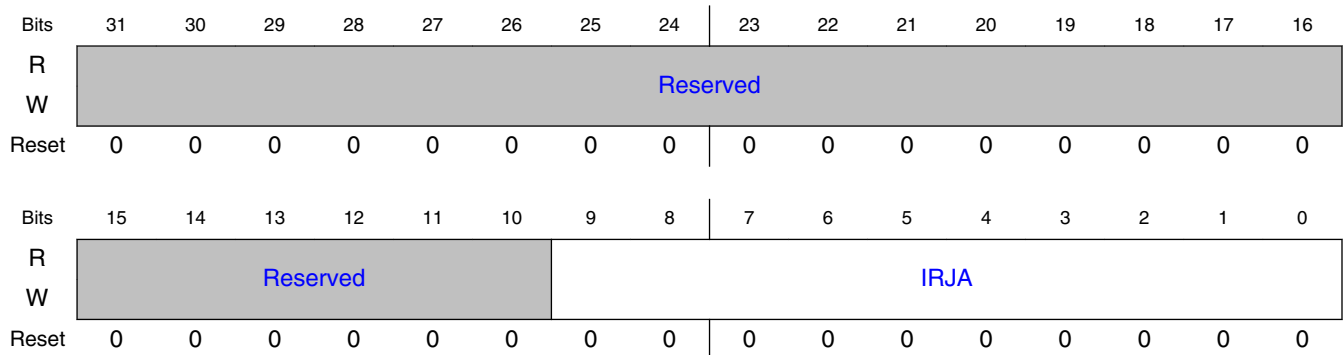
Register	Offset	Description
IRJAR_JRa	1_001Ch + (a × 1_0000h)	(used by JR a)

14.133.2 Function

The Input Ring Jobs Added Register tells SEC how many new jobs were added to the input ring. Because there are 4 Job Rings, there are 4 copies of this register. Software must write into this register the number of Job Descriptor addresses that software has added to the ring. When the Input Ring Jobs Added Register is written, SEC adds that new value to its count of the jobs available for processing and decrements the Input Ring Slots Available Register. The value in the Input Ring Jobs Added Register must not be larger than the value of the Input Ring Slots Available Register (see Section [Input Ring Slots Available Register for Job Ring a \(IRSAR_JR0 - IRSAR_JR3\)](#)). If more jobs are added than the value in the Input Ring Slots Available Register an "Added too many jobs" error (type 9h) will occur. This is a fatal error and will require a Job Ring reset or power on reset to correct. More information on job management can be found in [Job Ring interface](#).

When the Job Ring is allocated to TrustZone SecureWorld, IRJAR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section [Job Ring Registers](#).

14.133.3 Diagram



14.133.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 IRJA	Input Ring Jobs Added. (measured in number of entries)

14.134 Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3)

14.134.1 Offset

For a = 0 to 3:

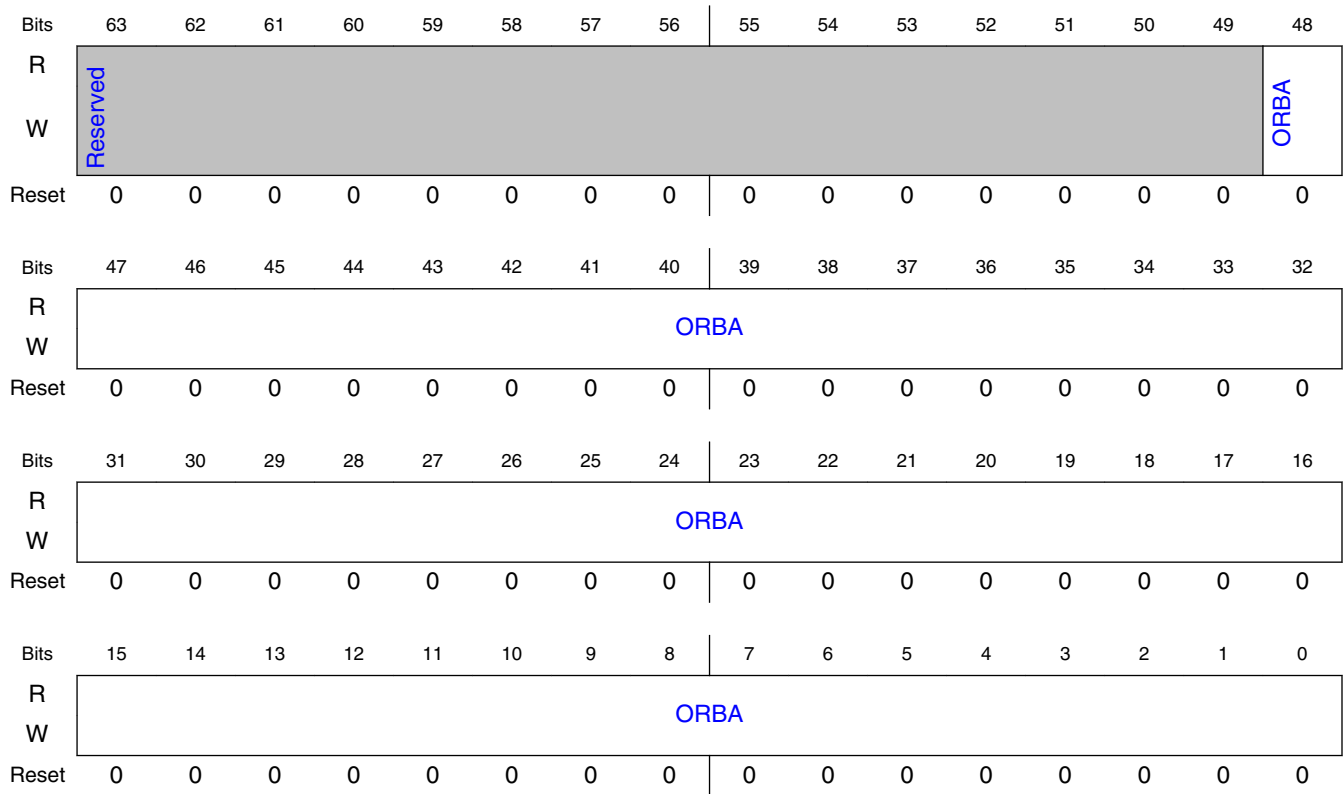
Register	Offset	Description
ORBAR_JRa	1_0020h + (a × 1_0000h)	Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.134.2 Function

The Output Ring Base Address Register holds the address of the output ring in memory (see [Job Ring interface](#)). SEC will use the number of bits configured by the Pointer Size bit of the [Master Configuration Register \(MCFGR\)](#). Because there are 4 Job Rings, there are 4 copies of this register. When the Job Ring is allocated to TrustZone SecureWorld, ORBAR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section [Job Ring Registers](#). This register can be written only when the Job Ring is halted or when there are no jobs from this ring in progress within SEC or in the input ring or output ring, else an output ring base address or size invalid write error will result and a Job Ring reset, software SEC reset or a power on reset will be required.

Writing this register resets the Output Ring Write Index register, therefore following a write to the ORBAR the new tail of the queue within the output ring will be located at the value just written to the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that Job Ring will either still be in the input ring or will be completed and written to the output ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the input ring before writing the new output ring base address. This would allow for a clean start with a new empty output ring. Note that if the output ring was not empty at the time the ORBAR was written, those old results entries will not be in the new output ring. The address written to the Output Ring Base Address register must be 4-byte aligned, else an error will result and the Job Ring will not process jobs until a valid address is written and the error is cleared. More information on job management can be found in [Job Ring interface](#).

14.134.3 Diagram



14.134.4 Fields

Field	Function
63-49 —	Reserved
48-0 ORBA	Output Ring Base Address.

14.135 Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR3)

14.135.1 Offset

For a = 0 to 3:

Register	Offset	Description
ORSR_JRa	1_002Ch + (a × 1_0000h)	(used by JR a)

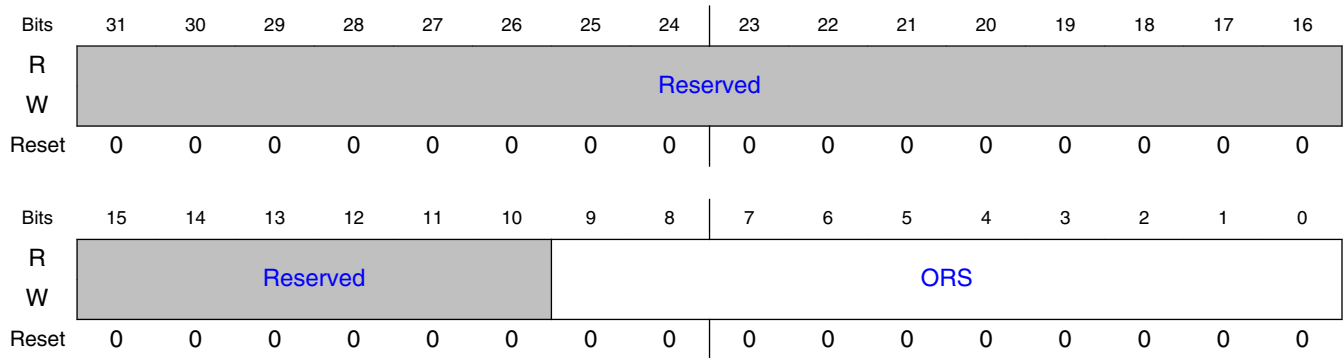
14.135.2 Function

The Output Ring Size Register holds the current size of the output ring, measured in number of entries. Each entry in the output ring consists of one descriptor address pointer plus one 32-bit results status word, plus an optional word indicating the length of the SEQ sequence, if any, associated with this job (see [INCL_SEQ_OUT](#) field in the section [Job Ring Configuration Register for Job Ring a, most-significant half \(JRCFGR_JR0_MS - JRCFGR_JR3_MS\)](#)). The size of a descriptor pointer is defined by the Pointer Size bit of the [Master Configuration Register \(MCFGR\)](#). If PS=0, pointers are 32 bits. If PS=1, pointers are 49 bits. See [Address pointers](#) for a discussion of address pointers. Because there are 4 Job Rings, there are 4 copies of this register. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section [Job Ring Registers](#). This register can be written only when the Job Ring is halted or when there are no jobs from this ring in the input ring or output ring or in progress within SEC, else an *output ring base address or size invalid write error* will result and a Job Ring reset, software SEC reset or a power on reset will be required.

Writing this register resets the Output Ring Write Index register, therefore following a write to the ORSR the new tail of the queue within the output ring will be located at the value stored in the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that Job Ring will either still be in the input ring or will be completed and written to the output ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the input ring before writing the new output ring base address. This would allow for a clean start with a new empty output ring. Note that if the output ring was not empty at the time the ORSR was written, those old results entries will not be in the new output ring. If the output ring is not empty when the ORSR is written, software may need to process or relocate those entries to avoid losing job results.

More information on job management can be found in [Job Ring interface](#).

14.135.3 Diagram



14.135.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 ORS	Output Ring Size. (measured in number of entries)

14.136 Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR3)

14.136.1 Offset

For a = 0 to 3:

Register	Offset	Description
ORJRR_JRa	1_0034h + (a × 1_0000h)	(used by JR a)

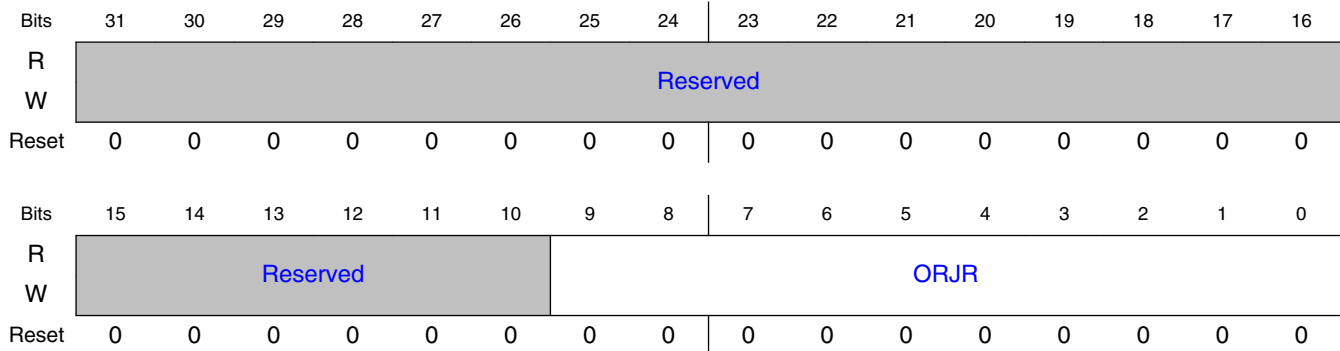
14.136.2 Function

The Output Ring Jobs Removed Register tells SEC how many jobs were removed from the output ring for processing by software. Because there are 4 Job Rings, there are 4 copies of this register. Software must write into this register the number of entries that software has removed from the ring. When the Output Ring Jobs Removed Register is written, SEC will subtract this amount from the Output Ring Slots Full Register (see Section [Output Ring Slots Full Register for Job Ring a \(ORSFR_JR0 - ORSFR_JR3\)](#)). The value of the Output Ring Jobs Removed Register must not be larger than the value in the Output Ring Slots Full Register. If a value larger than the Output Ring Slots Full Register is written to the ORJRR, a removed too many jobs error will occur and a Job Ring reset, software SEC reset or a power on reset will be required.

When the Job Ring is allocated to TrustZone SecureWorld, ORJRR may only be written with a transaction with ns=0. If virtualization is enabled, the Job Ring must be started in order to write the register. See Section [Job Ring Registers](#).

More information on job management can be found in [Job Ring interface](#).

14.136.3 Diagram



14.136.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 ORJR	Output Ring Jobs Removed. (measured in number of entries)

14.137 Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR3)

14.137.1 Offset

For a = 0 to 3:

Register	Offset	Description
ORSFR_JRa	1_003Ch + (a × 1_0000h)	(used by JR a)

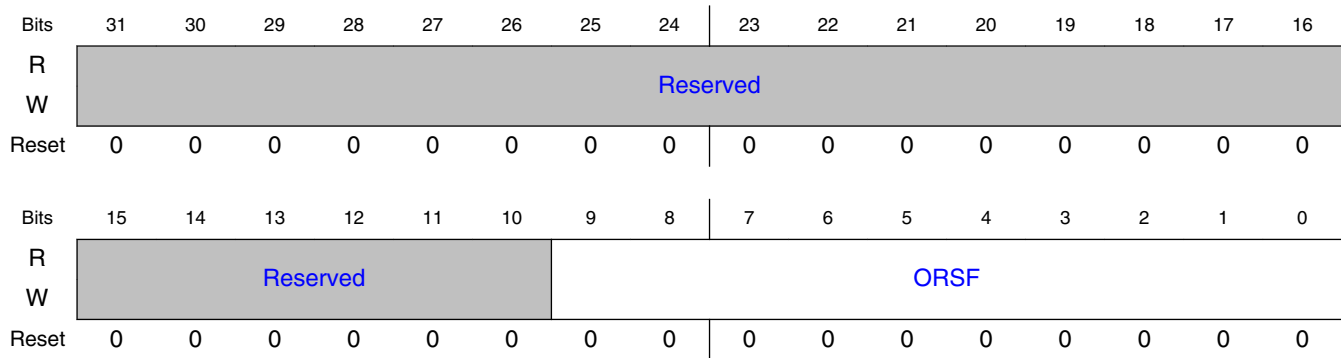
14.137.2 Function

The Output Ring Slots Full Register tells the software how many completed jobs SEC has placed in the output ring. Because there are 4 Job Rings, there are 4 copies of this register. SEC will increment this register as it completes a Descriptor and adds it to the output ring. SEC will decrement this register when software writes a new value to the Output Ring Jobs Removed Register (see Section [Output Ring Jobs Removed Register for Job Ring a \(ORJRR_JR0 - ORJRR_JR3\)](#)). The value in the Output Ring Slots Full Register cannot be larger than the value in the Output Ring Size Register (see Section [Output Ring Size Register for Job Ring a \(ORSR_JR0 - ORSR_JR3\)](#)).

The Job Ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, ORSFR may only be written with a transaction with ns=0. See Section [Job Ring Registers](#).

More information on job management can be found in [Job Ring interface](#).

14.137.3 Diagram



14.137.4 Fields

Field	Function
31-10 —	Reserved. Must be 0.
9-0 ORSF	Output Ring Slots Full. (measured in number of entries)

14.138 Job Ring Output Status Register for Job Ring a (JRST AR_JR0 - JRSTAR_JR3)

14.138.1 Offset

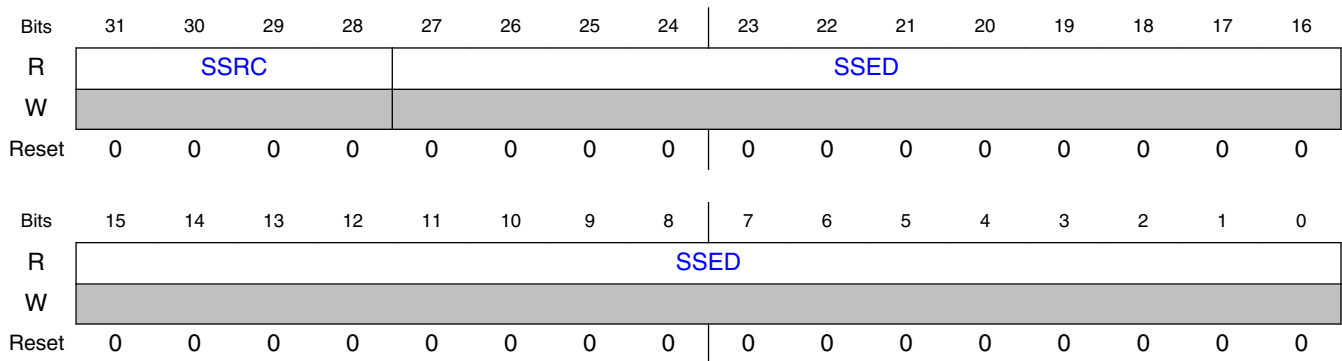
For a = 0 to 3:

Register	Offset	Description
JRSTAR_JRa	1_0044h + (a × 1_0000h)	(used by JR a)

14.138.2 Function

This register is used to show the status of the last job that was completed. Because there are 4 Job Rings, there are 4 copies of this register. Although it is possible to read the job completion status directly from this register, in normal circumstances this is not useful because the status value will quickly be overwritten when the next job completes. Bits 0-31 of this register are written into the output ring after the completion of a job, and software should read the status from there. More information on Job Ring management can be found in Section [Job Ring interface](#). Only one type of error will be valid at a time. The status code and various other information related to the status are given in the SSED field.

14.138.3 Diagram



14.138.4 Fields

Field	Function
31-28 SSRC	Status source. These bits define which source is reporting the status. All other values - reserved 0000b - No Status Source (No Error or Status Reported) 0001b - AI Source (AIOP Interface Error Reported) 0010b - CCB Status Source (CCB Error Reported) 0011b - Jump Halt User Status Source (User-Provided Status Reported) 0100b - DECO Status Source (DECO Error Reported) 0101b - QI Status Source (Queue Manager Interface Error Reported) 0110b - Job Ring Status Source (Job Ring Error Reported) 0111b - Jump Halt Condition Codes (Condition Code Status Reported)
27-0 SSED	Source-specific error details. The format of this field depends on the status source specified in the SSRC field. The interpretation of the SSED field for all status sources is shown in Job termination status/error codes .

14.139 Job Ring Interrupt Status Register for Job Ring a (JRINTR_JR0 - JRINTR_JR3)

14.139.1 Offset

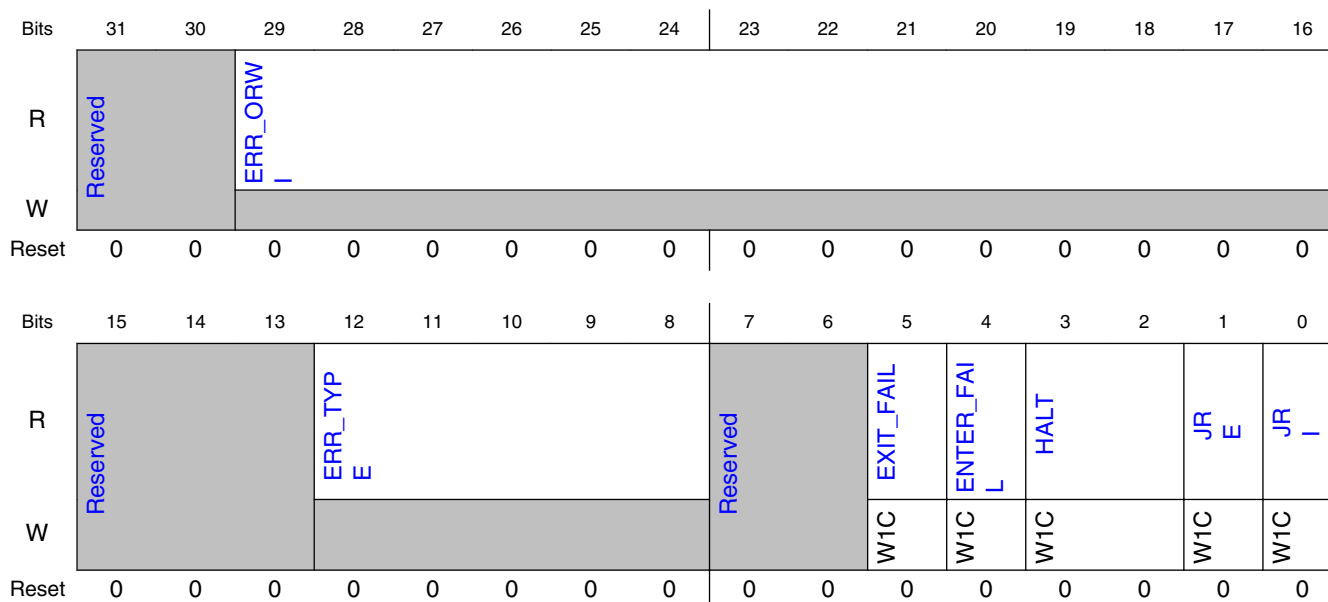
For a = 0 to 3:

Register	Offset	Description
JRINTR_JRa	1_004Ch + (a × 1_0000h)	(used by JR a)

14.139.2 Function

The Job Ring Interrupt Status Register indicates whether SEC has asserted an interrupt for a particular Job Ring, whether software has requested that the Job Ring be halted, whether the Job Ring is now halted, and whether there is an error in this Job Ring. If there was an error, the type of error is indicated. The error bit in the JRINT Register doesn't assert when there is a non-zero job completion status. It only asserts for the types of errors reported in the ERR_TYPE field in this register. Because there are 4 Job Rings, there are 4 copies of this register.

14.139.3 Diagram



14.139.4 Fields

Field	Function
31-30 —	Reserved
29-16 ERR_ORWI	Output ring write index with error. Set only when ERR_TYPE=0001. This indicates the location in the output ring that was being written when the error occurred. It is the offset in bytes from the Output Ring Base Address (see Section Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3)).
15-13 —	Reserved
12-8 ERR_TYPE	<p>Error type. Set only when JRE bit is also set. Indicates the type of error when it cannot be reported in the Job Ring Status Register (see Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3).)</p> <ul style="list-style-type: none"> 00001b - Error writing status to Output Ring 00011b - Bad input ring base address (not on a 4-byte boundary). 00100b - Bad output ring base address (not on a 4-byte boundary). 00101b - Invalid write to Input Ring Base Address Register or Input Ring Size Register. Can be written when there are no jobs in the input ring or when the Job Ring is halted. These are fatal and will likely result in not being able to get all jobs out into the output ring for processing by software. Resetting the job ring will almost certainly be necessary. 00110b - Invalid write to Output Ring Base Address Register or Output Ring Size Register. Can be written when there are no jobs in the output ring and no jobs from this queue are already processing in SEC (in the holding tanks or DECOs), or when the Job Ring is halted. 00111b - Job Ring reset released before Job Ring is halted. 01000b - Removed too many jobs (ORJRR larger than ORSFR).

Table continues on the next page...

Field	Function
	<p>01001b - Added too many jobs (IRJAR larger than IRSAR).</p> <p>01010b - Writing ORSF > ORS In these error cases the write is ignored, the interrupt is asserted (unless masked) and the error bit and error_type fields are set in the Job Ring Interrupt Status Register.</p> <p>01011b - Writing IRSA > IRS</p> <p>01100b - Writing ORWI > ORS in bytes</p> <p>01101b - Writing IRR1 > IRS in bytes</p> <p>01110b - Writing IRSA when ring is active</p> <p>01111b - Writing IRR1 when ring is active</p> <p>10000b - Writing ORSF when ring is active</p> <p>10001b - Writing ORWI when ring is active</p>
7-6 —	Reserved
5 EXIT_FAIL	Exit SecMon Fail State. This bit asserts when SecMon Fail State is exited. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the EXIT_FAIL bit will clear it.
4 ENTER_FAIL	Enter SecMon Fail State. This bit asserts when SecMon Fail State is entered. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the ENTER_FAIL bit will clear it.
3-2 HALT	<p>Halt the Job Ring.</p> <p>If reading HALT returns 01:</p> <p>Software has requested that SEC flush the jobs in this Job Ring and halt processing jobs in this Job Ring (by writing to the RESET bit in the Job Ring Command register.).</p> <p>If reading HALT returns 10:</p> <p>SEC has flushed all jobs from this Job Ring and has halted processing jobs in this Job Ring. If there is not enough room in the output ring for all the flushed jobs, HALT will continue to return 01 until software has removed enough jobs so that all the flushed jobs can be written to the output ring.</p> <p>Software writes a "1" to the MSB of HALT (bit 3) to clear the HALT field and resume processing jobs in this Job Ring. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that SEC has flushed all jobs from this Job Ring.</p> <p>If SecMon indicates a FAIL MODE and the FAIL MODE bit is not set in the Job Ring Configuration register, a Job Ring halt will be initiated and the HALT status will return 01. When the halt process is complete, the HALT status will be 10. The HALT status cannot be cleared until SecMon transitions out of FAIL MODE. If the FAIL MODE bit is set in the Job Ring Configuration register, the Job Ring is not halted in FAIL MODE.</p>
1 JRE	Job Ring Error. A Job Ring error occurred. The error code is indicated in the ERR_TYPE field in this register. Write a 1 to this bit to clear the error indication.
0 JRI	Job Ring Interrupt. SEC has asserted the interrupt request signal for this Job Ring. Write a 1 to this bit to clear the interrupt request.

14.140 Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS)

14.140.1 Offset

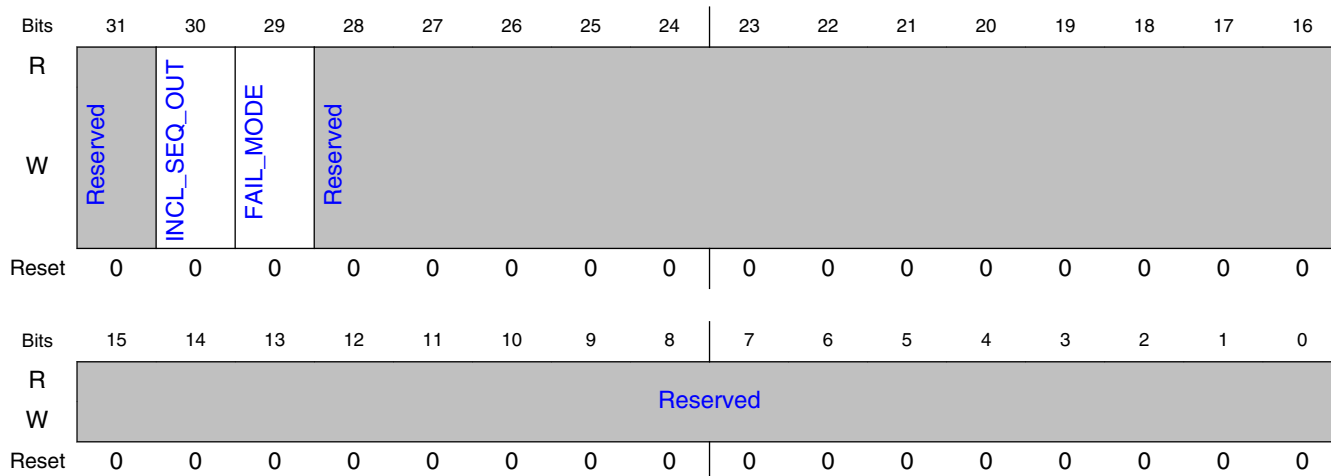
For a = 0 to 3:

Register	Offset	Description
JRCFGR_JRa_MS	1_0050h + (a × 1_0000h)	(used by JR a)

14.140.2 Function

Software uses the Job Ring Configuration Register to configure the interrupt handling and error handling specific to a Job Ring. Because there are 4 Job Rings, there are 4 copies of this register. Since there are more than 32 bits in the JRCFG Register, it is accessed as two 32-bit words.

14.140.3 Diagram



14.140.4 Fields

Field	Function
31	Reserved
—	

Table continues on the next page...

Field	Function
30 INCL_SEQ_OUT	Include Sequence Out Length. If this bit is set to 1, entries in the job ring's output ring will include a 32-bit word indicating the number of bytes written out via SEQ STORE and SEQ FIFO STORE commands in this job. If this bit is set to 0, the SEQ OUT Length is omitted from the entries. The setting of this bit can be changed only during ring configuration, when no jobs are running in SEC, else an error will be flagged.
29 FAIL_MODE	Fail mode control. If this bit is set to 1 and SecMon indicates a FAIL MODE, the Job Ring will assert its interrupt and set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The Job Ring will not halt, but will continue to process any available jobs. DECO will return these jobs with a FAIL MODE error. If SecMon transitions out of FAIL MODE, the Job Ring will assert its interrupt and set the EXIT_FAIL bit in the Job Ring Interrupt Status register. If this bit is set to 0 and SecMon indicates a FAIL MODE, the Job Ring will set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The Job Ring will halt until SecMon transitions out of FAIL MODE. When the Job Ring has halted, it will assert its interrupt. If SecMon transitions out of FAIL MODE, the Job Ring will set the EXIT_FAIL bit in the Job Ring Interrupt Status register.
28-0 —	Reserved. Must be 0.

14.141 Job Ring Configuration Register for Job Ring a, least-significant half (JR CFGR_JRa_LS - JR CFGR_JR3_LS)

14.141.1 Offset

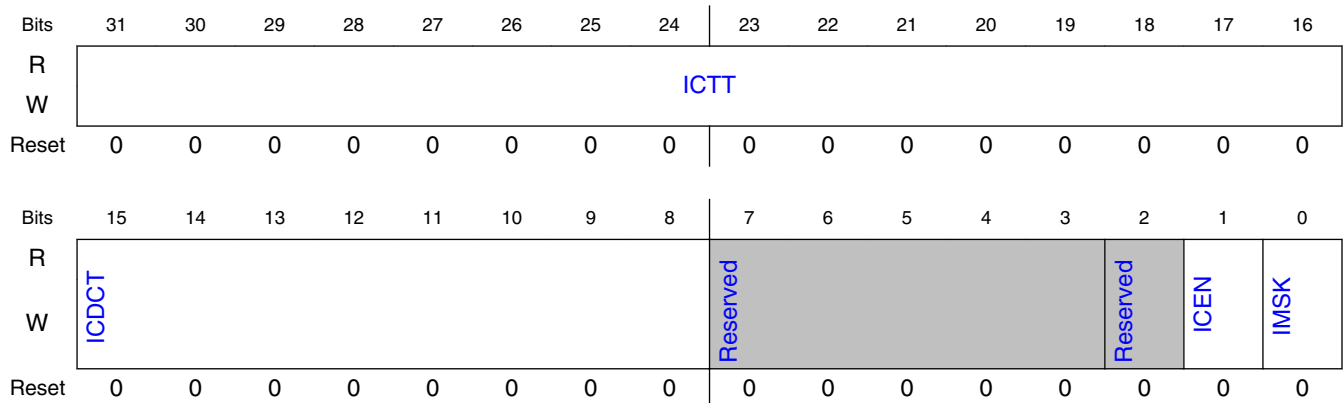
For a = 0 to 3:

Register	Offset	Description
JR CFGR_JRa_LS	1_0054h + (a × 1_0000h)	(used by JR a)

14.141.2 Function

Software uses the Job Ring Configuration Register to configure the interrupt handling and error handling specific to a Job Ring. Because there are 4 Job Rings, there are 4 copies of this register. Since there are more than 32 bits in the JR CFG Register, it is accessed as two 32-bit words.

14.141.3 Diagram



14.141.4 Fields

Field	Function
31-16 ICTT	Interrupt Coalescing Timer Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines the maximum amount of time after processing a Descriptor before raising an interrupt. If Descriptors have been processed but the Descriptor count threshold has not been met, an interrupt is raised when the interrupt coalescing timer expires. The interrupt coalescing timer is stopped when the Output Ring Slots Full Register is 0. The timer is reset and stopped once an interrupt has been asserted or whenever the Output Ring Jobs Removed Register is written by software. Counting resumes from zero after a reset if the counter is still enabled. The timer begins counting once the next Descriptor is moved to the output ring. Note that it is possible for one or more Descriptors to be moved to the output ring after software has read the Output Ring Slots Full Register and before software has written the Output Ring Jobs Removed Register. This would cause the timer to be reset to 0, but still counting. In this situation an interrupt would be raised when the timer expires at the full threshold value (unless the interrupt was raised earlier due to the Descriptor Count Threshold). The threshold value is represented in units equal to 64 SEC interface clocks. Valid values for this field are from 1 to 65535. A value of 0 results in behavior identical to that when interrupt coalescing is disabled.
15-8 ICDCT	Interrupt Coalescing Descriptor Count Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines how many Descriptors are completed before raising an interrupt. Valid values for this field are from 0 to 255. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the threshold value is reached each time that a Job Descriptor is completed. A value of 0 is treated in the same manner as a value of 1. The value of ICDCT is ignored if ICEN=0.
7-3 —	Reserved
2 —	Reserved
1 ICEN	Interrupt Coalescing Enable. 0b - Interrupt coalescing is disabled. If the IMSK bit is cleared, an interrupt is asserted whenever a job is written to the output ring. ICDCT is ignored. Note that if software removes one or more jobs and clears the interrupt but the output rings slots full is still greater than 0 (ORSF > 0), then the interrupt will clear but reassert on the next clock cycle.

Table continues on the next page...

Field	Function
	1b - Interrupt coalescing is enabled. If the IMASK bit is cleared, an interrupt is asserted whenever the threshold number of frames is reached (ICDCT) or when the threshold timer expires (ICTT). Note that if software removes one or more jobs and clears the interrupt but the interrupt coalescing threshold is still met ($ORSF \geq ICDCT$), then the interrupt will clear but reassert on the next clock cycle.
0 IMSK	Interrupt Mask. Mask the interrupt that is associated with the particular processor. 0b - Interrupt enabled. 1b - Interrupt masked.

14.142 Input Ring Read Index Register for Job Ring a (IRRIR_JR0 - IRRIR_JR3)

14.142.1 Offset

For a = 0 to 3:

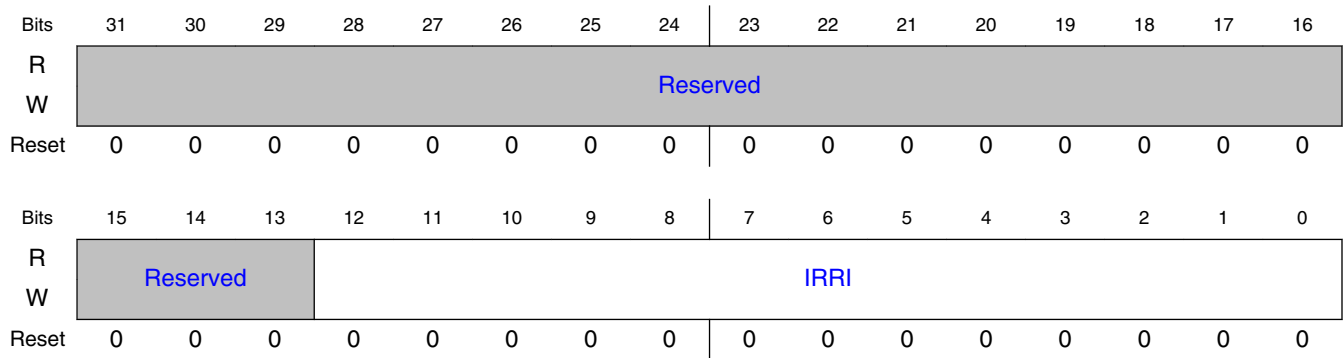
Register	Offset	Description
IRRIR_JRa	$1_005Ch + (a \times 1_0000h)$	(used by JR a)

14.142.2 Function

The Input Ring Read Index Register points to the head of the queue within the Input Ring. At this address there will be a pointer to the next Job Descriptor that SEC will fetch from this Job Ring. After SEC reads a Job Descriptor from the Job Ring SEC increments this register based on the Pointer Size bit of the Master Configuration Register described in Section [Master Configuration Register \(MCFGR\)](#). If PS=0 (pointers are 32 bits), the increment is 4. If PS=1 (pointers are 49 bits), the increment is 8. The index will be added to the Input Ring Base Address to get the physical address. Because there are 4 Job Rings, there are 4 copies of this register. More information on job management can be found in [Job Ring interface](#).

The Job Ring must be started in order to write the IRRIR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, IRRIR may only be written with a transaction with ns=0. See Section [Job Ring Registers](#).

14.142.3 Diagram



14.142.4 Fields

Field	Function
31-13 —	Reserved
12-0 IRRI	Input Ring Read Index.

14.143 Output Ring Write Index Register for Job Ring a (ORWIR_JR0 - ORWIR_JR3)

14.143.1 Offset

For a = 0 to 3:

Register	Offset	Description
ORWIR_JRa	1_0064h + (a × 1_0000h)	(used by JR a)

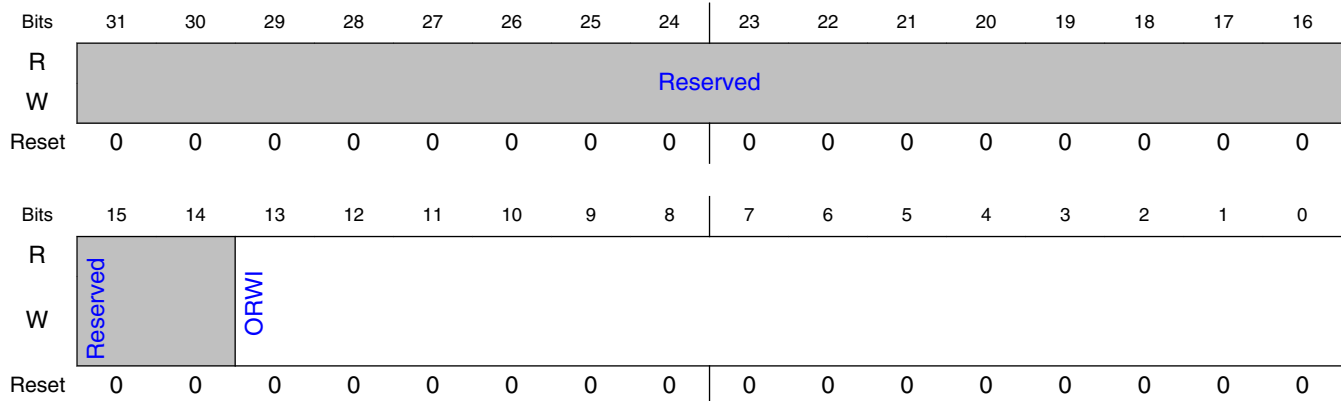
14.143.2 Function

The Output Ring Write Index Register points to the tail of the queue within the output ring. Because there are 4 Job Rings, there are 4 copies of this register. The Output Ring Write Index Register is added to the Output Ring Base Address Register to get the physical address. At this address SEC writes a pointer to the last Descriptor that SEC has processed. At the next entry in the ring SEC writes the completion status of that Descriptor. Every time that a Descriptor has been processed SEC increments the value in the Output Ring Write Index Register by the size of the pointer¹ plus the size of the 4-byte completion status word plus an additional 4 bytes if the INCL_SEQ_OUT bit in the JRCRGR is 1. So if INCL_SEQ_OUT=0 the increment will be 8 (if PS=0, i.e. 32-bit addresses) or 12 (if PS=1, i.e. 49-bit addresses). If INCL_SEQ_OUT=1, the increment will be 12 (if PS=0, i.e. 32-bit addresses) or 16 (if PS=1, i.e. 49-bit addresses). For a discussion of address pointers see [Address pointers](#).

The Job Ring must be started in order to write the ORWIR register. This register is read-only when virtualization is disabled. When the Job Ring is allocated to TrustZone SecureWorld, ORWIR may only be written with a transaction with ns=0. See Section [Job Ring Registers](#).

More information on job management can be found in [Job Ring interface](#).

14.143.3 Diagram



1. The size of the pointer is defined by the Pointer Size bit of the Master Configuration Register described in Section [Master Configuration Register \(MCFGR\)](#).

14.143.4 Fields

Field	Function
31-14 —	Reserved
13-0 ORWI	Output Ring Write Index. The pointer to the next entry in the output ring.

14.144 Job Ring Command Register for Job Ring a (JR0 - JR3)

14.144.1 Offset

For a = 0 to 3:

Register	Offset	Description
JR0 - JR3	1_006Ch + (a × 1_0000h)	(used by JR a)

14.144.2 Function

Software can use this register to issue a park, flush or reset command to a Job Ring. A flush command is issued by writing a 1 to JR0[RESET] when JR0[RESET]=0. A flush is defined as stalling any jobs currently in the input ring and terminating (with an error code) any jobs currently in progress in the holding tanks or DECOs. The terminated jobs will be written to the output ring with a status indicating that they were terminated by a flush request. Note that these flushed jobs will count towards the Interrupt Coalescing Descriptor Count. If there is not sufficient space in the output ring for all the flushed jobs, Job Ring flushing will be paused until software has made enough space in the Output Ring. After a flush completes, the halt can be cleared and job processing will resume, or a reset can be requested.

A park command will stall any jobs in the job ring that have not yet been fetched, but will allow all the jobs in progress to complete normally. A park command may be issued only if virtualization is enabled. If virtualization is disabled, any writes to the PARK bit are ignored.

During the time between the write to PARK and all of the in-progress jobs completing, the HALT field in the Job Ring Interrupt Status register will return 01b indicating that the Job Ring was asked to stop processing jobs. When all jobs are complete and the Job Ring has halted, the Job Ring Interrupt Status register will indicate this by setting the HALT field to 10b. Once the Job Ring indicates that it has halted, it is safe to read the values of the Job Ring registers to save the Job Ring state. The following register values should be saved:

- JRCFGR_JR - Job Ring Configuration Register for the Job Ring
- IRBAR_JR - Input Ring Base Address Register for the Job Ring
- IRSR_JR - Input Ring Size Register for the Job Ring
- ORBAR_JR - Output Ring Base Address Register for the Job Ring
- ORSR_JR - Output Ring Size Register for the Job Ring
- IRSAR_JR - Input Ring Slots Available Register for the Job Ring
- ORSFR_JR - Output Ring Slots Full Register for the Job Ring
- IRRIR_JR - Input Ring Read Index Register for the Job Ring
- ORWIR_JR - Output Ring Write Index Register for the Job Ring

Once the state is saved, the Job Ring may be reassigned. To reassign the Job Ring, the registers that were saved should be rewritten with new values. When reprogramming, note that IRS must be written before IRSA or IRRI, and ORS must be written before ORSF or ORWI. IRSA should be written last because this is the register that indicates to the Job Ring that it has jobs to process. Failure to write the registers in the correct order may result in one of the following errors: IRSA>IRS, IRRI>IRS, ORSF>ORS, or ORWI>ORS. Once the Job Ring is reprogrammed, park status can be released so that the Job Ring can start running again. To do this, write a "1" to the MSB of the HALT field in the Job Ring Interrupt Status register. Note that if software tries to release parking status before the Job Ring has halted, a fatal error will occur (type 00111). This is the same error type as releasing the Job Ring from reset status before the ring has halted.

A reset command is issued by writing a 1 to JRCCR[RESET] when JRCCR[RESET]=1. A reset command will clear all registers in the Job Ring except the following:

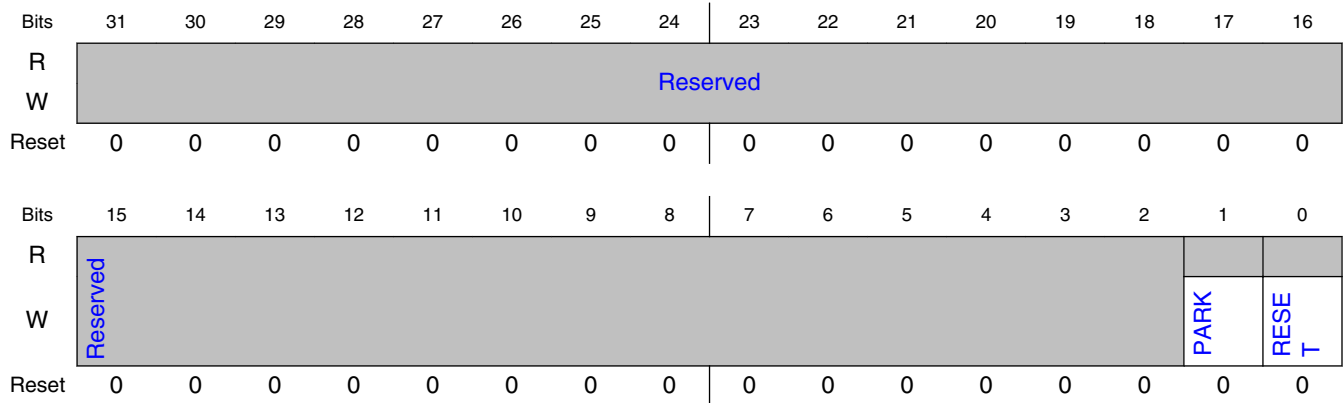
- Input Ring Base Address
- Input Ring Size
- Output Ring Base Address
- Output Ring Size
- Job Ring Configuration.

A reset can be initiated only after a flush has been requested and completed as indicated by the HALT field in the Job Ring Interrupt Status Register. After a reset, job processing will resume when the Input Ring Jobs Added Register is written to indicate that new jobs are available. If both PARK and RESET are written to 1, the PARK is ignored and the Job Ring is reset.

Job Ring a Address-Array Valid Register (JR0AAV - JR3AAV)

Because there are 4 Job Rings, there are 4 copies of this register.

14.144.3 Diagram



14.144.4 Fields

Field	Function
31-2 —	Reserved. Always 0.
1 PARK	Park - When PARK is 0, software writes a 1 to PARK to request that the Job Ring be "parked", i.e. quiesced. All jobs currently "in flight" (in holding tanks, DECOs or waiting for status results to be written out) are completed, but no new jobs are fetched from the input ring. When the job ring has completed parking, the HALT field in the Job Ring Interrupt Status register will change from 01b to 10b.
0 RESET	Reset - When RESET is 0, software writes a 1 to RESET to request a flush of the Job Ring. If software wants to initiate a reset of the Job Ring, software writes a 1 to the RESET bit after a flush (when RESET is already 1). The reset will clear the RESET bit and other registers in the job ring. If no reset is required, software writes a 1 to the MSB of the HALT field in the Job Ring Interrupt Status Register to cause the Job Ring to resume processing jobs. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that the SEC has flushed all jobs from this Job Ring.

14.145 Job Ring a Address-Array Valid Register (JR0AAV - JR3AAV)

14.145.1 Offset

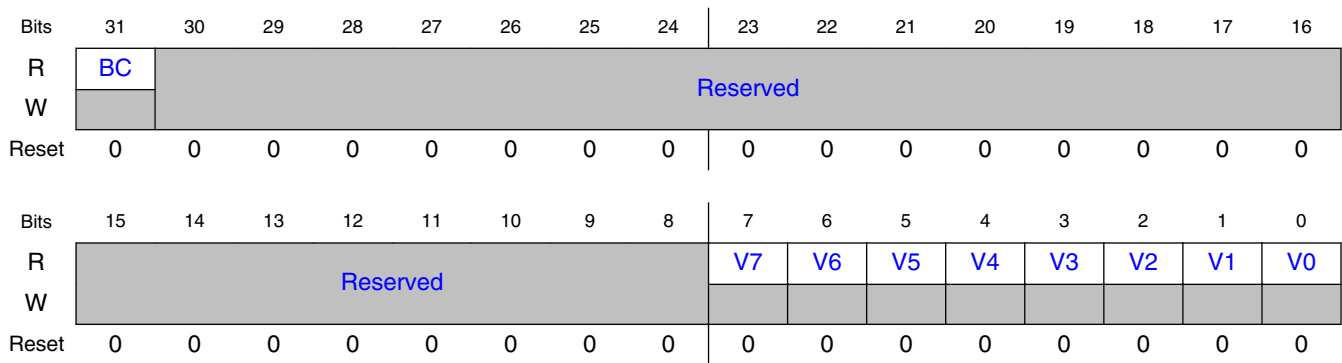
For a = 0 to 3:

Register	Offset	Description
JRaAAV	1_0704h + (a × 1_0000h)	Used with Job Ring a

14.145.2 Function

The Job Ring Address-Array Valid register indicates stored in the Job Ring Address-Array Address Registers. The register is intended to be used when debugging descriptor execution via a Job Ring. The Debug Control Register can be used to stop SEC processing before reading the job ring debug registers so that a consistent set of values can be read.

14.145.3 Diagram



14.145.4 Fields

Field	Function
31 BC	Been Changed. The BC bit is used to verify that consistent data has been read from the Address Array Registers. BC is set to 0 when JR0AAA0 is read, and BC is then set to 1 if the content of any of the JRAAAx registers or the JRAAVS register changes (due to new addresses being loaded into AA or existing addresses being sent to a holding tank) before the JRAAVS is read. So if BC is 1 after this sequence of register reads, some of the data that was read may be inconsistent with other data that was read. In this case the address Array registers should be read again.
30-8 —	Reserved
7 V7	Valid 7. When V7=1, Job Ring Address-Array Address Register 7 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.

Table continues on the next page...

Job Ring a Address-Array Address b Register (JR0AAA0 - JR3AAA7)

Field	Function
6 V6	Valid 6. When V6=1, Job Ring Address-Array Address Register 6 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
5 V5	Valid 5. When V5=1, Job Ring Address-Array Address Register 5 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
4 V4	Valid 4. When V4=1, Job Ring Address-Array Address Register 4 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
3 V3	Valid 3. When V3=1, Job Ring Address-Array Address Register 3 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
2 V2	Valid 2. When V2=1, Job Ring Address-Array Address Register 2 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
1 V1	Valid 1. When V1=1, Job Ring Address-Array Address Register 1 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.
0 V0	Valid 0. When V0=1, Job Ring Address-Array Address Register 0 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers.

14.146 Job Ring a Address-Array Address b Register (JR0AAA0 - JR3AAA7)

14.146.1 Offset

For a = 0 to 3; b = 0 to 7:

Register	Offset	Description
JRaAAAb	$1_0800h + (a \times 1_0000h) + (b \times 8h)$	Used with Job Ring a. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR) .

14.146.2 Function

The JRAAA registers are intended to be used when debugging descriptor execution via a Job Ring. The Debug Control Register can be used to stop SEC processing before reading the Job Ring debug registers so that a consistent set of values can be read. As discussed in [Job scheduling](#), the job Queue Controller buffers up to four Job Descriptors from one Job Ring before servicing the next Job Ring in round-robin fashion. For performance reasons SEC reads multiple input ring entries whenever possible, so SEC may read up to four job descriptor addresses in a single bus burst. These registers store the job descriptor addresses after the job queue controller fetches the descriptor address from the input ring and before assigning the descriptor to a Holding Tank.

14.146.3 Diagram



14.146.4 Fields

Field	Function
63-49 —	Reserved.
48-0 JD_ADDR	Job Descriptor Address.

14.147 Recoverable Error Interrupt Record 0 for Job Ring a (REIR0JR0 - REIR0JR3)

14.147.1 Offset

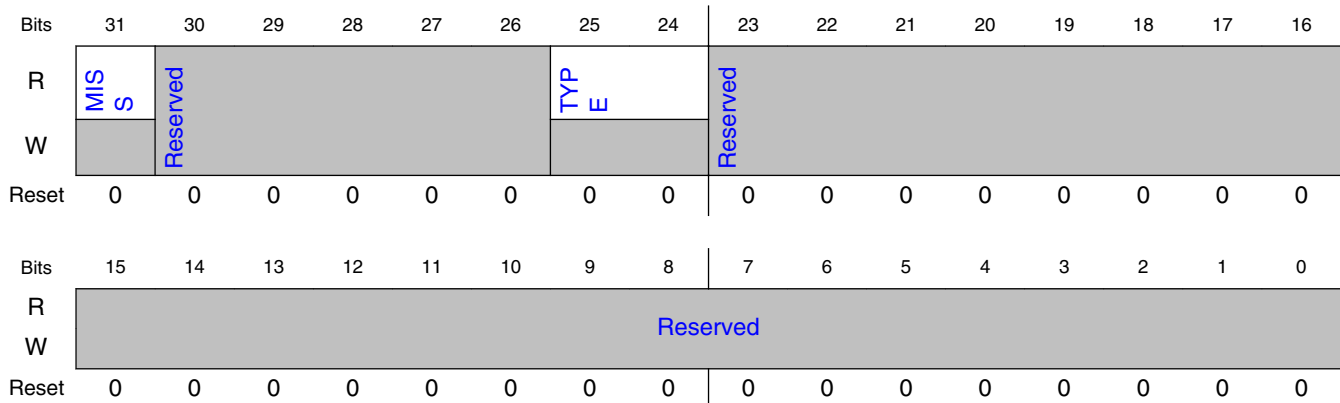
For a = 0 to 3:

Register	Offset	Description
REIR0JRa	1_0E00h + (a × 1_0000h)	(used by JR a)

14.147.2 Function

If a recoverable error occurs related to execution of a job from a Job Ring, error information will be captured in the JR's REIR registers. Data for a second recoverable error related to jobs from JR will not be captured until the REIR0JR is written. If another bus error from JR occurs before then, the double error status bit (MISS) in REIR0JR will be set. When REIR0JR is written, all of JR's REIRJR registers are cleared and error capture is re-enabled.

14.147.3 Diagram



14.147.4 Fields

Field	Function
31 MISS	If MISS=1, a second recoverable error associated with JR occurred before REIROJR was written following a previous JR recoverable error.
30-26 —	Reserved
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 0 : reserved If TYPE = 1 : memory access error If TYPE = 2 : reserved If TYPE = 3 : reserved
23-0 —	Reserved

14.148 Recoverable Error Interrupt Record 2 for Job Ring a (REIR2JR0 - REIR2JR3)

14.148.1 Offset

For a = 0 to 3:

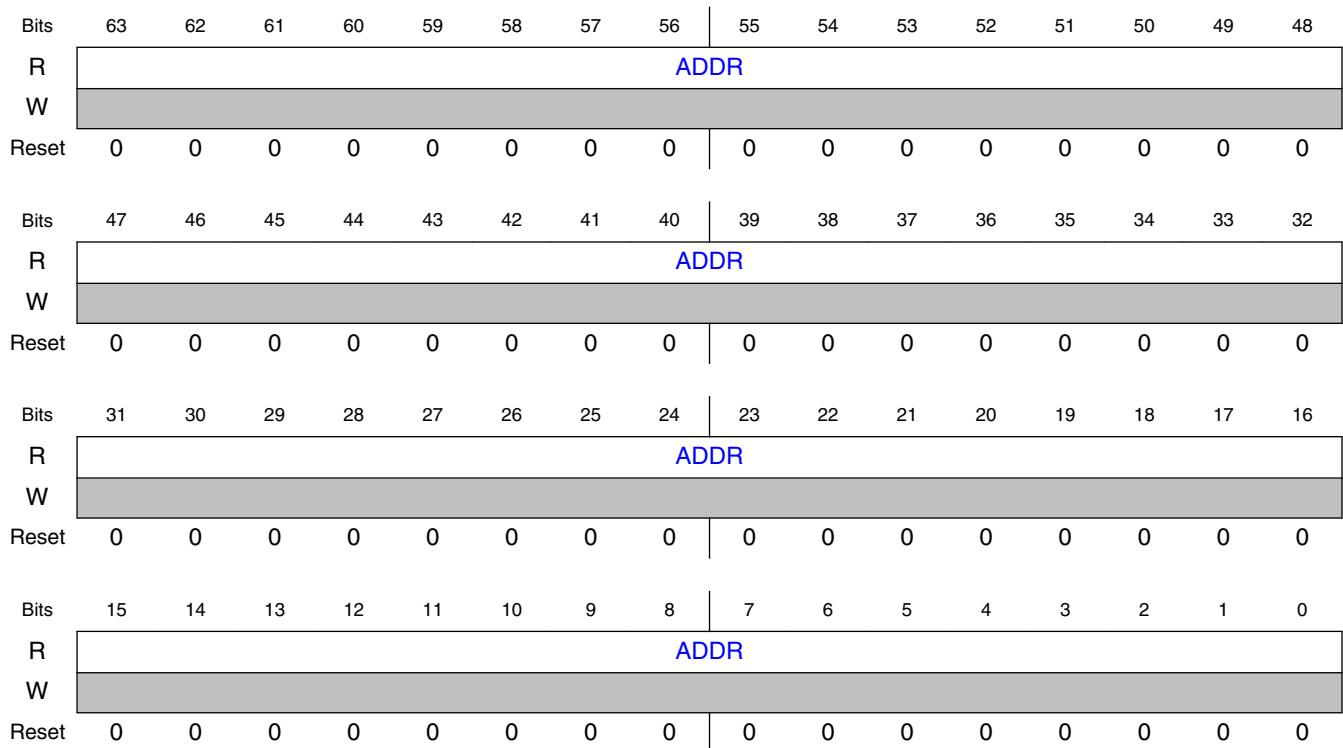
Recoverable Error Interrupt Record 2 for Job Ring a (REIR2JR0 - REIR2JR3)

Register	Offset	Description
REIR2JRa	1_0E08h + (a × 1_0000h)	Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.148.2 Function

See the description for [Recoverable Error Interrupt Record 0 for Job Ring a \(REIR0JR0 - REIR0JR3\)](#).

14.148.3 Diagram



14.148.4 Fields

Field	Function
63-0 ADDR	Address associated with the recoverable JR error.

14.149 Recoverable Error Interrupt Record 4 for Job Ring a (REIR4JR0 - REIR4JR3)

14.149.1 Offset

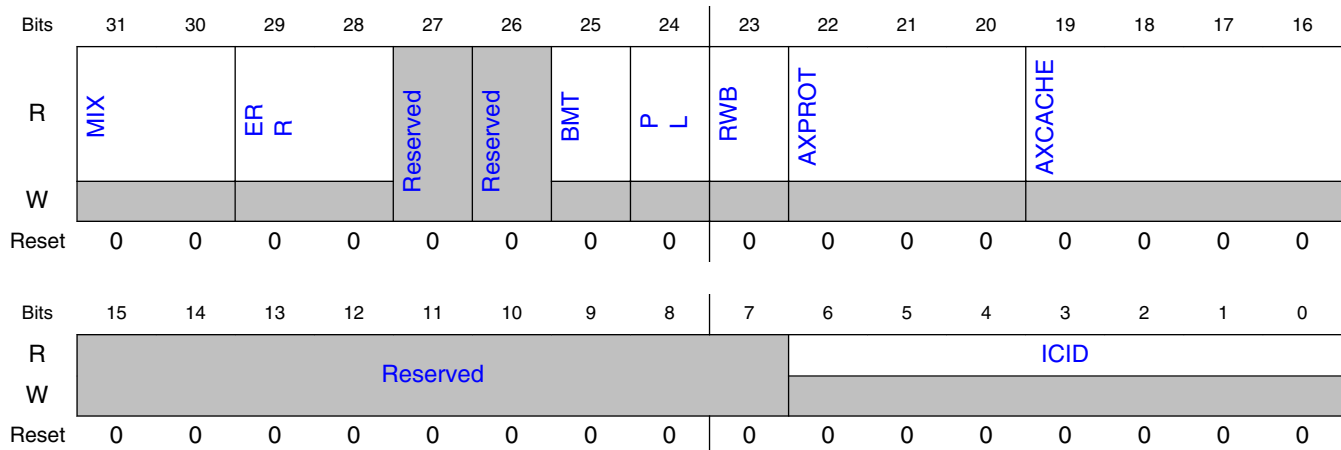
For a = 0 to 3:

Register	Offset	Description
REIR4JRa	1_0E10h + (a × 1_0000h)	(used by JR a)

14.149.2 Function

See the description for [Recoverable Error Interrupt Record 0 for Job Ring a \(REIR0JR0 - REIR0JR3\)](#).

14.149.3 Diagram



14.149.4 Fields

Field	Function
31-30	This field holds the memory interface index associated with the recoverable error.

Table continues on the next page...

Recoverable Error Interrupt Record 5 for Job Ring a (REIR5JR0 - REIR5JR3)

Field	Function
MIX	
29-28 ERR	This field holds the AXI error response associated with the recoverable error.
27 —	Reserved
26 —	Reserved
25 BMT	This field holds the bypass memory translation transaction attribute used for the memory access.
24 PL	This field holds the privilege level transaction attribute used for the memory access.
23 RWB	This field specifies whether the memory access was a read or write.
22-20 AXPROT	This field holds the AXI protection transaction attribute used for the memory access.
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15-7 —	Reserved
6-0 ICID	This field holds the ICID associated with the recoverable error.

14.150 Recoverable Error Interrupt Record 5 for Job Ring a (REIR5JR0 - REIR5JR3)

14.150.1 Offset

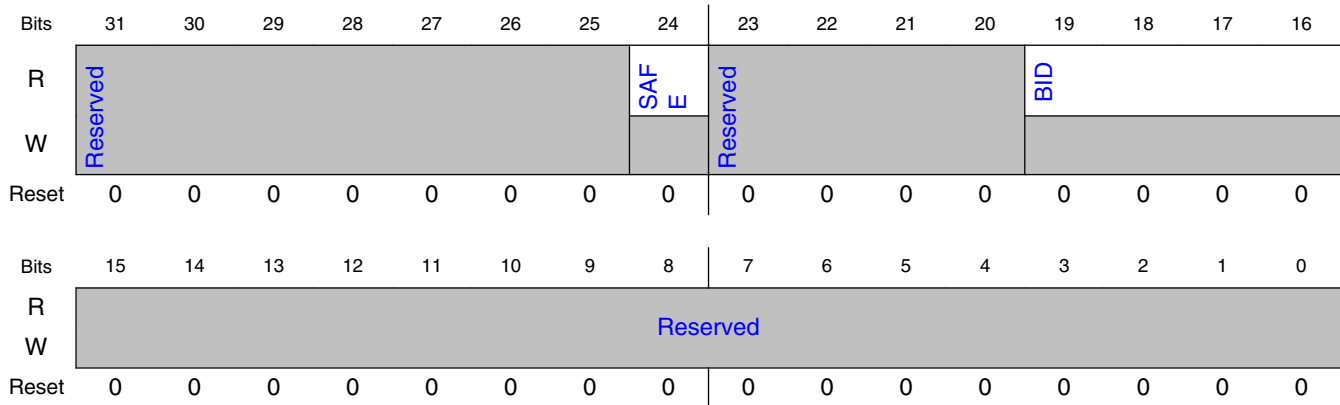
For a = 0 to 3:

Register	Offset	Description
REIR5JRa	1_0E14h + (a × 1_0000h)	(used by JR a)

14.150.2 Function

See the description for [Recoverable Error Interrupt Record 0 for Job Ring a \(REIR0JR0 - REIR0JR3\)](#).

14.150.3 Diagram



14.150.4 Fields

Field	Function
31-25 —	Reserved
24 SAFE	For errors of REIR0JR[TYPE] = 00b SAFE indicates whether the AXI transaction associated with the recoverable error was a "safe" transaction.
23-20 —	Reserved
19-16 BID	This field holds the block identifier (see Table 14-1) of the source of the AXI transaction associated with the recoverable error.
15-0 —	Reserved

14.151 AIOP Interface Configuration Register (AICFG)

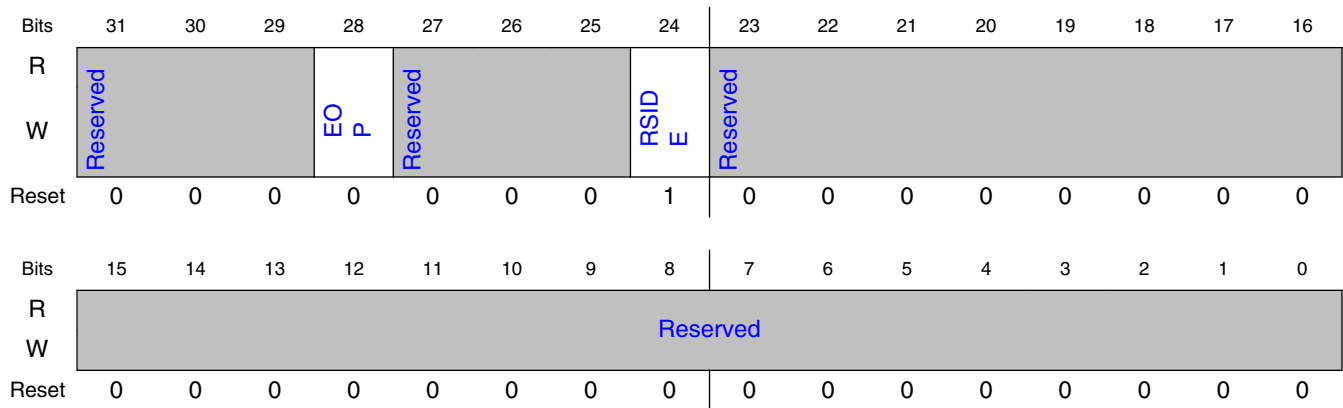
14.151.1 Offset

Register	Offset
AICFG	5_0000h

14.151.2 Function

Some characteristics of the AIOP Interface are controlled via the AIOP Interface Configuration Register.

14.151.3 Diagram



14.151.4 Fields

Field	Function
31-29 —	Reserved
28 EOP	Enable Order Preservation. If EOP=0 the dequeue order of jobs using the same Flow Context (FLC) address is not preserved when the result responses are enqueued. Note that this is the default setting. If EOP=1 the dequeue order of jobs using the same Flow Context (FLC) address is preserved.
27-25 —	Reserved
24 RSIDE	Report SEC ID on Error. When RSIDE=1 and the job termination status is non-zero, the 32-bit SEC error status encoded in the FRC field of the enqueued Frame Descriptor is reformatted as follows:

Table continues on the next page...

Field	Function
	Bits 31:24 are moved to bits 23:16 (which were reserved). The SEC Source ID, 03h, is inserted in bits 30:24. If the job termination status indicates an error or a warning, bit 31 is 1, otherwise bit 31 is 0.
23-0 —	Reserved.

14.152 AIOP Interface Control Register (AICTL)

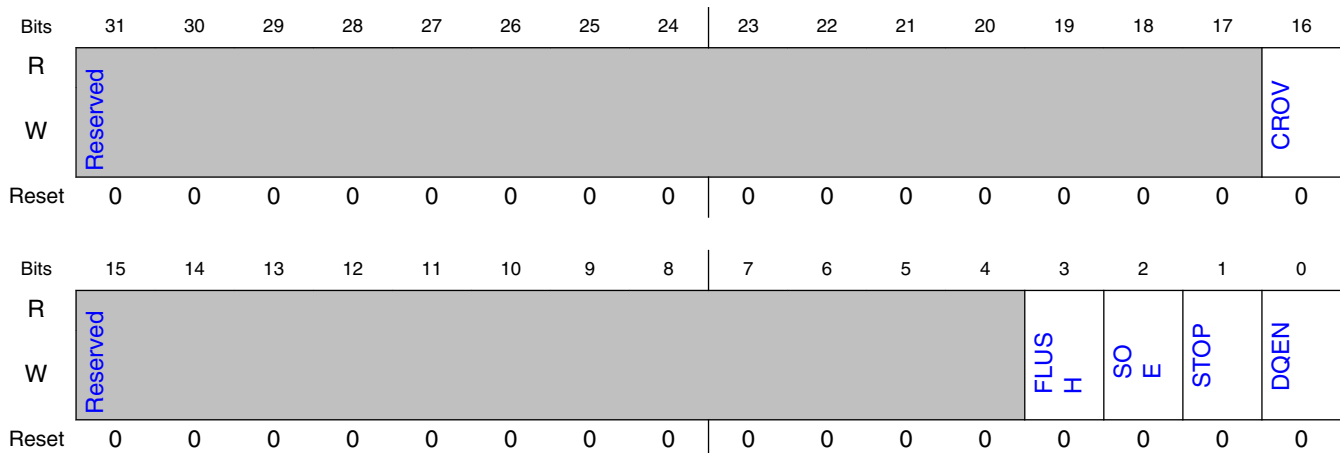
14.152.1 Offset

Register	Offset
AICTL	5_0004h

14.152.2 Function

The AIOP Interface can be stopped and started, and certain other features of the AIOP Interface can be configured using this register.

14.152.3 Diagram



14.152.4 Fields

Field	Function
31-17 —	Reserved
16 CROV	Critical Resource Override. If set, critical resource loading is not considered when selecting the next job for transfer to the job queue controller for job scheduling. If clear, critical resource loading may affect job selection priority. If the Flow Context associated with a job specifies a critical resource ID and all instances of that resource are in use by jobs in other flows that were already transferred to the job queue controller, the job is assigned the lowest selection priority.
15-4 —	Reserved
3 FLUSH	Flush the AI jobs. When FLUSH is set, AI asserts a signal that tells the DECOs to terminate any jobs currently executing with a DNR error status. DQEN should be set to 0 to disable dequeues. When all AI jobs have been enqueued, FLUSH is automatically reset to 0. If FLUSH is cleared by the user before some jobs are terminated, those jobs will finish normally.
2 SOE	Stop on error. If set, an AI-detected error will cause AI to stop all operations, including frame dequeue and enqueue operations and transfer of jobs to the job queue controller. When internal operations reach a stable, stopped state, STOPDOE and STOPD in the AI Status Register will be 1 to indicate that AI is stopped due to an error. See the AIOP Interface Status Register (see Section AIOP Interface Status Register (AISTA)) for a description of the errors. If the REI registers are programmed to halt SEC after a recoverable error and that recoverable error occurs, this will cause the DEBUGCTL[STOP] bit to assert. SEC will acknowledge that the stop is complete by setting the DEBUGCTL[STOP_ACK] bit. The DEBUGCTL[STOP] bit must be cleared in order to restart SEC.
1 STOP	Stop. Write 1 to STOP to direct AI to stop all AI operations. When internal operations reach a stable, stopped state, STOPD in the AI Status Register will be 1. Writing a 1 a second time (with DQEN = 0) will flush all jobs and reset AI. AISTA[STOPD] will be 0 while the AI jobs are flushed. When all AI jobs have been enqueued AI will reset itself, restoring all registers to their default/reset state, including resetting AICTL[STOP] to 0.
0 DQEN	Dequeue enable. If set, the AI will dequeue frames, if available, from the AIOP.

14.153 AIOP Interface Status Register (AISTA)

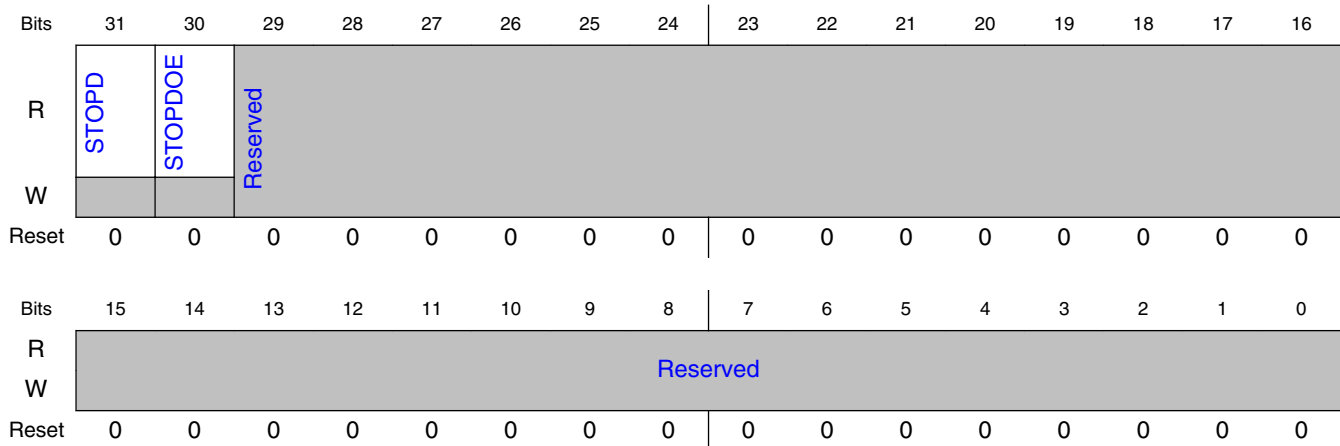
14.153.1 Offset

Register	Offset
AISTA	5_000Ch

14.153.2 Function

Software can determine the current status of the AIOP Interface by reading the AIOP Interface Status Register.

14.153.3 Diagram



14.153.4 Fields

Field	Function
31 STOPD	Stopped. Frame dequeue and enqueue operations and transfer of jobs to the Job Queue Controller has stopped. This condition will occur after the AICTL Register STP bit is set or after an error bit is set with the AICTL Register SOE bit set. There may be a delay of several cycles from the time the stopping event occurs until STOPD is asserted, while the AIOP Interface state machine finishes current operations.
30 STOPDOE	Stopped on Error. STOPDOE is asserted when AI is stopped because AICTL[SOE] is set and error status is non-zero for one or more jobs. It will be cleared when AICTL[SOE] is reset or there are no jobs with non-zero error status.
29-0 —	Reserved

14.154 AIOP Interface Dequeue Event Filter Control Register 0 (ADQEFC0)

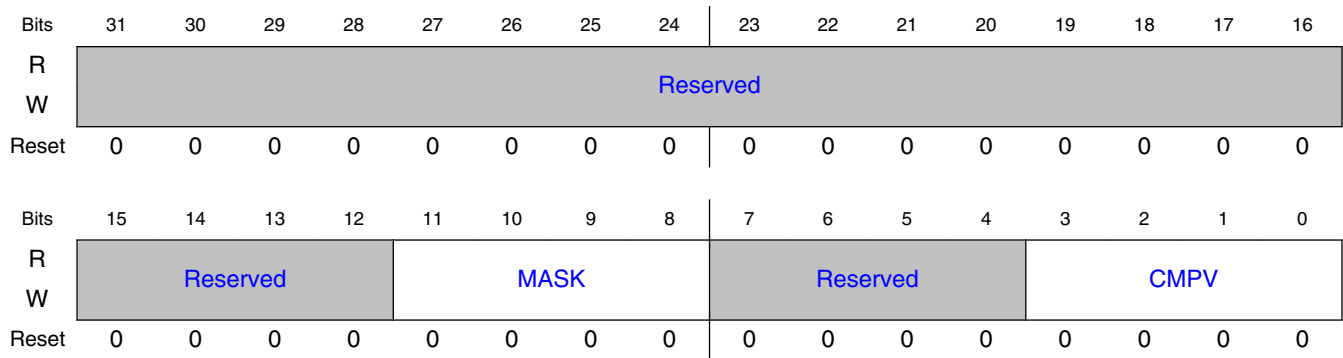
14.154.1 Offset

Register	Offset
ADQEFC0	5_0030h

14.154.2 Function

AIOP Interface Dequeue Event Filter Control Register 0. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 0.

14.154.3 Diagram



14.154.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison

Table continues on the next page...

Field	Function
	1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.155 AIOP Interface Dequeue Event Filter Control Register 1 (ADQEFC1)

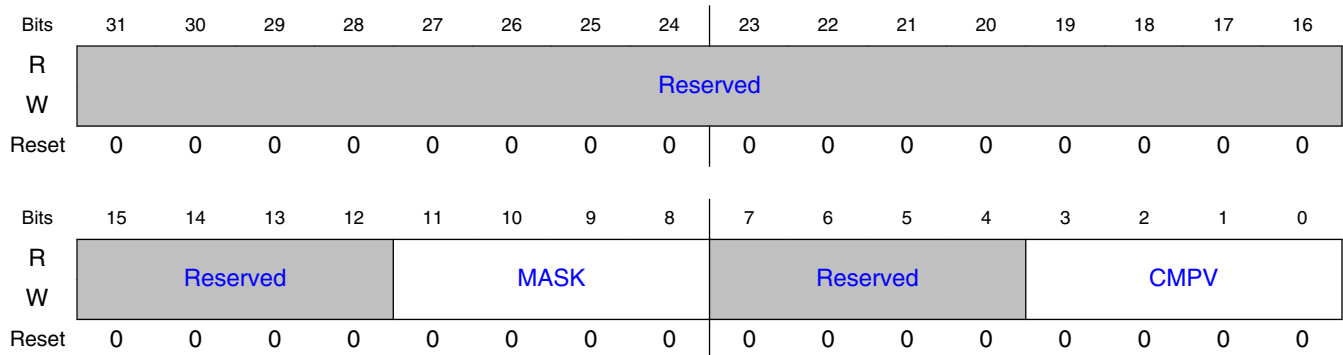
14.155.1 Offset

Register	Offset
ADQEFC1	5_0034h

14.155.2 Function

AIOP Interface Dequeue Event Filter Control Register 1. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 1.

14.155.3 Diagram



14.155.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.156 AIOP Interface Dequeue Event Filter Control Register 2 (ADQEFC2)

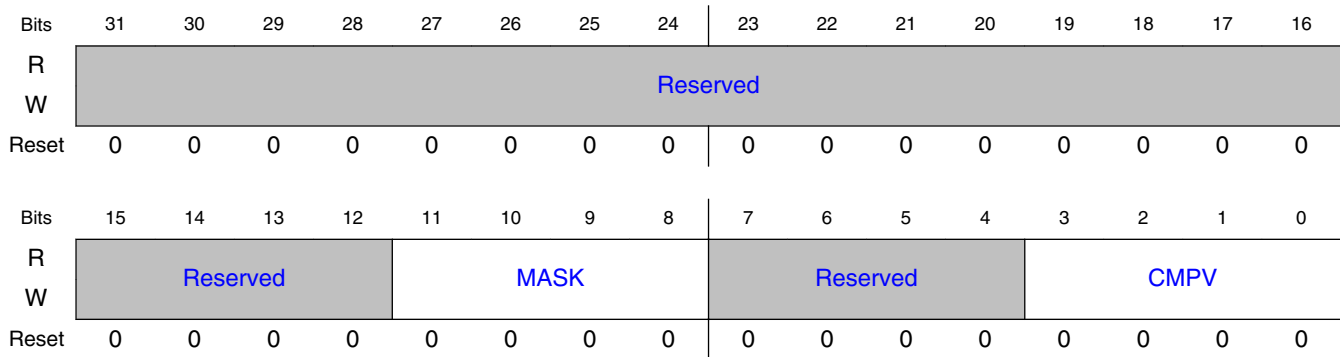
14.156.1 Offset

Register	Offset
ADQEFC2	5_0038h

14.156.2 Function

AIOP Interface Dequeue Event Filter Control Register 2. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 2.

14.156.3 Diagram



14.156.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.157 AIOP Interface Enqueue Event Filter Control Register 0 (AEQEFC0)

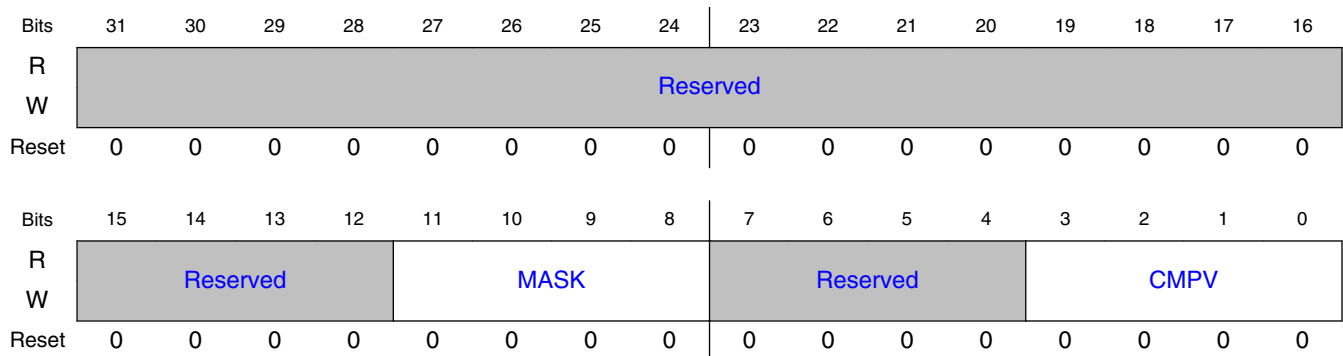
14.157.1 Offset

Register	Offset
AEQEFC0	5_0050h

14.157.2 Function

AIOP Interface Enqueue Event Filter Control Register 0. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 4.

14.157.3 Diagram



14.157.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame's current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame's Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.158 AIOP Interface Enqueue Event Filter Control Register 1 (AEQEFC1)

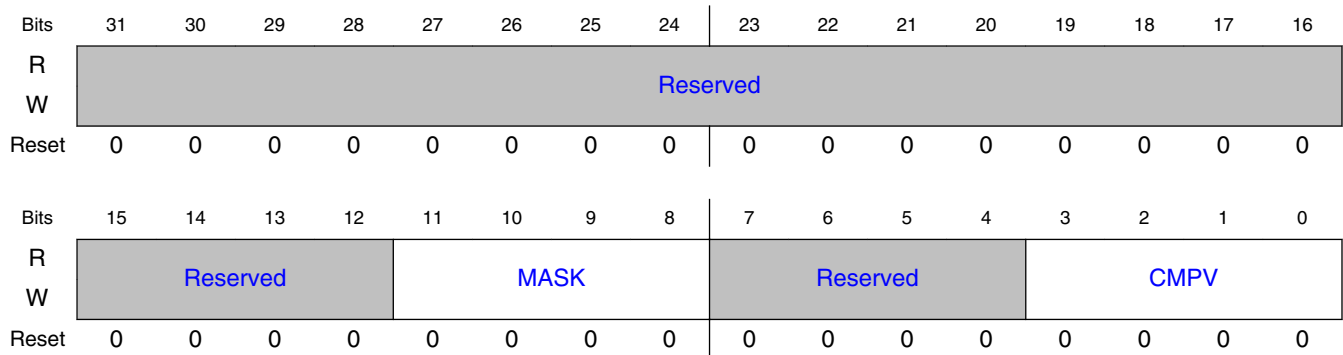
14.158.1 Offset

Register	Offset
AEQEFC1	5_0054h

14.158.2 Function

AIOP Interface Enqueue Event Filter Control Register 1. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 4.

14.158.3 Diagram



14.158.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.159 AIOP Interface Enqueue Event Filter Control Register 2 (AEQEFC2)

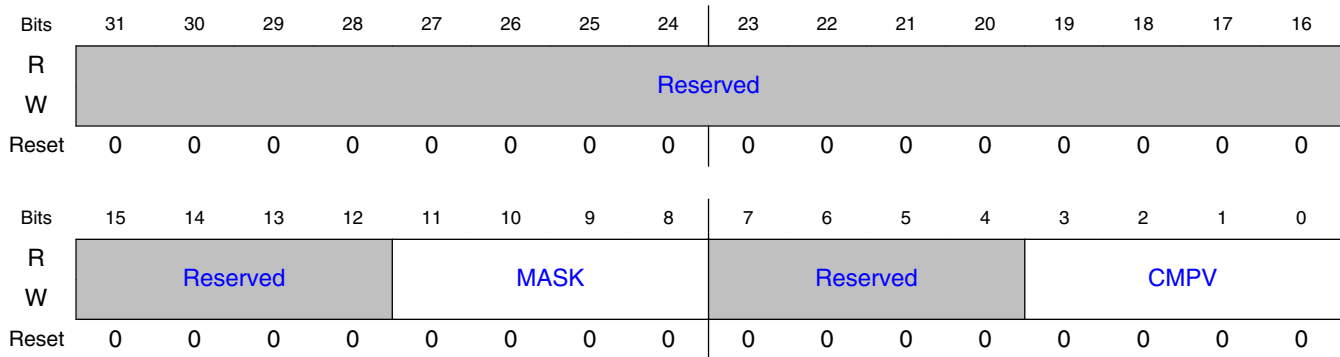
14.159.1 Offset

Register	Offset
AEQEFC2	5_0058h

14.159.2 Function

AIOP Interface Enqueue Event Filter Control Register 2. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on AI Performance Monitor event bus bit 5.

14.159.3 Diagram



14.159.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.160 Jobs in Use Register for AIOP Interface (JOBS_IN_USE_AI)

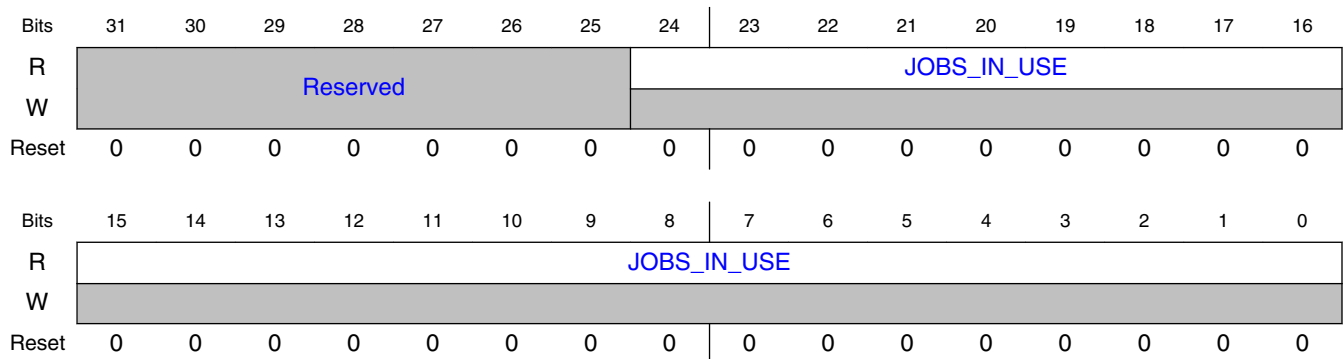
14.160.1 Offset

Register	Offset
JOBS_IN_USE_AI	5_0500h

14.160.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_IN_USE_AI register indicates whether certain job buffers are in use for active AI jobs. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.160.3 Diagram



14.160.4 Fields

Field	Function
31-25	Reserved

Table continues on the next page...

Field	Function
—	
24-0 JOBS_IN_USE	The bits in this field indicate whether certain job buffers are in use for active AI jobs. Bit n indicates the active/inactive status of job buffer n.) Bit n=0 means the job buffer is not in use. Bit n=1 means there is an active job in the job buffer. Note that there is no job buffer 0.

14.161 Jobs Ready Register for AIOP Interface (JOBS_READY_AI)

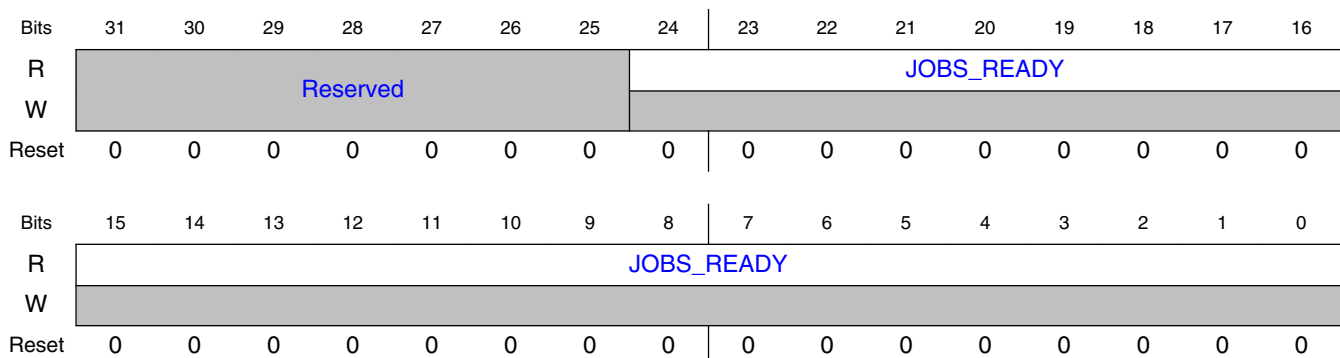
14.161.1 Offset

Register	Offset
JOBS_READY_AI	5_0508h

14.161.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_READY_AI register indicates whether certain AI job buffers contain jobs that are ready for transfer to the job queue controller. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.161.3 Diagram



14.161.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_READY	The bits in this field indicate whether certain job buffers hold AI jobs that are ready for transfer to the job queue controller. Bit n indicates the job-ready status of job buffer n.) Bit n=1 means that the job in the job buffer is ready for transfer. Bit n=0 means that the job is not ready or the job buffer is not in use. Note that there is no job buffer 0.

14.162 Jobs Transfer Blocking Disabled Register for AIOP Interface (JOBS_XFR_BLK_DIS_AI)

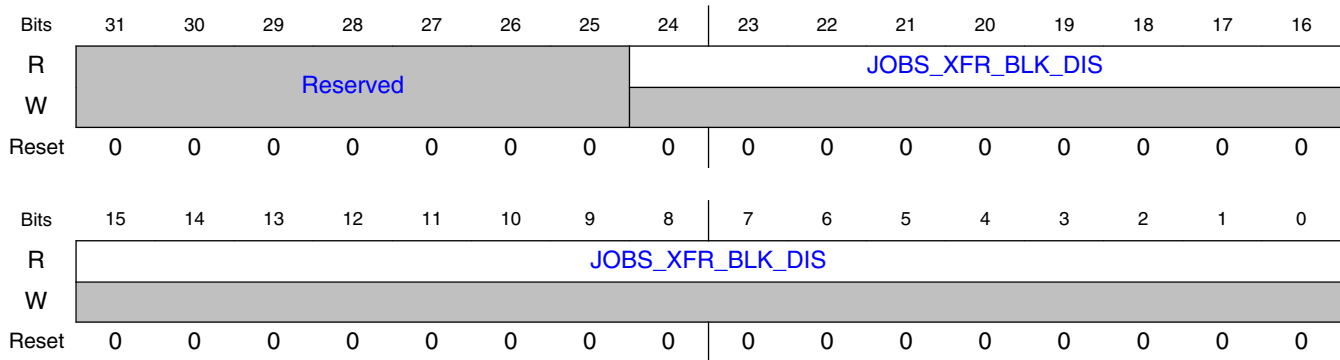
14.162.1 Offset

Register	Offset
JOBS_XFR_BLK_DIS_AI	5_0510h

14.162.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_XFR_BLK_DIS_AI register indicates whether certain AI job buffers contain jobs that have transfer blocking disabled. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.162.3 Diagram



14.162.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_XFR_BLK_DIS	The bits in this field indicate whether certain AI job buffers hold jobs with transfer blocking disabled. Bit n indicates the transfer-blocking-disabled status of job buffer n.) Bit n=1 means that the job in the job buffer has transfer blocking disabled. Bit n=0 means that the job does not have transfer blocking disabled or the job buffer is not in use. Note that there is no job buffer 0.

14.163 Jobs Transferred Register for AIOP Interface (JOBS_XFRD_AI)

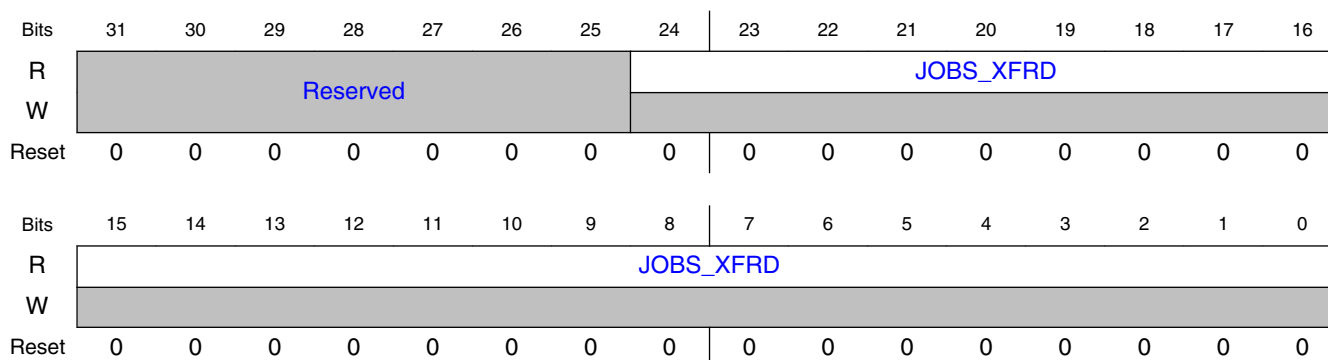
14.163.1 Offset

Register	Offset
JOBS_XFRD_AI	5_0518h

14.163.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_XFRD_AI register indicates whether certain AI job buffers contain jobs that have been transferred to the job queue controller. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.163.3 Diagram



14.163.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_XFRD	The bits in this field indicate whether certain job buffers hold AI jobs that have been transferred to the job controller. Bit n indicates the transferred status of job buffer n.) Bit n=1 means that the job in the job buffer has been transferred. Bit n=0 means that the job has not been transferred or the job buffer is not in use. Note that there is no job buffer 0.

14.164 Jobs Executing Register for AIOP Interface (JOBS_EXEC_AI)

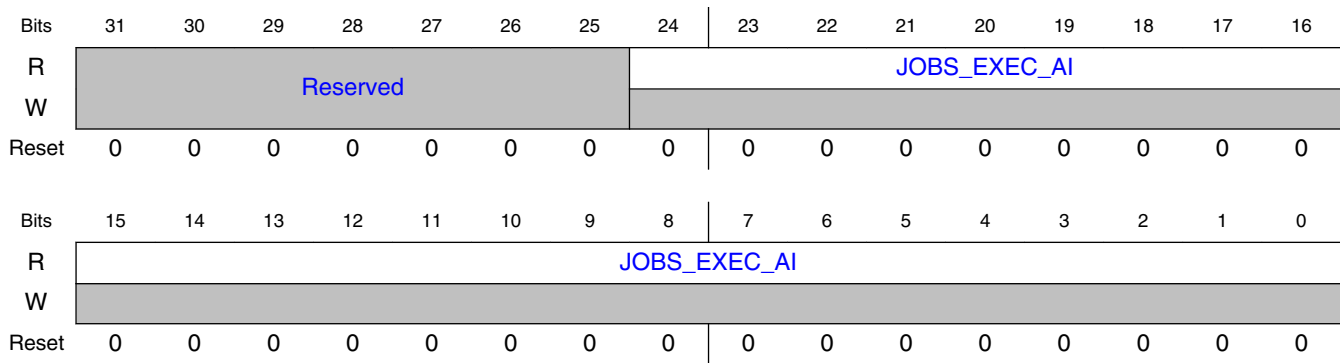
14.164.1 Offset

Register	Offset
JOBS_EXEC_AI	5_0520h

14.164.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_EXEC_AI register indicates whether certain AI job buffers contain jobs that are currently executing in a DECO. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.164.3 Diagram



14.164.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_EXEC_AI	The bits in this field indicate whether certain job buffers hold AI jobs that are currently executing in a DECO. Bit n indicates the executing status of job buffer n.) Bit n=1 means that the job in the job buffer is executing. Bit n=0 means that the job is not executing or the job buffer is not in use. Note that there is no job buffer 0.

14.165 Jobs Done Register for AIOP Interface (JOBS_DONE_AI)

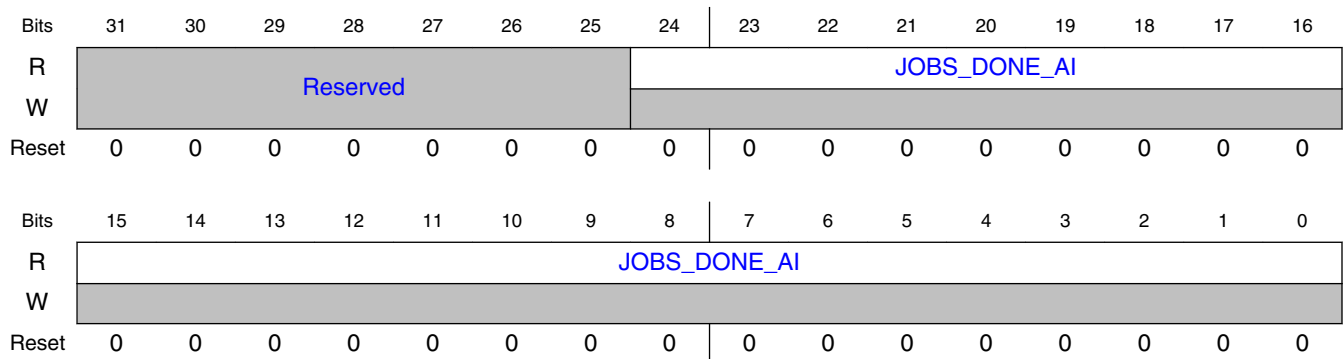
14.165.1 Offset

Register	Offset
JOBS_DONE_AI	5_0528h

14.165.2 Function

Data for each active job managed by AI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_DONE_AI register indicates whether certain AI job buffers contain jobs that are done executing. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.165.3 Diagram



14.165.4 Fields

Field	Function
31-25	Reserved

Table continues on the next page...

Field	Function
—	
24-0 JOBS_DONE_A I	The bits in this field indicate whether certain job buffers hold AI jobs that are done. Bit n indicates the done status of job buffer n.) Bit n=1 means that the job in the job buffer is done. Bit n=0 means that the job is not done or the job buffer is not in use. Note that there is no job buffer 0.

14.166 Job Select Register for AIOP Interface (JOB_SELECT_AI)

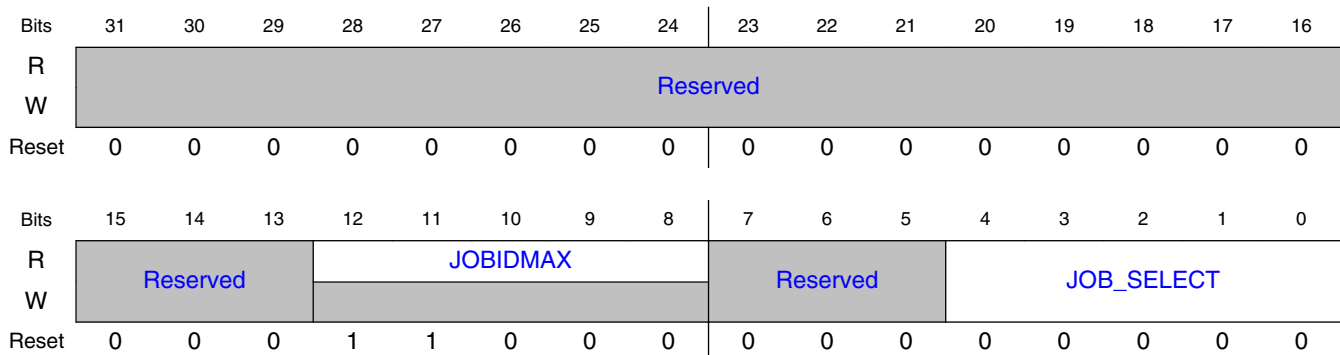
14.166.1 Offset

Register	Offset
JOB_SELECT_AI	5_0570h

14.166.2 Function

This register is used to specify the job for which status is displayed in the AI Job Data Registers. For example, to get the address of the dequeued FD for AI job n, write n to the JOB_SELECT field of this register and then read the Job Data Registers 9 and 10. This register also contains a read-only field that defines the maximum job ID for AI jobs.

14.166.3 Diagram



14.166.4 Fields

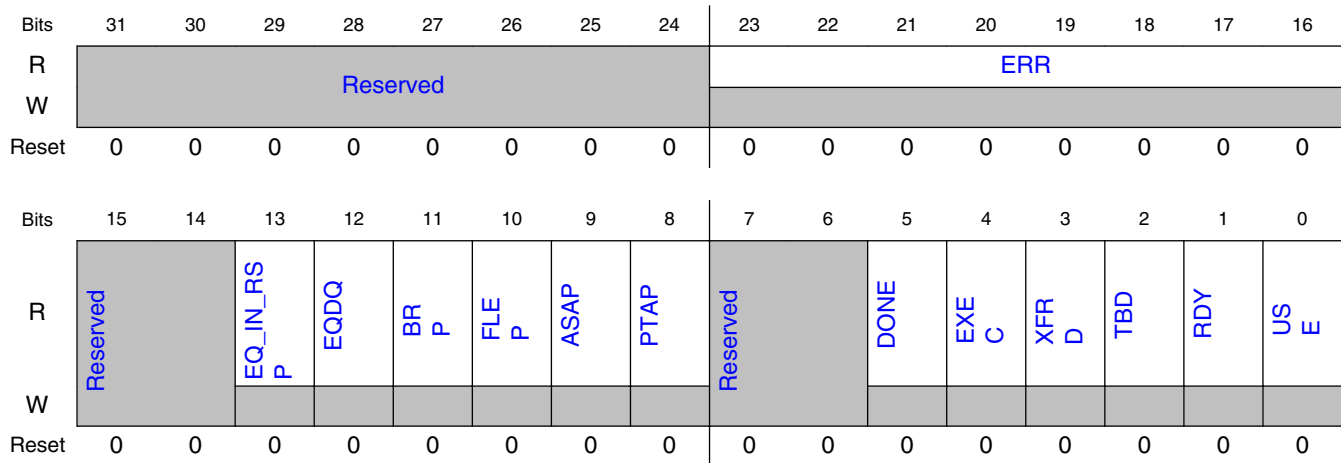
Field	Function
31-13 —	Reserved.
12-8 JOBIDMAX	JOBIDMAX specifies the maximum job ID for AI jobs.
7-5 —	Reserved.
4-0 JOB_SELECT	Bits 0 through 4 select the job for which data is displayed in the AI Job Data Registers. As jobs are processed, the contents of these registers can change at any time. To get a consistent snapshot of all the data for a job, AI should be stopped when the Job Data Registers are accessed. This can be done by setting AICTL[STOP] and then waiting until AISTA[STOPD] is 1. Writing the value 0 or a value greater than JOBIDMAX to the JOB_SELECT field may cause garbage data to be displayed in the Job Data Registers.

14.167 AIOP Interface Job Data Register 0 (AIJOBDO)

14.167.1 Offset

Register	Offset	Description
AIJOBDO	5_0574h	This register is used for debugging AIOP Interface jobs. Registers AIJOBDO .. AIJOBDO33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.167.2 Diagram



14.167.3 Fields

Field	Function
31-24 —	Reserved.
23-16 ERR	If this field is 00h, no error has occurred for this job. If the field is not 00h, this field contains an error code for this job. The codes in this field have the same definition as the least-significant 8 bits of Job termination status word, Source 5h (QI) .
15-14 —	Reserved.
13 EQ_IN_RSP	If 1, enqueue frame using input partition response parameters for this job.
12 EQDQ	If 1, enqueued dequeued frame for this job.
11 BRP	If 1, a buffer release is pending for this job.
10 FLEP	If 1, an FLE write is pending for this job.
9 ASAP	If 1, an ASA write is pending for this job.
8 PTAP	If 1, the PTA copy is pending for this job.
7-6 —	Reserved.
5	If 1, this job is done.

Table continues on the next page...

AIOP Interface Job Data Register 1 (AIJOB1)

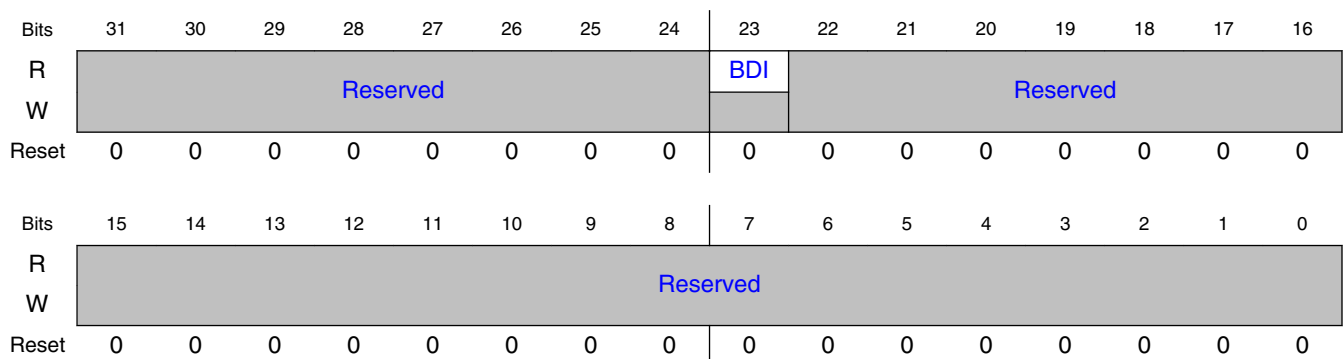
Field	Function
DONE	
4 EXEC	If 1, this job is executing.
3 XFRD	If 1, this job has been transferred to the Job Controller.
2 TBD	If 1, this job has transfer blocking disabled.
1 RDY	If 1, this job is ready for transfer to the Job Controller.
0 USE	If 1, the selected AI job buffer is currently in use.

14.168 AIOP Interface Job Data Register 1 (AIJOB1)

14.168.1 Offset

Register	Offset	Description
AIJOB1	5_0578h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.168.2 Diagram



14.168.3 Fields

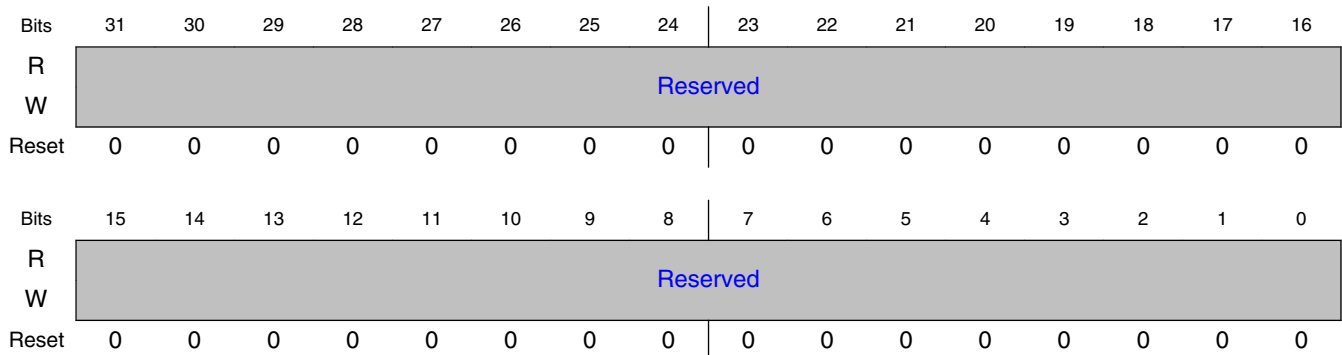
Field	Function
31-24 —	Reserved.
23 BDI	If 1, datapath isolation is bypassed for this job.
22-0 —	Reserved.

14.169 AIOP Interface Job Data Register 2 (AIJOB2)

14.169.1 Offset

Register	Offset	Description
AIJOB2	5_057Ch	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.169.2 Diagram



14.169.3 Fields

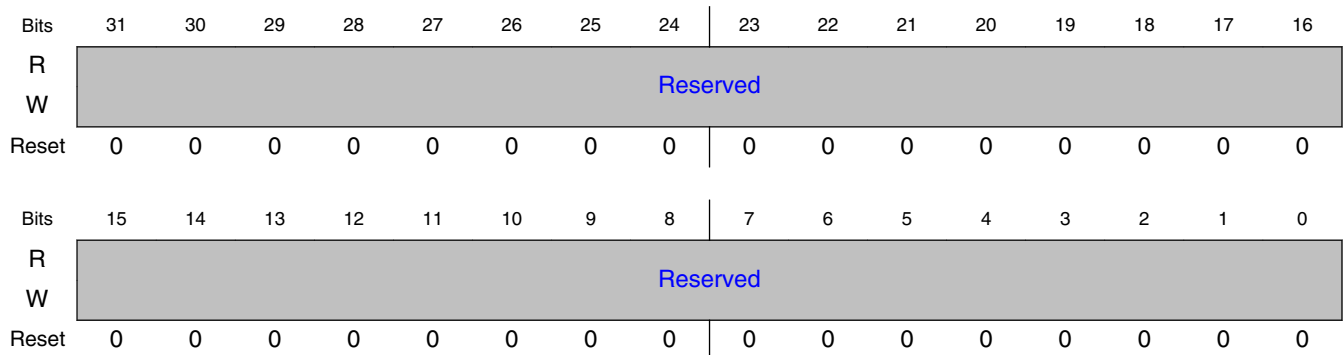
Field	Function
31-0 —	Reserved.

14.170 AIOP Interface Job Data Register 3 (AIJOBBD3)

14.170.1 Offset

Register	Offset	Description
AIJOBBD3	5_0580h	This register is used for debugging AIOP Interface jobs. Registers AIJOBBD0 .. AIJOBBD33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.170.2 Diagram



14.170.3 Fields

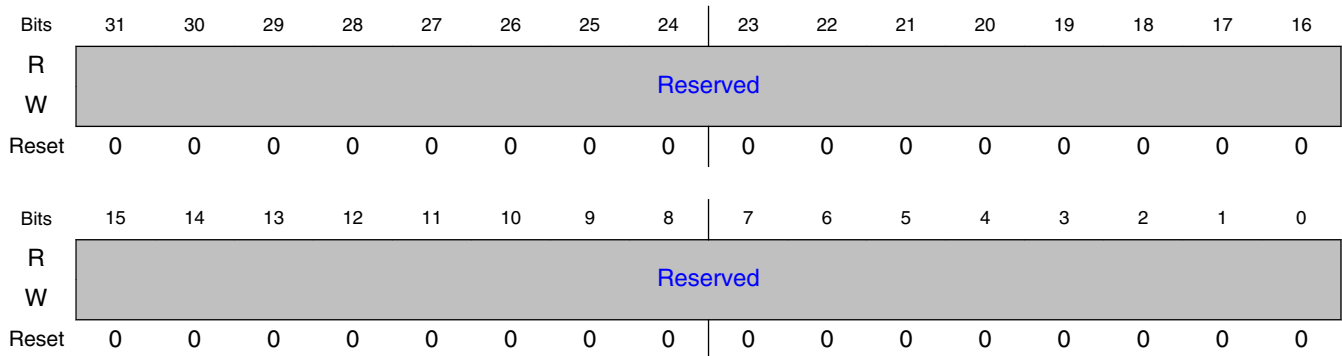
Field	Function
31-0 —	Reserved.

14.171 AIOP Interface Job Data Register 4 (AIJOB4)

14.171.1 Offset

Register	Offset	Description
AIJOB4	5_0584h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.171.2 Diagram



14.171.3 Fields

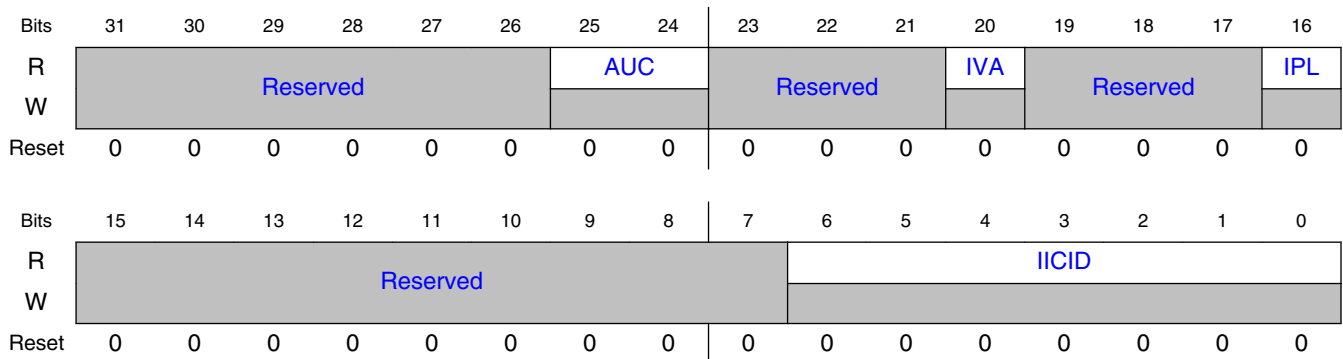
Field	Function
31-0	Reserved.
—	

14.172 AIOP Interface Job Data Register 5 (AIJOB5)

14.172.1 Offset

Register	Offset	Description
AIJOB5	5_0588h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.172.2 Diagram



14.172.3 Fields

Field	Function
31-26 —	Reserved.
25-24 AUC	The Input Partition AMQ Usage Control for this job.
23-21 —	Reserved.
20 IVA	The Input Partition virtual address enable? for this job.
19-17 —	Reserved.
16 IPL	The Input Partition ICID for this job.
15-7 —	Reserved.

Table continues on the next page...

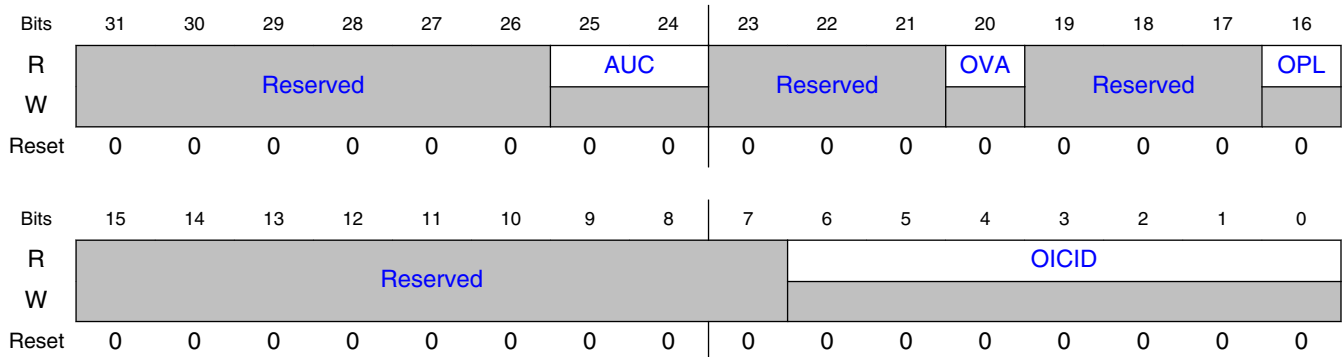
Field	Function
6-0 IICID	The ICID of the Input Partition for this job.

14.173 AIOP Interface Job Data Register 6 (AIJOB6)

14.173.1 Offset

Register	Offset	Description
AIJOB6	5_058Ch	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.173.2 Diagram



14.173.3 Fields

Field	Function
31-26 —	Reserved.
25-24 AUC	The Output Partition AMQ Usage Control for this job.
23-21	Reserved.

Table continues on the next page...

AIOP Interface Job Data Register 7 (AIJOB7)

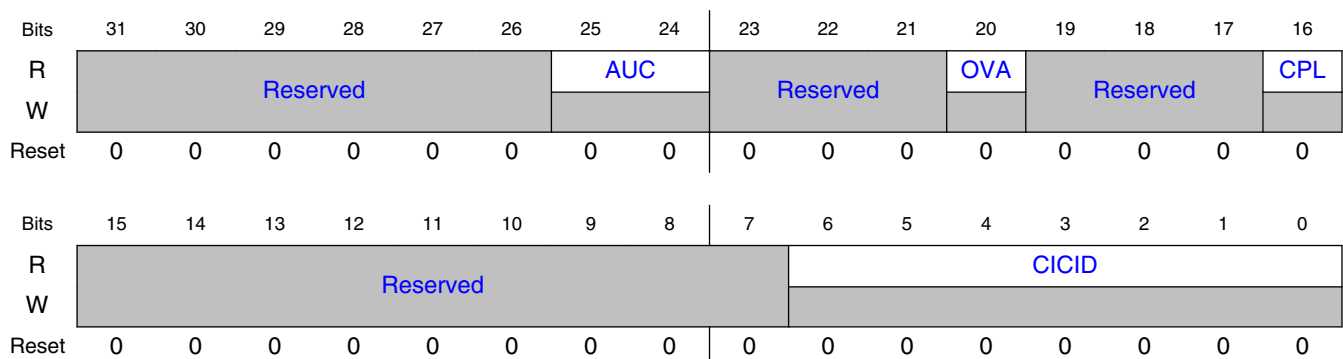
Field	Function
—	
20 OVA	The Output Partition virtual address enable? for this job.
19-17 —	Reserved.
16 OPL	The Output Partition Privilege Level for this job.
15-7 —	Reserved.
6-0 OICID	The Output Partition ICID for this job.

14.174 AIOP Interface Job Data Register 7 (AIJOB7)

14.174.1 Offset

Register	Offset	Description
AIJOB7	5_0590h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.174.2 Diagram



14.174.3 Fields

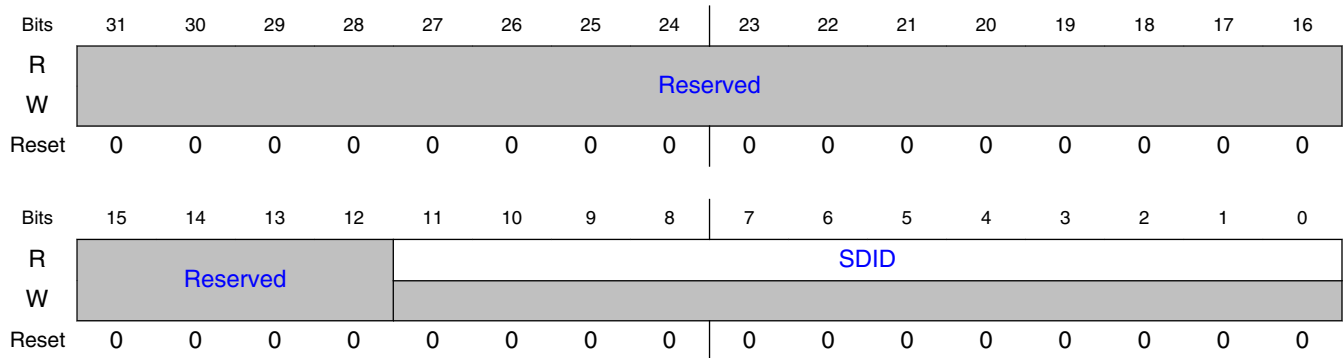
Field	Function
31-26 —	Reserved.
25-24 AUC	The Control Partition AMQ Usage Control for this job.
23-21 —	Reserved.
20 OVA	The Control Partition virtual address enable? for this job.
19-17 —	Reserved.
16 CPL	The Control Partition Privilege Level for this job.
15-7 —	Reserved.
6-0 CICID	The Control Partition ICID for this job.

14.175 AIOP Interface Job Data Register 8 (AIJOB8)

14.175.1 Offset

Register	Offset	Description
AIJOB8	5_0594h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.175.2 Diagram



14.175.3 Fields

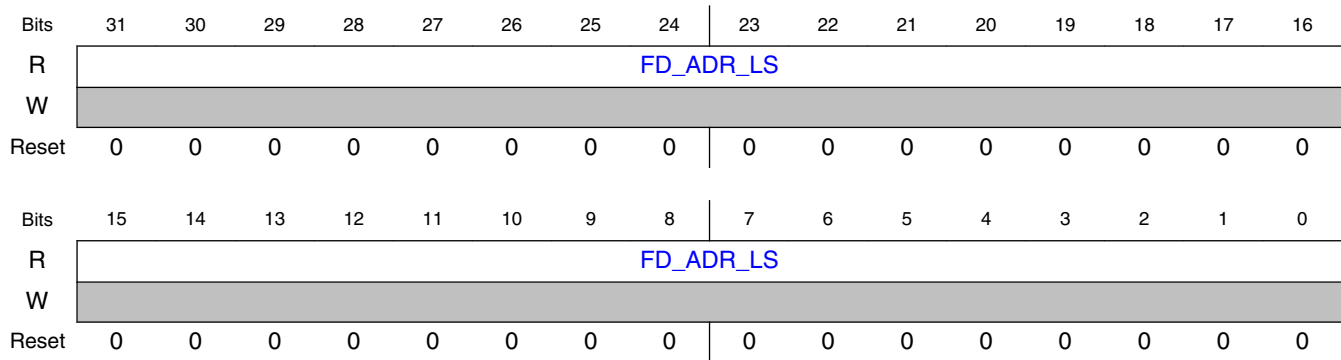
Field	Function
31-12 —	Reserved.
11-0 SDID	The Security Domain Identifier (SDID) for this job.

14.176 AIOP Interface Job Data Register 9 (AIJOB9)

14.176.1 Offset

Register	Offset	Description
AIJOB9	5_0598h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.176.2 Diagram



14.176.3 Fields

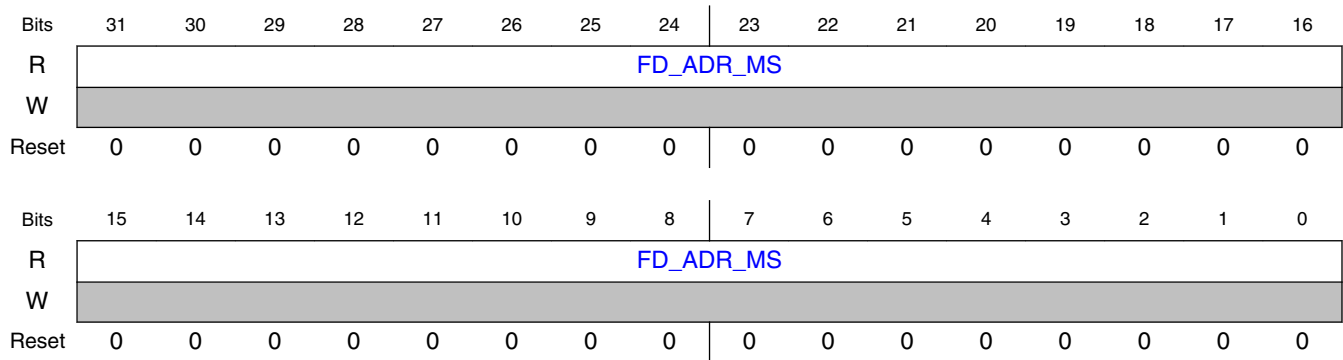
Field	Function
31-0 FD_ADR_LS	The least-significant 32 bits of the address of the dequeued frame for this job.

14.177 AIOP Interface Job Data Register 10 (AIJOB10)

14.177.1 Offset

Register	Offset	Description
AIJOB10	5_059Ch	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.177.2 Diagram



14.177.3 Fields

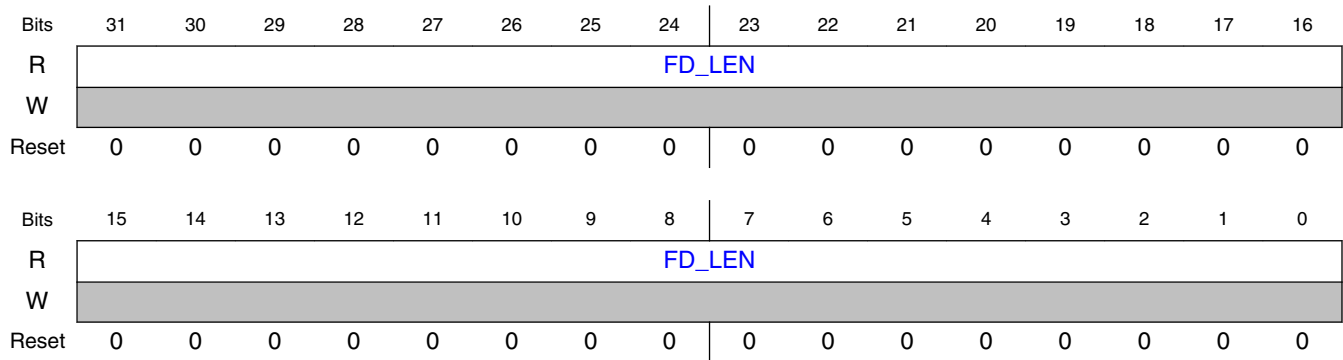
Field	Function
31-0 FD_ADR_MS	The most-significant 32 bits of the address of the dequeued frame for this job.

14.178 AIOP Interface Job Data Register 11 (AIJOB11)

14.178.1 Offset

Register	Offset	Description
AIJOB11	5_05A0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.178.2 Diagram



14.178.3 Fields

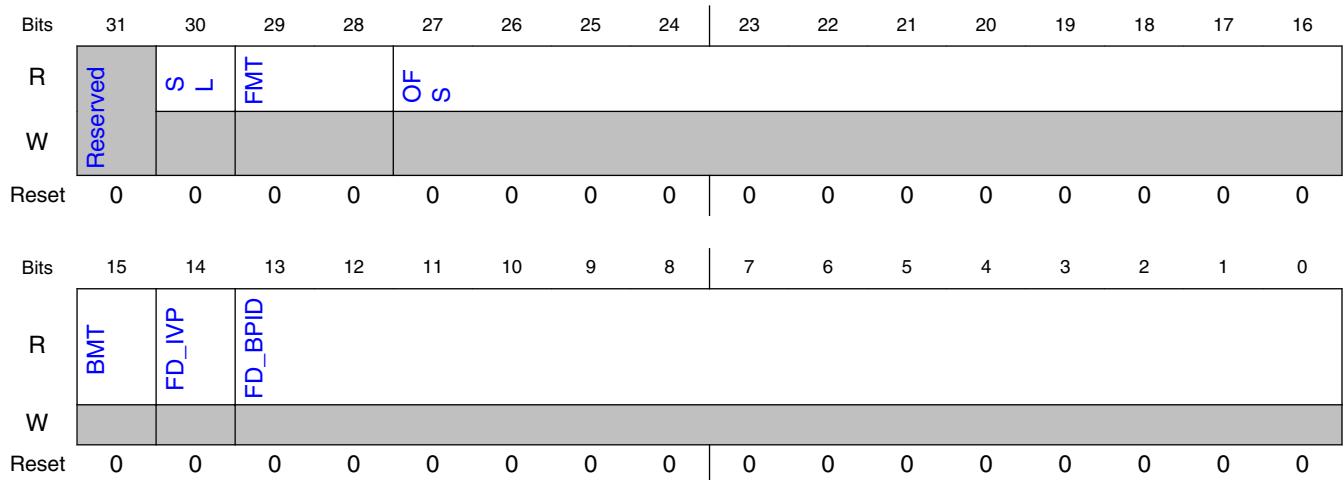
Field	Function
31-0 FD_LEN	The length of the dequeued frame for this job.

14.179 AIOP Interface Job Data Register 12 (AIJOB12)

14.179.1 Offset

Register	Offset	Description
AIJOB12	5_05A4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.179.2 Diagram



14.179.3 Fields

Field	Function
31 —	Reserved
30 SL	The length of the dequeued frame for this job is in short-length form.
29-28 FMT	The format of the dequeued frame for this job.
27-16 OFS	The data offset for the dequeued frame for this job.
15 BMT	If 1, memory address translation is bypassed for this job.
14 FD_IVP	If 1, the dequeued frame for this job is invalid.
13-0 FD_BPID	The Buffer Pool ID for this job.

14.180 AIOP Interface Job Data Register 13 (AIJOB13)

14.180.1 Offset

Register	Offset	Description
AIJOB13	5_05A8h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.180.2 Diagram

Bits	31	30	29	28	27	26	25	24		23	22	21	20	19	18	17	16
R	FD_FRC																
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0
R	FD_FRC																
W																	
Reset	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0

14.180.3 Fields

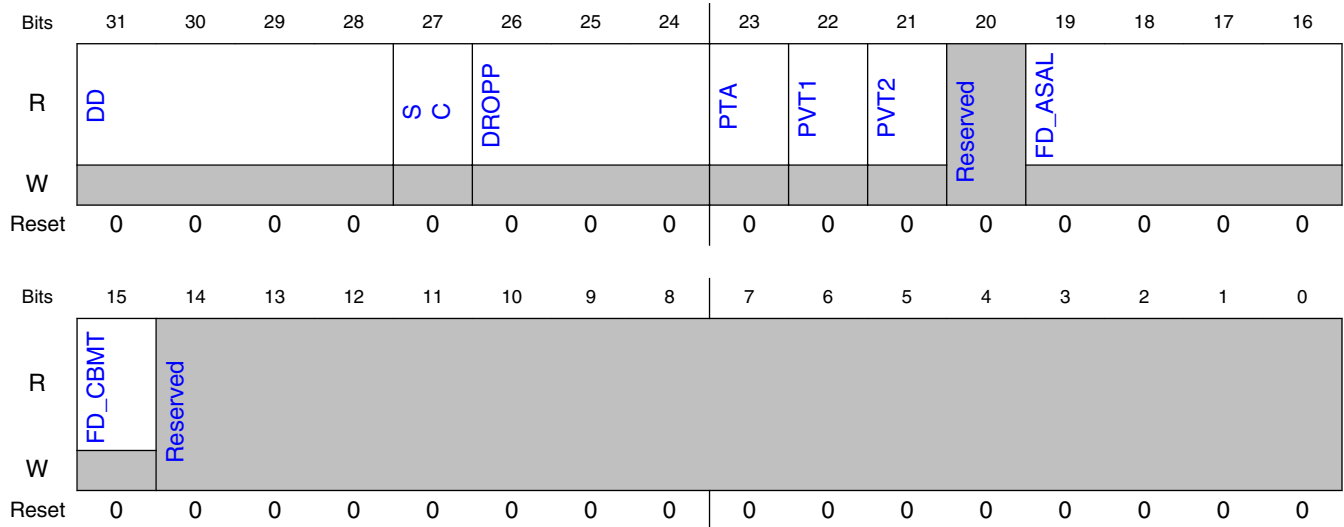
Field	Function
31-0 FD_FRC	The context for the dequeued frame for this job.

14.181 AIOP Interface Job Data Register 14 (AIJOB14)

14.181.1 Offset

Register	Offset	Description
AIJOB14	5_05ACh	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.181.2 Diagram



14.181.3 Fields

Field	Function
31-28 DD	The dynamic debug mark for the dequeued frame for this job.
27 SC	The stashing control for the dequeued frame for this job.
26-24 DROPP	The drop priority for the dequeued frame for this job.
23 PTA	Pass-Through Annotation for the dequeued frame for this job.
22 PVT1	If 1, the dequeued frame pass-through annotation 1 is valid for this job.
21 PVT2	If 1, the dequeued frame pass-through annotation 2 is valid for this job.
20 —	Reserved
19-16 FD_ASAL	The length of the dequeued frame accelerator-specific annotation.
15 FD_CBMT	If 1, bypass memory address translation for the dequeued frame flow context for this job.

Table continues on the next page...

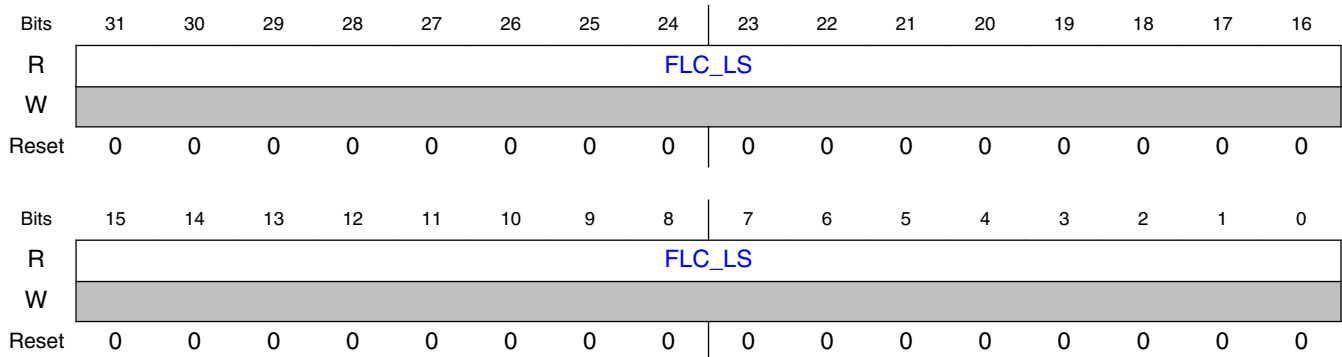
Field	Function
14-0 —	Reserved

14.182 AIOP Interface Job Data Register 15 (AIJOB15)

14.182.1 Offset

Register	Offset	Description
AIJOB15	5_05B0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.182.2 Diagram



14.182.3 Fields

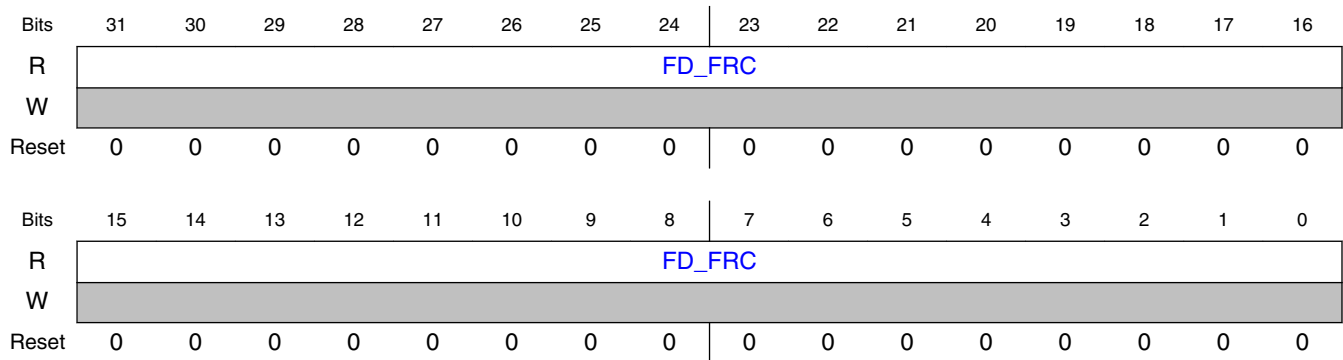
Field	Function
31-0 FLC_LS	The least-significant 32 bits of the flow context for the dequeued frame for this job.

14.183 AIOP Interface Job Data Register 16 (AIJOB16)

14.183.1 Offset

Register	Offset	Description
AIJOB16	5_05B4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.183.2 Diagram



14.183.3 Fields

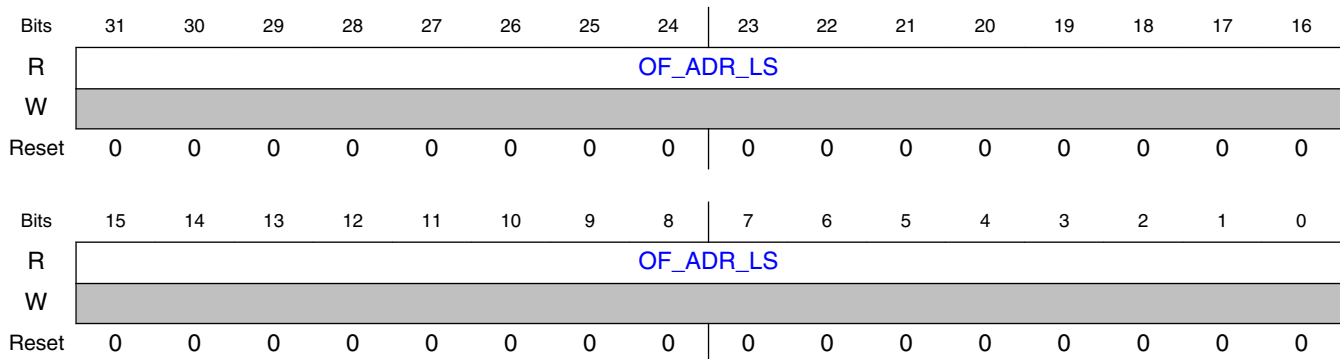
Field	Function
31-0 FD_FRC	The most-significant 32 bits of the flow context for the dequeued frame for this job.

14.184 AIOP Interface Job Data Register 17 (AIJOB17)

14.184.1 Offset

Register	Offset	Description
AIJOB17	5_05B8h	This register is used for debugging AIOF Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.184.2 Diagram



14.184.3 Fields

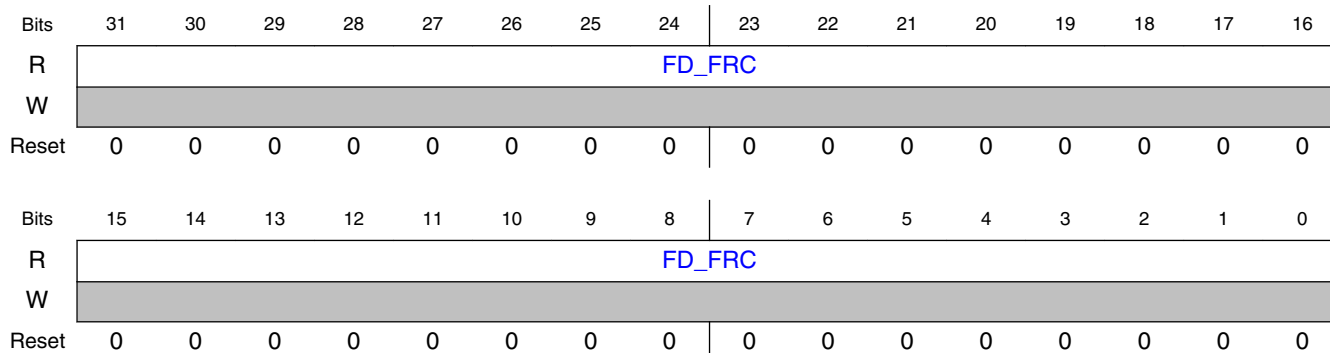
Field	Function
31-0 OF_ADR_LS	The least-significant 32 bits of the output frame address for this job.

14.185 AIOF Interface Job Data Register 18 (AIJOB18)

14.185.1 Offset

Register	Offset	Description
AIJOB18	5_05BCh	This register is used for debugging AIOF Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.185.2 Diagram



14.185.3 Fields

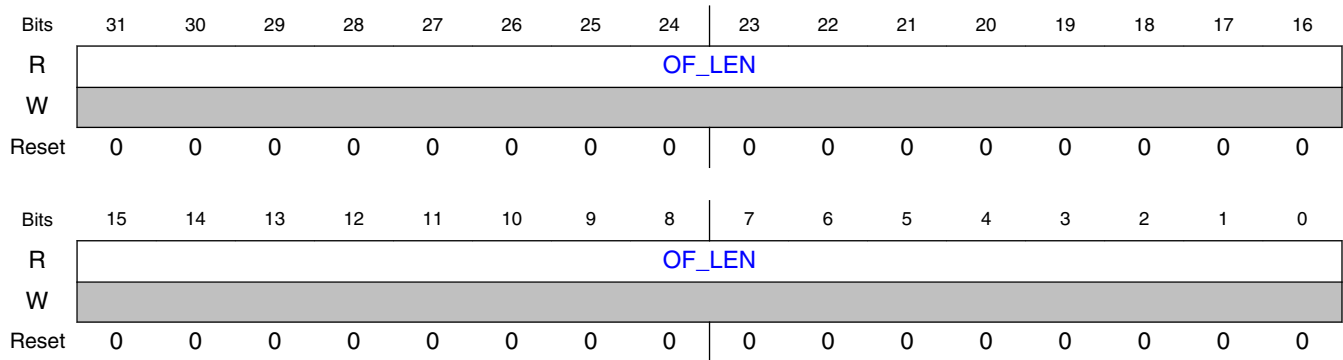
Field	Function
31-0 FD_FRC	The most-significant 32 bits of the output frame address for this job.

14.186 AIOP Interface Job Data Register 19 (AIJOB19)

14.186.1 Offset

Register	Offset	Description
AIJOB19	5_05C0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.186.2 Diagram



14.186.3 Fields

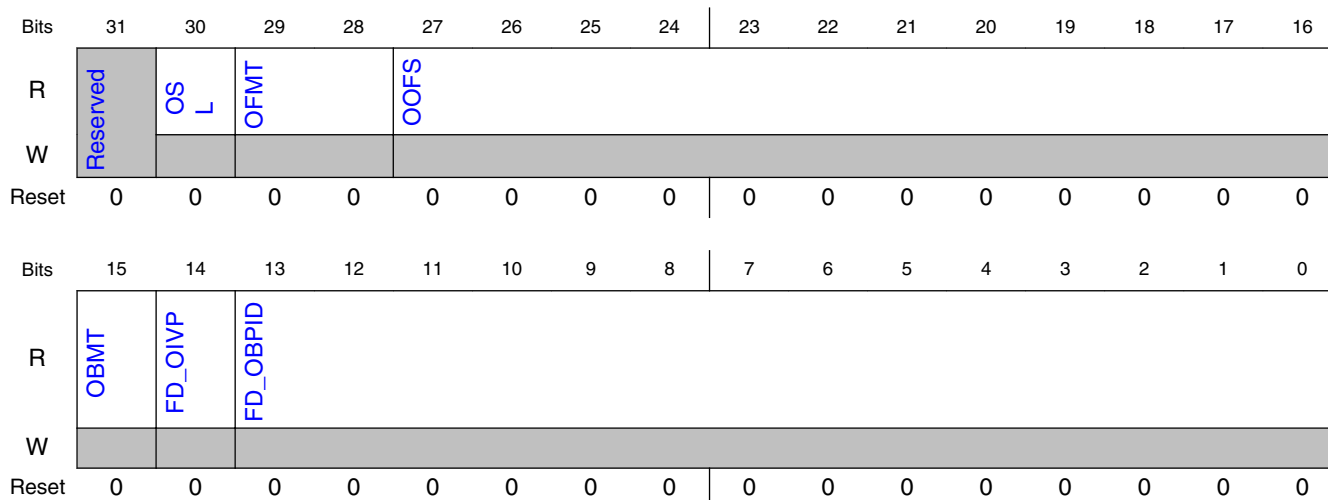
Field	Function
31-0 OF_LEN	The data length of the output frame for this job.

14.187 AIOP Interface Job Data Register 20 (AIJOB20)

14.187.1 Offset

Register	Offset	Description
AIJOB20	5_05C4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.187.2 Diagram



14.187.3 Fields

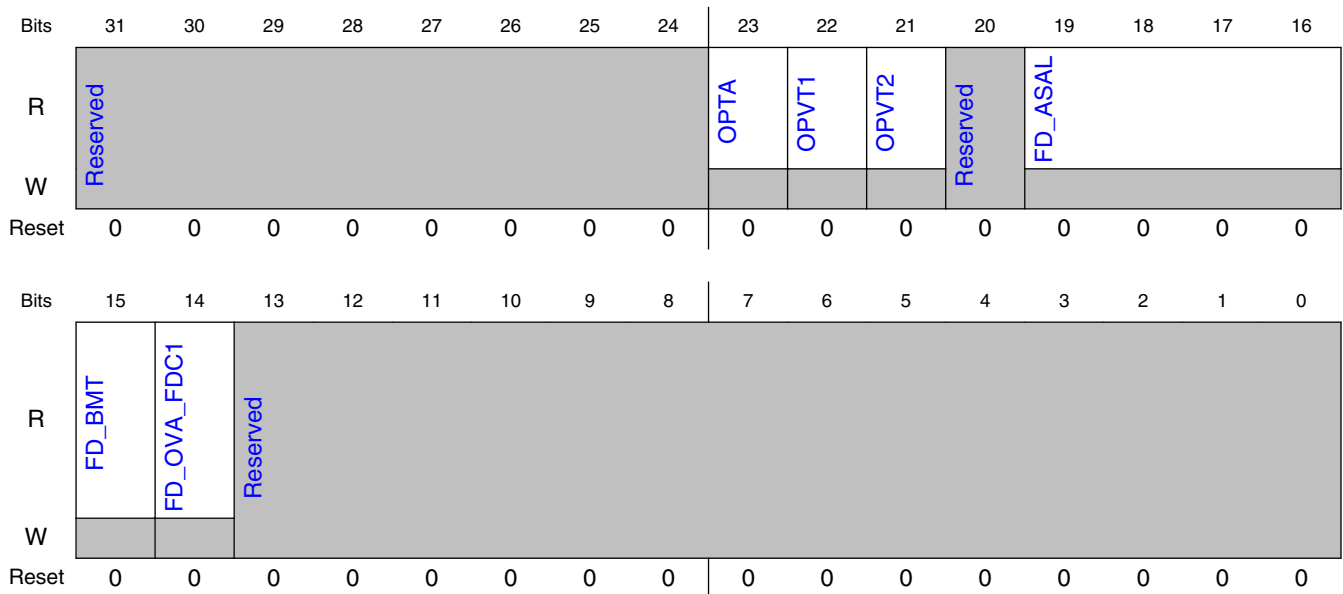
Field	Function
31 —	Reserved
30 OSL	The length of the output frame for this job is in short-length form.
29-28 OFMT	The format of the output frame for this job.
27-16 OOFS	The data offset for the output frame for this job.
15 OBMT	If 1, memory address translation is bypassed for the output frame for this job.
14 FD_OIVP	If 1, the output frame for this job is invalid.
13-0 FD_OBPID	The Output Frame Buffer Pool ID for this job.

14.188 AIOP Interface Job Data Register 21 (AIJOB21)

14.188.1 Offset

Register	Offset	Description
AIJOB21	5_05C8h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.188.2 Diagram



14.188.3 Fields

Field	Function
31-24 —	Reserved
23 OPTA	Pass-Through Annotation for the output frame for this job.
22 OPVT1	If 1, the output frame pass-through annotation 1 is valid for this job.
21 OPVT2	If 1, the output frame pass-through annotation 2 is valid for this job.
20	Reserved

Table continues on the next page...

AIOP Interface Job Data Register 22 (AIJOB22)

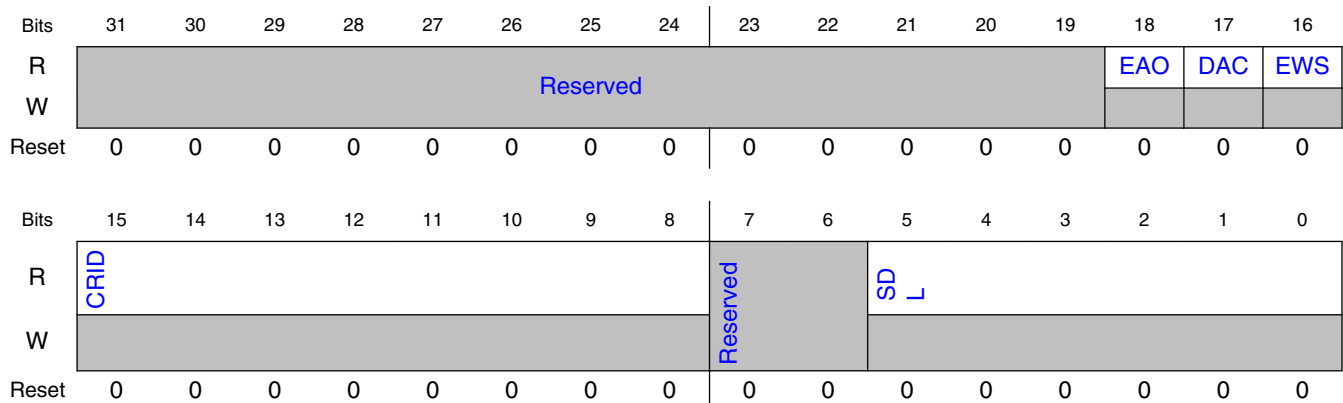
Field	Function
—	
19-16 FD_ASAL	The length of the accelerator-specific annotation.
15 FD_BMT	If 1, bypass memory address translation for this job.
14 FD_OVA_FDC1	If framelist format, this is the FL OF FD-compatibility field 1, otherwise this indicates that the output partition uses a virtual address.
13-0 —	Reserved

14.189 AIOP Interface Job Data Register 22 (AIJOB22)

14.189.1 Offset

Register	Offset	Description
AIJOB22	5_05CCh	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.189.2 Diagram



14.189.3 Fields

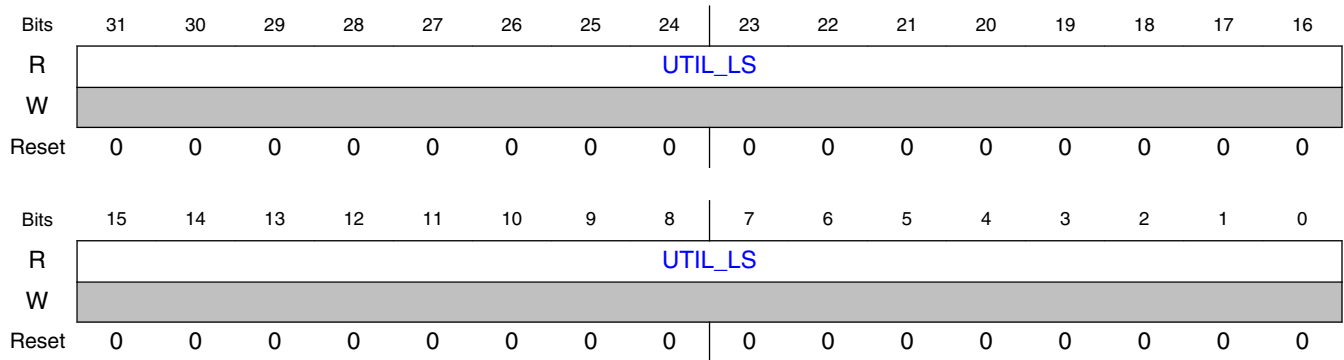
Field	Function
31-19 —	Reserved
18 EAO	If 1, annotation output is enabled for this job.
17 DAC	If 1, annotation (PTA) copy is disabled for this job.
16 EWS	If 1, AXI Write-Safe is enabled.
15-8 CRID	The identifier for a critical resource.
7-6 —	Reserved
5-0 SDL	The length of the shared descriptor.

14.190 AIOP Interface Job Data Register 23 (AIJOB23)

14.190.1 Offset

Register	Offset	Description
AIJOB23	5_05D0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.190.2 Diagram



14.190.3 Fields

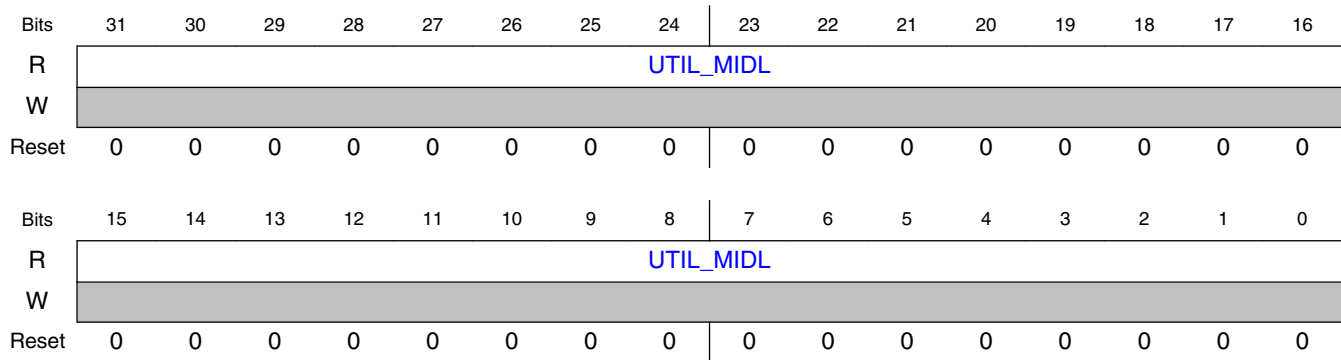
Field	Function
31-0 UTIL_LS	The least-significant 32 bits of the utility buffer for this job.

14.191 AIOP Interface Job Data Register 24 (AIJOB24)

14.191.1 Offset

Register	Offset	Description
AIJOB24	5_05D4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.191.2 Diagram



14.191.3 Fields

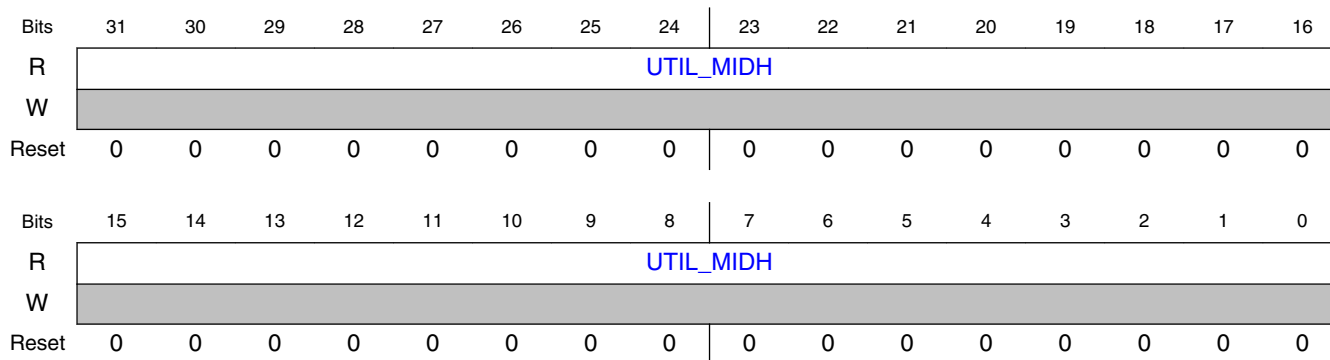
Field	Function
31-0 UTIL_MIDL	The next-to-least-significant 32 bits (63..32) of the utility buffer for this job.

14.192 AIOP Interface Job Data Register 25 (AIJOB25)

14.192.1 Offset

Register	Offset	Description
AIJOB25	5_05D8h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.192.2 Diagram



14.192.3 Fields

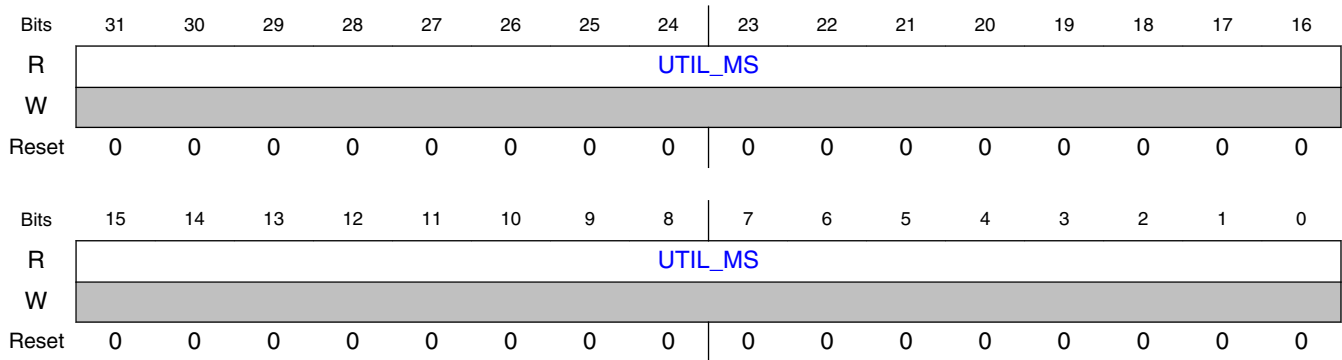
Field	Function
31-0 UTIL_MIDH	The next-to-most-significant 32 bits (95..64) of the utility buffer for this job.

14.193 AIOP Interface Job Data Register 26 (AIJOB26)

14.193.1 Offset

Register	Offset	Description
AIJOB26	5_05DCh	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.193.2 Diagram



14.193.3 Fields

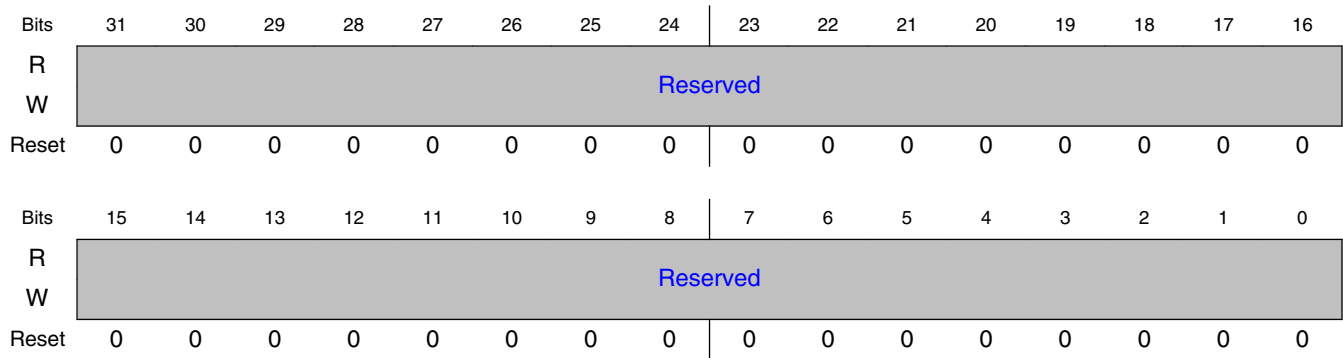
Field	Function
31-0 UTIL_MS	The most-significant 32 bits (127..96) of the utility buffer for this job.

14.194 AIOP Interface Job Data Register 27 (AIJOB27)

14.194.1 Offset

Register	Offset	Description
AIJOB27	5_05E0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.194.2 Diagram



14.194.3 Fields

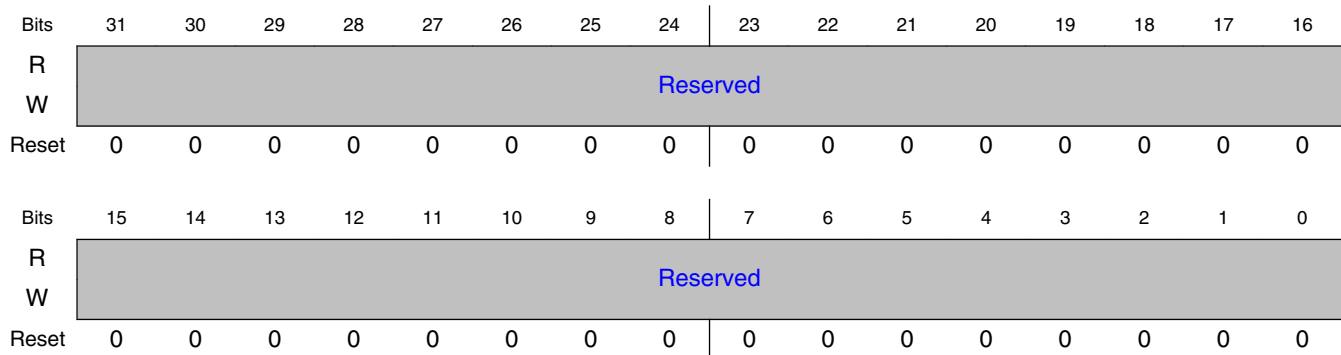
Field	Function
31-0	Reserved.
—	

14.195 AIOP Interface Job Data Register 28 (AIJOB28)

14.195.1 Offset

Register	Offset	Description
AIJOB28	5_05E4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.195.2 Diagram



14.195.3 Fields

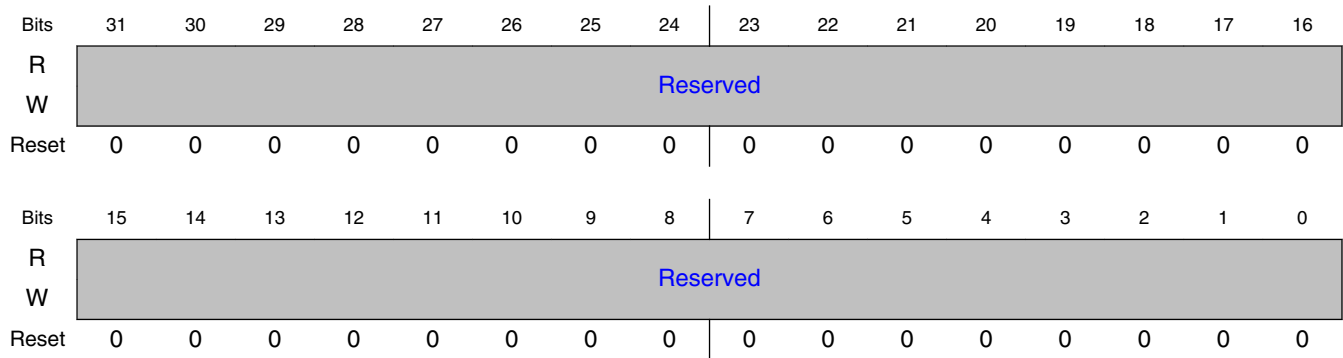
Field	Function
31-0	Reserved.
—	

14.196 AIOP Interface Job Data Register 29 (AIJOB29)

14.196.1 Offset

Register	Offset	Description
AIJOB29	5_05E8h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.196.2 Diagram



14.196.3 Fields

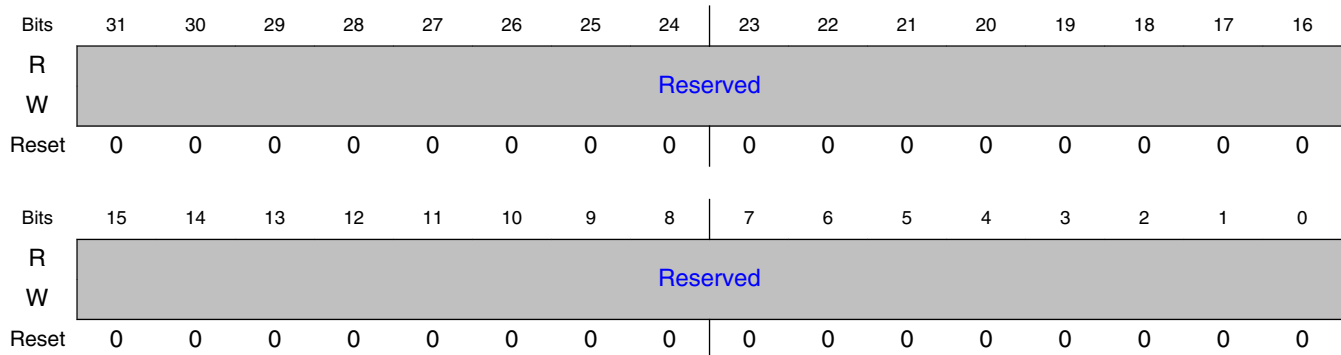
Field	Function
31-0	Reserved.
—	

14.197 AIOP Interface Job Data Register 30 (AIJOB30)

14.197.1 Offset

Register	Offset	Description
AIJOB30	5_05ECh	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.197.2 Diagram



14.197.3 Fields

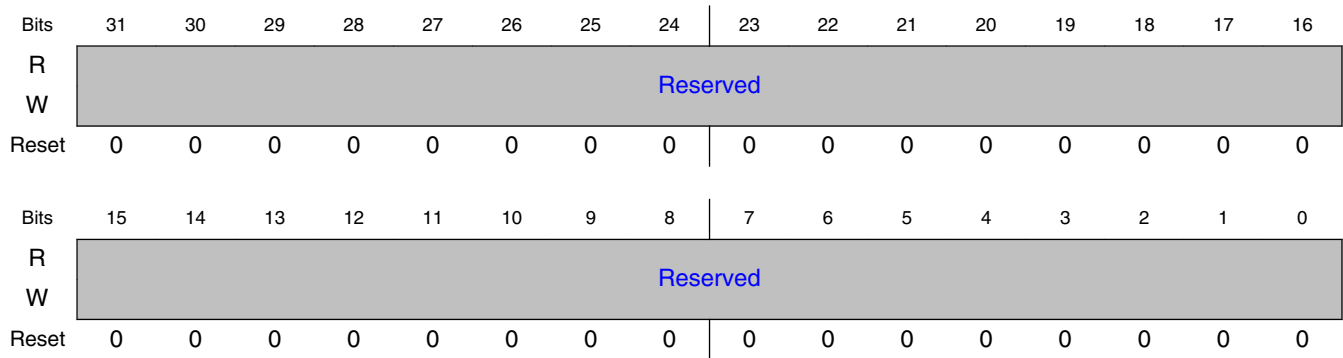
Field	Function
31-0	Reserved.
—	

14.198 AIOP Interface Job Data Register 31 (AIJOB31)

14.198.1 Offset

Register	Offset	Description
AIJOB31	5_05F0h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.198.2 Diagram



14.198.3 Fields

Field	Function
31-0	Reserved.
—	

14.199 AIOP Interface Job Data Register 32 (AIJOB32)

14.199.1 Offset

Register	Offset	Description
AIJOB32	5_05F4h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.199.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved								DQ_IDX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PREV_IDX								SFAID							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.199.3 Fields

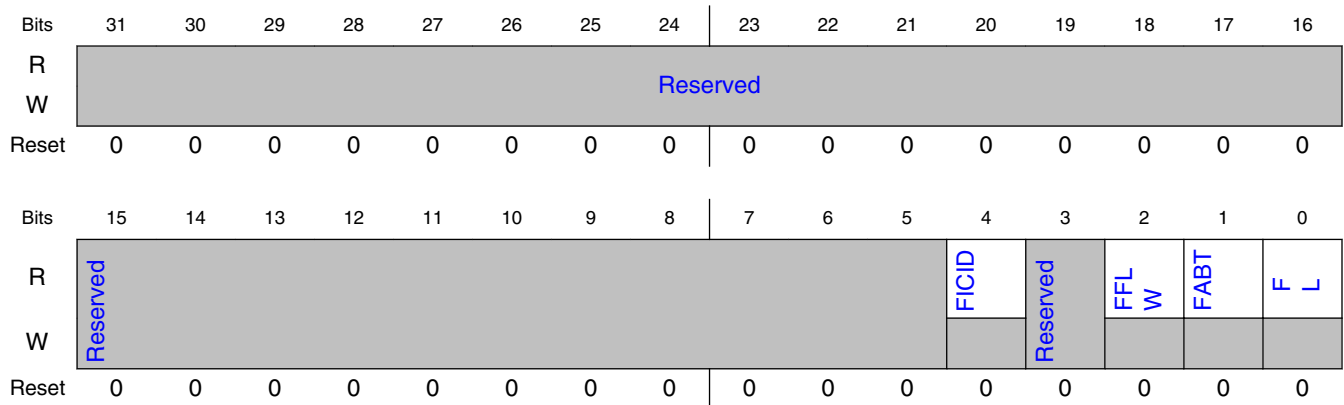
Field	Function
31-24 —	Reserved.
23-16 DQ_IDX	The dequeue order index of the job awaiting transfer to the Job Queue Controller.
15-8 PREV_IDX	The job buffer index of the previous job in the enqueue order list.
7-0 SFAID	The job buffer index of the next job in the enqueue order list.

14.200 AIOP Interface Job Data Register 33 (AIJOB33)

14.200.1 Offset

Register	Offset	Description
AIJOB33	5_05F8h	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.200.2 Diagram



14.200.3 Fields

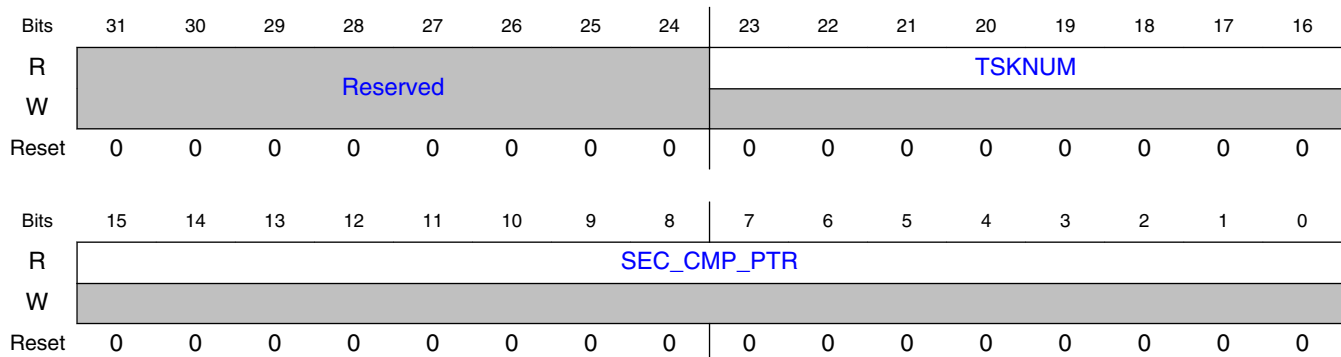
Field	Function
31-5 —	Reserved.
4 FICID	This job requested a flush of all AI jobs with the specified ICID (FRC 0000_1000b and 0000_1001b).
3 —	Reserved.
2 FFLW	This job requested a flow flush (FRC 0000_1100b).
1 FABT	This job requested either ICID flush with abort (FRC 0000_1001b) or input queue flush with abort (FRC 0000_1011b).
0 FL	The ID of the Subportal for this job.

14.201 AIOP Interface Job Data Register 34 (AIJOB34)

14.201.1 Offset

Register	Offset	Description
AIJOB34	5_05FCh	This register is used for debugging AIOP Interface jobs. Registers AIJOB0 .. AIJOB33 display job status data for the AI job selected via the JOB_SELECT_AI register.

14.201.2 Diagram



14.201.3 Fields

Field	Function
31-24 —	Reserved.
23-16 TSKNUM	The task number.
15-0 SEC_CMP_PTR	The SEC completion pointer.

14.202 Recoverable Error Interrupt Record 0 for the AIOP Interface (REIR0AI)

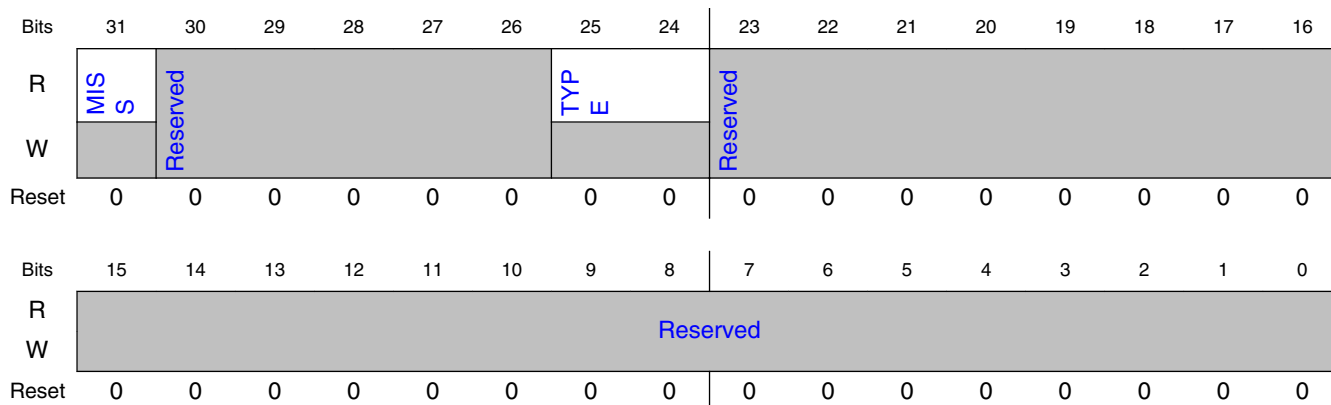
14.202.1 Offset

Register	Offset
REIROAI	5_0700h

14.202.2 Function

Two types of recoverable errors are defined for AI: system bus access (memory read/write) errors and ICID-related isolation errors. If a recoverable error occurs related to execution of a job from AI, error information will be captured in the AI's REIR registers. Data for a second AI recoverable error will not be captured unless REIROAI was written (with any value) prior to the occurrence of that error. If another recoverable error from AI occurs before that write, the missed error status bit (MISS) in REIROAI will be set. When REIROAI is written, all of AI's REIRAI registers are cleared and error capture is re-enabled.

14.202.3 Diagram



14.202.4 Fields

Field	Function
31 MISS	If MISS=1, a second AI recoverable error occurred before REIROAI was written to re-enable error data capture.
30-26	Reserved

Table continues on the next page...

Field	Function
—	
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 0 : no error If TYPE = 1 : memory access error If TYPE = 2 : isolation error If TYPE = 3 : reserved
23-0 —	Reserved

14.203 Recoverable Error Interrupt Record 1 for the AIOP Interface (REIR1AI)

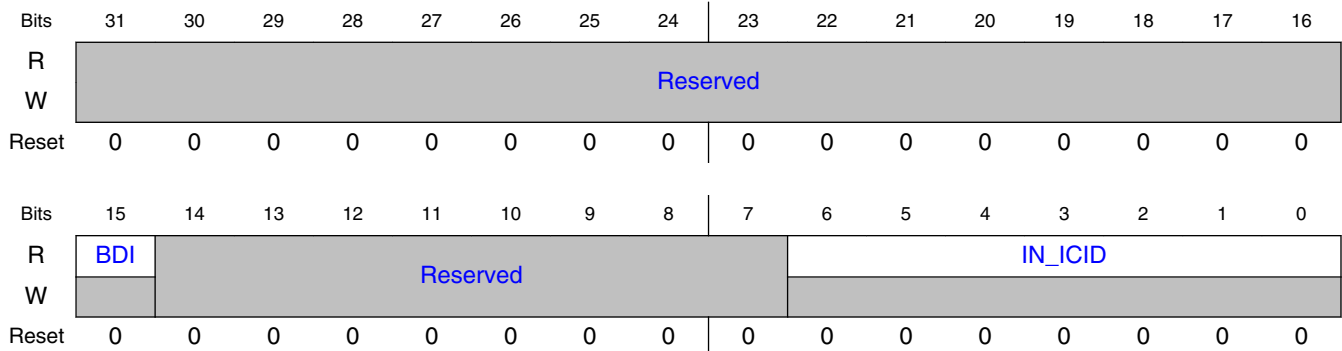
14.203.1 Offset

Register	Offset
REIR1AI	5_0704h

14.203.2 Function

See the description for REIR0AI.

14.203.3 Diagram



14.203.4 Fields

Field	Function
31-16 —	Reserved
15 BDI	For TYPE 2 errors this field holds the BDI for the job associated with the recoverable error. For TYPE 1 errors this field will return 0b.
14-7 —	Reserved
6-0 IN_ICID	For TYPE 2 errors this field holds the Input ICID for the job associated with the recoverable error. For TYPE 1 errors this field will return 0h.

14.204 Recoverable Error Interrupt Record 2 for the AIOP Interface (REIR2AI)

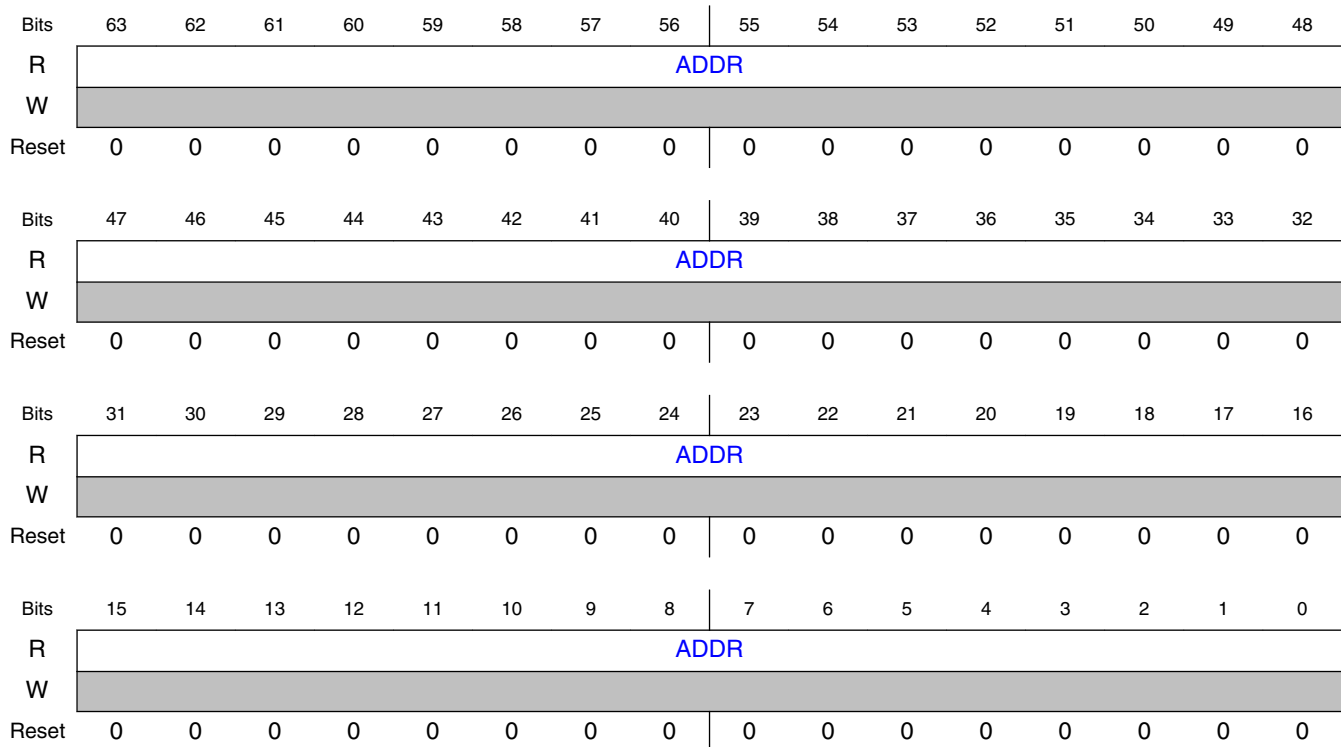
14.204.1 Offset

Register	Offset
REIR2AI	5_0708h

14.204.2 Function

See the description for REIR0AI.

14.204.3 Diagram



14.204.4 Fields

Field	Function
63-0 ADDR	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR) .

14.205 Recoverable Error Interrupt Record 4 for the AIOP Interface (REIR4AI)

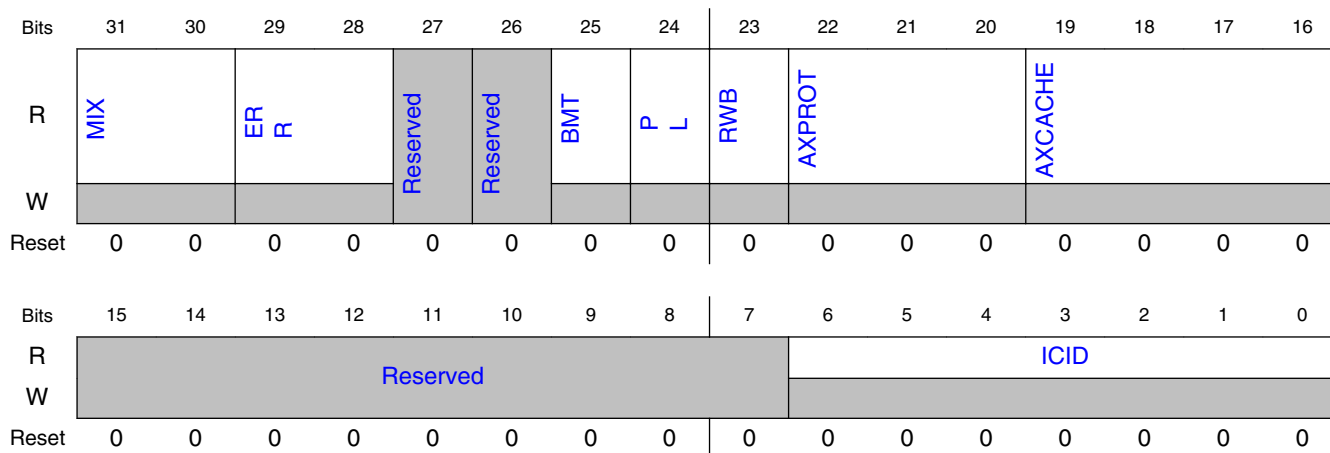
14.205.1 Offset

Register	Offset
REIR4AI	5_0710h

14.205.2 Function

See the description for REIR0AI. For Type 1 errors this register contains the 32-bit AI error status code. For Type 2 errors see the register layout that follows.

14.205.3 Diagram



14.205.4 Fields

Field	Function
31-30 MIX	This field holds the memory interface index associated with the recoverable error.
29-28 ERR	This field holds the AXI error response associated with the recoverable error.
27 —	Reserved
26 —	Reserved
25 BMT	This field holds the bypass memory translation transaction attribute used for the memory access.
24 PL	This field holds the privilege level transaction attribute used for the memory access.
23 RWB	This field specifies whether the memory access was a read or write.

Table continues on the next page...

Field	Function
22-20 AXPROT	This field holds the AXI protection transaction attribute used for the memory access.
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15-7 —	Reserved
6-0 ICID	For errors of Type 1 this field holds the ICID transaction attribute associated with the recoverable error.

14.206 Recoverable Error Interrupt Record 5 for the AIOPI Interface (REIR5AI)

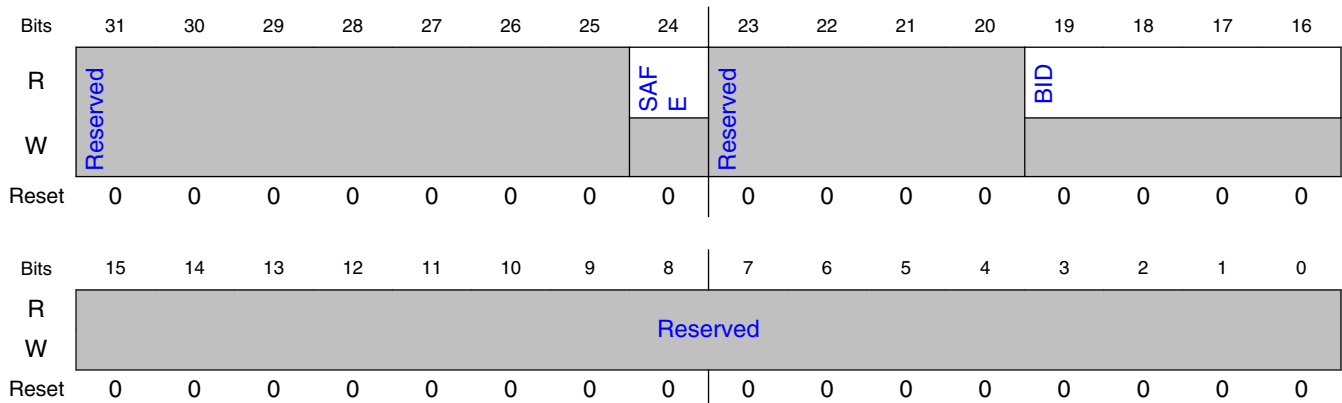
14.206.1 Offset

Register	Offset
REIR5AI	5_0714h

14.206.2 Function

See the description for REIROAI.

14.206.3 Diagram



14.206.4 Fields

Field	Function
31-25 —	Reserved
24 SAFE	For Type 1 errors SAFE indicates whether the AXI transaction associated with the recoverable error was a read-safe or write-safe transaction. For Type 2 errors this field is 0h.
23-20 —	Reserved
19-16 BID	For Type 1 errors this field holds the block identifier (see Table 14-1) of the source of the AXI transaction associated with the recoverable error. For Type 2 errors this field is 0h.
15-0 —	Reserved

14.207 RTIC Status Register (RSTA)

14.207.1 Offset

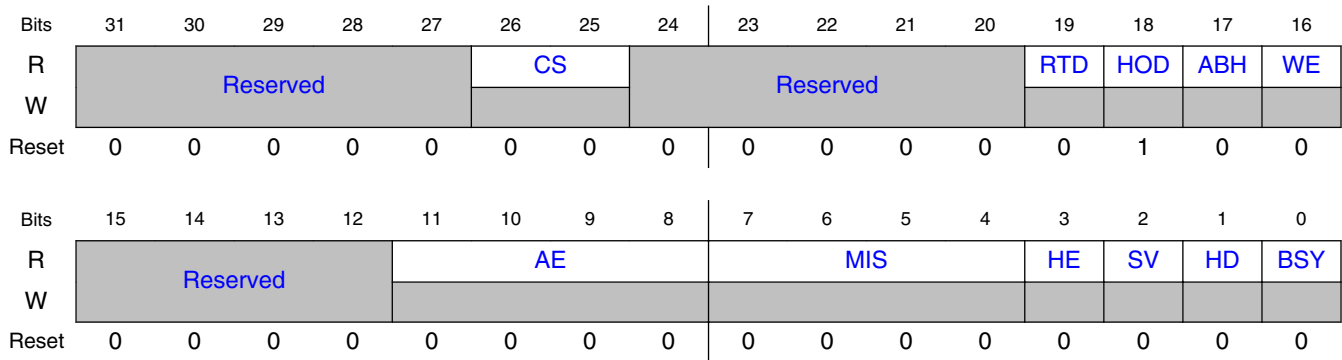
Register	Offset
RSTA	6_0004h

14.207.2 Function

This section describes the registers of the Run Time Integrity Checker (RTIC). A functional description of the RTIC can be found in [Run-time integrity checker \(RTIC\)](#). Note the use of the RTIC is optional, to support platform assurance.

The Run Time Integrity Checking Status Register is a read-only register that gives software information about the internal states of RTIC. Reading the RTIC Status Register will clear all errors and the RTIC interrupt. Due to timing issues, instead of polling this register software should read the RTIC Status Register after an RTIC done interrupt.

14.207.3 Diagram



14.207.4 Fields

Field	Function
31-27 —	Reserved
26-25 CS	RTIC Current State. Indicates the current state of the RTIC controller. 00b - Idle State 01b - Single Hash State 10b - Run-time State 11b - Error State
24-20 —	Reserved
19 RTD	Run Time Blocks Disabled. When RTIC is in Run Time mode, a 1 in the RTD field indicates that all the Memory Blocks are Disabled for Run Time Operation.
18 HOD	Hash Once Blocks Disabled. All the Memory Blocks are Disabled for Hash Once Operation. This bit is set when RTIC is either in an Idle State or Hash Once State and none of the Memory Blocks have been enabled for Hash Once Operation.
17 ABH	All Blocks Hashed. This is a bit that is used for debugging. This bit toggles during run-time mode every time RTIC completes hashing A-D memory blocks and starts over at the beginning again.
16 WE	RTIC Watchdog Error. RTIC Watchdog timer has tripped during run-time hashing. This indicates that all enabled memory segments did not finish a round of hashing prior to the RTIC watchdog timer completing. 0b - No RTIC Watchdog timer error has occurred. 1b - RTIC Watchdog timer has expired prior to completing a round of hashing.
15-12 —	Reserved
11-8 AE	Address Error. Indicates an illegal address was read from a peripheral memory block. This is caused by an invalid start address in the Address 1/2 fields or a value in the Length 1/2 fields that caused the RTIC to read outside a peripheral memory's valid address space. If an address error occurs, the illegal address will be captured in the SEC Fault Address Register (Section Fault Address Register (FAR)).

Table continues on the next page...

RTIC Status Register (RSTA)

Field	Function
	<p>Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error:</p> <p>xxx1 - Memory Block A Address Error xx1x - Memory Block B Address Error x1xx - Memory Block C Address Error 1xxx - Memory Block D Address Error</p> <p>The settings for each bit are as follows:</p> <p>0000b - All reads by RTIC were valid. 0001b - An illegal address was accessed by the RTIC</p>
7-4 MIS	<p>Memory Integrity Status. Indicates memory block(s) with error. Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error:</p> <p>xxx1 - Memory Block A Hash Error xx1x - Memory Block B Hash Error x1xx - Memory Block C Hash Error 1xxx - Memory Block D Hash Error</p> <p>The settings for each bit are as follows:</p> <p>0000b - Memory Block X is valid or state unknown 0001b - Memory Block X has been corrupted</p>
3 HE	<p>Hashing Error. Indicates that a unlocked memory block has been corrupted during run time or that an address error has occurred. The unlocked memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. The security violation signal will be asserted. RTIC will generate a done interrupt and disable checking the memory block that caused the failure.</p> <p>Unlocked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)).</p> <p>0b - Memory block contents authenticated. 1b - Memory block hash doesn't match reference value.</p>
2 SV	<p>Security Violation. Indicates that a locked RTIC memory block has been corrupted during run-time, an address error has occurred, or an RTIC Watchdog timeout has occurred. The memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. If an RTIC Watchdog timeout error occurred then the WE bit will be set. A security violation can only be cleared by a hardware reset.</p> <p>Locked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)).</p> <p>0b - Memory block contents authenticated. 1b - Memory block hash doesn't match reference value.</p>
1 HD	<p>Hash Once Operation Completed (Hash Done). processor may read hash values. If an error occurs during hashing or no memory blocks are enabled for one-time hash, this bit will not be set even if the RTIC hardware interrupts are asserted. This bit is cleared by setting the CINT bit in the RTIC Command Register (see Section RTIC Command Register (RCMD)) or when the RTIC enters the run-time checking state.</p> <p>0b - Boot authentication disabled 1b - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode.</p>
0	RTIC Idle/Busy Status. When busy, the RTIC cannot be written to.

Field	Function
BSY	0b - RTIC Idle. 1b - RTIC Busy.

14.208 RTIC Command Register (RCMD)

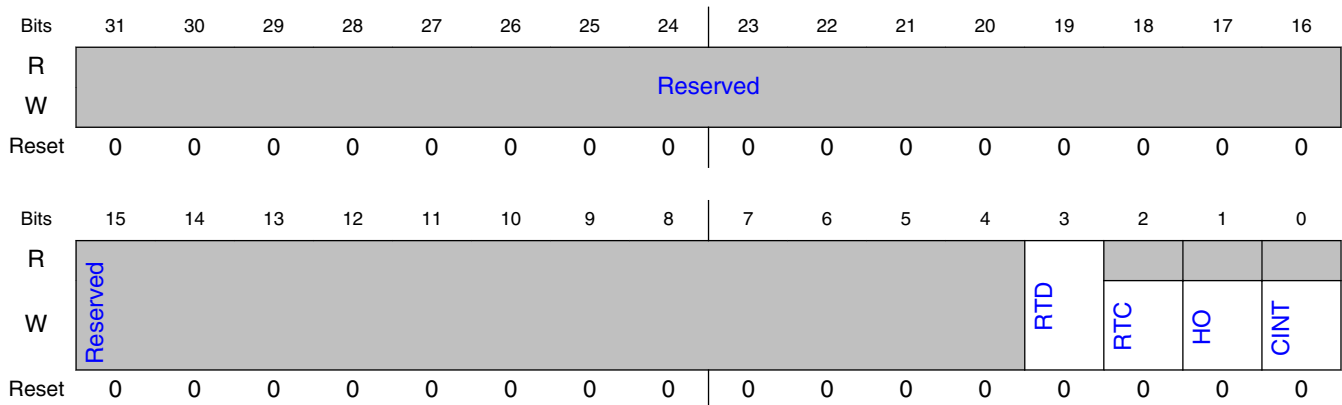
14.208.1 Offset

Register	Offset
RCMD	6_000Ch

14.208.2 Function

The Run Time Integrity Checking Command Register is used to issue commands to the RTIC. This register is used to instruct the RTIC to perform different functions. This register is only writeable when RTIC is in an idle state.

14.208.3 Diagram



14.208.4 Fields

Field	Function
31-4 —	Reserved
3 RTD	<p>Run Time Disable. Does not allow RTIC to be put into Run-Time mode. This bit will have higher priority in the case where both Run Time Check and Run Time Disable are set on the same write. Run Time Disable is ignored if RTIC is already in the Run Time Mode.</p> <p>0b - Allow Run Time Mode 1b - Prevent Run Time Mode</p>
2 RTC	<p>Run time check. Starts run-time integrity checking for any blocks having the corresponding RTME bit =1 (see RTIC Status Register (RSTA)). Some of the RTIC registers become read-only. This bit is self-clearing and always returns a logic-0 when read. Setting this bit will clear the ipi_done_int hardware interrupt as well as the HASH DONE bit in the RTIC Status Register. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode.</p> <p>If no memory blocks are enabled, setting the RUN TIME CHK bit will cause the RTIC to enter an idle state while waiting for a memory segment to be enabled. Some registers will be read only. No data is hashed and no interrupts or errors will be generated.</p> <p>0b - Run-time checking disabled 1b - Verify run-time memory blocks continually</p>
1 HO	<p>Hash once. Starts one-time hash/boot code authentication for any blocks having the corresponding HOME bit =1 (see RTIC Status Register (RSTA)). The resulting hash value is stored in the Hash Register File. This bit is self-clearing and always returns a logic-0 when read. If no memory blocks are enabled, a done interrupt will be immediately generated. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode.</p> <p>0b - Boot authentication disabled 1b - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode.</p>
0 CINT	<p>Clear Interrupt. Clears RTIC hardware interrupt signal. This bit is self-clearing and always returns a logic 0 when read.</p> <p>0b - Do not clear interrupt 1b - Clear interrupt. This bit cannot be modified during run-time checking mode</p>

14.209 RTIC Control Register (RCTL)

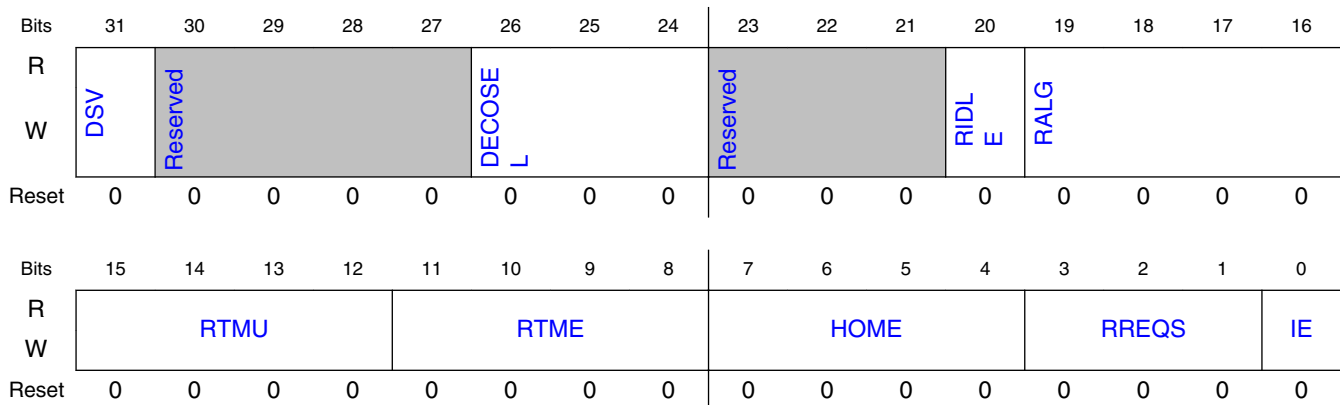
14.209.1 Offset

Register	Offset
RCTL	6_0014h

14.209.2 Function

The RTIC is configured by writing to the Run Time Integrity Checking Control Register. No bits in this register are writable unless RTIC is idle or, if RTIC is in Run-Time Mode, unless the control bits for the memory block are disabled and unlocked.

14.209.3 Diagram



14.209.4 Fields

Field	Function
31 DSV	DECOSEL Valid. If DSV=1, the DECOSEL field indicates the number of the DECO in which RTIC descriptors are to be run. This is intended for use during debugging. If DSV=0, the DECOSEL field is ignored and RTIC descriptors are run in any available DECO.
30-27 —	Reserved
26-24 DECOSEL	DECO Select. If DSV=1, DECOSEL indicates the number of the DECO in which RTIC descriptors are to be run. This is intended for use during debugging. If DSV=0, the DECOSEL field is ignored and RTIC descriptors are run in any available DECO. DECO Select is interpreted as shown below. Note that the use of any value other than those listed will generate an error. 000b - run RTIC descriptors in DECO 0 001b - run RTIC descriptors in DECO 1 010b - run RTIC descriptors in DECO 2 011b - run RTIC descriptors in DECO 3 100b - run RTIC descriptors in DECO 4 101b - run RTIC descriptors in DECO 5
23-21 —	Reserved

Table continues on the next page...

RTIC Control Register (RCTL)

Field	Function								
20 RIDLE	RTIC setting for the IPG_IDLE signal. If RIDLE=1, the signal ipg_idle will be negated if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits is 1). If RIDLE=0 and SEC is otherwise idle, the signal ipg_idle will still occasionally negate while RTIC is actually hashing a chunk of memory.								
19-16 RALG	RTIC Algorithm Select. Selects which algorithms should be used per Memory Block. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)								
	<table border="1"> <tr> <td>xxx0b - SHA-256 selected for Memory Block A</td> <td>xxx1b - SHA-512 selected for Memory Block A</td> </tr> <tr> <td>xx0xb - SHA-256 selected for Memory Block B</td> <td>xx1xb - SHA-512 selected for Memory Block B</td> </tr> <tr> <td>x0xxb - SHA-256 selected for Memory Block C</td> <td>x1xxb - SHA-512 selected for Memory Block C</td> </tr> <tr> <td>0xxxb - SHA-256 selected for Memory Block D</td> <td>1xxxb - SHA-512 selected for Memory Block D</td> </tr> </table>	xxx0b - SHA-256 selected for Memory Block A	xxx1b - SHA-512 selected for Memory Block A	xx0xb - SHA-256 selected for Memory Block B	xx1xb - SHA-512 selected for Memory Block B	x0xxb - SHA-256 selected for Memory Block C	x1xxb - SHA-512 selected for Memory Block C	0xxxb - SHA-256 selected for Memory Block D	1xxxb - SHA-512 selected for Memory Block D
xxx0b - SHA-256 selected for Memory Block A	xxx1b - SHA-512 selected for Memory Block A								
xx0xb - SHA-256 selected for Memory Block B	xx1xb - SHA-512 selected for Memory Block B								
x0xxb - SHA-256 selected for Memory Block C	x1xxb - SHA-512 selected for Memory Block C								
0xxxb - SHA-256 selected for Memory Block D	1xxxb - SHA-512 selected for Memory Block D								
15-12 RTMU	Run Time Memory Unlock. Unlocks memory block(s) for run-time hashing. If a memory block is unlocked it can be enabled and disabled at any time even if RTIC Run-Time Mode has started. These bits are not writable once RTIC Run-Time Mode has started. These bits are intended to allow some of the RTIC memory blocks to be used during RTIC Run-Time Mode by trusted software to verify the integrity of dynamically loaded software. The remaining (locked) memory blocks would be used to verify the integrity of the operating system and the trusted software itself. xxx1b - Unlock Memory Block A xx1xb - Unlock Memory Block B x1xxb - Unlock Memory Block C 1xxxb - Unlock Memory Block D								
11-8 RTME	Run Time Memory Enable. Enables memory block(s) for run-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description) xxx1 - Enable Memory Block A xx1x - Enable Memory Block B x1xx - Enable Memory Block C 1xxx - Enable Memory Block D								
7-4 HOME	Hash Once Memory Enable. Enables memory block(s) for one-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description) xxx1 - Enable Memory Block A xx1x - Enable Memory Block B x1xx - Enable Memory Block C 1xxx - Enable Memory Block D								
3-1 RREQS	RTIC Request Size. These bits are used during run-time mode to specify how many blocks of data are hashed every time the throttle counter expires. A block size is determined by the Algorithm that is selected. Block Size: SHA-256 = 64 bytes SHA-512 = 128 bytes Values:								

Table continues on the next page...

Field	Function
	000b - 1 Block 001b - 1 Block 010b - 2 Blocks 011b - 3 Blocks 100b - 4 Blocks 101b - 5 Blocks 110b - 6 Blocks 111b - 7 Blocks
0 IE	Interrupt Enable. Enables the RTIC interrupt. This bit is writable only while RTIC is in an idle state. Hardware interrupts are disabled by default after reset. 0b - Interrupts disabled 1b - Interrupts enabled

14.210 RTIC Throttle Register (RTHR)

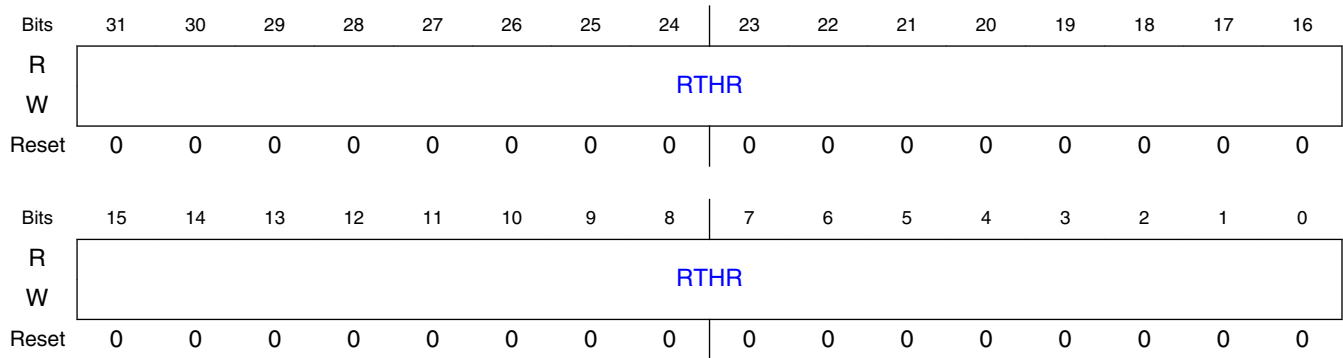
14.210.1 Offset

Register	Offset
RTHR	6_001Ch

14.210.2 Function

The Run Time Integrity Checking Throttle Register can be set to specify how many clock cycles to wait between RTIC hashing operations when RTIC is in run-time mode. The register becomes read-only when RTIC is in run-time mode.

14.210.3 Diagram



14.210.4 Fields

Field	Function
31-0 RTHR	Run Time Mode DMA Throttle. Programmable Timer that can be set to specify how many cycles of the system clock to wait between RTIC hashing operations during run time mode. At boot time, this register would generally be set to a value that will allow all four memory blocks to be hashed in a reasonable time without high bus utilization.

14.211 RTIC Watchdog Timer (RWDOG)

14.211.1 Offset

Register	Offset	Description
RWDOG	6_0028h	When the endianness is in the default configuration, this address is for the least-significant 32 bits. The most-significant 32 bits can be accessed at this address +4h. This address is for the most-significant 32 bits; the least-significant 32 bits can be accessed at this address +4h.

14.211.2 Function

The RTIC Watchdog Register holds the starting value for the RTIC Watchdog Timer, which is used during Run Time Mode to prevent a denial of service attack on RTIC. When RTIC is in Run Time Mode, the RTIC Watchdog Timer begins counting down when run time hashing starts on the first memory block and it resets to the starting value when the last memory block has been hashed. If the RTIC Watchdog Timer times out prior to the last memory block's completion then an RTIC Watchdog error will be generated. Note that the RTIC Watchdog Register is not writable after RTIC enters Run Time Mode, so prior to placing RTIC into Run Time Mode software must write a large enough value into the register to prevent the RTIC Watchdog Timer from expiring under normal conditions. Upon entering low-power mode the RTIC Watchdog Timer will stop counting until low-power mode is exited. Upon exiting low-power mode, the RTIC Watchdog Timer will resume from where it left off.

14.211.3 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	Reserved															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	RWDG															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RWDG															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RWDG															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.211.4 Fields

Field	Function
63-48 —	Reserved
47-0 RWDOG	Run Time Watchdog Time-Out value. This holds the starting value of the RTIC Run Time Watchdog Timer.

14.212 RTIC Memory Block a Address b Register (RMAA0 - RMDA1)

14.212.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

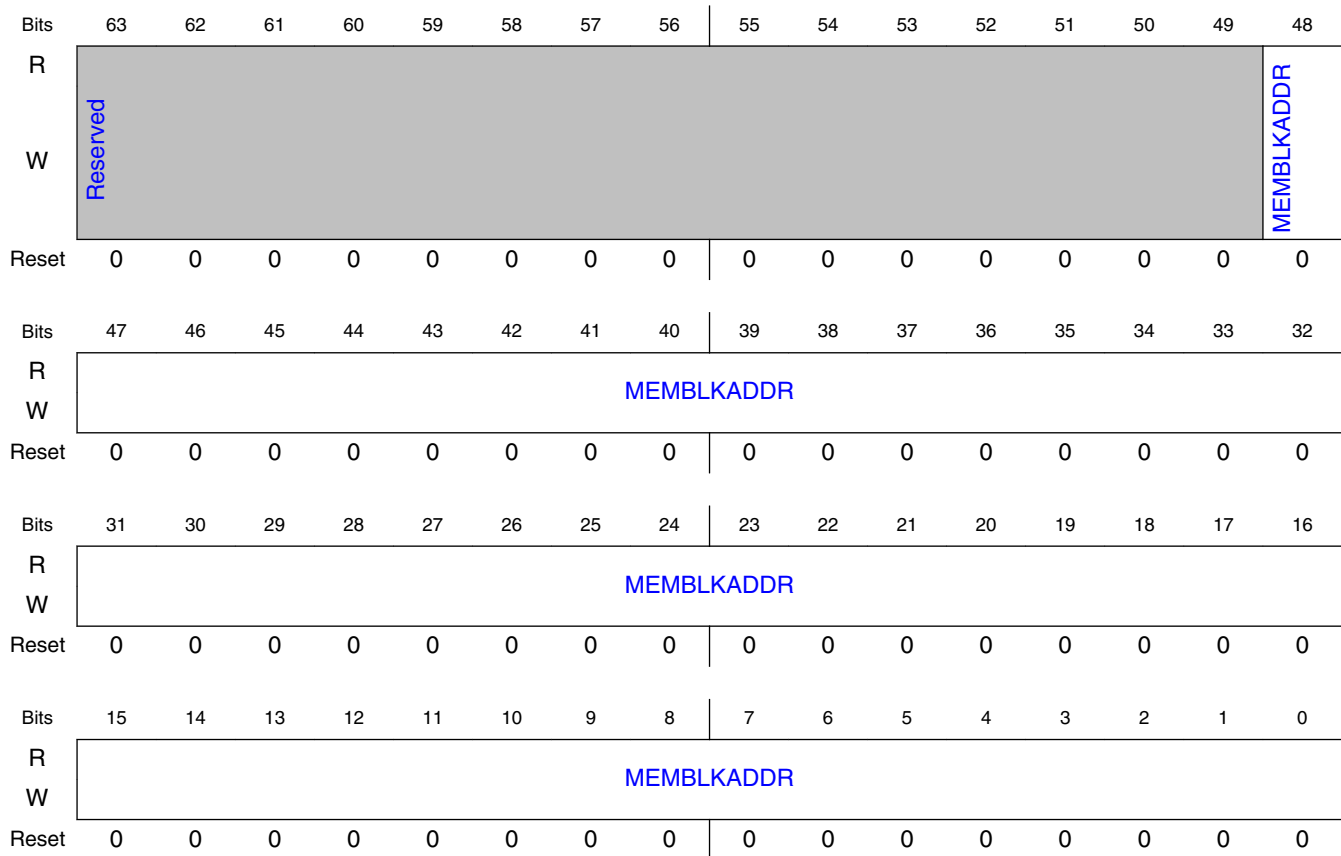
Register	Offset	Description
RMaAb	$6_0100h + (a \times 20h) + (b \times 10h)$	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.212.2 Function

For an explanation of the RTIC Memory Block registers, see [RTIC Memory Block Address/Length Registers](#)

The RTIC Memory Block a Address b Register (RMaAb) specifies the starting address of segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The length of data referred to by this pointer (see [Address pointers](#).) is found in the RTIC Memory Block a Length b Register (RMaLb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section [RTIC Control Register \(RCTL\)](#)) for the corresponding memory block.

14.212.3 Diagram



14.212.4 Fields

Field	Function
63-49 —	Reserved.
48-0 MEMBLKADDR	Memory Block Address. The MEMBLKADDR field of RMaAb holds the starting address of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D).

14.213 RTIC Memory Block a Length b Register (RMAL0 - RMDL1)

14.213.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

Register	Offset
RMALb	6_010Ch + (a × 20h) + (b × 10h)

14.213.2 Function

For an explanation of the RTIC Memory Block registers, see [RTIC Memory Block Address/Length Registers](#)

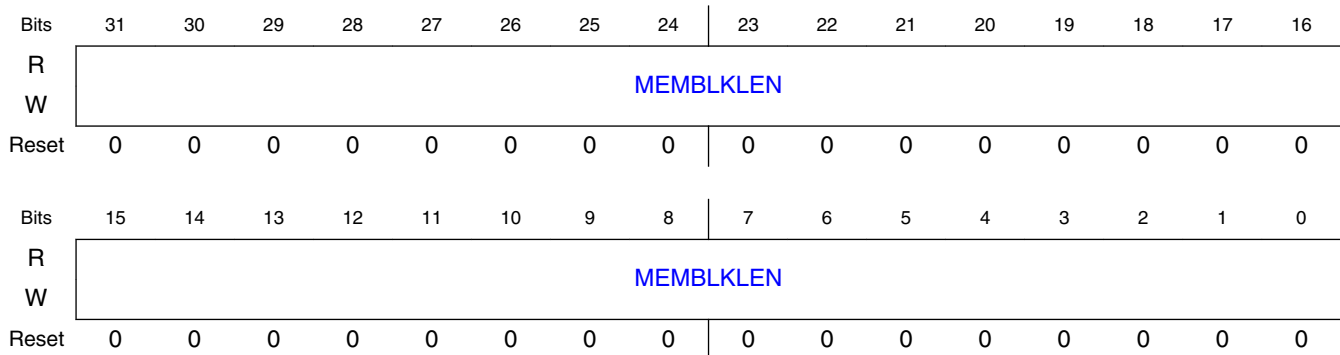
The RTIC Memory Block a Length b Register (RMALb) specifies the number of bytes to hash in segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The starting address of segment b of RTIC Memory Block a is specified in the RTIC Memory Block a Address b Register (RMAAb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writeable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section [RTIC Control Register \(RCTL\)](#)) for the corresponding memory block.

Note that programming a memory segment (A, B, C, D) to have a zero length (length_1 and length_2) will cause RTIC to generate a bad descriptor.

In RTIC versions RMJV= 0 and RMNV <=1 this can be detected by means of a watchdog timer. In hash-once operation this will be detected only if the DECO watchdog timer is enabled. This will cause the descriptor that is programmed by RTIC to be detected by the watchdog and flagged as an Address Error in the status register. In run-time operation the bad descriptor will be flagged by either the RTIC watchdog timer or the DECO watchdog timer. If the RTIC watchdog timer detects this condition then it will be flagged as an RTIC Watchdog Error. If instead the DECO watchdog catches it, then it will be flagged as an Address Error.

In later versions of RTIC bad RTIC descriptors will be flagged immediately as Address Errors.

14.213.3 Diagram



14.213.4 Fields

Field	Function
31-0 MEMBLKLEN	Memory Block Lengths. The MEMBLKLEN field of RMaLb holds the length, in bytes, of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D).

14.214 RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

14.214.1 Offset

Register	Offset	Description
RAMDB_0	6_0200h	RTIC Mem Block A Hash Result Big Endian Format Word 0
RAMDB_1	6_0204h	RTIC Mem Block A Hash Result Big Endian Format Word 1
RAMDB_2	6_0208h	RTIC Mem Block A Hash Result Big Endian Format Word 2
RAMDB_3	6_020Ch	RTIC Mem Block A Hash Result Big Endian Format Word 3
RAMDB_4	6_0210h	RTIC Mem Block A Hash Result Big Endian Format Word 4
RAMDB_5	6_0214h	RTIC Mem Block A Hash Result Big Endian Format Word 5

Table continues on the next page...

RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RAMDB_6	6_0218h	RTIC Mem Block A Hash Result Big Endian Format Word 6
RAMDB_7	6_021Ch	RTIC Mem Block A Hash Result Big Endian Format Word 7
RAMDB_8	6_0220h	RTIC Mem Block A Hash Result Big Endian Format Word 8
RAMDB_9	6_0224h	RTIC Mem Block A Hash Result Big Endian Format Word 9
RAMDB_10	6_0228h	RTIC Mem Block A Hash Result Big Endian Format Word 10
RAMDB_11	6_022Ch	RTIC Mem Block A Hash Result Big Endian Format Word 11
RAMDB_12	6_0230h	RTIC Mem Block A Hash Result Big Endian Format Word 12
RAMDB_13	6_0234h	RTIC Mem Block A Hash Result Big Endian Format Word 13
RAMDB_14	6_0238h	RTIC Mem Block A Hash Result Big Endian Format Word 14
RAMDB_15	6_023Ch	RTIC Mem Block A Hash Result Big Endian Format Word 15
RAMDB_16	6_0240h	RTIC Mem Block A Hash Result Big Endian Format Word 16
RAMDB_17	6_0244h	RTIC Mem Block A Hash Result Big Endian Format Word 17
RAMDB_18	6_0248h	RTIC Mem Block A Hash Result Big Endian Format Word 18
RAMDB_19	6_024Ch	RTIC Mem Block A Hash Result Big Endian Format Word 19
RAMDB_20	6_0250h	RTIC Mem Block A Hash Result Big Endian Format Word 20
RAMDB_21	6_0254h	RTIC Mem Block A Hash Result Big Endian Format Word 21
RAMDB_22	6_0258h	RTIC Mem Block A Hash Result Big Endian Format Word 22
RAMDB_23	6_025Ch	RTIC Mem Block A Hash Result Big Endian Format Word 23
RAMDB_24	6_0260h	RTIC Mem Block A Hash Result Big Endian Format Word 24
RAMDB_25	6_0264h	RTIC Mem Block A Hash Result Big Endian Format Word 25
RAMDB_26	6_0268h	RTIC Mem Block A Hash Result Big Endian Format Word 26
RAMDB_27	6_026Ch	RTIC Mem Block A Hash Result Big Endian Format Word 27
RAMDB_28	6_0270h	RTIC Mem Block A Hash Result Big Endian Format Word 28
RAMDB_29	6_0274h	RTIC Mem Block A Hash Result Big Endian Format Word 29

Table continues on the next page...

Register	Offset	Description
RAMDB_30	6_0278h	RTIC Mem Block A Hash Result Big Endian Format Word 30
RAMDB_31	6_027Ch	RTIC Mem Block A Hash Result Big Endian Format Word 31
RAMDL_0	6_0280h	RTIC Mem Block A Hash Result Little Endian Format Word 0
RAMDL_1	6_0284h	RTIC Mem Block A Hash Result Little Endian Format Word 1
RAMDL_2	6_0288h	RTIC Mem Block A Hash Result Little Endian Format Word 2
RAMDL_3	6_028Ch	RTIC Mem Block A Hash Result Little Endian Format Word 3
RAMDL_4	6_0290h	RTIC Mem Block A Hash Result Little Endian Format Word 4
RAMDL_5	6_0294h	RTIC Mem Block A Hash Result Little Endian Format Word 5
RAMDL_6	6_0298h	RTIC Mem Block A Hash Result Little Endian Format Word 6
RAMDL_7	6_029Ch	RTIC Mem Block A Hash Result Little Endian Format Word 7
RAMDL_8	6_02A0h	RTIC Mem Block A Hash Result Little Endian Format Word 8
RAMDL_9	6_02A4h	RTIC Mem Block A Hash Result Little Endian Format Word 9
RAMDL_10	6_02A8h	RTIC Mem Block A Hash Result Little Endian Format Word 10
RAMDL_11	6_02ACh	RTIC Mem Block A Hash Result Little Endian Format Word 11
RAMDL_12	6_02B0h	RTIC Mem Block A Hash Result Little Endian Format Word 12
RAMDL_13	6_02B4h	RTIC Mem Block A Hash Result Little Endian Format Word 13
RAMDL_14	6_02B8h	RTIC Mem Block A Hash Result Little Endian Format Word 14
RAMDL_15	6_02BCh	RTIC Mem Block A Hash Result Little Endian Format Word 15
RAMDL_16	6_02C0h	RTIC Mem Block A Hash Result Little Endian Format Word 16
RAMDL_17	6_02C4h	RTIC Mem Block A Hash Result Little Endian Format Word 17
RAMDL_18	6_02C8h	RTIC Mem Block A Hash Result Little Endian Format Word 18
RAMDL_19	6_02CCh	RTIC Mem Block A Hash Result Little Endian Format Word 19
RAMDL_20	6_02D0h	RTIC Mem Block A Hash Result Little Endian Format Word 20
RAMDL_21	6_02D4h	RTIC Mem Block A Hash Result Little Endian Format Word 21

Table continues on the next page...

RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RAMDL_22	6_02D8h	RTIC Mem Block A Hash Result Little Endian Format Word 22
RAMDL_23	6_02DCh	RTIC Mem Block A Hash Result Little Endian Format Word 23
RAMDL_24	6_02E0h	RTIC Mem Block A Hash Result Little Endian Format Word 24
RAMDL_25	6_02E4h	RTIC Mem Block A Hash Result Little Endian Format Word 25
RAMDL_26	6_02E8h	RTIC Mem Block A Hash Result Little Endian Format Word 26
RAMDL_27	6_02ECh	RTIC Mem Block A Hash Result Little Endian Format Word 27
RAMDL_28	6_02F0h	RTIC Mem Block A Hash Result Little Endian Format Word 28
RAMDL_29	6_02F4h	RTIC Mem Block A Hash Result Little Endian Format Word 29
RAMDL_30	6_02F8h	RTIC Mem Block A Hash Result Little Endian Format Word 30
RAMDL_31	6_02FCh	RTIC Mem Block A Hash Result Little Endian Format Word 31
RBMDB_0	6_0300h	RTIC Mem Block B Hash Result Big Endian Format Word 0
RBMDB_1	6_0304h	RTIC Mem Block B Hash Result Big Endian Format Word 1
RBMDB_2	6_0308h	RTIC Mem Block B Hash Result Big Endian Format Word 2
RBMDB_3	6_030Ch	RTIC Mem Block B Hash Result Big Endian Format Word 3
RBMDB_4	6_0310h	RTIC Mem Block B Hash Result Big Endian Format Word 4
RBMDB_5	6_0314h	RTIC Mem Block B Hash Result Big Endian Format Word 5
RBMDB_6	6_0318h	RTIC Mem Block B Hash Result Big Endian Format Word 6
RBMDB_7	6_031Ch	RTIC Mem Block B Hash Result Big Endian Format Word 7
RBMDB_8	6_0320h	RTIC Mem Block B Hash Result Big Endian Format Word 8
RBMDB_9	6_0324h	RTIC Mem Block B Hash Result Big Endian Format Word 9
RBMDB_10	6_0328h	RTIC Mem Block B Hash Result Big Endian Format Word 10
RBMDB_11	6_032Ch	RTIC Mem Block B Hash Result Big Endian Format Word 11
RBMDB_12	6_0330h	RTIC Mem Block B Hash Result Big Endian Format Word 12
RBMDB_13	6_0334h	RTIC Mem Block B Hash Result Big Endian Format Word 13

Table continues on the next page...

Register	Offset	Description
RBMDB_14	6_0338h	RTIC Mem Block B Hash Result Big Endian Format Word 14
RBMDB_15	6_033Ch	RTIC Mem Block B Hash Result Big Endian Format Word 15
RBMDB_16	6_0340h	RTIC Mem Block B Hash Result Big Endian Format Word 16
RBMDB_17	6_0344h	RTIC Mem Block B Hash Result Big Endian Format Word 17
RBMDB_18	6_0348h	RTIC Mem Block B Hash Result Big Endian Format Word 18
RBMDB_19	6_034Ch	RTIC Mem Block B Hash Result Big Endian Format Word 19
RBMDB_20	6_0350h	RTIC Mem Block B Hash Result Big Endian Format Word 20
RBMDB_21	6_0354h	RTIC Mem Block B Hash Result Big Endian Format Word 21
RBMDB_22	6_0358h	RTIC Mem Block B Hash Result Big Endian Format Word 22
RBMDB_23	6_035Ch	RTIC Mem Block B Hash Result Big Endian Format Word 23
RBMDB_24	6_0360h	RTIC Mem Block B Hash Result Big Endian Format Word 24
RBMDB_25	6_0364h	RTIC Mem Block B Hash Result Big Endian Format Word 25
RBMDB_26	6_0368h	RTIC Mem Block B Hash Result Big Endian Format Word 26
RBMDB_27	6_036Ch	RTIC Mem Block B Hash Result Big Endian Format Word 27
RBMDB_28	6_0370h	RTIC Mem Block B Hash Result Big Endian Format Word 28
RBMDB_29	6_0374h	RTIC Mem Block B Hash Result Big Endian Format Word 29
RBMDB_30	6_0378h	RTIC Mem Block B Hash Result Big Endian Format Word 30
RBMDB_31	6_037Ch	RTIC Mem Block B Hash Result Big Endian Format Word 31
RBMDL_0	6_0380h	RTIC Mem Block B Hash Result Little Endian Format Word 0
RBMDL_1	6_0384h	RTIC Mem Block B Hash Result Little Endian Format Word 1
RBMDL_2	6_0388h	RTIC Mem Block B Hash Result Little Endian Format Word 2
RBMDL_3	6_038Ch	RTIC Mem Block B Hash Result Little Endian Format Word 3
RBMDL_4	6_0390h	RTIC Mem Block B Hash Result Little Endian Format Word 4
RBMDL_5	6_0394h	RTIC Mem Block B Hash Result Little Endian Format Word 5

Table continues on the next page...

RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RBMDL_6	6_0398h	RTIC Mem Block B Hash Result Little Endian Format Word 6
RBMDL_7	6_039Ch	RTIC Mem Block B Hash Result Little Endian Format Word 7
RBMDL_8	6_03A0h	RTIC Mem Block B Hash Result Little Endian Format Word 8
RBMDL_9	6_03A4h	RTIC Mem Block B Hash Result Little Endian Format Word 9
RBMDL_10	6_03A8h	RTIC Mem Block B Hash Result Little Endian Format Word 10
RBMDL_11	6_03ACh	RTIC Mem Block B Hash Result Little Endian Format Word 11
RBMDL_12	6_03B0h	RTIC Mem Block B Hash Result Little Endian Format Word 12
RBMDL_13	6_03B4h	RTIC Mem Block B Hash Result Little Endian Format Word 13
RBMDL_14	6_03B8h	RTIC Mem Block B Hash Result Little Endian Format Word 14
RBMDL_15	6_03BCh	RTIC Mem Block B Hash Result Little Endian Format Word 15
RBMDL_16	6_03C0h	RTIC Mem Block B Hash Result Little Endian Format Word 16
RBMDL_17	6_03C4h	RTIC Mem Block B Hash Result Little Endian Format Word 17
RBMDL_18	6_03C8h	RTIC Mem Block B Hash Result Little Endian Format Word 18
RBMDL_19	6_03CCh	RTIC Mem Block B Hash Result Little Endian Format Word 19
RBMDL_20	6_03D0h	RTIC Mem Block B Hash Result Little Endian Format Word 20
RBMDL_21	6_03D4h	RTIC Mem Block B Hash Result Little Endian Format Word 21
RBMDL_22	6_03D8h	RTIC Mem Block B Hash Result Little Endian Format Word 22
RBMDL_23	6_03DCh	RTIC Mem Block B Hash Result Little Endian Format Word 23
RBMDL_24	6_03E0h	RTIC Mem Block B Hash Result Little Endian Format Word 24
RBMDL_25	6_03E4h	RTIC Mem Block B Hash Result Little Endian Format Word 25
RBMDL_26	6_03E8h	RTIC Mem Block B Hash Result Little Endian Format Word 26
RBMDL_27	6_03ECh	RTIC Mem Block B Hash Result Little Endian Format Word 27
RBMDL_28	6_03F0h	RTIC Mem Block B Hash Result Little Endian Format Word 28
RBMDL_29	6_03F4h	RTIC Mem Block B Hash Result Little Endian Format Word 29

Table continues on the next page...

Register	Offset	Description
RBMDL_30	6_03F8h	RTIC Mem Block B Hash Result Little Endian Format Word 30
RBMDL_31	6_03FCh	RTIC Mem Block B Hash Result Little Endian Format Word 31
RCMDB_0	6_0400h	RTIC Mem Block C Hash Result Big Endian Format Word 0
RCMDB_1	6_0404h	RTIC Mem Block C Hash Result Big Endian Format Word 1
RCMDB_2	6_0408h	RTIC Mem Block C Hash Result Big Endian Format Word 2
RCMDB_3	6_040Ch	RTIC Mem Block C Hash Result Big Endian Format Word 3
RCMDB_4	6_0410h	RTIC Mem Block C Hash Result Big Endian Format Word 4
RCMDB_5	6_0414h	RTIC Mem Block C Hash Result Big Endian Format Word 5
RCMDB_6	6_0418h	RTIC Mem Block C Hash Result Big Endian Format Word 6
RCMDB_7	6_041Ch	RTIC Mem Block C Hash Result Big Endian Format Word 7
RCMDB_8	6_0420h	RTIC Mem Block C Hash Result Big Endian Format Word 8
RCMDB_9	6_0424h	RTIC Mem Block C Hash Result Big Endian Format Word 9
RCMDB_10	6_0428h	RTIC Mem Block C Hash Result Big Endian Format Word 10
RCMDB_11	6_042Ch	RTIC Mem Block C Hash Result Big Endian Format Word 11
RCMDB_12	6_0430h	RTIC Mem Block C Hash Result Big Endian Format Word 12
RCMDB_13	6_0434h	RTIC Mem Block C Hash Result Big Endian Format Word 13
RCMDB_14	6_0438h	RTIC Mem Block C Hash Result Big Endian Format Word 14
RCMDB_15	6_043Ch	RTIC Mem Block C Hash Result Big Endian Format Word 15
RCMDB_16	6_0440h	RTIC Mem Block C Hash Result Big Endian Format Word 16
RCMDB_17	6_0444h	RTIC Mem Block C Hash Result Big Endian Format Word 17
RCMDB_18	6_0448h	RTIC Mem Block C Hash Result Big Endian Format Word 18
RCMDB_19	6_044Ch	RTIC Mem Block C Hash Result Big Endian Format Word 19
RCMDB_20	6_0450h	RTIC Mem Block C Hash Result Big Endian Format Word 20
RCMDB_21	6_0454h	RTIC Mem Block C Hash Result Big Endian Format Word 21

Table continues on the next page...

RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RCMDB_22	6_0458h	RTIC Mem Block C Hash Result Big Endian Format Word 22
RCMDB_23	6_045Ch	RTIC Mem Block C Hash Result Big Endian Format Word 23
RCMDB_24	6_0460h	RTIC Mem Block C Hash Result Big Endian Format Word 24
RCMDB_25	6_0464h	RTIC Mem Block C Hash Result Big Endian Format Word 25
RCMDB_26	6_0468h	RTIC Mem Block C Hash Result Big Endian Format Word 26
RCMDB_27	6_046Ch	RTIC Mem Block C Hash Result Big Endian Format Word 27
RCMDB_28	6_0470h	RTIC Mem Block C Hash Result Big Endian Format Word 28
RCMDB_29	6_0474h	RTIC Mem Block C Hash Result Big Endian Format Word 29
RCMDB_30	6_0478h	RTIC Mem Block C Hash Result Big Endian Format Word 30
RCMDB_31	6_047Ch	RTIC Mem Block C Hash Result Big Endian Format Word 31
RCMDL_0	6_0480h	RTIC Mem Block C Hash Result Little Endian Format Word 0
RCMDL_1	6_0484h	RTIC Mem Block C Hash Result Little Endian Format Word 1
RCMDL_2	6_0488h	RTIC Mem Block C Hash Result Little Endian Format Word 2
RCMDL_3	6_048Ch	RTIC Mem Block C Hash Result Little Endian Format Word 3
RCMDL_4	6_0490h	RTIC Mem Block C Hash Result Little Endian Format Word 4
RCMDL_5	6_0494h	RTIC Mem Block C Hash Result Little Endian Format Word 5
RCMDL_6	6_0498h	RTIC Mem Block C Hash Result Little Endian Format Word 6
RCMDL_7	6_049Ch	RTIC Mem Block C Hash Result Little Endian Format Word 7
RCMDL_8	6_04A0h	RTIC Mem Block C Hash Result Little Endian Format Word 8
RCMDL_9	6_04A4h	RTIC Mem Block C Hash Result Little Endian Format Word 9
RCMDL_10	6_04A8h	RTIC Mem Block C Hash Result Little Endian Format Word 10
RCMDL_11	6_04ACh	RTIC Mem Block C Hash Result Little Endian Format Word 11
RCMDL_12	6_04B0h	RTIC Mem Block C Hash Result Little Endian Format Word 12
RCMDL_13	6_04B4h	RTIC Mem Block C Hash Result Little Endian Format Word 13

Table continues on the next page...

Register	Offset	Description
RCMDL_14	6_04B8h	RTIC Mem Block C Hash Result Little Endian Format Word 14
RCMDL_15	6_04BCCh	RTIC Mem Block C Hash Result Little Endian Format Word 15
RCMDL_16	6_04C0h	RTIC Mem Block C Hash Result Little Endian Format Word 16
RCMDL_17	6_04C4h	RTIC Mem Block C Hash Result Little Endian Format Word 17
RCMDL_18	6_04C8h	RTIC Mem Block C Hash Result Little Endian Format Word 18
RCMDL_19	6_04CCh	RTIC Mem Block C Hash Result Little Endian Format Word 19
RCMDL_20	6_04D0h	RTIC Mem Block C Hash Result Little Endian Format Word 20
RCMDL_21	6_04D4h	RTIC Mem Block C Hash Result Little Endian Format Word 21
RCMDL_22	6_04D8h	RTIC Mem Block C Hash Result Little Endian Format Word 22
RCMDL_23	6_04DCh	RTIC Mem Block C Hash Result Little Endian Format Word 23
RCMDL_24	6_04E0h	RTIC Mem Block C Hash Result Little Endian Format Word 24
RCMDL_25	6_04E4h	RTIC Mem Block C Hash Result Little Endian Format Word 25
RCMDL_26	6_04E8h	RTIC Mem Block C Hash Result Little Endian Format Word 26
RCMDL_27	6_04ECh	RTIC Mem Block C Hash Result Little Endian Format Word 27
RCMDL_28	6_04F0h	RTIC Mem Block C Hash Result Little Endian Format Word 28
RCMDL_29	6_04F4h	RTIC Mem Block C Hash Result Little Endian Format Word 29
RCMDL_30	6_04F8h	RTIC Mem Block C Hash Result Little Endian Format Word 30
RCMDL_31	6_04FCh	RTIC Mem Block C Hash Result Little Endian Format Word 31
RDMDB_0	6_0500h	RTIC Mem Block D Hash Result Big Endian Format Word 0
RDMDB_1	6_0504h	RTIC Mem Block D Hash Result Big Endian Format Word 1
RDMDB_2	6_0508h	RTIC Mem Block D Hash Result Big Endian Format Word 2
RDMDB_3	6_050Ch	RTIC Mem Block D Hash Result Big Endian Format Word 3
RDMDB_4	6_0510h	RTIC Mem Block D Hash Result Big Endian Format Word 4
RDMDB_5	6_0514h	RTIC Mem Block D Hash Result Big Endian Format Word 5

Table continues on the next page...

RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RDMDL_6	6_0518h	RTIC Mem Block D Hash Result Big Endian Format Word 6
RDMDL_7	6_051Ch	RTIC Mem Block D Hash Result Big Endian Format Word 7
RDMDL_8	6_0520h	RTIC Mem Block D Hash Result Big Endian Format Word 8
RDMDL_9	6_0524h	RTIC Mem Block D Hash Result Big Endian Format Word 9
RDMDL_10	6_0528h	RTIC Mem Block D Hash Result Big Endian Format Word 10
RDMDL_11	6_052Ch	RTIC Mem Block D Hash Result Big Endian Format Word 11
RDMDL_12	6_0530h	RTIC Mem Block D Hash Result Big Endian Format Word 12
RDMDL_13	6_0534h	RTIC Mem Block D Hash Result Big Endian Format Word 13
RDMDL_14	6_0538h	RTIC Mem Block D Hash Result Big Endian Format Word 14
RDMDL_15	6_053Ch	RTIC Mem Block D Hash Result Big Endian Format Word 15
RDMDL_16	6_0540h	RTIC Mem Block D Hash Result Big Endian Format Word 16
RDMDL_17	6_0544h	RTIC Mem Block D Hash Result Big Endian Format Word 17
RDMDL_18	6_0548h	RTIC Mem Block D Hash Result Big Endian Format Word 18
RDMDL_19	6_054Ch	RTIC Mem Block D Hash Result Big Endian Format Word 19
RDMDL_20	6_0550h	RTIC Mem Block D Hash Result Big Endian Format Word 20
RDMDL_21	6_0554h	RTIC Mem Block D Hash Result Big Endian Format Word 21
RDMDL_22	6_0558h	RTIC Mem Block D Hash Result Big Endian Format Word 22
RDMDL_23	6_055Ch	RTIC Mem Block D Hash Result Big Endian Format Word 23
RDMDL_24	6_0560h	RTIC Mem Block D Hash Result Big Endian Format Word 24
RDMDL_25	6_0564h	RTIC Mem Block D Hash Result Big Endian Format Word 25
RDMDL_26	6_0568h	RTIC Mem Block D Hash Result Big Endian Format Word 26
RDMDL_27	6_056Ch	RTIC Mem Block D Hash Result Big Endian Format Word 27
RDMDL_28	6_0570h	RTIC Mem Block D Hash Result Big Endian Format Word 28
RDMDL_29	6_0574h	RTIC Mem Block D Hash Result Big Endian Format Word 29

Table continues on the next page...

Register	Offset	Description
RDMDB_30	6_0578h	RTIC Mem Block D Hash Result Big Endian Format Word 30
RDMDB_31	6_057Ch	RTIC Mem Block D Hash Result Big Endian Format Word 31
RDMDL_0	6_0580h	RTIC Mem Block D Hash Result Little Endian Format Word 0
RDMDL_1	6_0584h	RTIC Mem Block D Hash Result Little Endian Format Word 1
RDMDL_2	6_0588h	RTIC Mem Block D Hash Result Little Endian Format Word 2
RDMDL_3	6_058Ch	RTIC Mem Block D Hash Result Little Endian Format Word 3
RDMDL_4	6_0590h	RTIC Mem Block D Hash Result Little Endian Format Word 4
RDMDL_5	6_0594h	RTIC Mem Block D Hash Result Little Endian Format Word 5
RDMDL_6	6_0598h	RTIC Mem Block D Hash Result Little Endian Format Word 6
RDMDL_7	6_059Ch	RTIC Mem Block D Hash Result Little Endian Format Word 7
RDMDL_8	6_05A0h	RTIC Mem Block D Hash Result Little Endian Format Word 8
RDMDL_9	6_05A4h	RTIC Mem Block D Hash Result Little Endian Format Word 9
RDMDL_10	6_05A8h	RTIC Mem Block D Hash Result Little Endian Format Word 10
RDMDL_11	6_05ACh	RTIC Mem Block D Hash Result Little Endian Format Word 11
RDMDL_12	6_05B0h	RTIC Mem Block D Hash Result Little Endian Format Word 12
RDMDL_13	6_05B4h	RTIC Mem Block D Hash Result Little Endian Format Word 13
RDMDL_14	6_05B8h	RTIC Mem Block D Hash Result Little Endian Format Word 14
RDMDL_15	6_05BCh	RTIC Mem Block D Hash Result Little Endian Format Word 15
RDMDL_16	6_05C0h	RTIC Mem Block D Hash Result Little Endian Format Word 16
RDMDL_17	6_05C4h	RTIC Mem Block D Hash Result Little Endian Format Word 17
RDMDL_18	6_05C8h	RTIC Mem Block D Hash Result Little Endian Format Word 18
RDMDL_19	6_05CCh	RTIC Mem Block D Hash Result Little Endian Format Word 19
RDMDL_20	6_05D0h	RTIC Mem Block D Hash Result Little Endian Format Word 20
RDMDL_21	6_05D4h	RTIC Mem Block D Hash Result Little Endian Format Word 21

Table continues on the next page...

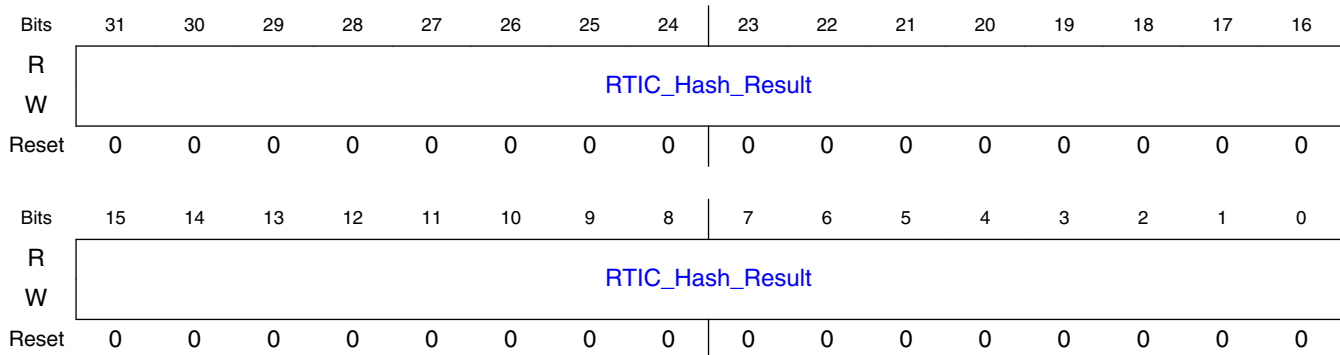
RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

Register	Offset	Description
RDMDL_22	6_05D8h	RTIC Mem Block D Hash Result Little Endian Format Word 22
RDMDL_23	6_05DCh	RTIC Mem Block D Hash Result Little Endian Format Word 23
RDMDL_24	6_05E0h	RTIC Mem Block D Hash Result Little Endian Format Word 24
RDMDL_25	6_05E4h	RTIC Mem Block D Hash Result Little Endian Format Word 25
RDMDL_26	6_05E8h	RTIC Mem Block D Hash Result Little Endian Format Word 26
RDMDL_27	6_05ECh	RTIC Mem Block D Hash Result Little Endian Format Word 27
RDMDL_28	6_05F0h	RTIC Mem Block D Hash Result Little Endian Format Word 28
RDMDL_29	6_05F4h	RTIC Mem Block D Hash Result Little Endian Format Word 29
RDMDL_30	6_05F8h	RTIC Mem Block D Hash Result Little Endian Format Word 30
RDMDL_31	6_05FCh	RTIC Mem Block D Hash Result Little Endian Format Word 31

14.214.2 Function

The results of the RTIC hashing operations are stored in the RTIC Hash Result Registers (256 bits for SHA-256, 512 bits for SHA-512). The hash result for Memory Block a (a= A,B,C,D) is accessed in contiguous word addresses beginning at the base address of RTIC Hash Result Register a. For each Memory Block, there are 2 addresses associated with RTIC Hash Result Register a. Reading successive words starting at the RaMDB address will return successive words, in big endian format, of the hash result for Memory Block a. Reading successive words starting at the RaMDL address will return successive words, in little endian format, of the hash result for Memory Block a.

14.214.3 Diagram



14.214.4 Fields

Field	Function
31-0 RTIC_Hash_Result	RTIC_Hash_Result

14.215 Recoverable Error Interrupt Record 0 for RTIC (REIR0RTIC)

14.215.1 Offset

Register	Offset
REIR0RTIC	6_0E00h

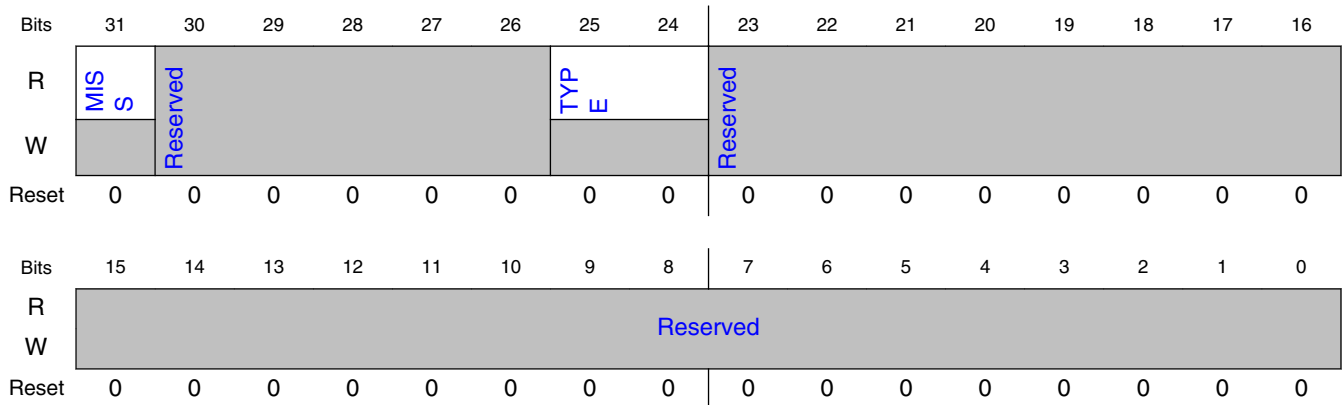
14.215.2 Function

If a recoverable error occurs related to execution of a job from RTIC, error information will be captured in RTIC's REIR registers. Data for a second recoverable error related to jobs from RTIC will not be captured until the REIR0RTIC is written. If another bus error

Recoverable Error Interrupt Record 2 for RTIC (REIR2RTIC)

from RTIC occurs before then, the double error status bit (MISS) in REIR0RTIC will be set. When REIR0RTIC is written, all of RTIC's REIRRTIC registers are cleared and error capture is re-enabled.

14.215.3 Diagram



14.215.4 Fields

Field	Function
31 MISS	If MISS=1, a second recoverable error associated with RTIC occurred before REIR0RTIC was written following a previous RTIC recoverable error.
30-26 —	Reserved
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 00b : reserved If TYPE = 01b : memory access error If TYPE = 10b : reserved If TYPE = 11b : reserved
23-0 —	Reserved

14.216 Recoverable Error Interrupt Record 2 for RTIC (REIR2RTIC)

14.216.1 Offset

Register	Offset	Description
REIR2RTIC	6_0E08h	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.216.2 Function

See the description for [Recoverable Error Interrupt Record 0 for RTIC \(REIR0RTIC\)](#).

14.216.3 Diagram

Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.216.4 Fields

Field	Function
63-0	This register holds the address associated with the recoverable error.

Recoverable Error Interrupt Record 4 for RTIC (REIR4RTIC)

Field	Function
ADDR	

14.217 Recoverable Error Interrupt Record 4 for RTIC (REIR4RTIC)

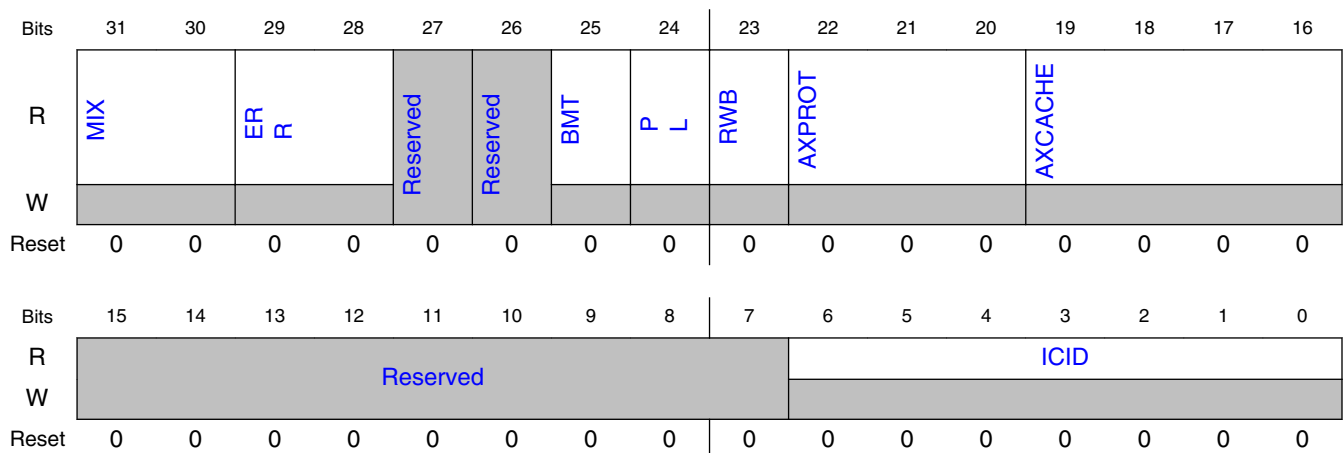
14.217.1 Offset

Register	Offset
REIR4RTIC	6_0E10h

14.217.2 Function

See the description for [Recoverable Error Interrupt Record 0 for RTIC \(REIR0RTIC\)](#).

14.217.3 Diagram



14.217.4 Fields

Field	Function
31-30 MIX	This field holds the memory interface index associated with the recoverable error.
29-28 ERR	This field holds the AXI error response associated with the recoverable error.
27 —	Reserved
26 —	Reserved
25 BMT	This field holds the bypass memory translation transaction attribute used for the memory access.
24 PL	This field holds the privilege level transaction attribute used for the memory access.
23 RWB	This field specifies whether the memory access was a read or write.
22-20 AXPROT	This field holds the AXI protection transaction attribute used for the memory access.
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15-7 —	Reserved
6-0 ICID	This field holds the ICID associated with the recoverable error.

14.218 Recoverable Error Interrupt Record 5 for RTIC (REIR5RTIC)

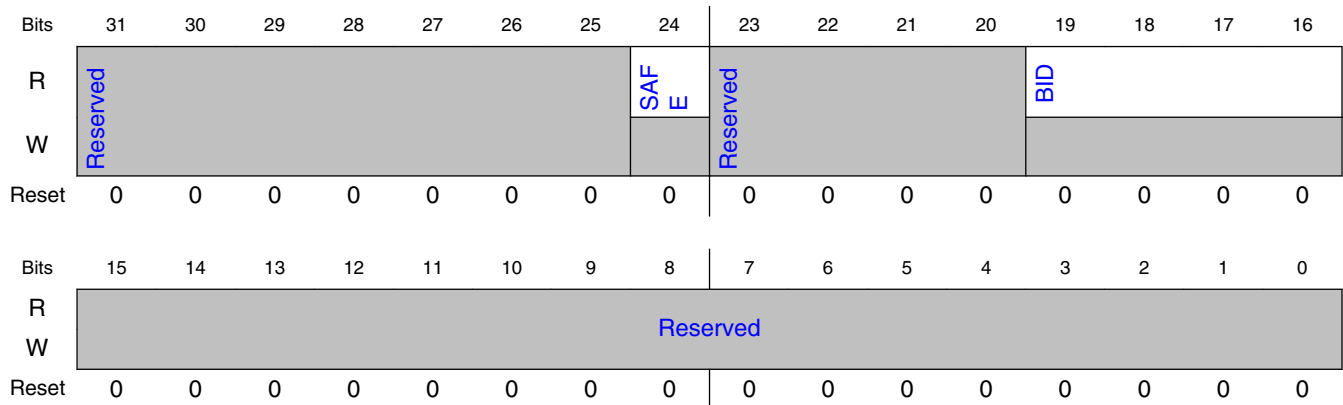
14.218.1 Offset

Register	Offset
REIR5RTIC	6_0E14h

14.218.2 Function

See the description for [Recoverable Error Interrupt Record 0 for RTIC \(REIR0RTIC\)](#).

14.218.3 Diagram



14.218.4 Fields

Field	Function
31-25 —	Reserved
24 SAFE	SAFE indicates whether the AXI transaction associated with the recoverable error was a "safe" transaction.
23-20 —	Reserved
19-16 BID	This field holds the block identifier (see Table 14-1) of the source of the AXI transaction associated with the recoverable error.
15-0 —	Reserved

14.219 Queue Interface Configuration Register (QICFG)

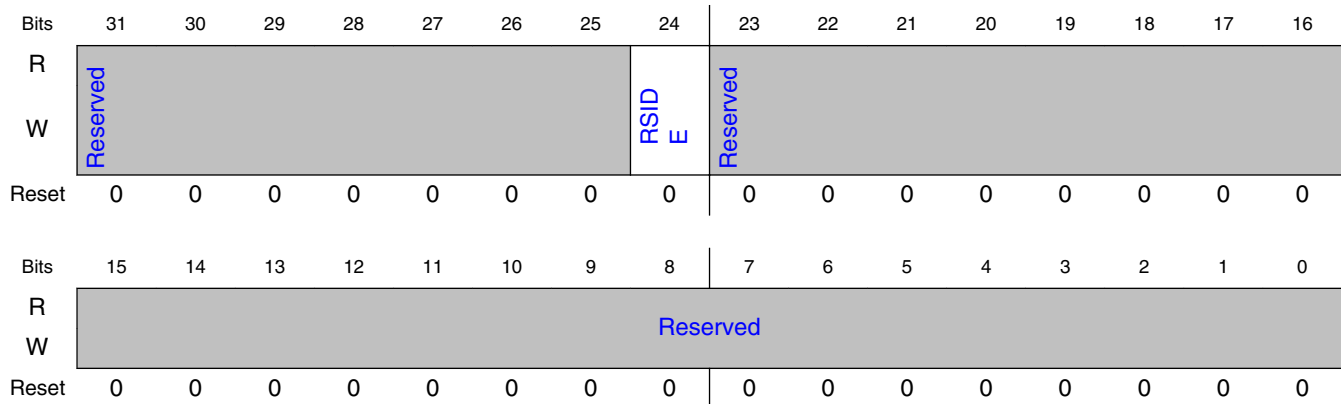
14.219.1 Offset

Register	Offset
QICFG	7_0000h

14.219.2 Function

Some characteristics of the Queue Interface are controlled via the Queue Interface Configuration Register.

14.219.3 Diagram



14.219.4 Fields

Field	Function
31-25 —	Reserved
24 RSIDE	Report SEC ID on Error. When RSIDE=1 and the job termination status is non-zero, the 32-bit SEC error status encoded in the FRC field of the enqueued Frame Descriptor is reformatted as follows: Bits 31:24 are moved to bits 23:16 (which were reserved). The SEC Source ID, 03h, is inserted in bits 30:24. If the job termination status indicates an error or a warning, bit 31 is 1, otherwise bit 31 is 0.
23-0 —	Reserved

14.220 Queue Interface Control Register (QICTL)

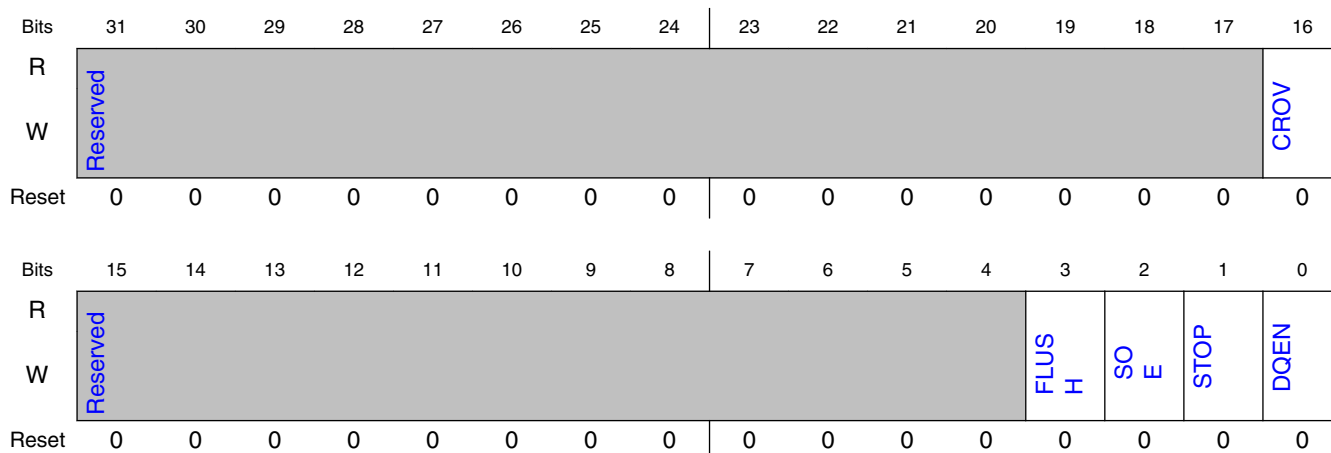
14.220.1 Offset

Register	Offset
QICTL	7_0004h

14.220.2 Function

The Queue Interface can be stopped and started, and certain other features of the Queue Interface can be configured using this register.

14.220.3 Diagram



14.220.4 Fields

Field	Function
31-17	Reserved
—	

Table continues on the next page...

Field	Function
16 CROV	Critical Resource Override. If set, critical resource loading is not considered when selecting the next job for transfer to the job queue controller for job scheduling. If clear, critical resource loading may affect job selection priority. If the Flow Context associated with a job specifies a critical resource ID and all instances of that resource are in use by jobs in other flows that were already transferred to the job queue controller, the job is assigned the lowest selection priority.
15-4 —	Reserved
3 FLUSH	Flush QI jobs. When FLUSH is set to 1, QI asserts a signal that tells the DECOs to terminate (with a DNR error status) any jobs currently executing. DQEN should be set to 0 to stop QI from issuing new dequeue commands. When all QI jobs have been enqueued, FLUSH is automatically reset to 0. If FLUSH is cleared by the user before some jobs are terminated, those jobs will finish normally.
2 SOE	Stop on error. If set, a QI error will cause QI to stop all operations, including frame dequeue and enqueue operations and transfer of jobs to the job queue controller. When internal operations reach a stable, stopped state, STOPDOE and STOPD in the QI Status Register will be 1 to indicate that QI is stopped due to an error. See the Queue Interface Status Register (see Section Queue Interface Status Register (QISTA)) for a description of the errors. If the REI registers are programmed to halt SEC after a recoverable error and that recoverable error occurs, this will cause the DEBUGCTL[STOP] bit to assert. SEC will acknowledge that the stop is complete by setting the DEBUGCTL[STOP_ACK] bit. The DEBUGCTL[STOP] bit must be cleared in order to restart SEC.
1 STOP	Stop. Write 1 to STOP to direct QI to stop all QI operations. When internal operations reach a stable, stopped state, STOPD in the QI Status Register will be 1. Writing a 1 a second time (with DQEN = 0) will flush all jobs and reset QI. QISTA[STOPD] will be 0 while the QI jobs are flushed. When all QI jobs have been enqueued QI will reset itself, restoring all registers to their default/reset state, including resetting QICTL[STOP] to 0.
0 DQEN	Dequeue enable. If set to 1 QI will issue dequeue commands to the Queue Manager.

14.221 Queue Interface Status Register (QISTA)

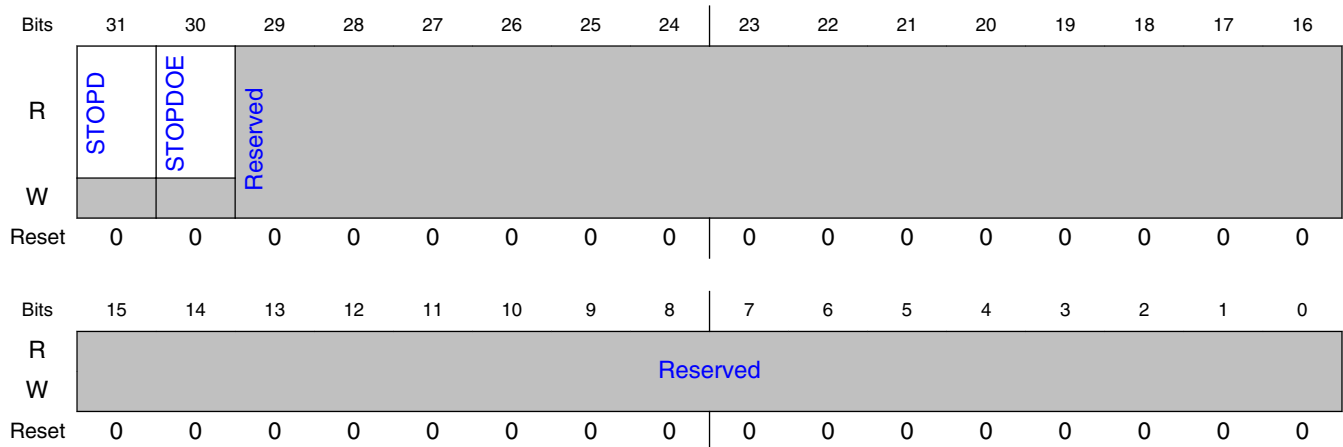
14.221.1 Offset

Register	Offset
QISTA	7_000Ch

14.221.2 Function

Software can determine the current status of the Queue Interface by reading the Queue Interface Status Register.

14.221.3 Diagram



14.221.4 Fields

Field	Function
31 STOPD	Stopped. Frame dequeue and enqueue operations and transfer of jobs to the Job Queue Controller has stopped. This condition will occur after the QICTL[STOP] bit is set or after an error bit is set with the QICTL[SOE] bit set. There may be a delay of several cycles from the time the stopping event occurs until STOPD is asserted, while Queue Interface state machine finishes current operations.
30 STOPDOE	Stopped on Error. STOPDOE is set to 1 when QI is stopped because QICTL[SOE] is set and error status is non-zero for one or more QI jobs. STOPDOE will be cleared to 0 when QICTL[SOE] is reset or there are no jobs with non-zero error status.
29-0 —	Reserved

14.222 Queue Interface Dequeue Configuration Register (QIDQC)

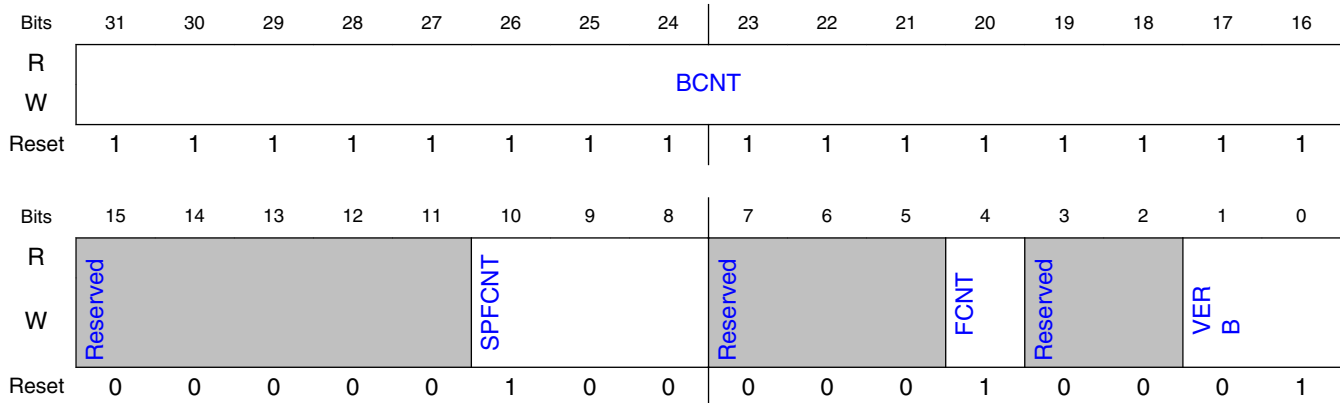
14.222.1 Offset

Register	Offset
QIDQC	7_0014h

14.222.2 Function

Queue Interface dequeue command parameters are specified with this register.

14.222.3 Diagram



14.222.4 Fields

Field	Function
31-16 BCNT	Dequeue Command byte count. This is the command byte count value used in Dequeue Commands to the Queue Manager. The default value of all 1's indicates that the byte count limit is not used. Smaller byte counts can be programmed to limit the number of large frames that are dequeued when the FCNT value is 1. Queue Manager will always dequeue at least one frame (unless there are no frames available). If the amount of data in the first frame is less than the requested byte count, and more frames are available, Queue Manager will provide a second frame in the dequeue response. If the amount of data in the first two frames is less than the requested byte count, and more frames are available, Queue Manager will provide a third frame in the dequeue response.
15-11 —	Reserved
10-8 SPFCNT	Subportal frame count threshold. Each dequeue command issued to the Queue Manager specifies a subportal ID. Queue Interface uses multiple subportals to get frames from different Frame Queues and it will only issue a Dequeue Command for a subportal if the number of frames being processed for that subportal is less than this frame count threshold. Setting this field to 0 will stop Dequeue Commands since the number of frames being processed is never less than 0.
7-5 —	Reserved
4 FCNT	Dequeue command frame count. This is the frame count value used in dequeue commands to the Queue Manager. See the Multi-core Datapath Acceleration Architecture Infrastructure Usage document for more detail on Frame Descriptors. 0b - Dequeue one Frame

Table continues on the next page...

Queue Interface Dequeue Event Filter Control Register 0 (QDQEFC0)

Field	Function
	1b - Dequeue up to three Frames. (QMan may supply less than three based on availability and FQ configuration)
3-2 —	Reserved
1-0 VERB	Dequeue command verb. These are the two least-significant bits of the verb used in dequeue commands to the Queue Manager. The default value indicates that the Queue Manager should perform a scheduled dequeue from the channel dedicated to SEC with priority precedence. For normal operation, only the default value should be programmed. Other values are allowed for test and debug. Note that unscheduled dequeues are not supported.

14.223 Queue Interface Dequeue Event Filter Control Register 0 (QDQEFC0)

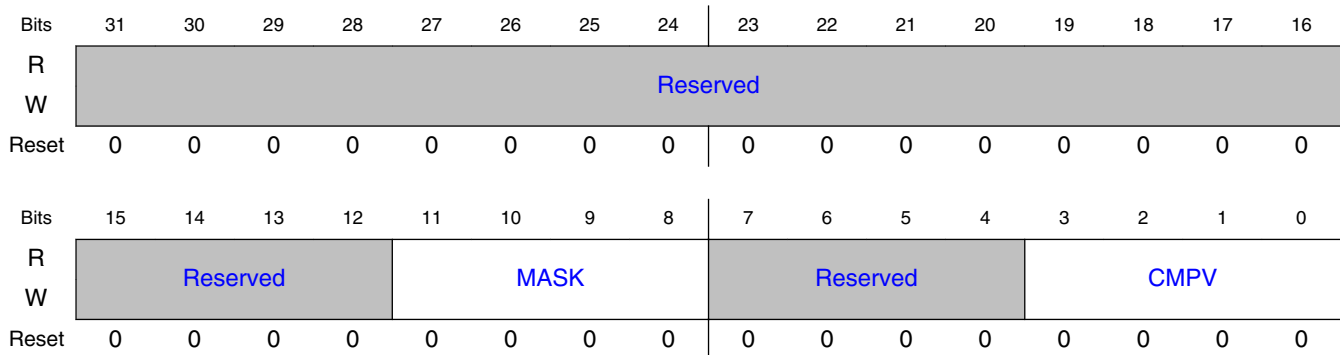
14.223.1 Offset

Register	Offset
QDQEFC0	7_0030h

14.223.2 Function

Queue Interface Dequeue Event Filter Control Register 0. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 0.

14.223.3 Diagram



14.223.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame's current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame's Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.224 Queue Interface Dequeue Event Filter Control Register 1 (QDQEFC1)

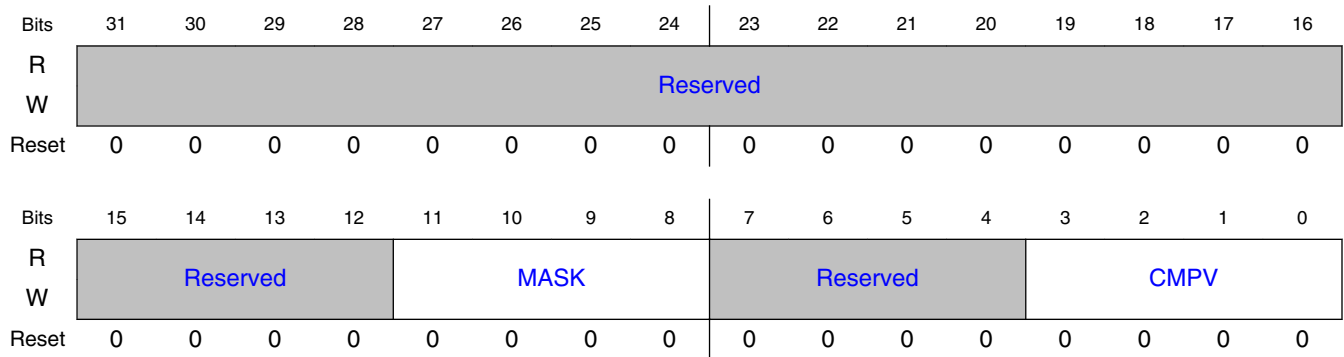
14.224.1 Offset

Register	Offset
QDQEFC1	7_0034h

14.224.2 Function

Queue Interface Dequeue Event Filter Control Register 1. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 1.

14.224.3 Diagram



14.224.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.225 Queue Interface Dequeue Event Filter Control Register 2 (QDQEFC2)

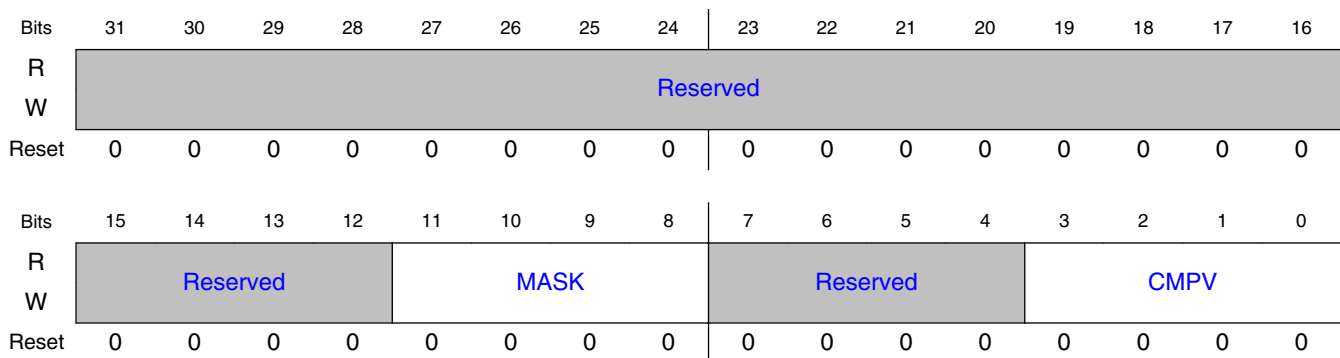
14.225.1 Offset

Register	Offset
QDQEFC2	7_0038h

14.225.2 Function

Queue Interface Dequeue Event Filter Control Register 2. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 2.

14.225.3 Diagram



14.225.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame's current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame's Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.226 Queue Interface Enqueue Event Filter Control Register 0 (QEQEFC0)

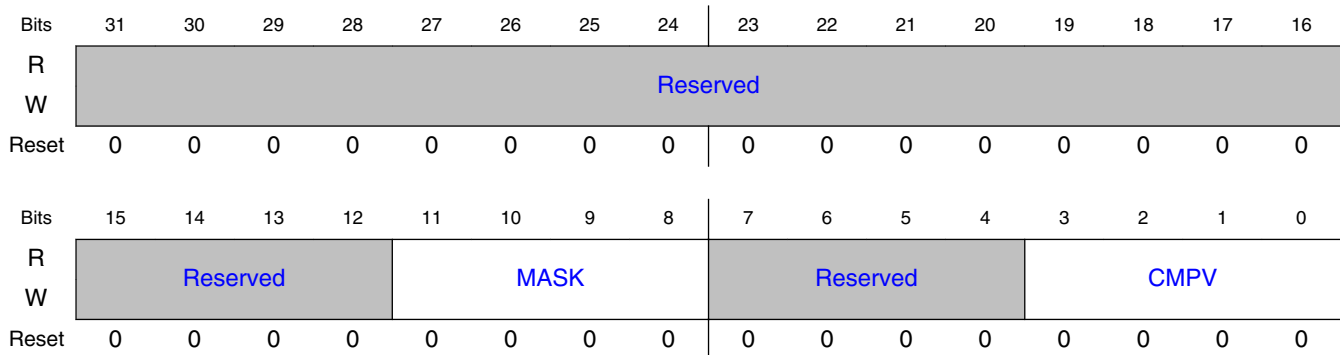
14.226.1 Offset

Register	Offset
QEQEFC0	7_0050h

14.226.2 Function

Queue Interface Enqueue Event Filter Control Register 0. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 3.

14.226.3 Diagram



14.226.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame's current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame's Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.227 Queue Interface Enqueue Event Filter Control Register 1 (QEQEFC1)

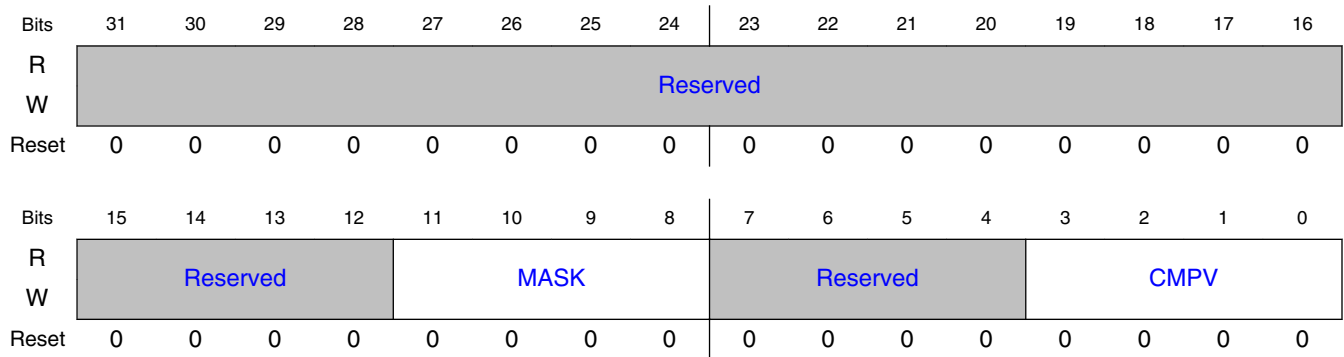
14.227.1 Offset

Register	Offset
QEQEFC1	7_0054h

14.227.2 Function

Queue Interface Enqueue Event Filter Control Register 1. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 4.

14.227.3 Diagram



14.227.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame’s current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame’s Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame’s Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame’s DD after masking.

14.228 Queue Interface Enqueue Event Filter Control Register 2 (QEQEFC2)

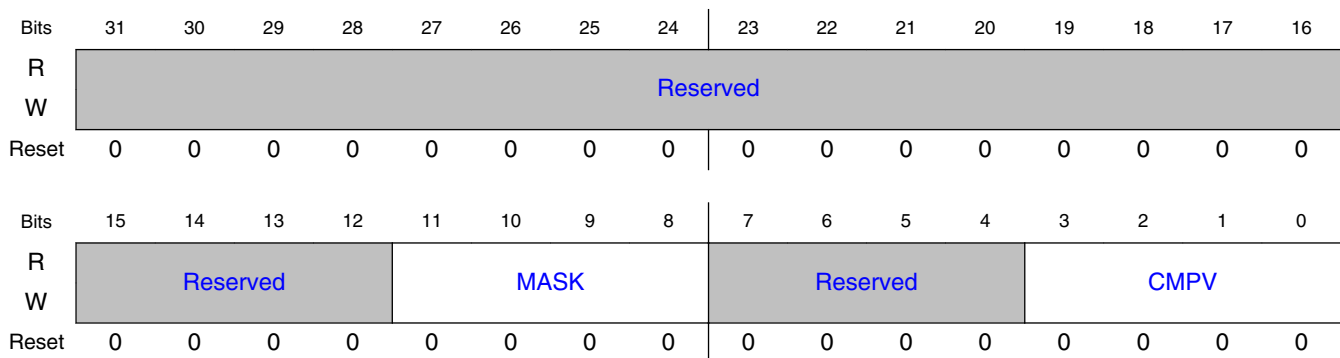
14.228.1 Offset

Register	Offset
QEQEFC2	7_0058h

14.228.2 Function

Queue Interface Enqueue Event Filter Control Register 2. Many of the Layerscape datapath blocks have the ability to generate datapath event signals (i.e. “performance monitor” signals) that are not only used for traditional performance counting of useful events but also for generating triggers into the Layerscape debug system. Although many datapath event signals are generated independent of the dynamic debug mark, it is often useful to have a small class of signals which are generated only on behalf of frames matching a particular mark using a simple criteria. This register is used to control signaling on QI Performance Monitor event bus bit 5.

14.228.3 Diagram



14.228.4 Fields

Field	Function
31-12 —	Reserved
11-8 MASK	Mask value. Used for masking a frame's current Dynamic Debug Mark value before being compared. For each bit in this field: 0 Corresponding bit in Frame's Dynamic Debug Mark is masked when performing comparison 1 Corresponding bit in Frame's Dynamic Debug Mark is compared against the value in CMPV When all bits in the MASK field are zeros, a match will not be detected.
7-4 —	Reserved
3-0 CMPV	Compare value. Provides the value to be compared against the current frame's DD after masking.

14.229 Jobs in Use Register for QM Interface (JOBS_IN_USE_QI)

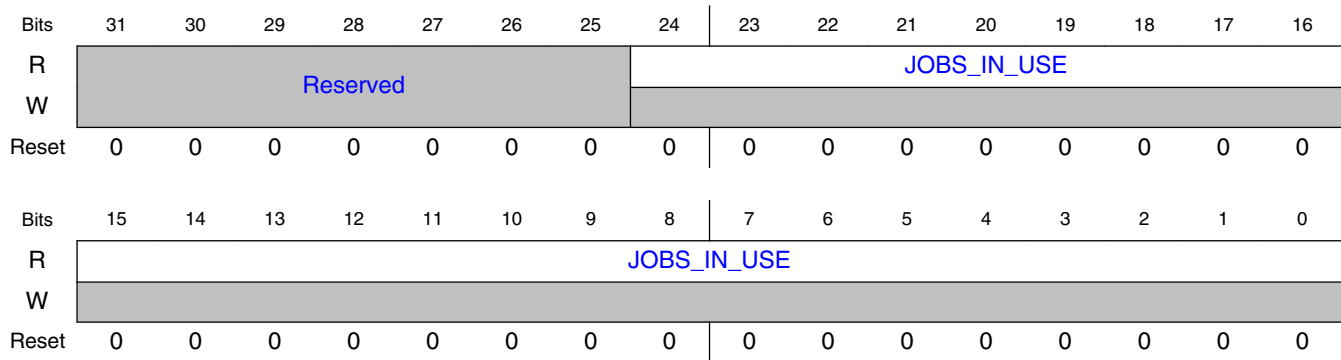
14.229.1 Offset

Register	Offset
JOBS_IN_USE_QI	7_0500h

14.229.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_IN_USE_QI register indicates whether certain job buffers are in use for active QI jobs. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.229.3 Diagram



14.229.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_IN_USE	The bits in this field indicate whether certain job buffers are in use for active QI jobs. Bit n indicates the active/inactive status of job buffer n.) Bit n=0 means the job buffer is not in use. Bit n=1 means there is an active job in the job buffer. Note that there is no job buffer 0.

14.230 Jobs Ready Register for QM Interface (JOBS_READY_QI)

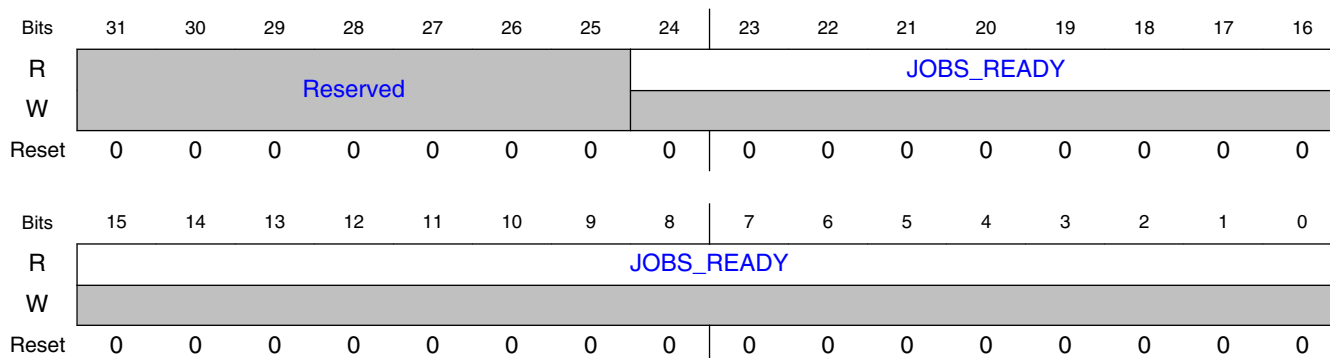
14.230.1 Offset

Register	Offset
JOBS_READY_QI	7_0508h

14.230.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_READY_QI register indicates whether certain QI job buffers contain jobs that are ready for transfer to the job queue controller. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.230.3 Diagram



14.230.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_READY	The bits in this field indicate whether certain job buffers hold QI jobs that are ready for transfer to the job queue controller. Bit n indicates the job-ready status of job buffer n.) Bit n=1 means that the job in the job buffer is ready for transfer. Bit n=0 means that the job is not ready or the job buffer is not in use. Note that there is no job buffer 0.

14.231 Jobs Transfer Blocking Disabled Register for QM Interface (JOBS_XFR_BLK_DIS_QI)

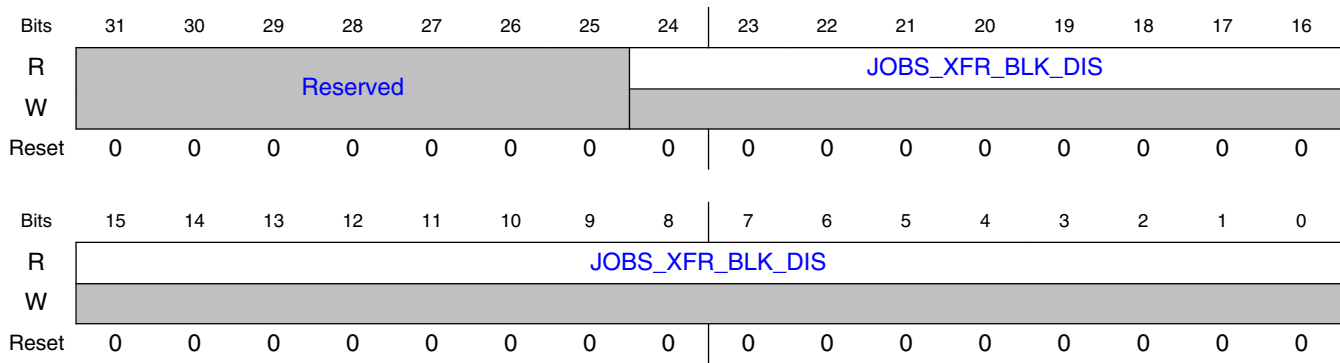
14.231.1 Offset

Register	Offset
JOBS_XFR_BLK_DIS_QI	7_0510h

14.231.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_XFR_BLK_DIS_QI register indicates whether certain QI job buffers contain jobs that have transfer blocking disabled. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.231.3 Diagram



14.231.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_XFR_BLK_DIS	The bits in this field indicate whether certain QI job buffers hold jobs with transfer blocking disabled. Bit n indicates the transfer-blocking-disabled status of job buffer n.) Bit n=1 means that the job in the job buffer has transfer blocking disabled. Bit n=0 means that the job does not have transfer blocking disabled or the job buffer is not in use. Note that there is no job buffer 0.

14.232 Jobs Transferred Register for QM Interface (JOBS_XFRD_QI)

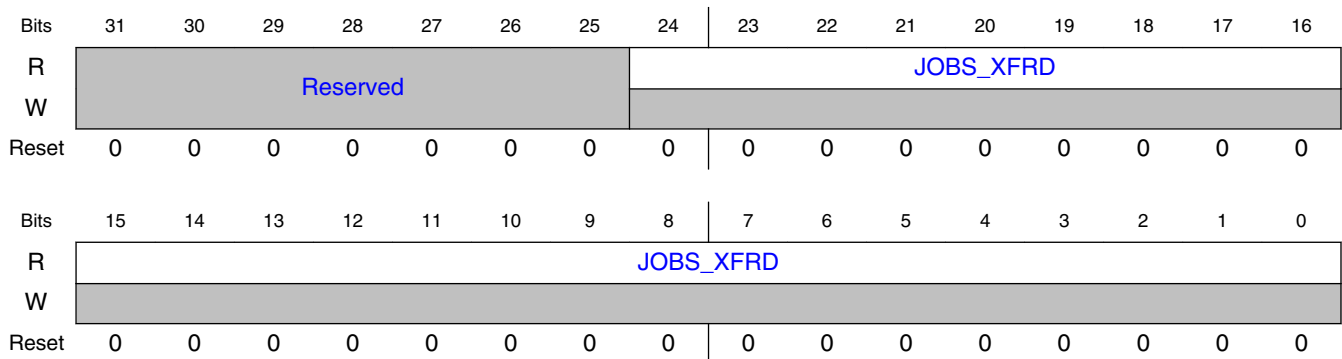
14.232.1 Offset

Register	Offset
JOBS_XFRD_QI	7_0518h

14.232.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_XFRD_QI register indicates whether certain QI job buffers contain jobs that have been transferred to the job queue controller. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.232.3 Diagram



14.232.4 Fields

Field	Function
31-25	Reserved

Table continues on the next page...

Field	Function
—	
24-0 JOBS_XFRD	The bits in this field indicate whether certain job buffers hold QI jobs that have been transferred to the job controller. Bit n indicates the transferred status of job buffer n.) Bit n=1 means that the job in the job buffer has been transferred. Bit n=0 means that the job has not been transferred or the job buffer is not in use. Note that there is no job buffer 0.

14.233 Jobs Executing Register for QM Interface (JOBS_EXEC_QI)

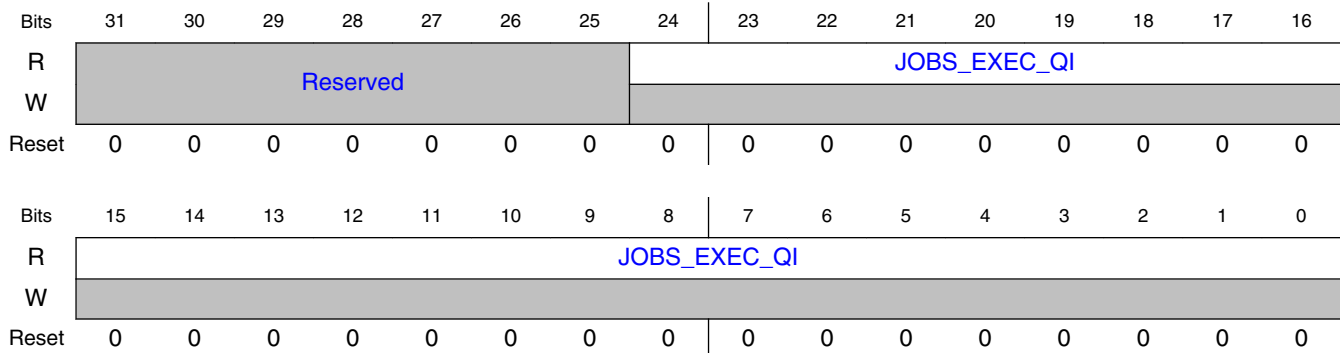
14.233.1 Offset

Register	Offset
JOBS_EXEC_QI	7_0520h

14.233.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_EXEC_QI register indicates whether certain QI job buffers contain jobs that are currently executing in a DECO. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.233.3 Diagram



14.233.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_EXEC_Q I	The bits in this field indicate whether certain job buffers hold QI jobs that are currently executing in a DECO. Bit n indicates the executing status of job buffer n.) Bit n=1 means that the job in the job buffer is executing. Bit n=0 means that the job is not executing or the job buffer is not in use. Note that there is no job buffer 0.

14.234 Jobs Done Register for QM Interface (JOBS_DONE_QI)

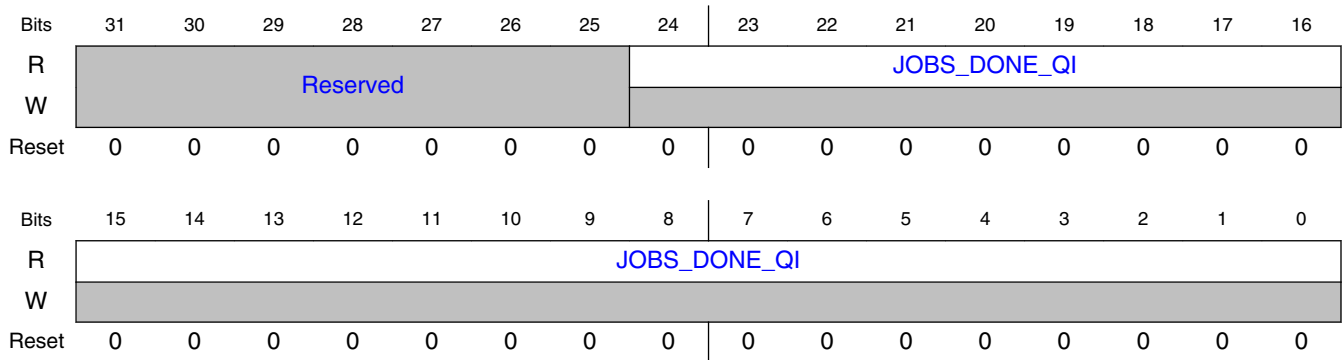
14.234.1 Offset

Register	Offset
JOBS_DONE_QI	7_0528h

14.234.2 Function

Data for each active job managed by QI is stored in a group of registers called a job buffer. Each job buffer has a unique identifier from 1 through 24, which can be used to refer to the job buffer. The JOBS_DONE_QI register indicates whether certain QI job buffers contain jobs that are done executing. Once set, these bits remain set until the job is done and the job buffer is freed for use by another job.

14.234.3 Diagram



14.234.4 Fields

Field	Function
31-25 —	Reserved
24-0 JOBS_DONE_QI	The bits in this field indicate whether certain job buffers hold QI jobs that are done. Bit n indicates the done status of job buffer n.) Bit n=1 means that the job in the job buffer is done. Bit n=0 means that the job is not done or the job buffer is not in use. Note that there is no job buffer 0.

14.235 Subportal Frame Count Register for Queue Interface (SUBPORT_FC_QI)

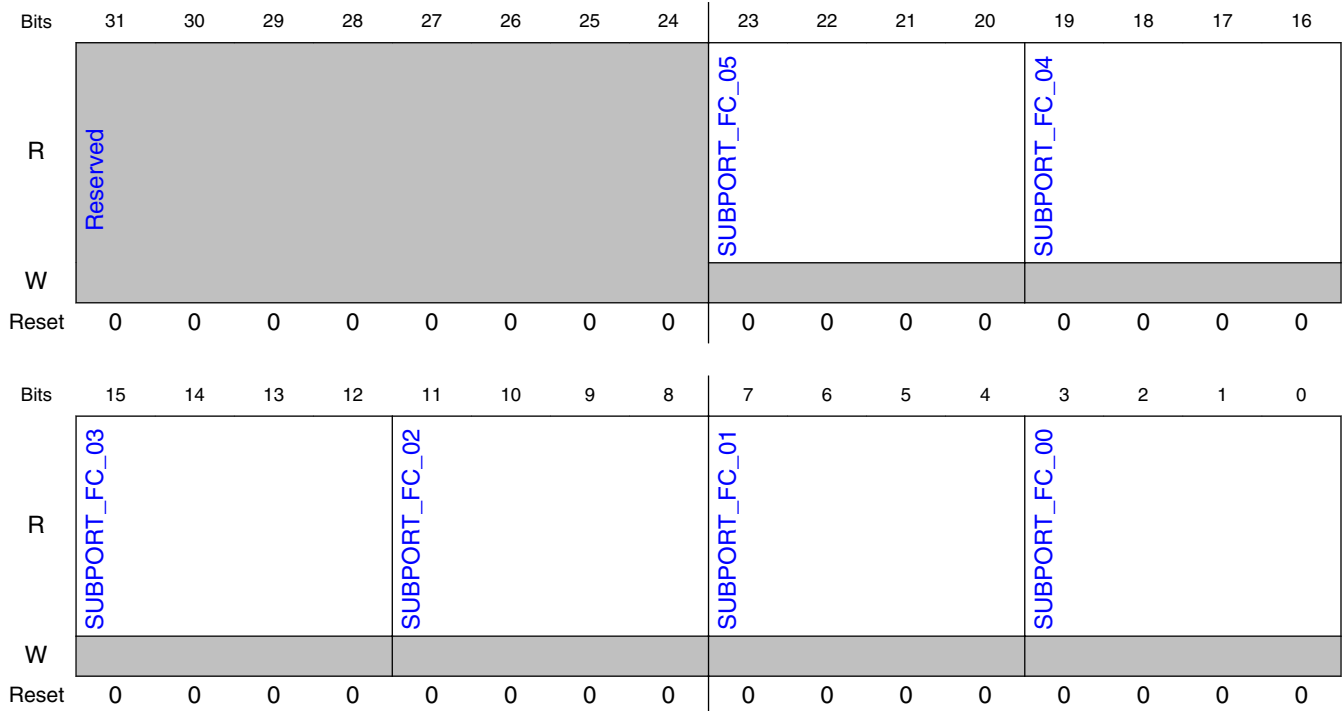
14.235.1 Offset

Register	Offset
SUBPORT_FC_QI	7_0530h

14.235.2 Function

This register shows the number of active frames in QI dequeued from each QMan subportal. The contents of this register change with every frame dequeue and enqueue.

14.235.3 Diagram



14.235.4 Fields

Field	Function
31-24 —	Reserved
23-20 SUBPORT_FC_05	SUBPORT_FC_05 shows frame count for subport 0x5. A value of 0xF indicates a frame count of 15 or more.
19-16 SUBPORT_FC_04	SUBPORT_FC_04 shows frame count for subport 0x4. A value of 0xF indicates a frame count of 15 or more.
15-12 SUBPORT_FC_03	SUBPORT_FC_03 shows frame count for subport 0x3. A value of 0xF indicates a frame count of 15 or more.
11-8 SUBPORT_FC_02	SUBPORT_FC_02 shows frame count for subport 0x2. A value of 0xF indicates a frame count of 15 or more.
7-4	SUBPORT_FC_01 shows frame count for subport 0x1. A value of 0xF indicates a frame count of 15 or more.

Table continues on the next page...

Field	Function
SUBPORT_FC_01	
3-0 SUBPORT_FC_00	SUBPORT_FC_00 shows frame count for subportal 0x0. A value of 0xF indicates a frame count of 15 or more.

14.236 Job Select Register for QI Interface (JOB_SELECT_Q I)

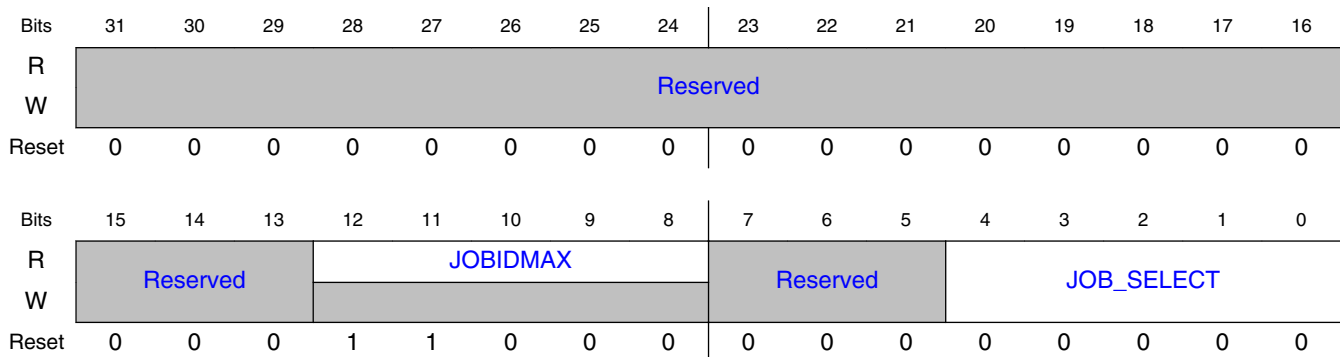
14.236.1 Offset

Register	Offset
JOB_SELECT_QI	7_0570h

14.236.2 Function

This register is used to specify the job for which status is displayed in the QI Job Data Registers. For example, to get the address of the dequeued FD for QI job n, write n to the JOB_SELECT field of this register and then read the Job Data Registers 9 and 10. This register also contains a read-only field that defines the maximum job ID for QI jobs.

14.236.3 Diagram



14.236.4 Fields

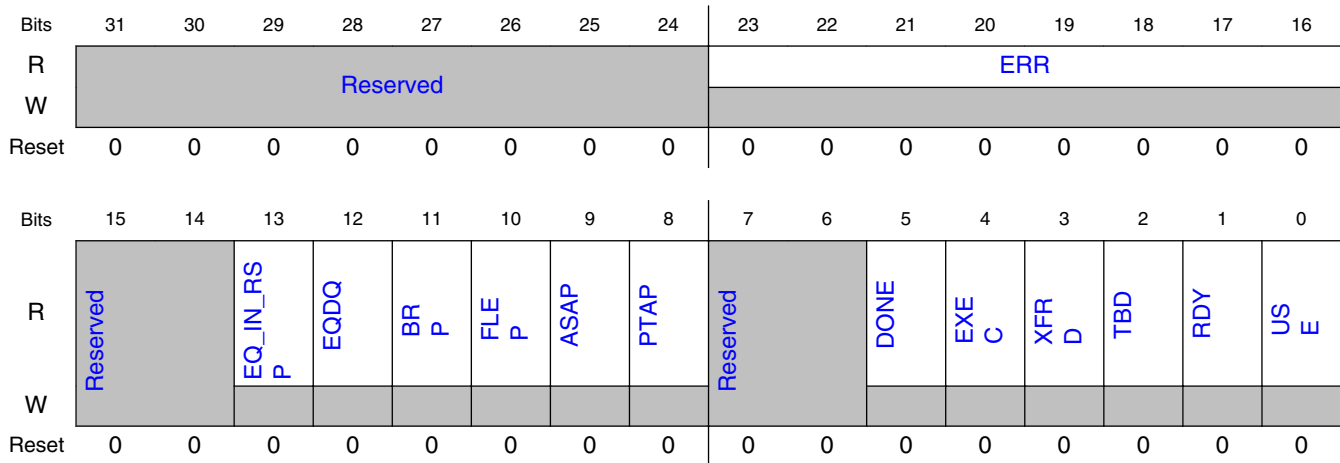
Field	Function
31-13 —	Reserved.
12-8 JOBIDMAX	JOBIDMAX specifies the maximum job ID for QI jobs.
7-5 —	Reserved.
4-0 JOB_SELECT	Bits 0 through 4 select the job for which data is displayed in the QI Job Data Registers. As jobs are processed, the contents of these registers can change at any time. To get a consistent snapshot of all the data for a job, QI should be stopped when the Job Data Registers are accessed. This can be done by setting QICTL[STOP] and then waiting until QISTA[STOPD] is 1. Writing the value 0 or a value greater than JOBIDMAX to the JOB_SELECT field may cause garbage data to be displayed in the Job Data Registers.

14.237 Queue Interface Job Data Register 0 (QIJOB0)

14.237.1 Offset

Register	Offset	Description
QIJOB0	7_0574h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.237.2 Diagram



14.237.3 Fields

Field	Function
31-24 —	Reserved.
23-16 ERR	If this field is 00h, no error has occurred for this job. If the field is not 00h, this field contains an error code for this job. The codes in this field have the same definition as the least-significant 8 bits of Job termination status word, Source 5h (QI) .
15-14 —	Reserved.
13 EQ_IN_RSP	If 1, enqueue frame using input partition response parameters for this job.
12 EQDQ	If 1, enqueued dequeued frame for this job.
11 BRP	If 1, a buffer release is pending for this job.
10 FLEP	If 1, an FLE write is pending for this job.
9 ASAP	If 1, an ASA write is pending for this job.
8 PTAP	If 1, the PTA copy is pending for this job.
7-6 —	Reserved.
5	If 1, this job is done.

Table continues on the next page...

Queue Interface Job Data Register 1 (QIJOB1)

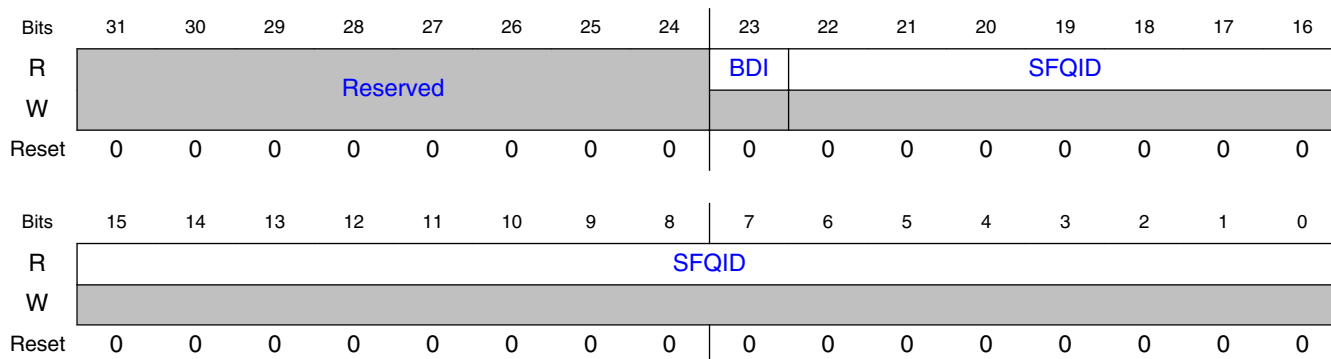
Field	Function
DONE	
4 EXEC	If 1, this job is executing.
3 XFRD	If 1, this job has been transferred to the Job Controller.
2 TBD	If 1, this job has transfer blocking disabled.
1 RDY	If 1, this job is ready for transfer to the Job Controller.
0 USE	If 1, the selected QI job buffer is currently in use.

14.238 Queue Interface Job Data Register 1 (QIJOB1)

14.238.1 Offset

Register	Offset	Description
QIJOB1	7_0578h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.238.2 Diagram



14.238.3 Fields

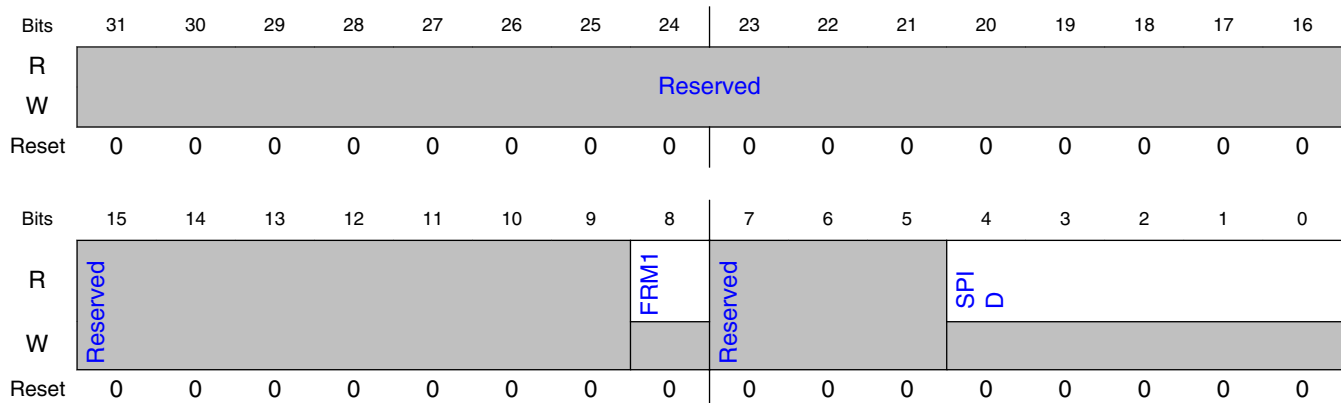
Field	Function
31-24 —	Reserved.
23 BDI	If 1, datapath isolation is bypassed for this job.
22-0 SFQID	The ID of the Source Frame Queue for this job.

14.239 Queue Interface Job Data Register 2 (QIJOB2)

14.239.1 Offset

Register	Offset	Description
QIJOB2	7_057Ch	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.239.2 Diagram



14.239.3 Fields

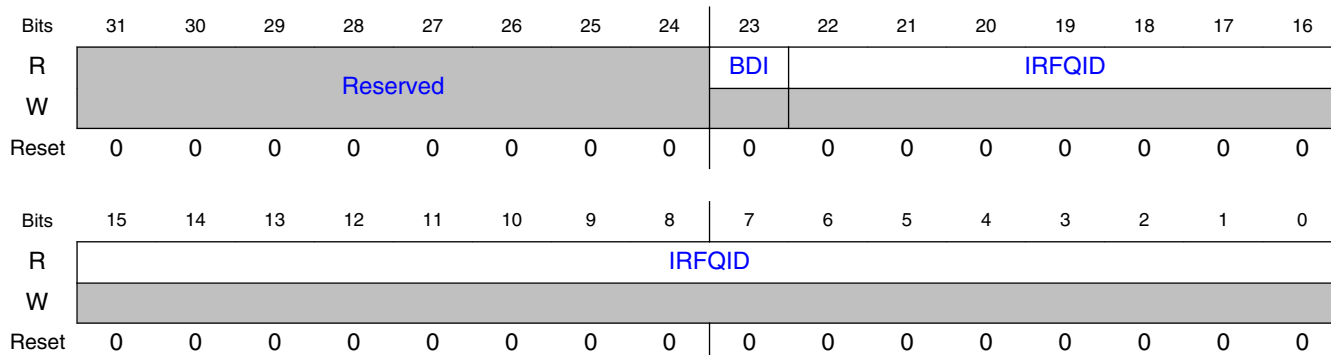
Field	Function
31-9 —	Reserved.
8 FRM1	The frame is first in a dequeue for this job.
7-5 —	Reserved.
4-0 SPID	The ID of the Subportal for this job.

14.240 Queue Interface Job Data Register 3 (QIJOB3)

14.240.1 Offset

Register	Offset	Description
QIJOB3	7_0580h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.240.2 Diagram



14.240.3 Fields

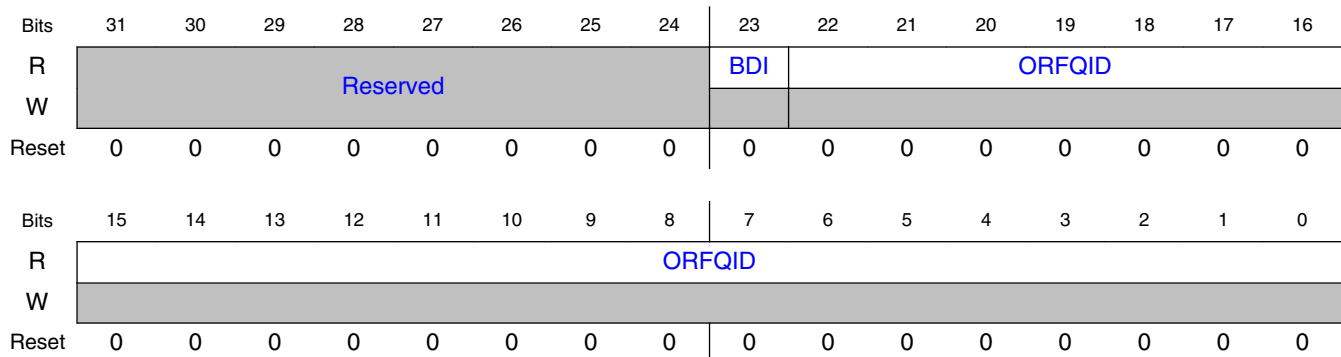
Field	Function
31-24 —	Reserved.
23 BDI	If 1, datapath isolation is bypassed for this job.
22-0 IRFQID	The ID of the Input Partition Return Frame Queue for this job.

14.241 Queue Interface Job Data Register 4 (QIJOB4)

14.241.1 Offset

Register	Offset	Description
QIJOB4	7_0584h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.241.2 Diagram



14.241.3 Fields

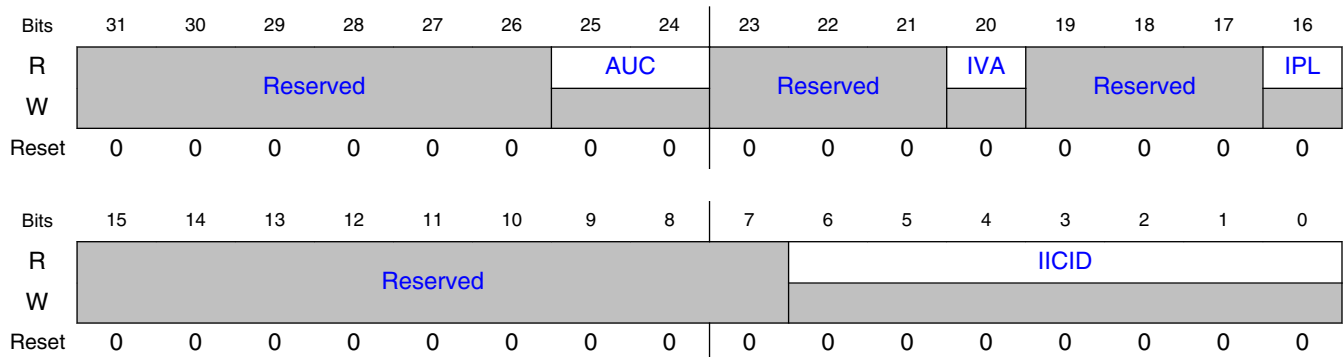
Field	Function
31-24 —	Reserved.
23 BDI	If 1, datapath isolation is bypassed for this job.
22-0 ORFQID	The ID of the Output Partition Return Frame Queue for this job.

14.242 Queue Interface Job Data Register 5 (QIJOB5)

14.242.1 Offset

Register	Offset	Description
QIJOB5	7_0588h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.242.2 Diagram



14.242.3 Fields

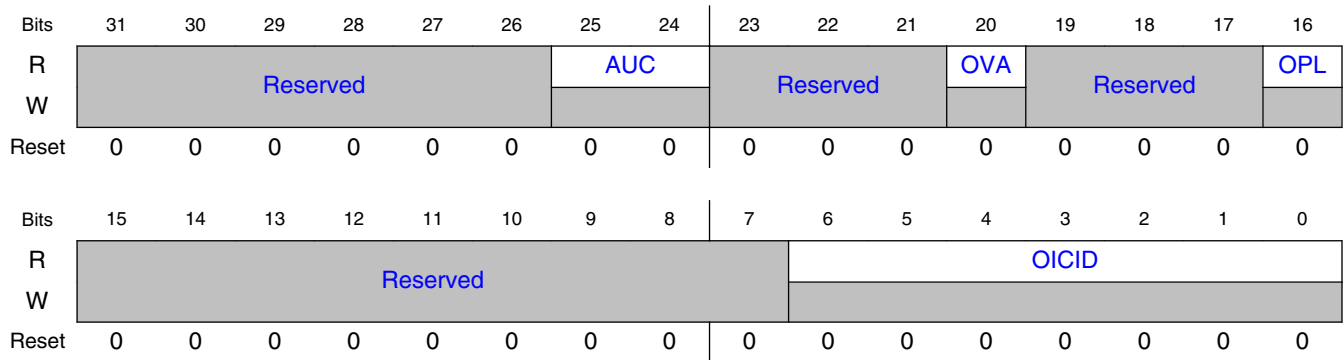
Field	Function
31-26 —	Reserved.
25-24 AUC	The Input Partition AMQ Usage Control for this job.
23-21 —	Reserved.
20 IVA	The Input Partition virtual address enable? for this job.
19-17 —	Reserved.
16 IPL	The Input Partition ICID for this job.
15-7 —	Reserved.
6-0 IICID	The ICID of the Input Partition for this job.

14.243 Queue Interface Job Data Register 6 (QIJOB6)

14.243.1 Offset

Register	Offset	Description
QIJOB6	7_058Ch	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.243.2 Diagram



14.243.3 Fields

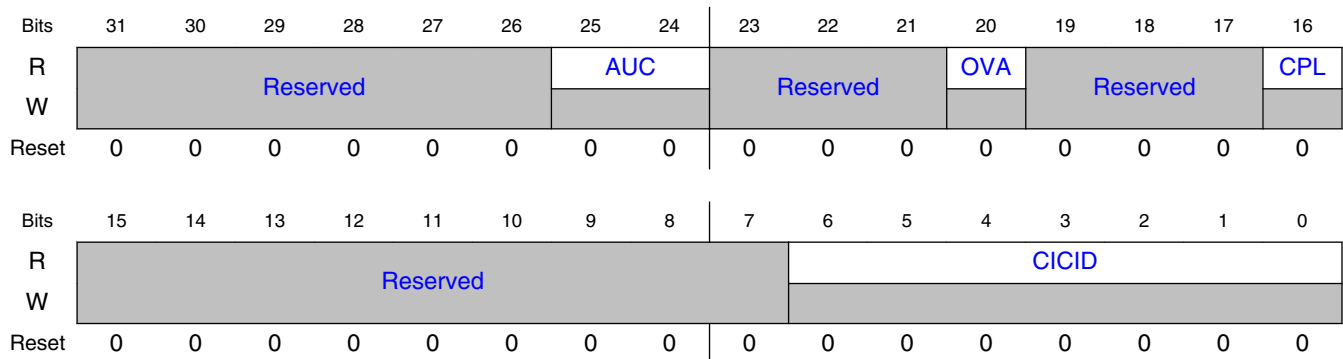
Field	Function
31-26 —	Reserved.
25-24 AUC	The Output Partition AMQ Usage Control for this job.
23-21 —	Reserved.
20 OVA	The Output Partition virtual address enable? for this job.
19-17 —	Reserved.
16 OPL	The Output Partition Privilege Level for this job.
15-7 —	Reserved.
6-0 OICID	The Output Partition ICID for this job.

14.244 Queue Interface Job Data Register 7 (QIJOB7)

14.244.1 Offset

Register	Offset	Description
QIJOB07	7_0590h	This register is used for debugging Queue Interface jobs. Registers QIJOB00 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.244.2 Diagram



14.244.3 Fields

Field	Function
31-26 —	Reserved.
25-24 AUC	The Control Partition AMQ Usage Control for this job.
23-21 —	Reserved.
20 OVA	The Control Partition virtual address enable? for this job.
19-17 —	Reserved.
16 CPL	The Control Partition Privilege Level for this job.
15-7 —	Reserved.

Table continues on the next page...

Queue Interface Job Data Register 8 (QIJOB8)

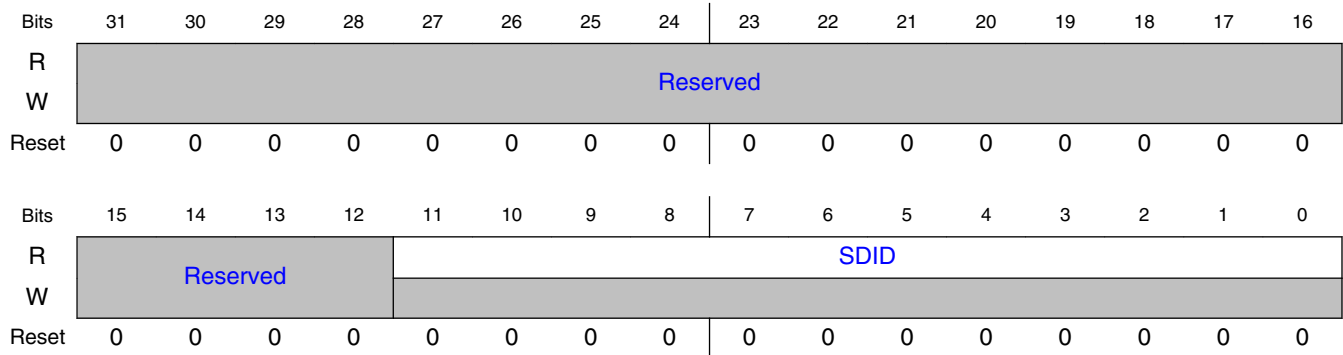
Field	Function
6-0 CICID	The Control Partition ICID for this job.

14.245 Queue Interface Job Data Register 8 (QIJOB8)

14.245.1 Offset

Register	Offset	Description
QIJOB8	7_0594h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.245.2 Diagram



14.245.3 Fields

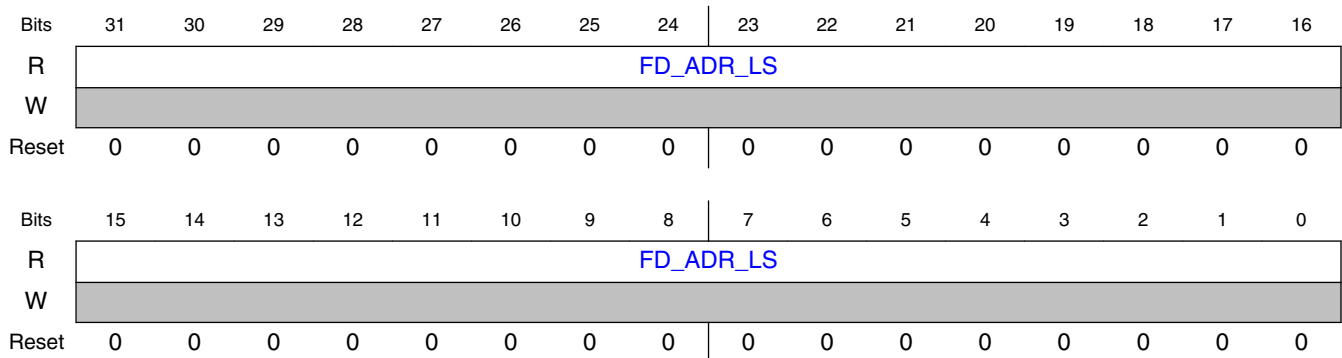
Field	Function
31-12 —	Reserved.
11-0 SDID	The Security Domain Identifier (SDID) for this job.

14.246 Queue Interface Job Data Register 9 (QIJOB9)

14.246.1 Offset

Register	Offset	Description
QIJOB9	7_0598h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.246.2 Diagram



14.246.3 Fields

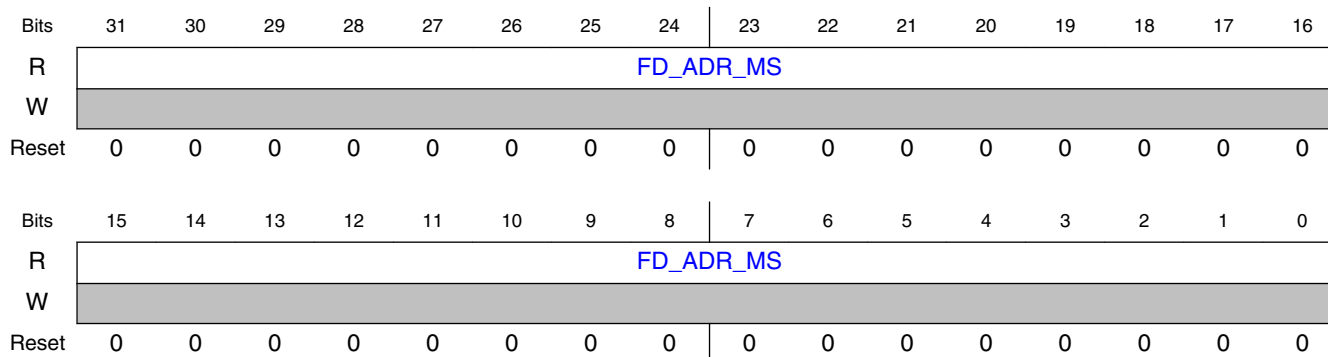
Field	Function
31-0 FD_ADR_LS	The least-significant 32 bits of the address of the dequeued frame for this job.

14.247 Queue Interface Job Data Register 10 (QIJOB10)

14.247.1 Offset

Register	Offset	Description
QIJOB10	7_059Ch	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.247.2 Diagram



14.247.3 Fields

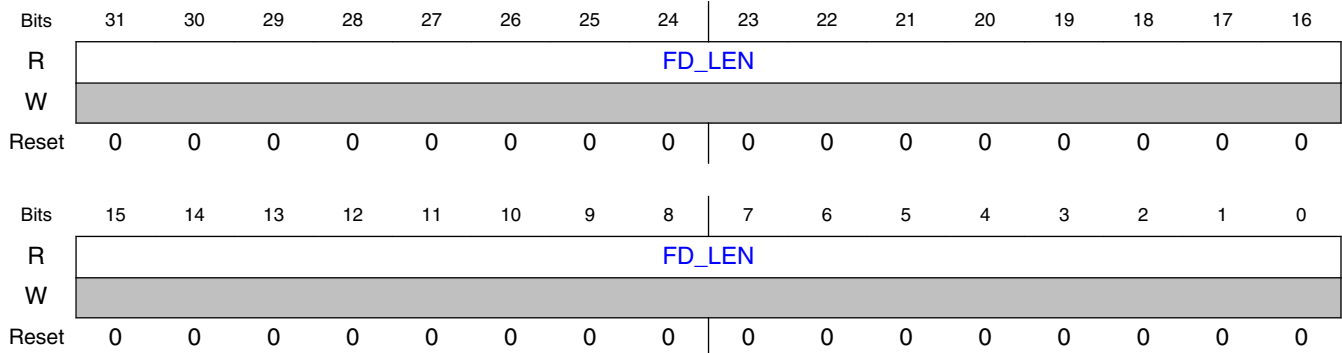
Field	Function
31-0 FD_ADR_MS	The most-significant 32 bits of the address of the dequeued frame for this job.

14.248 Queue Interface Job Data Register 11 (QIJOB11)

14.248.1 Offset

Register	Offset	Description
QIJOB11	7_05A0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.248.2 Diagram



14.248.3 Fields

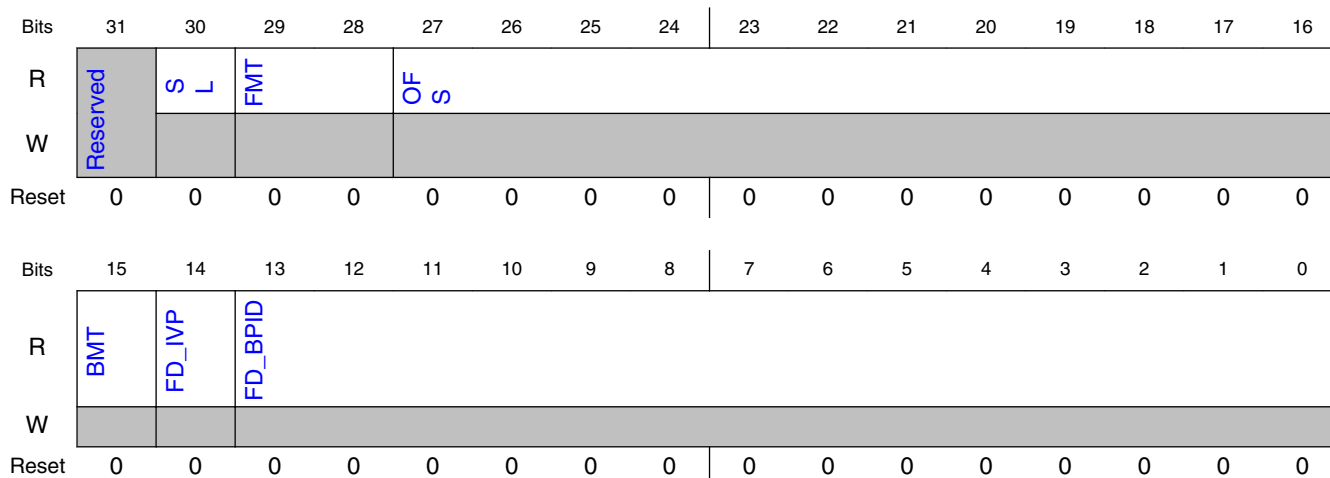
Field	Function
31-0 FD_LEN	The length of the dequeued frame for this job.

14.249 Queue Interface Job Data Register 12 (QIJOB12)

14.249.1 Offset

Register	Offset	Description
QIJOB12	7_05A4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.249.2 Diagram



14.249.3 Fields

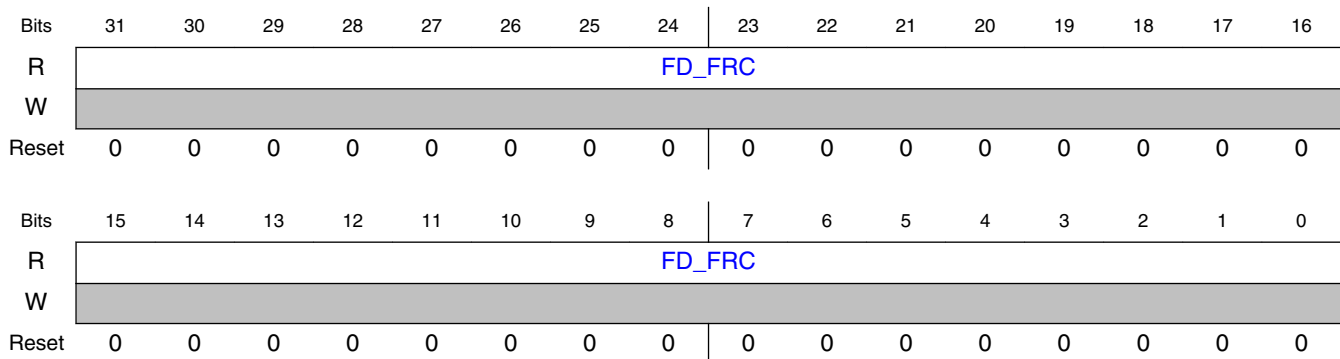
Field	Function
31 —	Reserved
30 SL	The length of the dequeued frame for this job is in short-length form.
29-28 FMT	The format of the dequeued frame for this job.
27-16 OFS	The data offset for the dequeued frame for this job.
15 BMT	If 1, memory address translation is bypassed for this job.
14 FD_IVP	If 1, the dequeued frame for this job is invalid.
13-0 FD_BPID	The Buffer Pool ID for this job.

14.250 Queue Interface Job Data Register 13 (QIJOB13)

14.250.1 Offset

Register	Offset	Description
QIJOB13	7_05A8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.250.2 Diagram



14.250.3 Fields

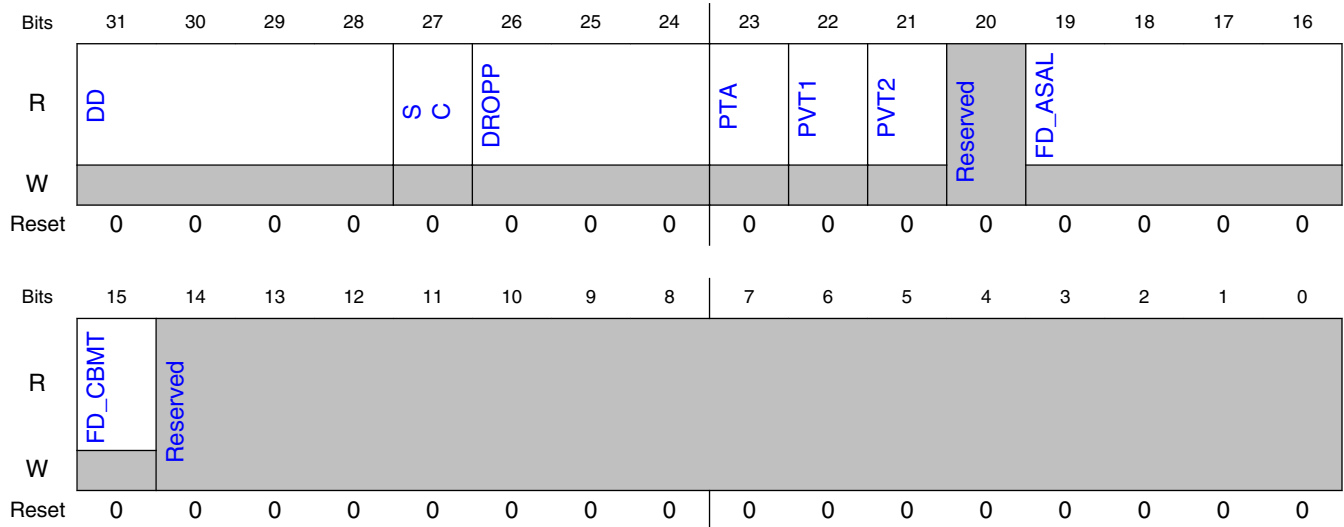
Field	Function
31-0 FD_FRC	The context for the dequeued frame for this job.

14.251 Queue Interface Job Data Register 14 (QIJOB14)

14.251.1 Offset

Register	Offset	Description
QIJOB14	7_05ACh	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.251.2 Diagram



14.251.3 Fields

Field	Function
31-28 DD	The dynamic debug mark for the dequeued frame for this job.
27 SC	The stashing control for the dequeued frame for this job.
26-24 DROPP	The drop priority for the dequeued frame for this job.
23 PTA	Pass-Through Annotation for the dequeued frame for this job.
22 PVT1	If 1, the dequeued frame pass-through annotation 1 is valid for this job.
21 PVT2	If 1, the dequeued frame pass-through annotation 2 is valid for this job.
20 —	Reserved
19-16 FD_ASAL	The length of the dequeued frame accelerator-specific annotation.
15 FD_CBMT	If 1, bypass memory address translation for the dequeued frame flow context for this job.

Table continues on the next page...

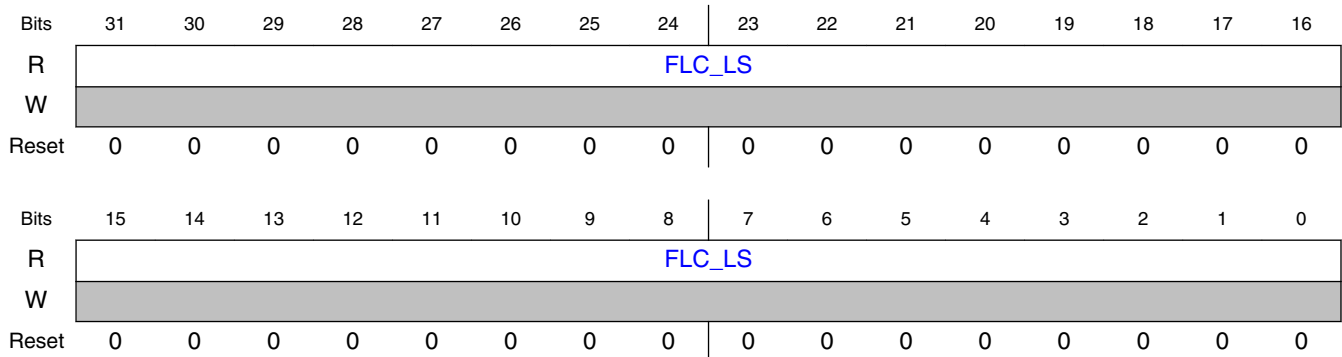
Field	Function
14-0 —	Reserved

14.252 Queue Interface Job Data Register 15 (QIJOB15)

14.252.1 Offset

Register	Offset	Description
QIJOB15	7_05B0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.252.2 Diagram



14.252.3 Fields

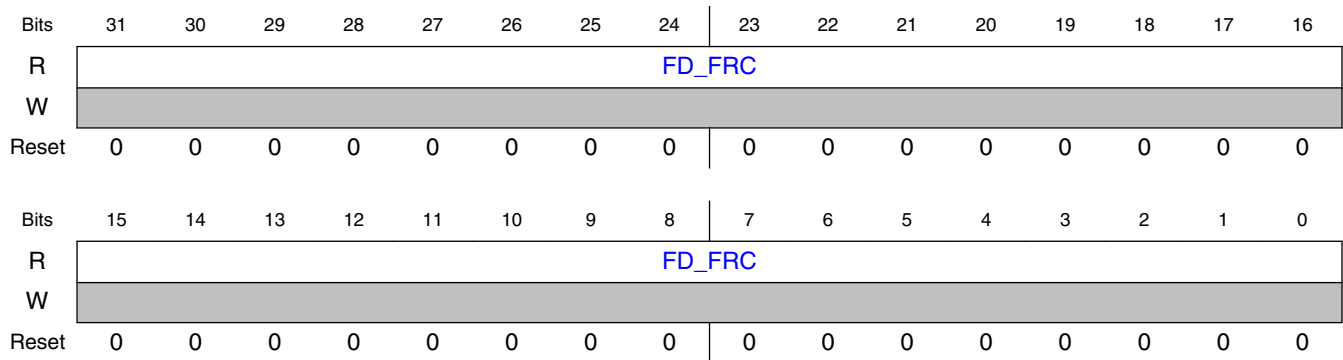
Field	Function
31-0 FLC_LS	The least-significant 32 bits of the flow context for the dequeued frame for this job.

14.253 Queue Interface Job Data Register 16 (QIJOB16)

14.253.1 Offset

Register	Offset	Description
QIJOB16	7_05B4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.253.2 Diagram



14.253.3 Fields

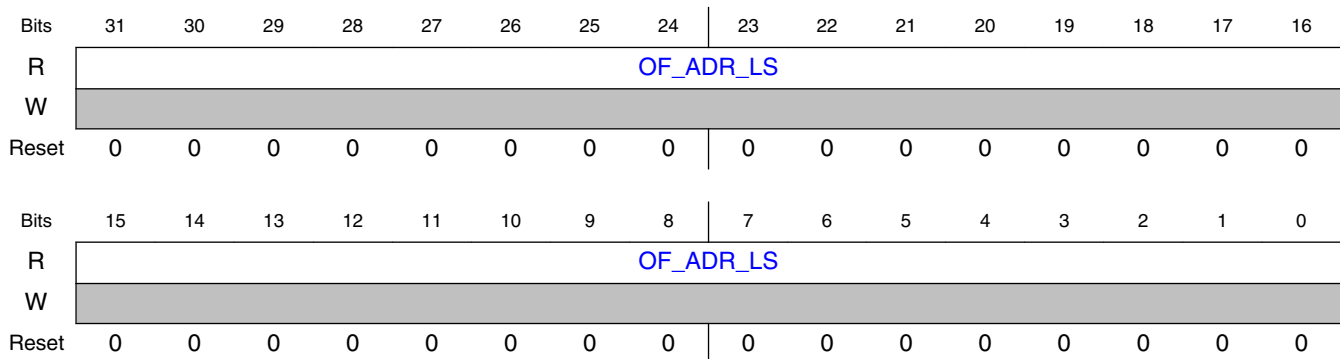
Field	Function
31-0 FD_FRC	The most-significant 32 bits of the flow context for the dequeued frame for this job.

14.254 Queue Interface Job Data Register 17 (QIJOB17)

14.254.1 Offset

Register	Offset	Description
QIJOB17	7_05B8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.254.2 Diagram



14.254.3 Fields

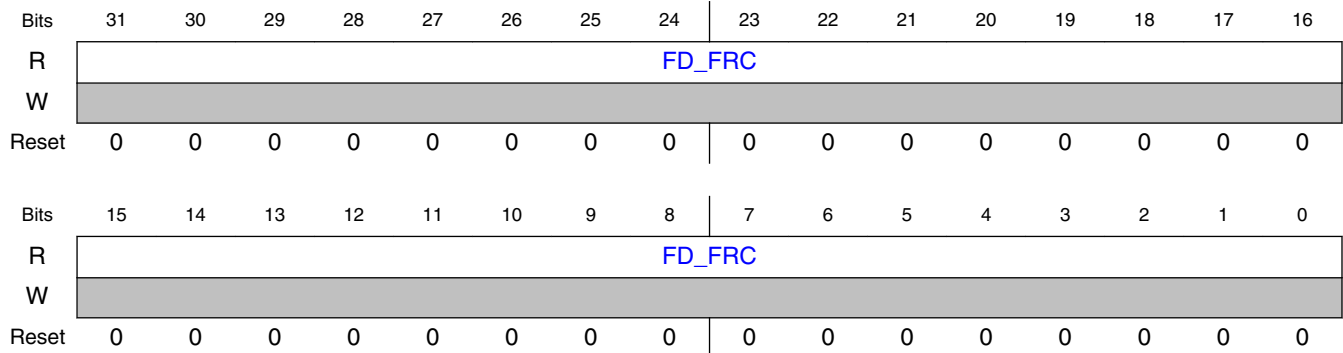
Field	Function
31-0 OF_ADR_LS	The least-significant 32 bits of the output frame address for this job.

14.255 Queue Interface Job Data Register 18 (QIJOB18)

14.255.1 Offset

Register	Offset	Description
QIJOB18	7_05BCh	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.255.2 Diagram



14.255.3 Fields

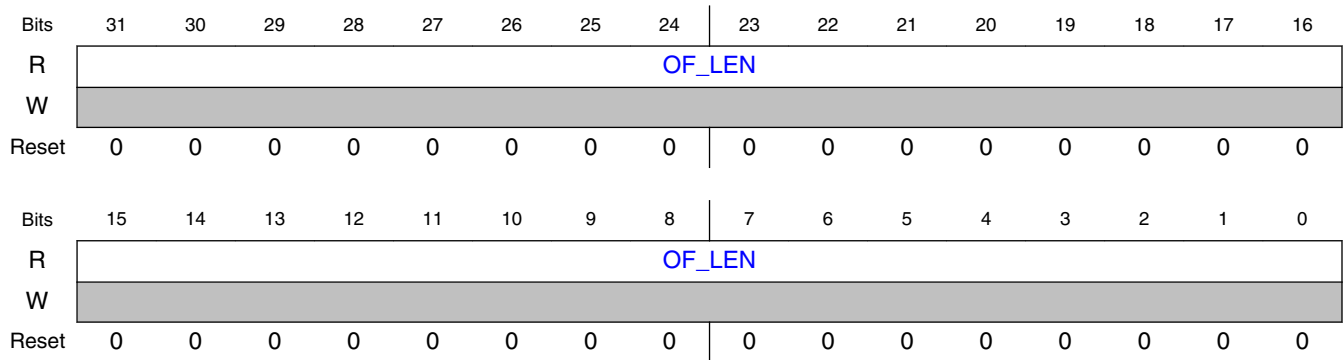
Field	Function
31-0 FD_FRC	The most-significant 32 bits of the output frame address for this job.

14.256 Queue Interface Job Data Register 19 (QIJOB19)

14.256.1 Offset

Register	Offset	Description
QIJOB19	7_05C0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.256.2 Diagram



14.256.3 Fields

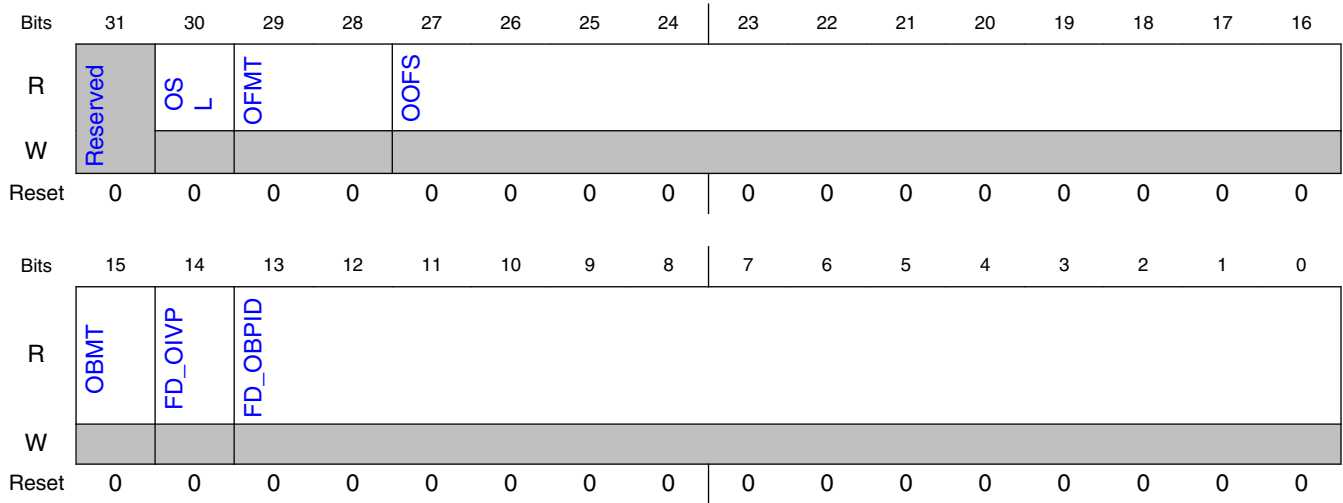
Field	Function
31-0 OF_LEN	The data length of the output frame for this job.

14.257 Queue Interface Job Data Register 20 (QIJOB20)

14.257.1 Offset

Register	Offset	Description
QIJOB20	7_05C4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.257.2 Diagram



14.257.3 Fields

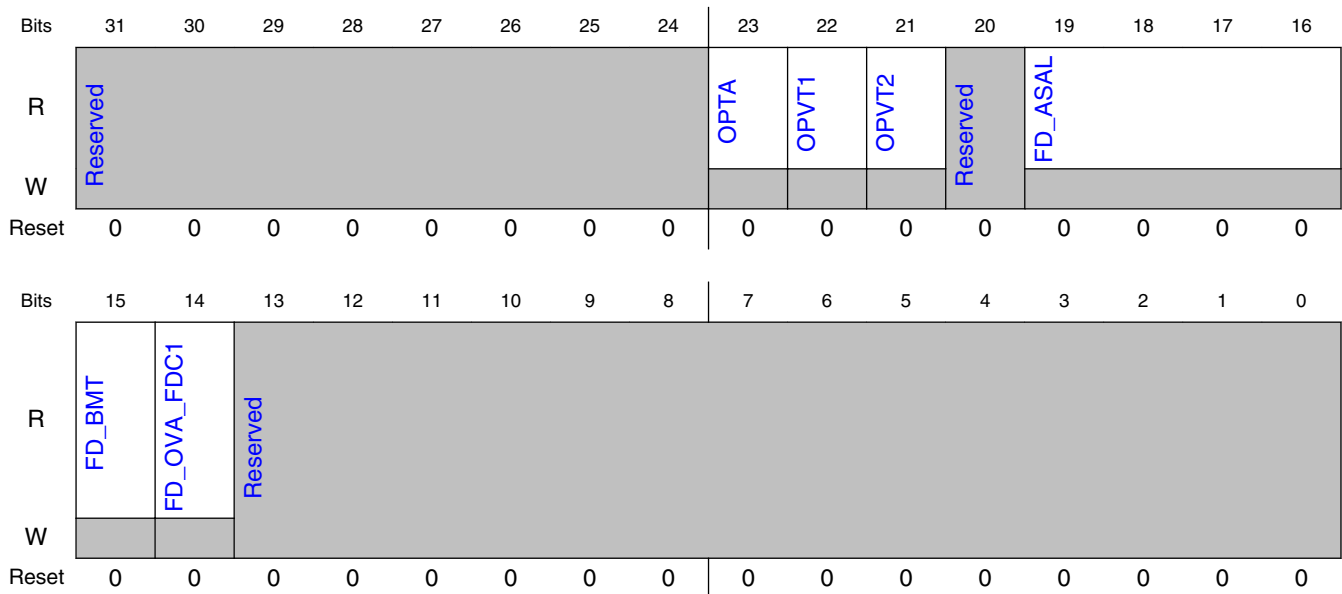
Field	Function
31 —	Reserved
30 OSL	The length of the output frame for this job is in short-length form.
29-28 OFMT	The format of the output frame for this job.
27-16 OOFS	The data offset for the output frame for this job.
15 OBMT	If 1, memory address translation is bypassed for the output frame for this job.
14 FD_OIVP	If 1, the output frame for this job is invalid.
13-0 FD_OBPID	The Output Frame Buffer Pool ID for this job.

14.258 Queue Interface Job Data Register 21 (QIJOB21)

14.258.1 Offset

Register	Offset	Description
QIJOB21	7_05C8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.258.2 Diagram



14.258.3 Fields

Field	Function
31-24 —	Reserved
23 OPTA	Pass-Through Annotation for the output frame for this job.
22 OPVT1	If 1, the output frame pass-through annotation 1 is valid for this job.
21 OPVT2	If 1, the output frame pass-through annotation 2 is valid for this job.
20	Reserved

Table continues on the next page...

Queue Interface Job Data Register 22 (QIJOB22)

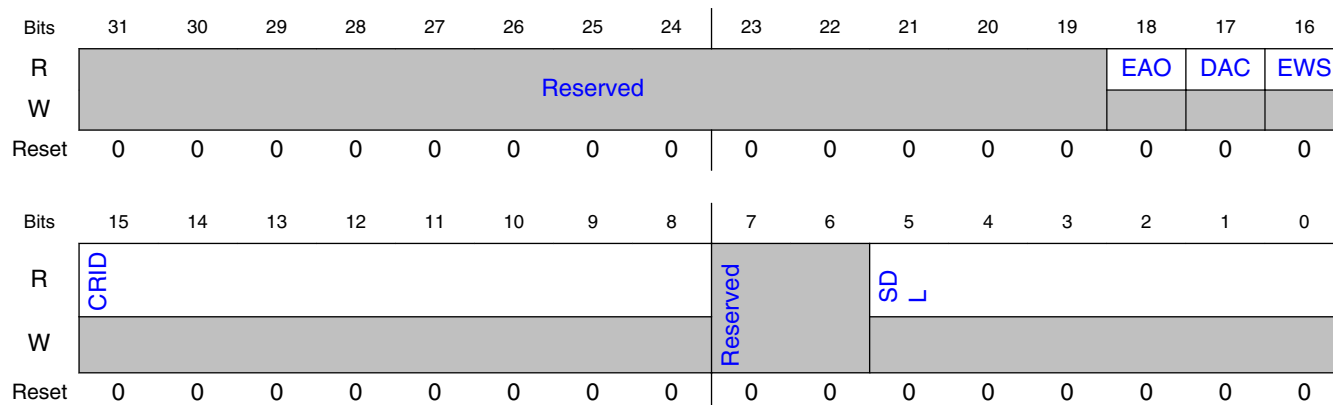
Field	Function
—	
19-16 FD_ASAL	The length of the accelerator-specific annotation.
15 FD_BMT	If 1, bypass memory address translation for this job.
14 FD_OVA_FDC1	If framelist format, this is the FL OF FD-compatibility field 1, otherwise this indicates that the output partition uses a virtual address.
13-0 —	Reserved

14.259 Queue Interface Job Data Register 22 (QIJOB22)

14.259.1 Offset

Register	Offset	Description
QIJOB22	7_05CCh	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.259.2 Diagram



14.259.3 Fields

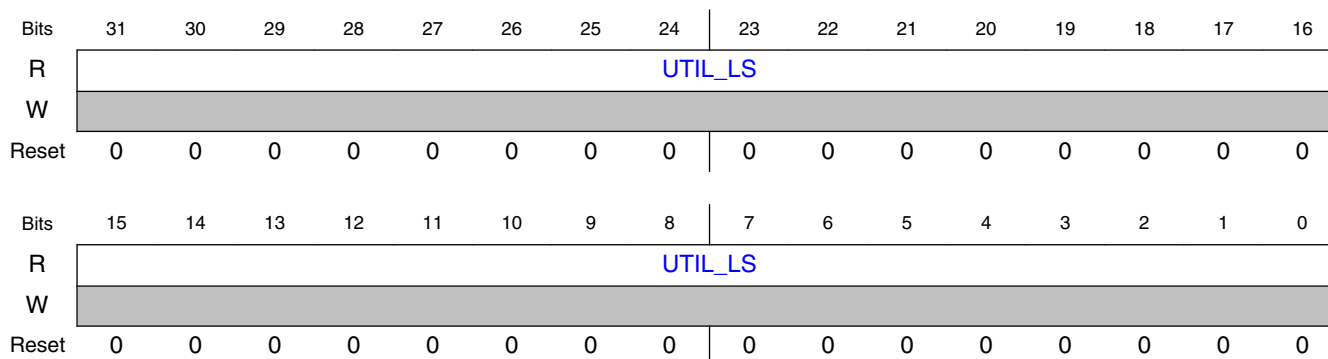
Field	Function
31-19 —	Reserved
18 EAO	If 1, annotation output is enabled for this job.
17 DAC	If 1, annotation (PTA) copy is disabled for this job.
16 EWS	If 1, AXI Write-Safe is enabled.
15-8 CRID	The identifier for a critical resource.
7-6 —	Reserved
5-0 SDL	The length of the shared descriptor.

14.260 Queue Interface Job Data Register 23 (QIJOB23)

14.260.1 Offset

Register	Offset	Description
QIJOB23	7_05D0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.260.2 Diagram



14.260.3 Fields

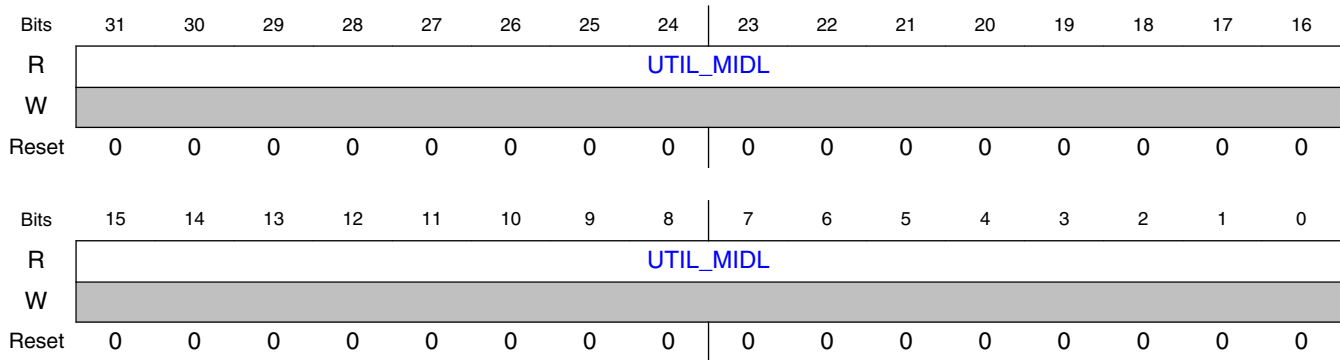
Field	Function
31-0 UTIL_LS	The least-significant 32 bits of the utility buffer for this job.

14.261 Queue Interface Job Data Register 24 (QIJOB24)

14.261.1 Offset

Register	Offset	Description
QIJOB24	7_05D4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.261.2 Diagram



14.261.3 Fields

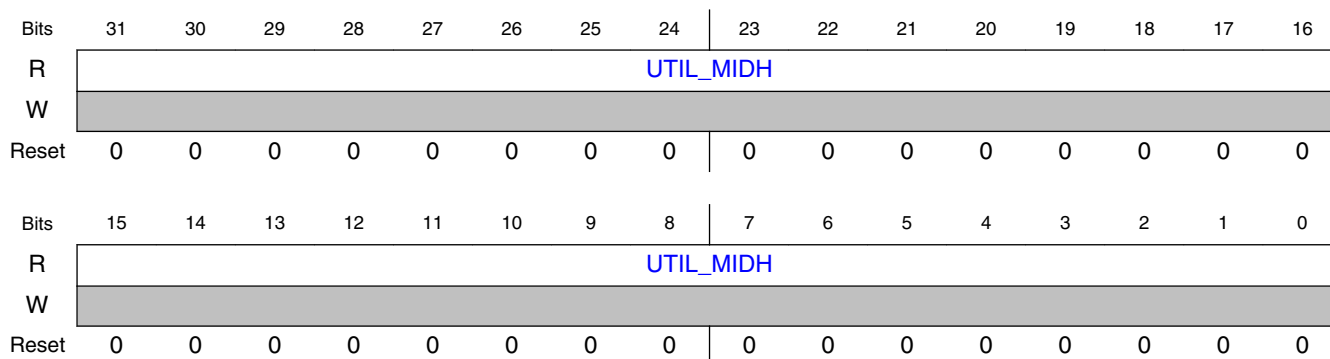
Field	Function
31-0 UTIL_MIDL	The next-to-least-significant 32 bits (63..32) of the utility buffer for this job.

14.262 Queue Interface Job Data Register 25 (QIJOB25)

14.262.1 Offset

Register	Offset	Description
QIJOB25	7_05D8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.262.2 Diagram



14.262.3 Fields

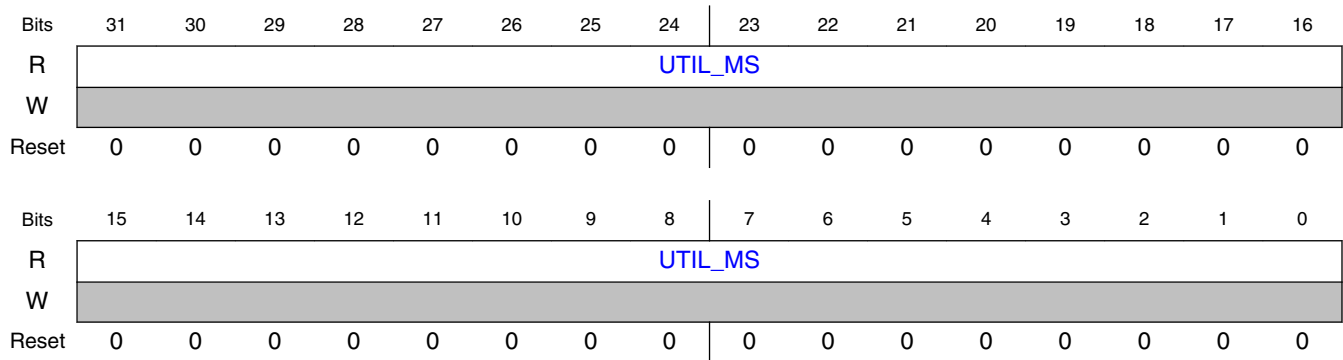
Field	Function
31-0 UTIL_MIDH	The next-to-most-significant 32 bits (95..64) of the utility buffer for this job.

14.263 Queue Interface Job Data Register 26 (QIJOB26)

14.263.1 Offset

Register	Offset	Description
QIJOB26	7_05DCh	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.263.2 Diagram



14.263.3 Fields

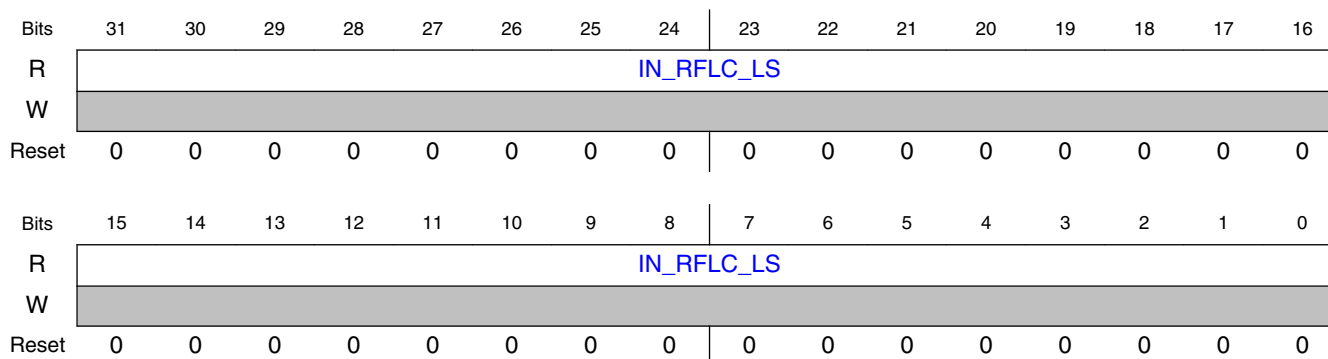
Field	Function
31-0 UTIL_MS	The most-significant 32 bits (127..96) of the utility buffer for this job.

14.264 Queue Interface Job Data Register 27 (QIJOB27)

14.264.1 Offset

Register	Offset	Description
QIJOB27	7_05E0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.264.2 Diagram



14.264.3 Fields

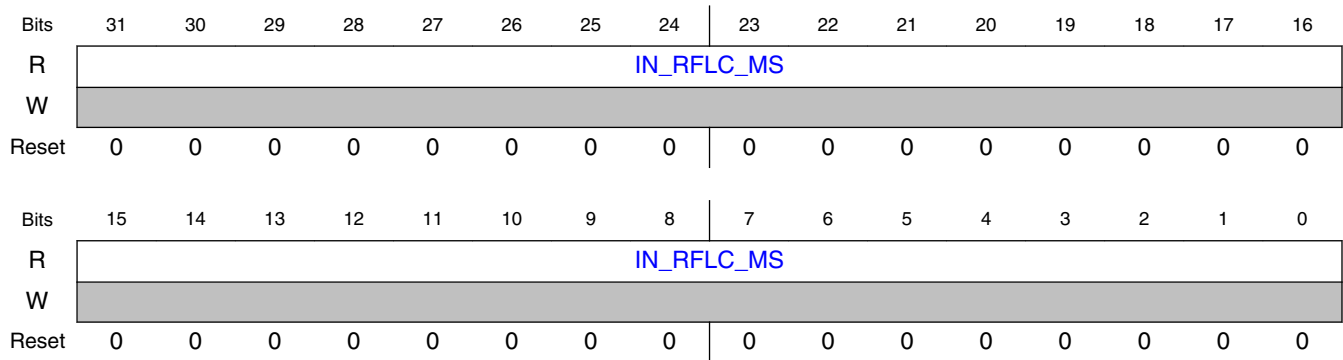
Field	Function
31-0 IN_RFLC_LS	The least-significant 32 bits of the input partition response flow context for this job.

14.265 Queue Interface Job Data Register 28 (QIJOB28)

14.265.1 Offset

Register	Offset	Description
QIJOB28	7_05E4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.265.2 Diagram



14.265.3 Fields

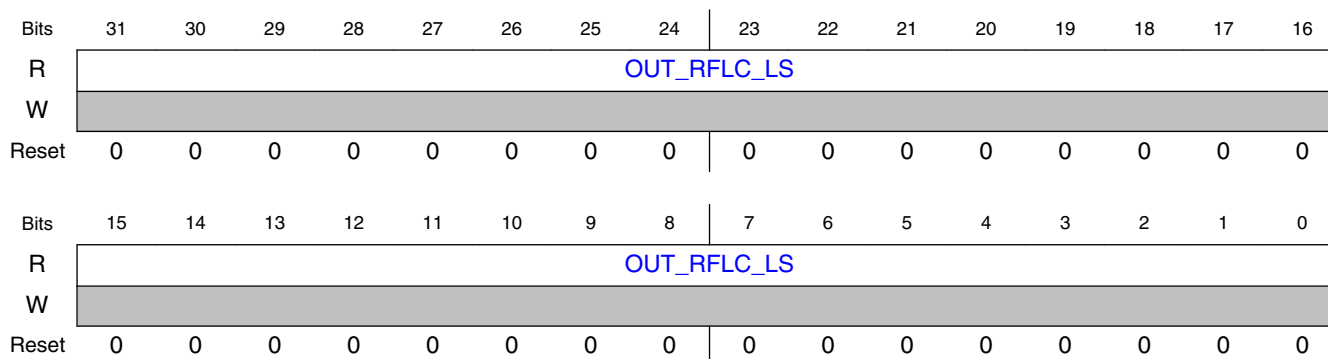
Field	Function
31-0 IN_RFLC_MS	The most-significant 32 bits of the input partition response flow context for this job.

14.266 Queue Interface Job Data Register 29 (QIJOB29)

14.266.1 Offset

Register	Offset	Description
QIJOB29	7_05E8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.266.2 Diagram



14.266.3 Fields

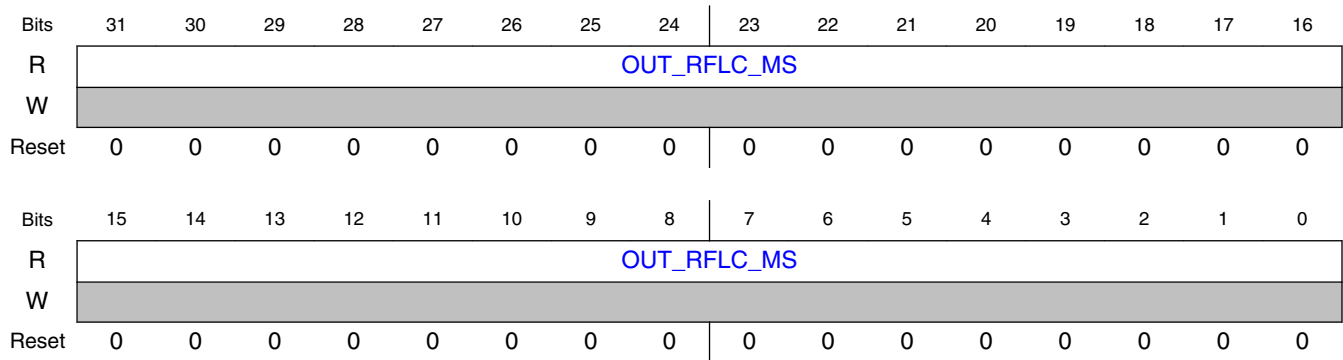
Field	Function
31-0 OUT_RFLC_LS	The least-significant 32 bits of the output partition response flow context for this job.

14.267 Queue Interface Job Data Register 30 (QIJOB30)

14.267.1 Offset

Register	Offset	Description
QIJOB30	7_05ECh	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.267.2 Diagram



14.267.3 Fields

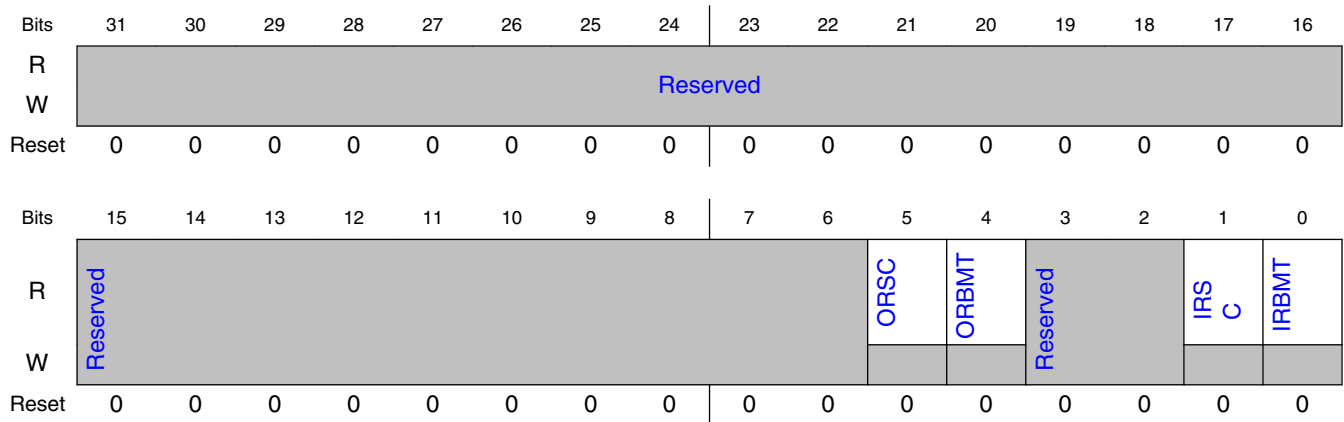
Field	Function
31-0 OUT_RFLC_MS	The most-significant 32 bits of the output partition response flow context for this job.

14.268 Queue Interface Job Data Register 31 (QIJOB31)

14.268.1 Offset

Register	Offset	Description
QIJOB31	7_05F0h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.268.2 Diagram



14.268.3 Fields

Field	Function
31-6 —	Reserved.
5 ORSC	If 1, the output partition response stashing control is enabled for this job.
4 ORBMT	If 1, memory address translation is bypassed for the output partition response for this job.
3-2 —	Reserved.
1 IRSC	If 1, the input partition response stashing control is enabled for this job.
0 IRBMT	If 1, memory address translation is bypassed for the input partition response for this job.

14.269 Queue Interface Job Data Register 32 (QIJOB32)

14.269.1 Offset

Register	Offset	Description
QIJOB32	7_05F4h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.269.2 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved								DQ_IDX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PREV_IDX								SFQID							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.269.3 Fields

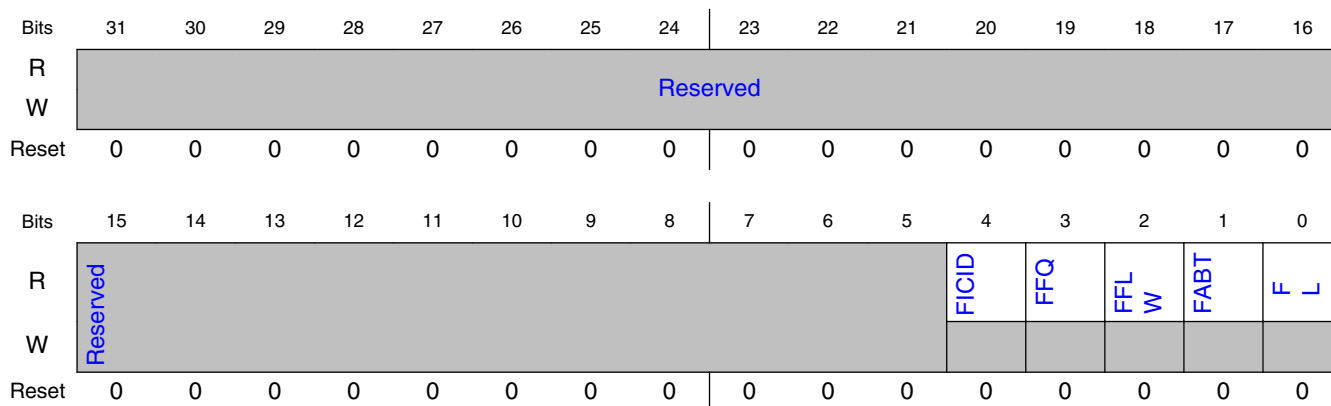
Field	Function
31-24 —	Reserved.
23-16 DQ_IDX	The dequeue order index of the job awaiting transfer to the Job Queue Controller.
15-8 PREV_IDX	The job buffer index of the previous job in the enqueue order list.
7-0 SFQID	The job buffer index of the next job in the enqueue order list.

14.270 Queue Interface Job Data Register 33 (QIJOB33)

14.270.1 Offset

Register	Offset	Description
QIJOB33	7_05F8h	This register is used for debugging Queue Interface jobs. Registers QIJOB0 .. QIJOB33 display job status data for the QI job selected via the JOB_SELECT_QI register.

14.270.2 Diagram



14.270.3 Fields

Field	Function
31-5 —	Reserved.
4 FICID	This job requested a flush of all QI jobs with the specified ICID (FRC 0000_1000b and 0000_1001b).
3 FFQ	This job requested a flush of all jobs for this frame queue (FRC 0000_1010b and 0000_1011b).
2 FFLW	This job requested a flow flush (FRC 0000_1100b).
1 FABT	This job requested either ICID flush with abort (FRC 0000_1001b) or input queue flush with abort (FRC 0000_1011b).
0 FL	The ID of the Subportal for this job.

14.271 Recoverable Error Interrupt Record 0 for the Queue Interface (REIR0QI)

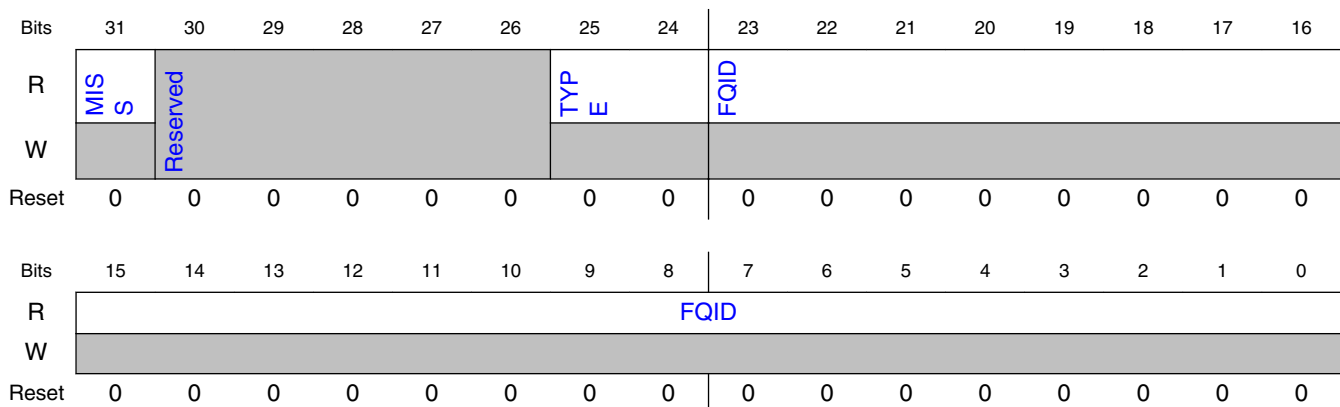
14.271.1 Offset

Register	Offset
REIR0QI	7_0700h

14.271.2 Function

Three types of recoverable errors are defined for QI: system bus access (memory read/write) errors, ICID-related isolation errors, and dropped frame errors. If a recoverable error occurs related to execution of a job from QI, error information will be captured in the QI's REIR registers. Data for a second QI recoverable error will not be captured unless REIR0QI was written (with any value) prior to the occurrence of that error. If another recoverable error occurs before that write, the missed error status bit (MISS) in REIR0QI will be set. When REIR0QI is written, all of QI's REIRQI registers are cleared and error capture is re-enabled.

14.271.3 Diagram



14.271.4 Fields

Field	Function
31 MISS	If MISS=1, a second QI recoverable error occurred before REIROQI was written to re-enable error data capture.
30-26 —	Reserved
25-24 TYPE	This field indicates the type of the recoverable error. If TYPE = 0 : reserved If TYPE = 1 : memory access error If TYPE = 2 : isolation error If TYPE = 3 : dropped frame error
23-0 FQID	For TYPE 2 and 3 errors, this field holds the Frame Queue identifier associated with the recoverable error. For TYPE 1 errors this field will return 000000h.

14.272 Recoverable Error Interrupt Record 1 for the Queue Interface (REIR1QI)

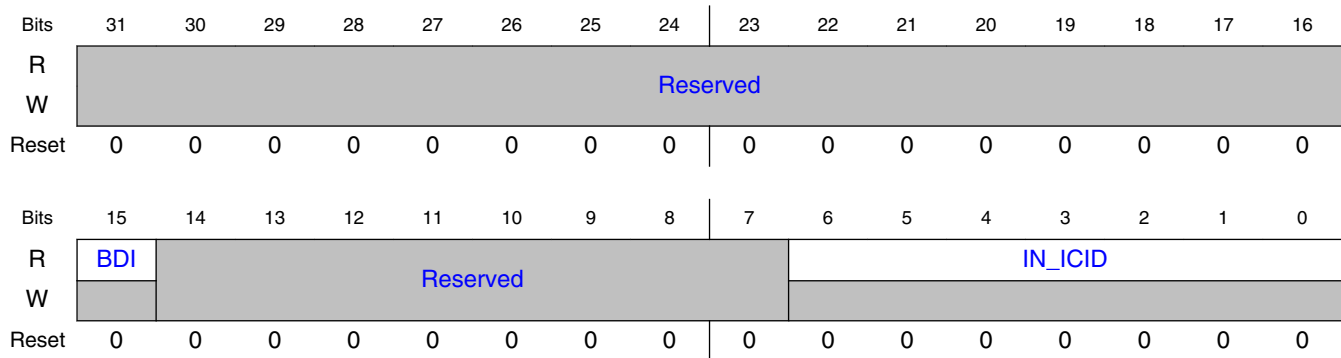
14.272.1 Offset

Register	Offset
REIR1QI	7_0704h

14.272.2 Function

This register is used to report information related to isolation errors (REIROQI.TYPE = 2). REIR1QI will return all zeros for memory access errors (TYPE = 1). See the description for [Recoverable Error Interrupt Record 0 for the Queue Interface \(REIROQI\)](#).

14.272.3 Diagram



14.272.4 Fields

Field	Function
31-16 —	Reserved
15 BDI	For TYPE 2 and 3 errors this field holds the BDI associated with the recoverable error. For TYPE 1 errors this field will return 0.
14-7 —	Reserved
6-0 IN_ICID	For TYPE 2 and 3 errors this field holds the Input_ICID associated with the recoverable error. For TYPE 1 errors this field will return 00h.

14.273 Recoverable Error Interrupt Record 2 for the Queue Interface (REIR2QI)

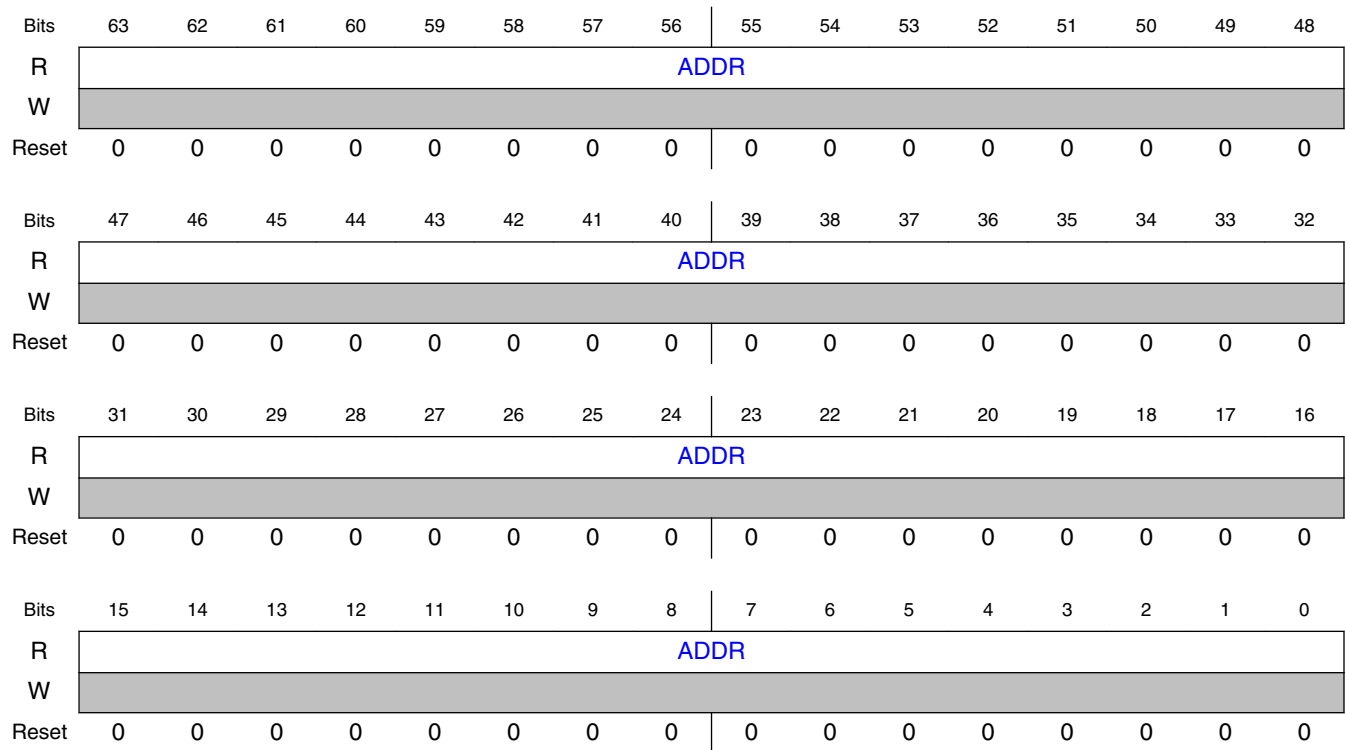
14.273.1 Offset

Register	Offset
REIR2QI	7_0708h

14.273.2 Function

See the description for [Recoverable Error Interrupt Record 0 for the Queue Interface \(REIR0QI\)](#).

14.273.3 Diagram



14.273.4 Fields

Field	Function
63-0 ADDR	REIR2QI holds the address associated with the recoverable QI error. For Type 1 errors the address is the memory location where the error occurred. For Type 2 or 3 errors the address is the Flow Context address. Note that this register may be double-word swapped. See MCFGR[DWT] (Master Configuration Register (MCFGR)).

14.274 Recoverable Error Interrupt Record 4 for the Queue Interface (REIR4QI)

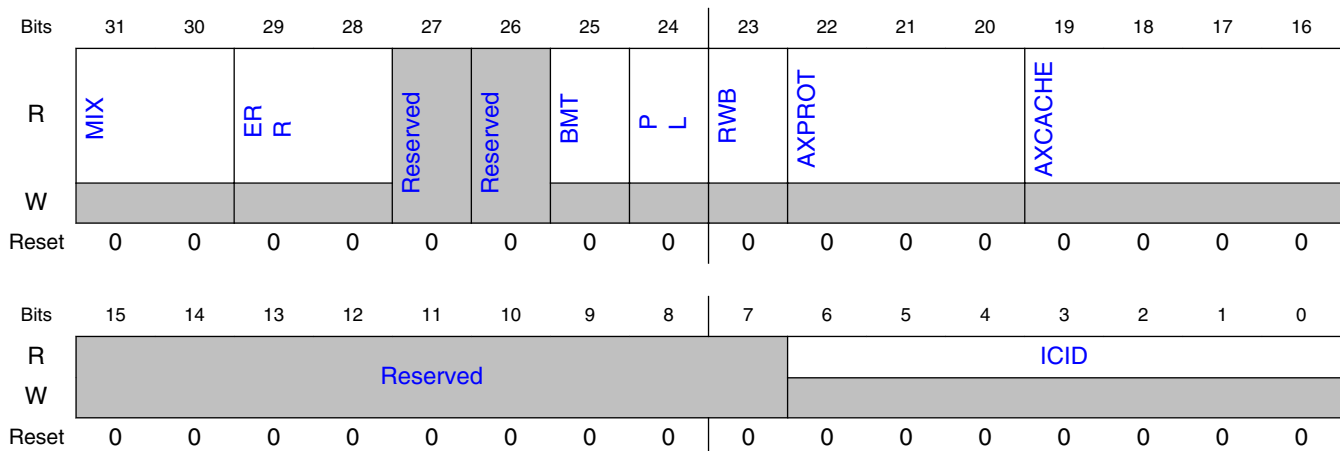
14.274.1 Offset

Register	Offset
REIR4QI	7_0710h

14.274.2 Function

See the description for [Recoverable Error Interrupt Record 0 for the Queue Interface \(REIR0QI\)](#). For Type 1 errors this register contains the 32-bit QI error status code. For Type 2 errors see the register layout that follows.

14.274.3 Diagram



14.274.4 Fields

Field	Function
31-30 MIX	This field holds the memory interface index associated with the recoverable error.
29-28	This field holds the AXI error response associated with the recoverable error.

Table continues on the next page...

Recoverable Error Interrupt Record 5 for the Queue Interface (REIR5QI)

Field	Function
ERR	
27 —	Reserved
26 —	Reserved
25 BMT	This field holds the bypass memory translation transaction attribute used for the memory access.
24 PL	This field holds the privilege level transaction attribute used for the memory access.
23 RWB	This field specifies whether the memory access was a read or write.
22-20 AXPROT	This field holds the AXI protection transaction attribute used for the memory access.
19-16 AXCACHE	This field holds the AXI cache control transaction attribute used for the memory access.
15-7 —	Reserved
6-0 ICID	For Type 1 errors this field holds the ICID transaction attribute associated with the recoverable error.

14.275 Recoverable Error Interrupt Record 5 for the Queue Interface (REIR5QI)

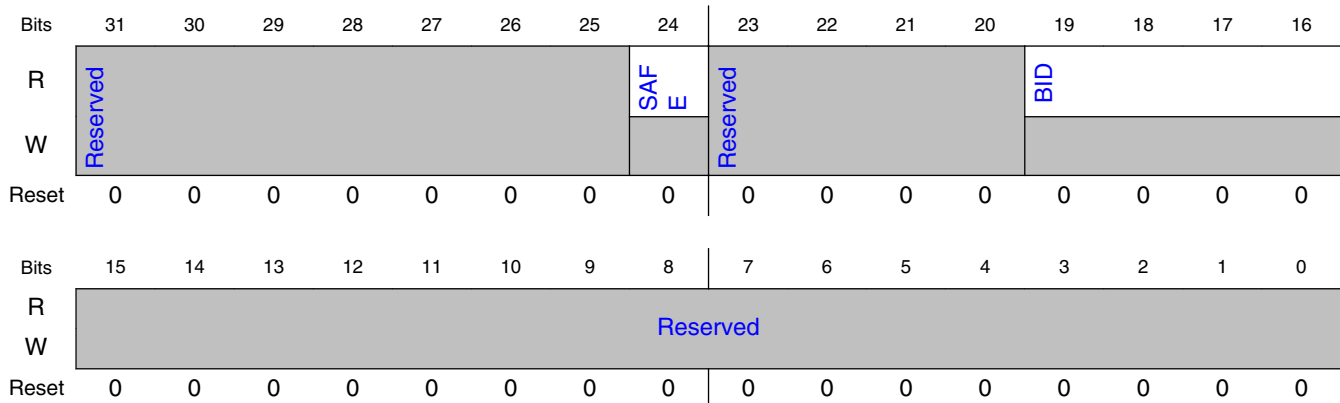
14.275.1 Offset

Register	Offset
REIR5QI	7_0714h

14.275.2 Function

See the description for [Recoverable Error Interrupt Record 0 for the Queue Interface \(REIR0QI\)](#). This register is used to report information related to memory access errors (REIR0QI.TYPE = 1). For errors that are not type 1, this register will return all zeros.

14.275.3 Diagram



14.275.4 Fields

Field	Function
31-25 —	Reserved
24 SAFE	For Type 1 errors SAFE indicates whether the AXI transaction associated with the recoverable error was a safe transaction (either read-safe or write-safe). For Type 2 or 3 errors, this field is 0.
23-20 —	Reserved
19-16 BID	For Type 1 errors BID holds the block identifier (see Table 14-1) of the source of the AXI transaction associated with the recoverable error. For Type 2 or 3 errors, this field is 0.
15-0 —	Reserved

14.276 CCB a Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK - C5C1MR_NPK)

14.276.1 Offset

For a = 0 to 5:

Register	Offset	Description
CaC1MR_NPK	8_0004h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.276.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section [CCB a Class 1 Mode Register Format for Public Key Algorithms \(C0C1MR_PK - C5C1MR_PK\)](#)), one for RNG (see Section [CCB a Class 1 Mode Register Format for RNG4 \(C0C1MR_RNG - C5C1MR_RNG\)](#)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

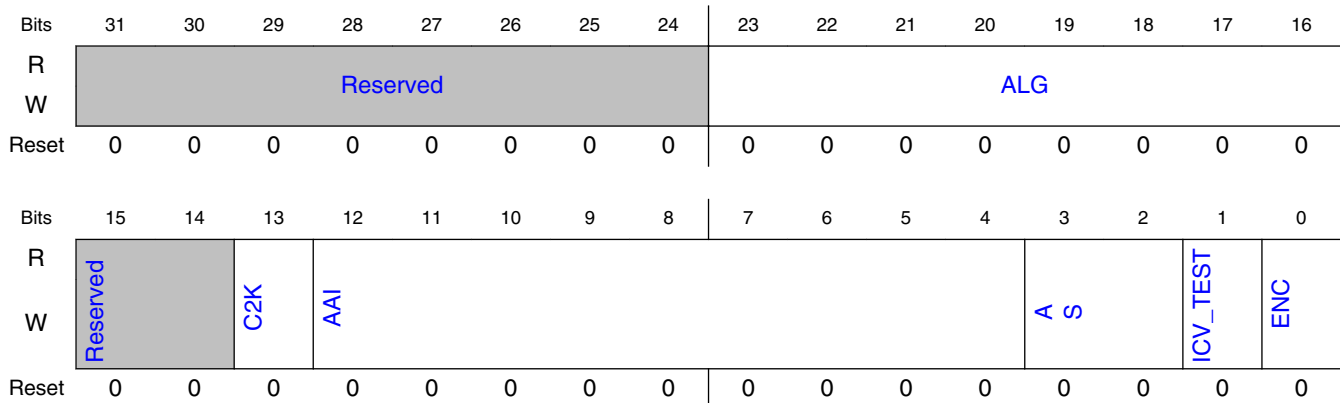
This section defines the format of the Class 1 Mode Register when used with non-public-key algorithms. The Non-Public-Key algorithms are those that do not use the PKHA.

Some examples of how to build the Class 1 Mode Register for non-Public Key algorithms:

Table 14-2. Class 1 Mode Register examples for non-Public Key algorithms

Crypto service performed	ALG Mnemonic	AAI Mnemonic	AS Mnemonic	ICV	Encrypt/ Decrypt/ Protect/ Authenticate	ALGORITHM OPERATION Command	32-bit Value Loaded into C1 Mode Reg
AES GCM	AES	GCM	Init / Finalize	yes	Decrypt	8201090Eh	0001090Eh
AES Counter with mod=2 ¹²⁸	AES	CTR Modulus 2 ¹²⁸	--	no	Encrypt	82010001h	00010001h
Kasumi f9	Kasumi	f9	Init / Finalize	yes	Authenticate	8207020Eh	0007020Eh
Triple DES OFB mode with key parity	DES	OFB	--	no	Decrypt	82021400h	00021400h

14.276.3 Diagram



14.276.4 Fields

Field	Function
31-24 —	Reserved. Must be 0.
23-16 ALG	Algorithm. This field specifies which algorithm is being selected. 00010000b - AES 00100000b - DES 00100001b - 3DES 01010000b - RNG 01100000b - SNOW f8 01110000b - Kasumi Encryption 10110000b - ZUC Encryption
15-14 —	Reserved. Must be 0.
13 C2K	Class 2 Key. This bit is ignored for all algorithms other than AES. 0b - AES will use the Class 1 Key for CCM and GCM modes. 1b - AES will use the Class 2 Key for CCM and GCM modes. Setting this bit =1 will result in a mode error for other AES modes.
12-4 AAI	Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. See also the section describing the appropriate CHA. For RNG OPERATION commands the AAI field is interpreted as shown in CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG) . NOTE: Some algorithms do not require additional algorithm information and in those cases this field should be all 0s. AAI Interpretation for AES Modes

Table continues on the next page...

Field	Function				
[For AES the MSB of AAI is the DK (Decrypt Key) bit.]					
	Code¹	Interpretation		Code¹	Interpretation
	00h	CTR (mod 2 ¹²⁸)		80h	CCM ²
	10h	CBC		90h	GCM ²
	20h	ECB		A0h	CBC_XCBC_MAC
	30h	CFB		B0h	CTR_XCBC_MAC
	40h	OFB		C0h	CBC_CMIC
	50h	XTS		D0h	CTR_CMIC_LTE
	60h	CMIC		E0h	CTR_CMIC
	70h	XCBC-MAC			
Setting the DK bit (i.e. ORing 100h with any AES code above) causes Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in Differences between the AES encrypt and decrypt keys					

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).
2. If C2K= 0, CCM and GCM use the key in the Class 1 Key Register. If C2K = 1, CCM and GCM use the key in the Class 2 Key Register.

Table 14-3. AAI Interpretation for Kasumi

Code ¹	Interpretation		Code ¹	Interpretation
10h	GSM		20h	EDGE
C08	f8		C8h	f9

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).

Table 14-4. AAI Interpretation for ZUC

Code ¹	Interpretation	For Authentication mode, see CCB a Class 2 Mode Register (C0C2MR - C5C2MR)
C0h	encryption mode	

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).

Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. See also the section describing the appropriate CHA.

NOTE: Some algorithms do not require additional algorithm information and in those cases this field should be all 0s.

NOTE: For RNG OPERATION commands the AAI field is interpreted as shown in the SK, AI, PS, OBP, NZ and SH fields shown in [CCB a Class 1 Mode Register Format for RNG4 \(C0C1MR_RNG - C5C1MR_RNG\)](#).

Table 14-5. AAI Interpretation for DES

Code ¹	Interpretation		Code ¹	Interpretation
10h	CBC		30h	CFB

Table continues on the next page...

Field	Function																															
	<p align="center">Table 14-5. AAI Interpretation for DES (continued)</p> <table border="1"> <thead> <tr> <th>Code¹</th> <th>Interpretation</th> <th></th> <th>Code¹</th> <th>Interpretation</th> </tr> </thead> <tbody> <tr> <td>20h</td> <td>ECB</td> <td></td> <td>40h</td> <td>OFB</td> </tr> <tr> <td colspan="5" style="text-align: center;">80h ORed with any DES code above: Check odd parity</td> </tr> </tbody> </table> <p>1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).</p> <p align="center">Table 14-6. AAI Interpretation for RNG</p> <table border="1"> <thead> <tr> <th>Code¹</th> <th>Interpretation</th> </tr> </thead> <tbody> <tr> <td>00h</td> <td>Random Numbers</td> </tr> <tr> <td>10h</td> <td>Random Numbers with No Zero Bytes</td> </tr> <tr> <td>20h</td> <td>Random Numbers with odd byte parity</td> </tr> </tbody> </table> <p align="center">Table 14-7. AAI Interpretation for SNOW 3G</p> <table border="1"> <thead> <tr> <th>Code¹</th> <th>Interpretation</th> <th></th> <th>For f9 mode, see CCB a Class 2 Mode Register (C0C2MR - C5C2MR)</th> </tr> </thead> <tbody> <tr> <td>C0h</td> <td>f8</td> <td></td> <td></td> </tr> </tbody> </table> <p>1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).</p>	Code ¹	Interpretation		Code ¹	Interpretation	20h	ECB		40h	OFB	80h ORed with any DES code above: Check odd parity					Code ¹	Interpretation	00h	Random Numbers	10h	Random Numbers with No Zero Bytes	20h	Random Numbers with odd byte parity	Code ¹	Interpretation		For f9 mode, see CCB a Class 2 Mode Register (C0C2MR - C5C2MR)	C0h	f8		
Code ¹	Interpretation		Code ¹	Interpretation																												
20h	ECB		40h	OFB																												
80h ORed with any DES code above: Check odd parity																																
Code ¹	Interpretation																															
00h	Random Numbers																															
10h	Random Numbers with No Zero Bytes																															
20h	Random Numbers with odd byte parity																															
Code ¹	Interpretation		For f9 mode, see CCB a Class 2 Mode Register (C0C2MR - C5C2MR)																													
C0h	f8																															
3-2 AS	<p>Algorithm State. This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG).</p> <p>00b - Update 01b - Initialize 10b - Finalize 11b - Initialize/Finalize</p>																															
1 ICV_TEST	<p>ICV Checking / Test AES fault detection.</p> <p>(This is the definition of this bit for CHAs other than RNG. For the definition of this bit in RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG))</p> <p>For algorithms other than AES ECB mode: ICV Checking</p> <p>This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.</p> <p>0 - Don't compare 1 - Compare</p> <p>For AES ECB mode: Test AES fault detection</p> <p>In AES ECB mode, this bit activates fault detection testing by injecting bit level errors into AES core logic as defined in the first 128 bits of the context.</p> <p>0 - Don't inject bit errors 1 - Inject bit errors</p>																															
0	Encrypt/Decrypt.																															

Field	Function
ENC	<p>(This is the definition of this bit for CHAs other than RNG.. For the definition of this bit in RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG).)</p> <p>This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes. However, for performance counting to be done correctly, this bit must be set appropriately even if the CHA or Algorithm does not use it to select cryptographic modes.</p> <p>0b - Decrypt. 1b - Encrypt.</p>

14.277 CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C5C1MR_PK)

14.277.1 Offset

For a = 0 to 5:

Register	Offset	Description
CaC1MR_PK	8_0004h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.277.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section [CCB a Class 1 Mode Register Format for Public Key Algorithms \(C0C1MR_PK - C5C1MR_PK\)](#)), one for RNG (see Section [CCB a Class 1 Mode Register Format for RNG4 \(C0C1MR_RNG - C5C1MR_RNG\)](#)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

The following figure shows the Class 1 Mode Register format that is used with public key algorithms, which are algorithms that use PKHA. The Class 1 Mode register is automatically cleared following a PKHA Command. The bit assignments for the

PKHA_MODE field shown in [CCB a Class 1 Mode Register Format for Public Key Algorithms \(C0C1MR_PK - C5C1MR_PK\)](#) will be different depending on which of the three types of PKHA functions is being called. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in [Table PKHA OPERATION: clear memory function](#), [Table PKHA OPERATION: Arithmetic Functions](#) and [Table PKHA OPERATION: copy memory functions](#).

14.277.3 Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved												PKHA_MODE_MS			
W	Reserved												PKHA_MODE_MS			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved				PKHA_MODE_LS											
W	Reserved				PKHA_MODE_LS											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.277.4 Fields

Field	Function
31-20 —	Reserved
19-16 PKHA_MODE_MS	PKHA_MODE most-significant 4 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION: clear memory function , Table PKHA OPERATION: Arithmetic Functions and Table PKHA OPERATION: copy memory functions .
15-12 —	Reserved
11-0 PKHA_MODE_LS	PKHA_MODE least significant 12 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION: clear memory function , Table PKHA OPERATION: Arithmetic Functions and Table PKHA OPERATION: copy memory functions .

14.278 CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG)

14.278.1 Offset

For a = 0 to 5:

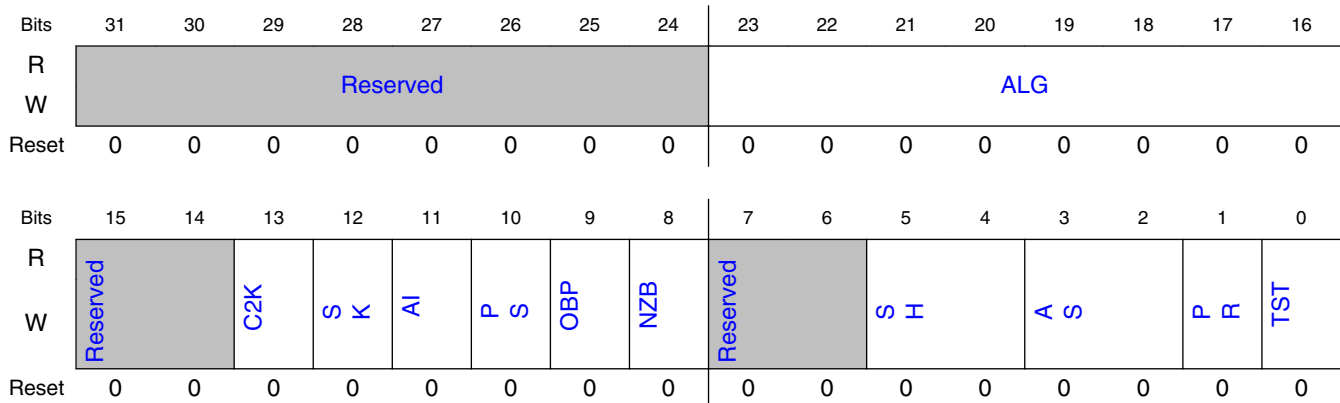
Register	Offset	Description
CaC1MR_RNG	8_0004h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.278.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section [CCB a Class 1 Mode Register Format for Public Key Algorithms \(C0C1MR_PK - C5C1MR_PK\)](#)), one for RNG (see Section [CCB a Class 1 Mode Register Format for RNG4 \(C0C1MR_RNG - C5C1MR_RNG\)](#)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

When the Class 1 Mode register is used to control the RNG, the following format is used.

14.278.3 Diagram



14.278.4 Fields

Field	Function
31-24 —	Reserved. Must be 0.
23-16 ALG	Algorithm. This field specifies which algorithm is being selected. 01010000b - RNG
15-14 —	Reserved. Must be 0.
13 C2K	Class 2 Key. This bit is ignored for all algorithms other than AES. 0b - AES will use the Class 1 Key for CCM and GCM modes. 1b - AES will use the Class 2 Key for CCM and GCM modes. Setting this bit =1 will result in a mode error for other AES modes.
12 SK	Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS!=00.
11 AI	Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the 256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Uninstantiate).
10 PS	Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional “entropy” input during instantiation. Note that the personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01.

Table continues on the next page...

CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C5C1MR_RNG)

Field	Function															
9 OBP	Odd Byte Parity. For RNG Operation commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00.															
8 NZB	NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00h, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00.															
7-6 —	Reserved. For RNG commands these bits of the AAI field are reserved.															
5-4 SH	State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented. 00b - State Handle 0 01b - State Handle 1 10b - Reserved 11b - Reserved															
3-2 AS	Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown below: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>AS Value</th> <th>State Handle is already instantiated</th> <th>State Handle is NOT already instantiated</th> </tr> </thead> <tbody> <tr> <td>00 Generate</td> <td>Generate random data per the mode in which the state handle was instantiated.</td> <td>Error</td> </tr> <tr> <td>01 Instantiate</td> <td>Error</td> <td>Instantiate the state handle in either test mode or non-deterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.</td> </tr> <tr> <td>10 Reseed</td> <td>Reseed the state handle.</td> <td>Error</td> </tr> <tr> <td>11 Uninstantiate</td> <td>Uninstantiate the state handle.</td> <td>Error</td> </tr> </tbody> </table> <p>1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.</p>	AS Value	State Handle is already instantiated	State Handle is NOT already instantiated	00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error	01 Instantiate	Error	Instantiate the state handle in either test mode or non-deterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.	10 Reseed	Reseed the state handle.	Error	11 Uninstantiate	Uninstantiate the state handle.	Error
AS Value	State Handle is already instantiated	State Handle is NOT already instantiated														
00 Generate	Generate random data per the mode in which the state handle was instantiated.	Error														
01 Instantiate	Error	Instantiate the state handle in either test mode or non-deterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.														
10 Reseed	Reseed the state handle.	Error														
11 Uninstantiate	Uninstantiate the state handle.	Error														
1 PR	Prediction Resistance. For RNG OPERATION commands this bit is interpreted as: <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>AS Value</th> <th>PR = 0</th> <th>PR = 1</th> </tr> </thead> <tbody> <tr> <td>00 Generate</td> <td>Do NOT reseed prior to generating new random data</td> <td>If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.</td> </tr> </tbody> </table>	AS Value	PR = 0	PR = 1	00 Generate	Do NOT reseed prior to generating new random data	If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.									
AS Value	PR = 0	PR = 1														
00 Generate	Do NOT reseed prior to generating new random data	If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.														

Table continues on the next page...

Field	Function		
	AS Value	PR = 0	PR = 1
	01 Instantiate	Instantiate the state handle to NOT support prediction resistance	Instantiate the state handle to support prediction resistance
	10 Reseed	Reseed the state handle. PR bit is ignored.	Reseed the state handle. PR bit is ignored.
	11 Uninstantiate	Uninstantiate the state handle. PR bit is ignored.	Uninstantiate the state handle. PR bit is ignored.
	<p>1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.</p>		
0 TST	Test Mode Request. For RNG OPERATION commands this bit is interpreted as:		
	AS Value	TST = 0	TST = 1
	00 Generate	<p>If the selected state handle is in non-deterministic mode, generate new random data.</p> <p>If the selected state handle is in deterministic mode, generate a Test error.¹</p>	<p>If the selected state handle is in deterministic mode, generate new random data.</p> <p>If the selected state handle is in non-deterministic mode, generate a Test error..</p>
	01 Instantiate	Instantiate the state handle in normal (non-deterministic) mode.	Instantiate the state handle in test (deterministic) mode.
	10 Reseed	<p>If the selected state handle is in non-deterministic mode, reseed the state handle.</p> <p>If the selected state handle is in deterministic mode, generate a Test error.</p>	<p>If the selected state handle is in non-deterministic mode, reseed the state handle.</p> <p>If the selected state handle is in deterministic mode, generate a Test error.</p>
	11 Uninstantiate	Uninstantiate the state handle. TST bit is ignored.	Uninstantiate the state handle. TST bit is ignored.
	<p>1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.</p>		

1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

14.279 CCB a Class 1 Key Size Register (C0C1KSR - C5C1KSR)

14.279.1 Offset

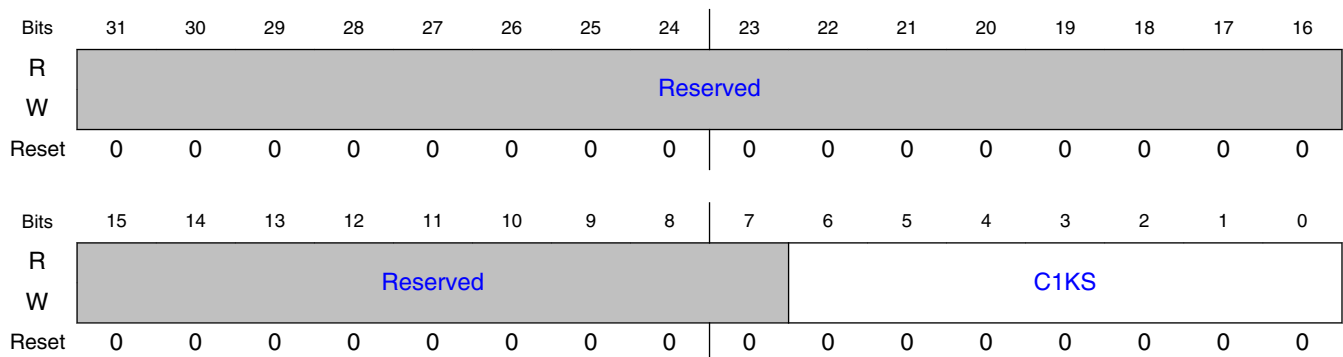
For a = 0 to 5:

Register	Offset	Description
CaC1KSR	8_000Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.279.2 Function

The Class 1 Key Size Register is used to tell the Class 1 CHA the size of the key that was loaded into the Class 1 Key Register. The Class 1 Key Size Register must be written after the key is written into the Class 1 Key Register. Writing to the Class 1 Key Size Register will prevent the user from modifying the Class 1 Key Register. The Class 1 Key Size Register is automatically written by the KEY Command except in the following cases. When the PKHA E-RAM is loaded the PKHA E Size Register is automatically loaded with the correct size, rather than loading the Class 1 Key Size Register.

14.279.3 Diagram



14.279.4 Fields

Field	Function
31-7 —	Reserved
6-0 C1KS	Class 1 Key Size. This is the size of a Class 1 Key measured in bytes Although the Class 1 Key Register holds only 32 bytes, it is possible to load a key as large as 9688 bytes. The first 32 bytes will be loaded into the Class 1 Key Register and the remaining bytes will be loaded into the Class 1 Context Register. The key bytes loaded into the Context Register will be treated as an "Extended Key Register" (i.e. as part of the Key Register) during cryptographic operations and when storing and loading Black Keys.

14.280 CCB a Class 1 Data Size Register (C0C1DSR - C5C1DSR)

14.280.1 Offset

For a = 0 to 5:

Register	Offset	Description
CaC1DSR	8_0010h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

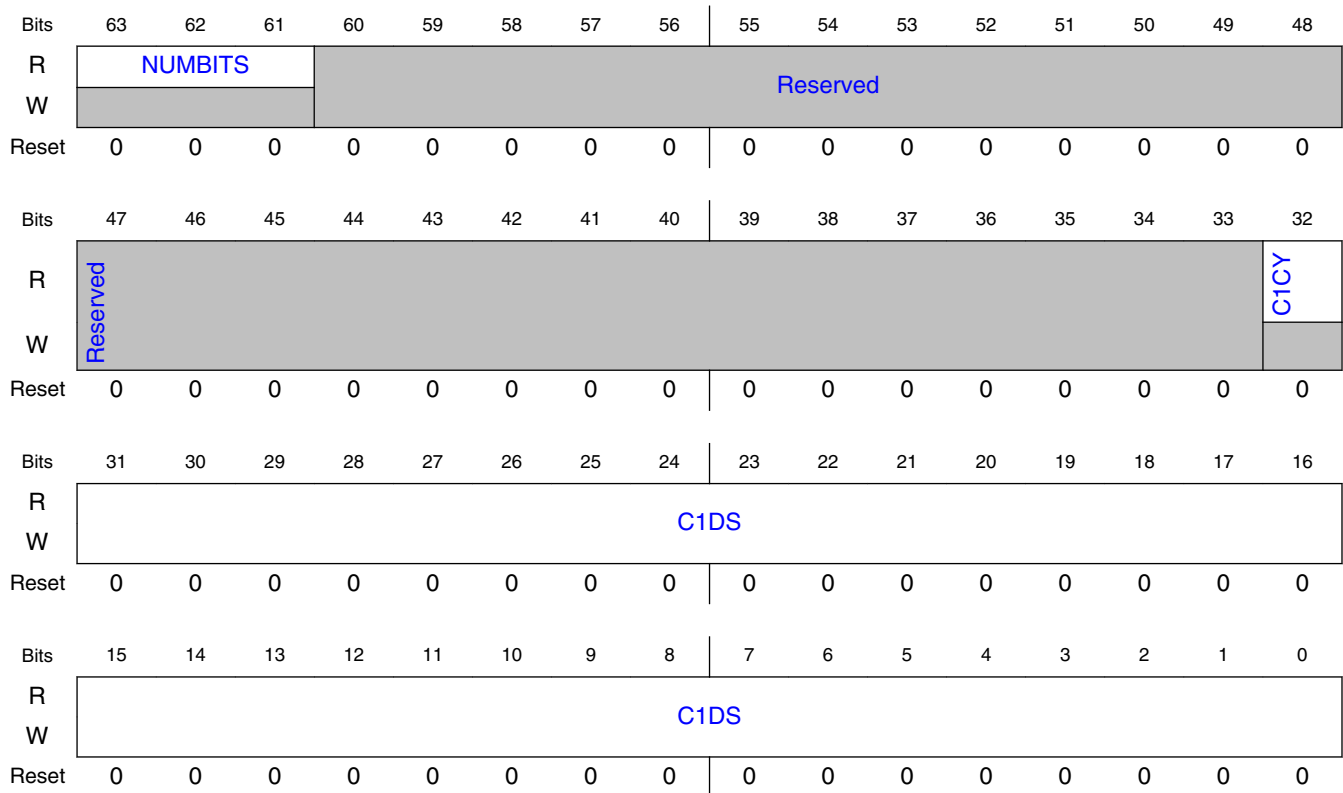
14.280.2 Function

The Class 1 Data Size Register is used to tell the Class 1 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C1CY and C1DS fields to form a data size that is measured in bits. Note that writing to the C1DS field in this register causes the written value to be added to the previous value in that field. That is, if the C1DS field currently has the value 14, writing 2 to the least-significant half of the Class 1 Data Size register (i.e. the C1DS field) will result in a value of 16 in the C1DS field. Although there is a C1CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C1CY and C1DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic iNformation FIFO

CCB a Class 1 Data Size Register (C0C1DSR - C5C1DSR)

entries are enabled. This register is cleared whenever a key is decrypted or encrypted. Since the Class 1 Data Size Registers hold more than 32 bits, they are accessed from the IP bus as two 32-bit registers.

14.280.3 Diagram



14.280.4 Fields

Field	Function
63-61 NUMBITS	Class 1 Data Size Number of bits. For bit-oriented operations, this value is appended to the C1CY and C1DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value (C1CY C1DS NUMBITS). Note that if NUMBITS is nonzero, C1DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value.
60-33 —	Reserved
32 C1CY	Class 1 Data Size Carry. Although this field is not writable, it will be set if a write to C1DS causes a carry out of the msb of C1DS.

Table continues on the next page...

Field	Function
31-0 C1DS	Class 1 Data Size. This is the number of whole bytes of data that will be consumed by the Class 1 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero.

14.281 CCB a Class 1 ICV Size Register (C0C1ICVSR - C5C1ICVSR)

14.281.1 Offset

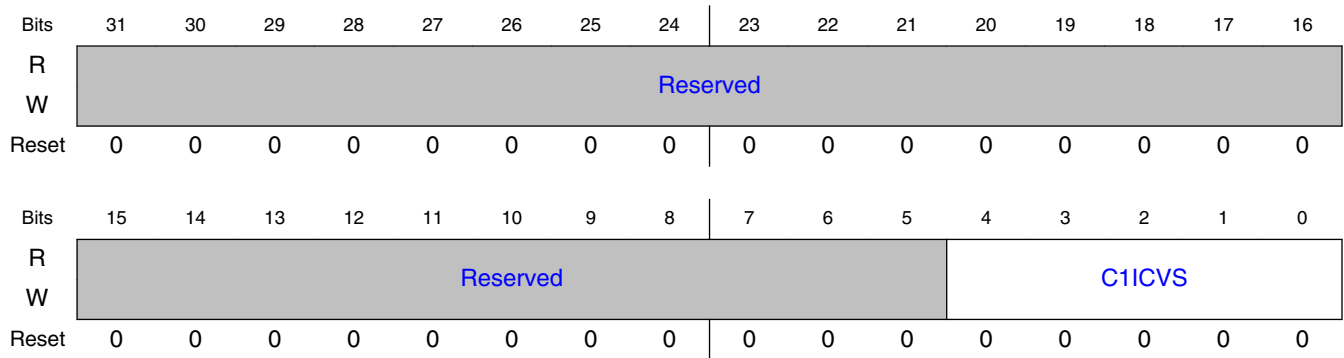
For a = 0 to 5:

Register	Offset	Description
CaC1ICVSR	8_001Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.281.2 Function

The Class 1 ICV Size Register indicates how much of the last block of ICV is valid when performing AES integrity check modes (e.g. AES-CMAC, AES-XCBC-MAC, AES-GCM). Like the Class 1 Data Size register, the Class 1 ICV Size register is additive. That is, any value written to the C1ICVS field will be added to the previous value in the field. This register must be written prior to the corresponding word of data being consumed by AES. In practical terms, this means the register must be written either prior to the corresponding data being written to the Input Data FIFO or prior to the iNformation FIFO entry for this data. FIFO LOAD commands can automatically load it when ICV is loaded.

14.281.3 Diagram



14.281.4 Fields

Field	Function
31-5 —	Reserved
4-0 C1ICVS	Class 1 ICV Size, in Bytes.

14.282 CCB a CHA Control Register (C0CCTRL - C5CCTRL)

14.282.1 Offset

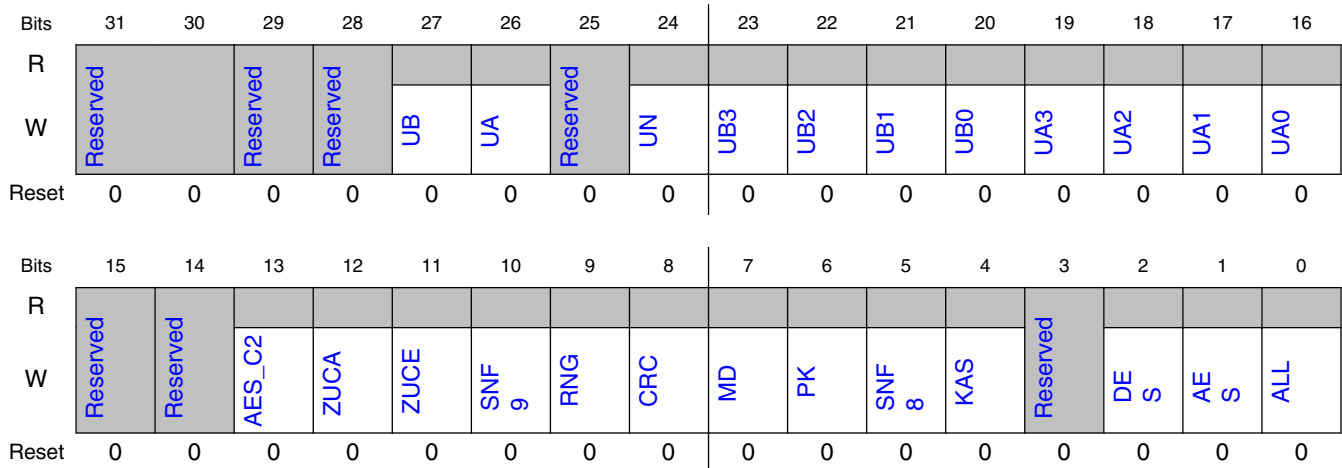
For a = 0 to 5:

Register	Offset	Description
CaCCTRL	8_0034h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.282.2 Function

The CHA Control Register is used to send control signals to the CHAs. This register is automatically written between Descriptors. Within a Descriptor, use the LOAD Command to reset blocks or unload memories.

14.282.3 Diagram



14.282.4 Fields

Field	Function
31-30 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
29 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
28 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
27 UB	Unload the PKHA B Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B memory into the Output Data FIFO.
26 UA	Unload the PKHA A Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A memory into the Output Data FIFO.
25 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
24 UN	Unload the PKHA N Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the N memory into the Output Data FIFO.

Table continues on the next page...

CCB a CHA Control Register (C0CCTRL - C5CCTRL)

Field	Function
23 UB3	Unload the PKHA B3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B3 memory into the Output Data FIFO.
22 UB2	Unload the PKHA B2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B2 memory into the Output Data FIFO.
21 UB1	Unload the PKHA B1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B1 memory into the Output Data FIFO.
20 UB0	Unload the PKHA B0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B0 memory into the Output Data FIFO.
19 UA3	Unload the PKHA A3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A3 memory into the Output Data FIFO.
18 UA2	Unload the PKHA A2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A2 memory into the Output Data FIFO.
17 UA1	Unload the PKHA A1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A1 memory into the Output Data FIFO.
16 UA0	Unload the PKHA A0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A0 memory into the Output Data FIFO.
15 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
14 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
13 AES_C2	Reset AES Class 2 CHA. Writing a 1 to this bit resets the AES Authentication (Class 2) accelerator. 0b - Do Not Reset 1b - Reset AES Authentication accelerator
12 ZUCA	Reset ZUCA. Writing a 1 to this bit resets the ZUC Authentication accelerator. 0b - Do Not Reset 1b - Reset ZUC Authentication accelerator
11 ZUCE	Reset ZUCE. Writing a 1 to this bit resets the ZUC Encryption accelerator. 0b - Do Not Reset 1b - Reset ZUC Encryption accelerator
10 SNF9	Reset SNOW f9. Writing a 1 to this bit resets the SNOW f9 Accelerator. 0b - Do Not Reset 1b - Reset SNOW f9 Accelerator
9 RNG	Reset Random Number Generator. Writing a 1 to this bit resets the Random Number Generator. 0b - Do Not Reset 1b - Reset Random Number Generator Block.
8 CRC	Reset CRCA. Writing a 1 to this bit resets the CRC Accelerator. 0b - Do Not Reset 1b - Reset CRC Accelerator
7 MD	Reset MDHA. Writing a 1 to this bit resets the Message Digest Hardware Accelerator. 0b - Do Not Reset 1b - Reset Message Digest Hardware Accelerator
6	Reset PKHA. Writing a 1 to this bit resets the Public Key Hardware Accelerator.

Table continues on the next page...

Field	Function
PK	0b - Do Not Reset 1b - Reset Public Key Hardware Accelerator
5 SNF8	Reset SNOW f8. Writing a 1 to this bit resets the SNOW f8 Hardware Accelerator. 0b - Do Not Reset 1b - Reset SNOW f8 Accelerator
4 KAS	Reset KFHA. Writing a 1 to this bit resets the Kasumi f8/f9 Hardware Accelerator. 0b - Do Not Reset 1b - Reset Kasumi f8/f9 Accelerator
3 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits.
2 DES	Reset DESA. Writing a 1 to this bit resets the DES Accelerator. 0b - Do Not Reset 1b - Reset DES Accelerator
1 AES	Reset AESA. Writing a 1 to this bit resets the AES Accelerator. 0b - Do Not Reset 1b - Reset AES Accelerator
0 ALL	Reset All Internal CHAs. Writing to this bit resets all CHAs in use by this CCB. 0b - Do Not Reset 1b - Reset all CHAs in use by this CCB.

14.283 CCB a Interrupt Control Register (C0ICTL - C5ICTL)

14.283.1 Offset

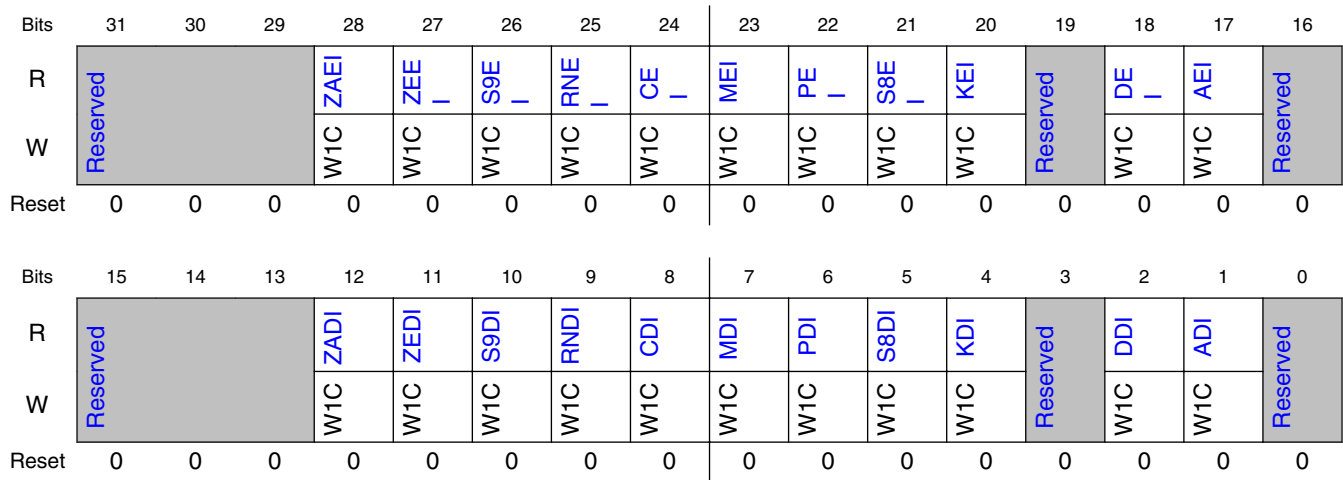
For a = 0 to 5:

Register	Offset	Description
CaICTL	8_003Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.283.2 Function

The IRQ Control Register shows the status of all CCB "done" interrupts and "error" interrupts and provides controls for clearing these interrupts.

14.283.3 Diagram



14.283.4 Fields

Field	Function									
31-29 —	Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits (shaded or marked “RSV”)									
28 ZAEI	ZUCA error Interrupt asserted. <table border="1"> <thead> <tr> <th>Value</th> <th>Read</th> <th>Write</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>No Error Write</td> <td>No change</td> </tr> <tr> <td>1</td> <td>ZUCA Error Interrupt asserted</td> <td>Clear the ZUCA Error Interrupt</td> </tr> </tbody> </table>	Value	Read	Write	0	No Error Write	No change	1	ZUCA Error Interrupt asserted	Clear the ZUCA Error Interrupt
Value	Read	Write								
0	No Error Write	No change								
1	ZUCA Error Interrupt asserted	Clear the ZUCA Error Interrupt								
27 ZEEI	ZUCE error Interrupt asserted. <table border="1"> <thead> <tr> <th>Value</th> <th>Read</th> <th>Write</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>No Error Write</td> <td>No change</td> </tr> <tr> <td>1</td> <td>ZUCE Error Interrupt asserted</td> <td>Clear the ZUCE Error Interrupt</td> </tr> </tbody> </table>	Value	Read	Write	0	No Error Write	No change	1	ZUCE Error Interrupt asserted	Clear the ZUCE Error Interrupt
Value	Read	Write								
0	No Error Write	No change								
1	ZUCE Error Interrupt asserted	Clear the ZUCE Error Interrupt								
26 S9EI	SNOW-f9 error Interrupt asserted. <table border="1"> <thead> <tr> <th>Value</th> <th>Read</th> <th>Write</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>No Error Write</td> <td>No change</td> </tr> <tr> <td>1</td> <td>SNOW-f9 Error Interrupt asserted</td> <td>Clear the SNOW-f9 Error Interrupt</td> </tr> </tbody> </table>	Value	Read	Write	0	No Error Write	No change	1	SNOW-f9 Error Interrupt asserted	Clear the SNOW-f9 Error Interrupt
Value	Read	Write								
0	No Error Write	No change								
1	SNOW-f9 Error Interrupt asserted	Clear the SNOW-f9 Error Interrupt								
25	RNG error Interrupt asserted.									

Table continues on the next page...

Field	Function		
	Value	Read	Write
RNEI	0	No Error Write	No change
	1	RNG Error Interrupt asserted	Clear the RNG Error Interrupt
24 CEI	CRCA error Interrupt asserted.		
	0	No Error Write	No change
	1	CRC Error Interrupt asserted	Clear the CRC Error Interrupt
23 MEI	MDHA (hashing) error Interrupt asserted.		
	0	No Error Write	No change
	1	MDHA Error Interrupt asserted	Clear the MDHA Error Interrupt
22 PEI	Public Key error Interrupt asserted.		
	0	No Error Write	No change
	1	PKHA Error Interrupt asserted	Clear the PKHA Error Interrupt
21 S8EI	SNOW-f8 error asserted.		
	0	No Error Write	No change
	1	SNOW-f8 Error Interrupt asserted	Clear the SNOW-f8 Error Interrupt
20 KEI	KFHA (Kasumi) error Interrupt asserted.		
	0	No Error Write	No change
	1	KFHA Error Interrupt asserted	Clear the KFHA Error Interrupt
19 —	Reserved		
18 DEI	DESA error Interrupt asserted.		
	0	No Error Write	No change

Table continues on the next page...

CCB a Interrupt Control Register (C0ICTL - C5ICTL)

Field	Function		
	Value	Read	Write
	1	DESA Error Interrupt asserted	Clear the DESA Error Interrupt
17 AEI	AESA error Interrupt asserted.		
	0	No Error Write	No change
	1	AESA Error Interrupt asserted	Clear the AESA Error Interrupt
16-13 —	Reserved		
12 ZADI	ZUCA Done interrupt.		
	0	No Error	No change
	1	ZUCA Done Interrupt asserted	Clear the ZUCA Done Interrupt
11 ZEDI	ZUCE Done interrupt.		
	0	No Error	No change
	1	ZUCE Error Interrupt asserted	Clear the ZUCE Done Interrupt
10 S9DI	SNOW-f9 done interrupt.		
	0	No Done Interrupt	No change
	1	SNOW-f9 Done Interrupt asserted	Clear the SNOW-f9 Done Interrupt
9 RNDI	RNG done interrupt.		
	0	No Done Interrupt	No change
	1	RNG Done Interrupt asserted	Clear the RNG Done Interrupt
8 CDI	CRCA done interrupt.		
	0	No Done Interrupt	No change
	1	CRCA Done Interrupt asserted	Clear the CRCA Done Interrupt

Table continues on the next page...

Field	Function		
7 MDI	MDHA (hashing) done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	MDHA Done Interrupt asserted	Clear the MDHA Done Interrupt
6 PDI	PKHA (Public Key) done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	PKHA Done Interrupt asserted	Clear the PKHA Done Interrupt
5 S8DI	SNOW-f8 done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	SNOW-f8 Done Interrupt asserted	Clear the SNOW-f8 Done Interrupt
Read: 0 No Done Interrupt Write: 0 No change 1 SNOW-f8 Done Interrupt asserted 1 Clear the SNOW-f8 Done Interrupt			
4 KDI	KFHA (Kasumi) done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	KFHA Done Interrupt asserted	Clear the KFHA Done Interrupt
3 —	Reserved		
2 DDI	DESA done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	DESA Done Interrupt asserted	Clear the DESA Done Interrupt
1 ADI	AESA done interrupt.		
	Value	Read	Write
	0	No Done Interrupt	No change
	1	AESA Done Interrupt asserted	Clear the AESA Done Interrupt
0	Reserved		

CCB a Clear Written Register (C0CWR - C5CWR)

Field	Function
—	

14.284 CCB a Clear Written Register (C0CWR - C5CWR)

14.284.1 Offset

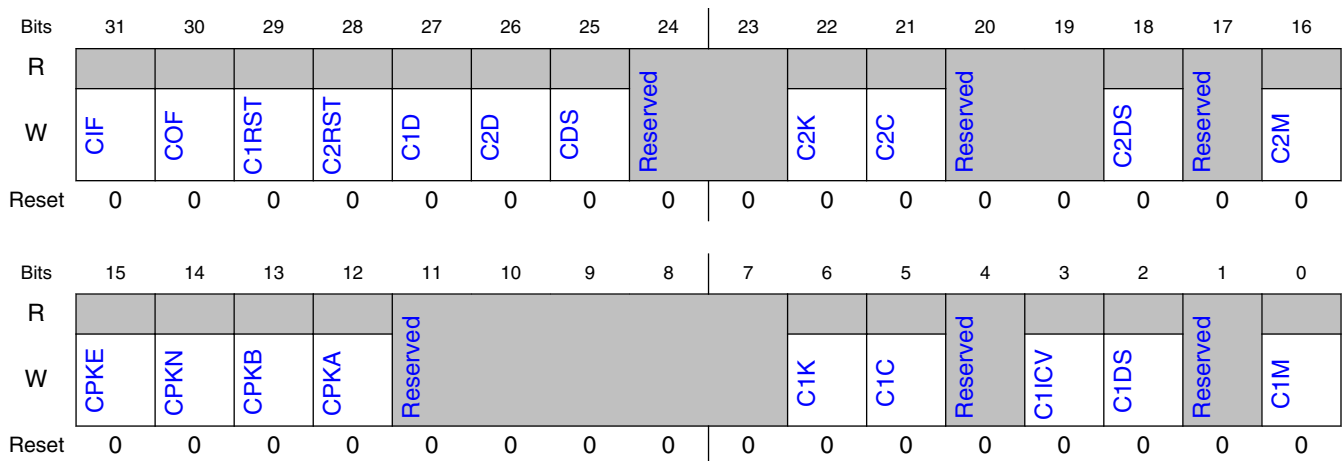
For a = 0 to 5:

Register	Offset	Description
CaCWR	8_0044h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.284.2 Function

The Clear Written Register is used to clear many of the internal registers. This register is automatically written, if necessary, by DECO between Shared Descriptors. All fields of this register are self-clearing.

14.284.3 Diagram



14.284.4 Fields

Field	Function
31 CIF	Clear Input FIFO (and NFIFO). Writing a 1 to this bit causes the Input Data FIFO and iNformation FIFO to be cleared.
30 COF	Clear Output FIFO. Writing a 1 to this bit causes the Output FIFO to be cleared.
29 C1RST	Reset Class 1 CHA. Writing a 1 to this bit causes a reset to any Class 1 CHA that is currently selected by this DECO.
28 C2RST	Reset Class 2 CHA. Writing a 1 to this bit causes a reset to any Class 2 CHA that is currently selected by this DECO.
27 C1D	Clear Class 1 Done Interrupt. Writing a 1 to this bit clears the Class 1 done interrupt.
26 C2D	Clear Class 2 Done Interrupt. Writing a 1 to this bit clears the Class 2 done interrupt.
25 CDS	Clear Descriptor Sharing signal. Writing a 1 to this bit clears the shared_descriptor signal in DECO. This signal tells DECO, and the protocols, whether this descriptor was shared from a previous run. If CDS is set via LOAD IMM to the Clear Written register the fact that this descriptor was shared will be forgotten and the descriptor will behave thereafter as if it was not shared. This is important in protocols where the protocol expects a "decrypt" key but an "encrypt" key is provided. This may occur when using RJD to re-key a flow. Note that writing 1 to this bit when the DECO/CCB is under direct software control will not clear sharing, but that is unimportant because sharing is not possible when the DECO is under direct software control.
24-23 —	Reserved
22 C2K	Clear the Class 2 Key Register. Writing a one to this bit causes the Class 2 Key and Key Size Registers to be cleared.
21 C2C	Clear the Class 2 Context Register. Writing a one to this bit causes the Class 2 Context Register to be cleared.
20-19 —	Reserved
18 C2DS	Clear the Class 2 Data Size Registers. Writing a one to this bit causes the Class 2 Data Size and ICV Size Registers to be cleared.
17 —	Reserved
16 C2M	Clear the Class 2 Mode Register. Writing a one to this bit causes the Class 2 Mode Register to be cleared.
15 CPKE	Clear the PKHA E Size Register. Writing a one to this bit causes the PKHA E Size Register to be cleared.
14 CPKN	Clear the PKHA N Size Register. Writing a one to this bit causes the PKHA N Size Register to be cleared.
13	Clear the PKHA B Size Register. Writing a one to this bit causes the PKHA B Size Register to be cleared.

Table continues on the next page...

CCB a Status and Error Register, most-significant half (C0CSTA_MS - C5CSTA_MS)

Field	Function
CPKB	
12 CPKA	Clear the PKHA A Size Register. Writing a one to this bit causes the PKHA A Size Register to be cleared.
11-7 —	Reserved
6 C1K	Clear the Class 1 Key Register. Writing a one to this bit causes the Class 1 Key and Key Size Registers to be cleared.
5 C1C	Clear the Class 1 Context Register. Writing a one to this bit causes the Class 1 Context Register to be cleared.
4 —	Reserved
3 C1ICV	Clear the Class 1 ICV Size Register. Writing a one to this bit causes the Class 1 ICV Size Register to be cleared.
2 C1DS	Clear the Class 1 Data Size Register. Writing a one to this bit causes the Class 1 Data Size Register to be cleared. This clears AAD Size as well.
1 —	Reserved
0 C1M	Clear the Class 1 Mode Register. Writing a one to this bit causes the Class 1 Mode Register to be cleared.

14.285 CCB a Status and Error Register, most-significant half (C0CSTA_MS - C5CSTA_MS)

14.285.1 Offset

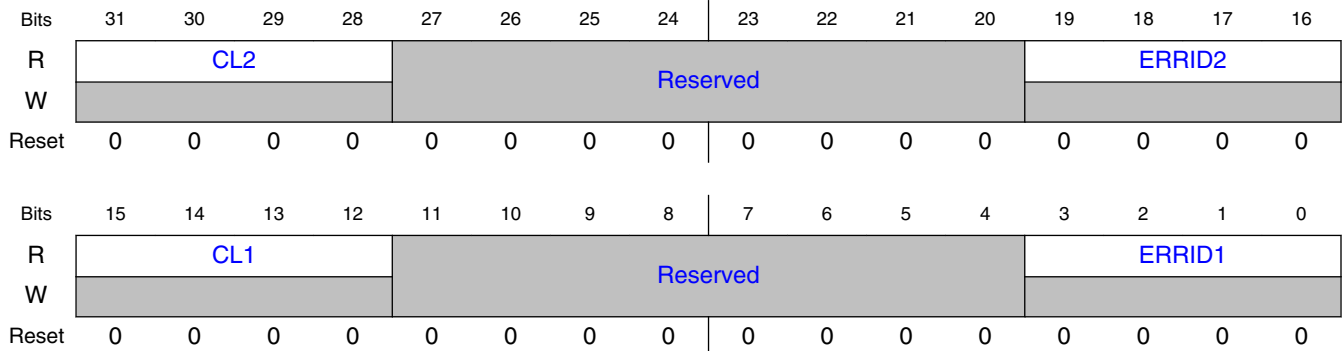
For a = 0 to 5:

Register	Offset	Description
CaCSTA_MS	8_0048h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.285.2 Function

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

14.285.3 Diagram



14.285.4 Fields

Field	Function
31-28 CL2	Class 2 Algorithms. The Class 2 Algorithms bits indicate which algorithm is asserting an error. Others reserved. 0100b - MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 and SHA-512/224, SHA-512/256 1001b - CRC 1010b - SNOW f9 1100b - ZUC Authentication
27-20 —	Reserved
19-16 ERRID2	Error ID 2. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL2 field Others Reserved 0001b - Mode Error 0010b - Data Size Error 0011b - Key Size Error 0110b - Data Arrived out of Sequence Error 1010b - ICV Check Failed 1011b - Internal Hardware Failure 1110b - Invalid CHA combination was selected. 1111b - Invalid CHA Selected
15-12 CL1	Class 1 algorithms. The Class 1 algorithms field indicates which algorithm is asserting an error. Others reserved 0001b - AES 0010b - DES 0101b - RNG 0110b - SNOW 0111b - Kasumi 1000b - Public Key 1011b - ZUC Encryption

Table continues on the next page...

CCB a Status and Error Register, least-significant half (C0CSTA_LS - C5CSTA_LS)

Field	Function
11-4 —	Reserved
3-0 ERRID1	<p>Error ID 1. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL1 field.</p> <p>Others reserved.</p> <ul style="list-style-type: none"> 0001b - Mode Error 0010b - Data Size Error, including PKHA N Memory Size Error 0011b - Key Size Error, including PKHA E Memory Size Error 0100b - PKHA A Memory Size Error 0101b - PKHA B Memory Size Error 0110b - Data Arrived out of Sequence Error 0111b - PKHA Divide by Zero Error 1000b - PKHA Modulus Even Error 1001b - DES Key Parity Error 1010b - ICV Check Failed 1011b - Internal Hardware Failure 1100b - CCM AAD Size Error (either 1. AAD flag in B0 =1 and no AAD type provided, 2. AAD flag in B0 = 0 and AAD provided, or 3. AAD flag in B0 =1 and not enough AAD provided - expecting more based on AAD size.) 1101b - Class 1 CHA is not reset 1110b - Invalid CHA combination was selected. 1111b - Invalid CHA Selected

14.286 CCB a Status and Error Register, least-significant half (C0CSTA_LS - C5CSTA_LS)

14.286.1 Offset

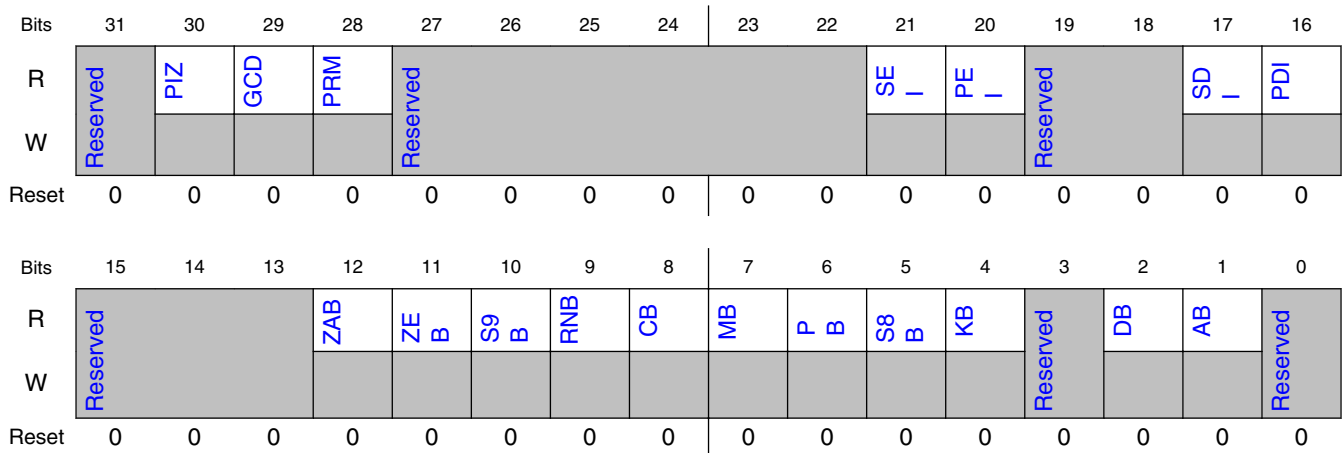
For a = 0 to 5:

Register	Offset	Description
CaCSTA_LS	8_004Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.286.2 Function

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

14.286.3 Diagram



14.286.4 Fields

Field	Function
31 —	Reserved
30 PIZ	Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity.
29 GCD	GCD is One. The greatest common divisor of two numbers is one (that is, the two numbers are relatively prime).
28 PRM	Public Key is Prime. The given number is probably prime (that is, it passes the Miller-Rabin primality test).
27-22 —	Reserved
21 SEI	Class 2 Error Interrupt. The Class 2 Error Interrupt has been asserted. 0b - No Error. 1b - Error Interrupt.
20 PEI	Class 1 Error Interrupt. The Class 1 Error Interrupt has been asserted. 0b - Not Error. 1b - Error Interrupt.
19-18 —	Reserved
17 SDI	Class 2 Done Interrupt. The Class 2 Done Interrupt has been asserted. 0b - Not Done. 1b - Done Interrupt.
16	Class 1 Done Interrupt. The Class 1 Done Interrupt has been asserted.

Table continues on the next page...

CCB a Status and Error Register, least-significant half (C0CSTA_LS - C5CSTA_LS)

Field	Function
PDI	0b - Not Done. 1b - Done Interrupt.
15-13 —	Reserved
12 ZAB	ZUCA Block Busy. This bit indicates that the ZUCA is busy. The CHA can either be busy processing data or resetting. 0b - ZUCA idle 1b - ZUCA busy.
11 ZEB	ZUCE Block Busy. This bit indicates that the ZUCE is busy. The CHA can either be busy processing data or resetting. 0b - ZUCE idle 1b - ZUCE busy.
10 S9B	SNOW f9 Busy. This bit indicates that the SNOW f9 Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - SNOW f9 Idle 1b - SNOW f9 Busy.
9 RNB	RNG Block Busy. This bit indicates that the RNG block is busy. The CHA can either be busy processing data or resetting. 0b - RNG Idle 1b - RNG Busy.
8 CB	CRC Block Busy. This bit indicates that the CRCA is busy. The CHA can either be busy processing data or resetting. 0b - CRCA Idle 1b - CRCA Busy
7 MB	MDHA Busy. This bit indicates that the MDHA is busy. The CHA can either be busy processing data or resetting. 0b - MDHA Idle 1b - MDHA Busy
6 PB	PKHA Busy. This bit indicates that the Public Key Hardware Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - PKHA Idle 1b - PKHA Busy.
5 S8B	SNOW f8. This bit indicates that the SNOW f8 Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - SNOW f8 Idle 1b - SNOW f8 Busy.
4 KB	KFHA Busy. This bit indicates that the Kasumi f8/f9 Hardware Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - KFHA Idle 1b - KFHA Busy
3 —	Reserved
2 DB	DESA Busy. This bit indicates that the DES Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - DESA Idle 1b - DESA Busy.

Table continues on the next page...

Field	Function
1 AB	AESA Busy. This bit indicates that the AES Accelerator is busy. The CHA can either be busy processing data or resetting. 0b - AESA Idle 1b - AESA Busy.
0 —	Reserved

14.287 CCB a AAD Size Register (C0AADSZR - C5AADSZR)

14.287.1 Offset

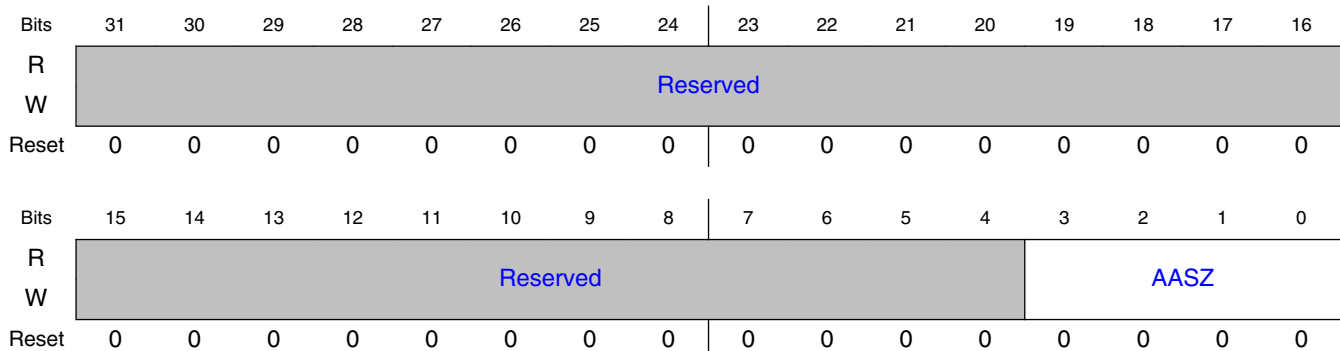
For a = 0 to 5:

Register	Offset	Description
CaAADSZR	8_005Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.287.2 Function

The AAD Size Register is used by AESA to determine how much of the last block of AAD is valid. Like the Class 1 Data Size Register, writing to this register causes the written value to be added to the previous value in the register. The register is automatically written by FIFO LOAD commands.

14.287.3 Diagram



14.287.4 Fields

Field	Function
31-4 —	Reserved
3-0 AASZ	AAD size in Bytes, mod 16.

14.288 Class 1 IV Size Register (C0C1IVSZR - C5C1IVSZR)

14.288.1 Offset

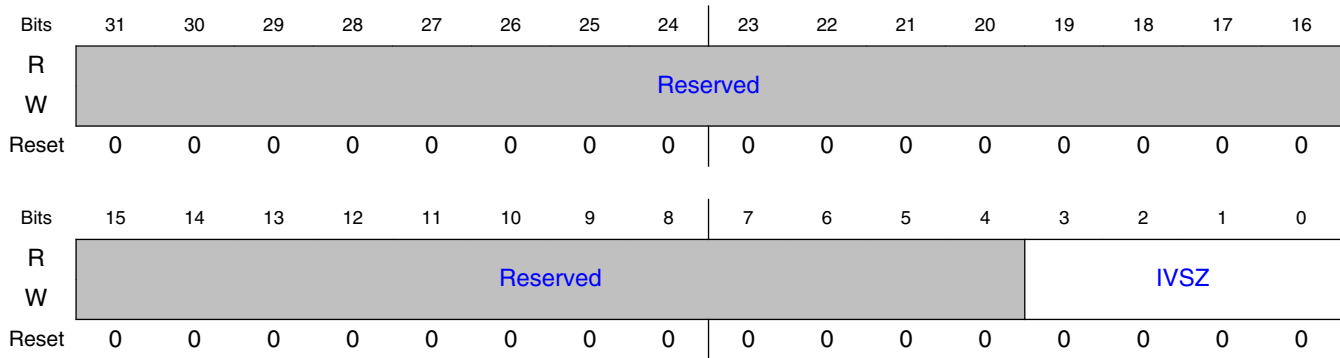
For a = 0 to 5:

Register	Offset	Description
CaC1IVSZR	8_0064h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.288.2 Function

The Class 1 IV Size Register tells the AES module how much of the last block of IV is valid. Like the Class 1 Data Size Register, writing to this register causes the written value to be added to the previous value in the register. The register is automatically written by FIFO LOAD commands.

14.288.3 Diagram



14.288.4 Fields

Field	Function
31-4 —	Reserved
3-0 IVSZ	IV size in bytes, mod 16.

14.289 PKHA A Size Register (C0PKASZR - C5PKASZR)

14.289.1 Offset

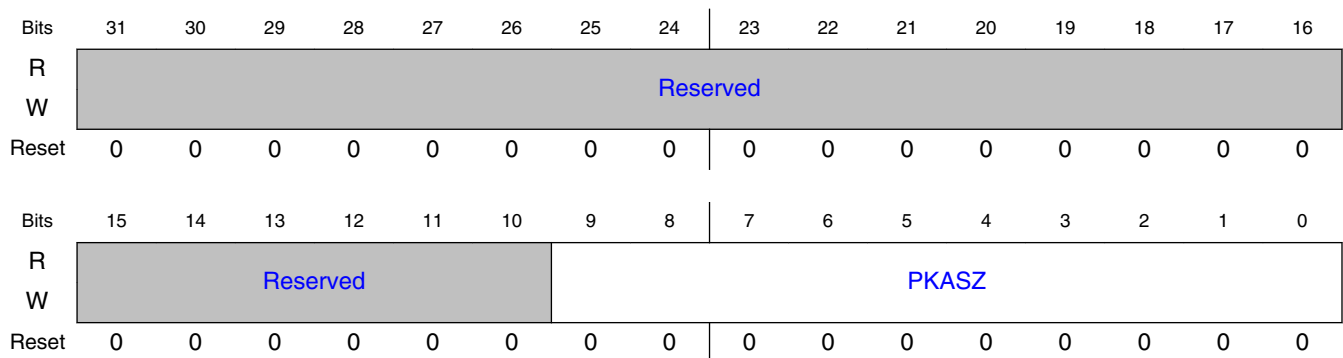
For a = 0 to 5:

Register	Offset	Description
CaPKASZR	8_0084h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.289.2 Function

The PKHA A Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA A Memory. The PKHA A Size Register must be written before the data is written into or read from the PKHA A Memory. This will reserve the PKHA for the current job. The PKHA A Size Register can be automatically written by the MOVE, FIFO LOAD and FIFO STORE commands.

14.289.3 Diagram



14.289.4 Fields

Field	Function
31-10 —	Reserved
9-0 PKASZ	PKHA A Memory key size in bytes.

14.290 PKHA B Size Register (C0PKBSZR - C5PKBSZR)

14.290.1 Offset

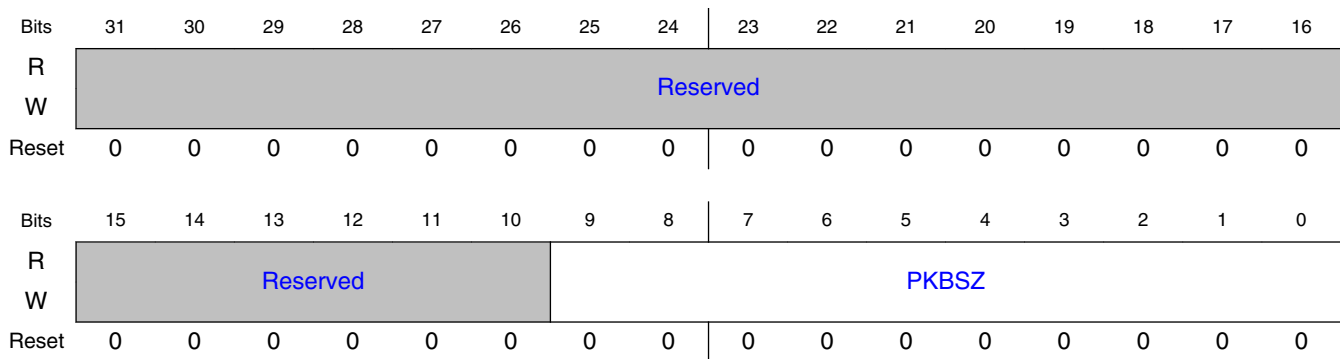
For a = 0 to 5:

Register	Offset	Description
CaPKBSZR	8_008Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.290.2 Function

The PKHA B Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA B Memory. The PKHA B Size Register must be written before the data is written into or read from the PKHA B Memory. This will reserve the PKHA for the current job. The PKHA B Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

14.290.3 Diagram



14.290.4 Fields

Field	Function
31-10 —	Reserved
9-0 PKBSZ	PKHA B Memory key size in bytes.

14.291 PKHA N Size Register (C0PKNSZR - C5PKNSZR)

14.291.1 Offset

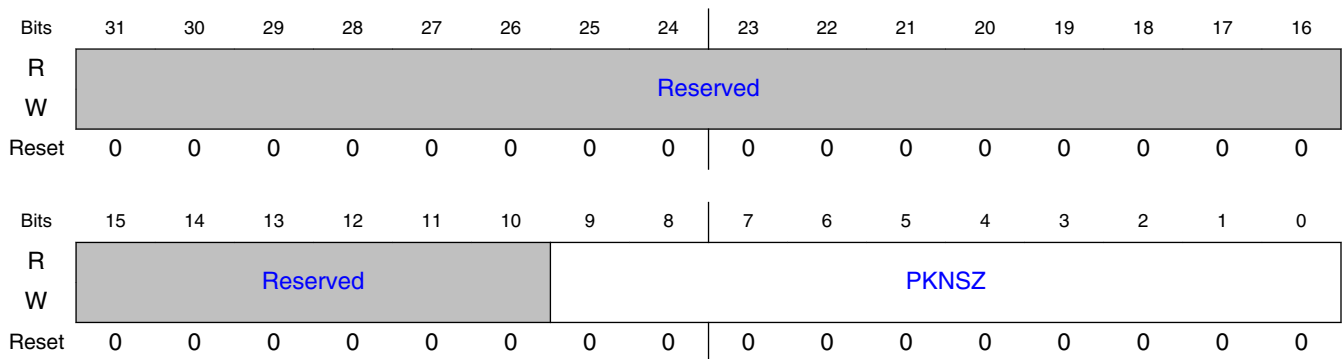
For a = 0 to 5:

Register	Offset	Description
CaPKNSZR	8_0094h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.291.2 Function

The PKHA N Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA N Memory. The PKHA N Size Register must be written before the data is written into or read from the PKHA N Memory. This will reserve the PKHA for the current job. The PKHA N Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

14.291.3 Diagram



14.291.4 Fields

Field	Function
31-10 —	Reserved
9-0 PKNSZ	PKHA N Memory key size in bytes.

14.292 PKHA E Size Register (C0PKESZR - C5PKESZR)

14.292.1 Offset

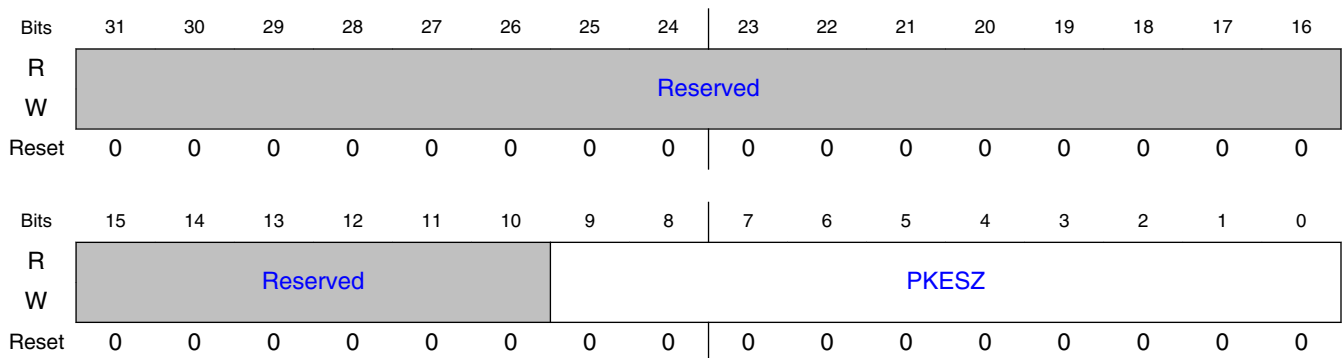
For a = 0 to 5:

Register	Offset	Description
CaPKESZR	8_009Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.292.2 Function

The PKHA E Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA E Memory. The PKHA E Size Register must be written before the data is written into or read from the PKHA E Memory. This will reserve the PKHA for the current job. The PKHA E Size Register is automatically written by the KEY Command.

14.292.3 Diagram



14.292.4 Fields

Field	Function
31-10	Reserved

Table continues on the next page...

CCB a Class 1 Context Register Word b (C0C1CTXR0 - C5C1CTXR15)

Field	Function
—	
9-0 PKESZ	PKHA E Memory key size in bytes.

14.293 CCB a Class 1 Context Register Word b (C0C1CTXR0 - C5C1CTXR15)

14.293.1 Offset

For a = 0 to 5; b = 0 to 15:

Register	Offset	Description
CaC1CTXRb	$8_0100h + (a \times 1_0000h) + (b \times 4h)$	Accessible only when RQDa and DENa are asserted in DECORR.

14.293.2 Function

The Class 1 Context Register holds the context for the Class 1 CHAs. This register is 512 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Context Register is accessible only as full-word reads or writes to sixteen 32-bit registers. The MSB is located at offset 0100h with respect to the register page. This register is cleared automatically when a Black Key is being encrypted or decrypted using AES-CCM.

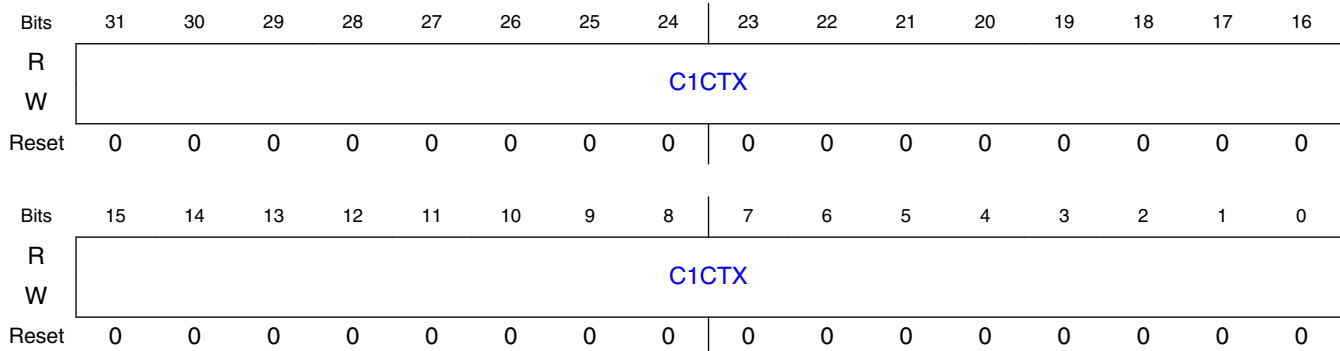
Note that part of the Class 1 Context Register is also used as an "Extended Key Register" when a key larger than 32 bytes is loaded into the Class 1 Key Register. As many 8-byte chunks of the Class 1 Context Register as required are allocated to the Extended Key Register, beginning with the most-significant byte of the register. The Extended Key Register bytes cannot be overwritten and will return 0 when read. The remaining bytes are still available for context data. Note that clearing the Class 1 Context Register will also clear the Extended Key Register.

Note that some commands must block until a previous load to the Class 1 Context Register has completed. Loading the Class 1 Context Register, whether via the KEY Command, LOAD Command or MOVE Command or as a consequence of its usage as the Extended Key Register, sets an internal blocking flag until the Class 1 Context Register load has completed.

The bit assignments of this register are dependent on the algorithm, and in some cases the mode of that algorithm. See the appropriate section for the Context Register format used for that algorithm:

- AES ECB: Section [AES ECB mode use of the Context Register](#)
- AES CBC, OFB and CFB128: Section [AES CBC, OFB, and CFB128 modes use of the Context Register](#)
- AES CTR: Section [AES CTR mode use of the Context Register](#)
- AES XTS: Section [AES XTS mode use of the Context Register](#)
- AES XCBC-MAC, CMAC: Section [AES XCBC-MAC and CMAC Modes use of the Context Register](#)
- AES CCM: Section [AES CCM mode use of the Context Register](#)
- AES GCM: Section [AES GCM mode use of the Mode Register](#)
- AES Optimization modes: [AES optimization modes use of the Context Register](#)
- DES: Section [DESA Context Register](#)
- Kasumi f8/f9: Section [KFHA use of the Context Register](#)
- Random Numbers: Section [RNG use of the Context Register](#)
- SNOW 3G f8: Section [SNOW 3G f8 use of the Context Register](#)
- SNOW 3G f9: Section [SNOW 3G f9 use of the Context Register](#)
- Triple DES: Section [DESA Context Register](#)
- ZUC Authentication: Section [ZUCA use of the Context Register](#)
- ZUC Encryption: Section [ZUCE use of the Context Register](#)

14.293.3 Diagram



14.293.4 Fields

Field	Function
31-0 C1CTX	Class 1 Context.

14.294 CCB a Class 1 Key Registers Word b (C0C1KR0 - C5C1KR7)

14.294.1 Offset

For a = 0 to 5; b = 0 to 7:

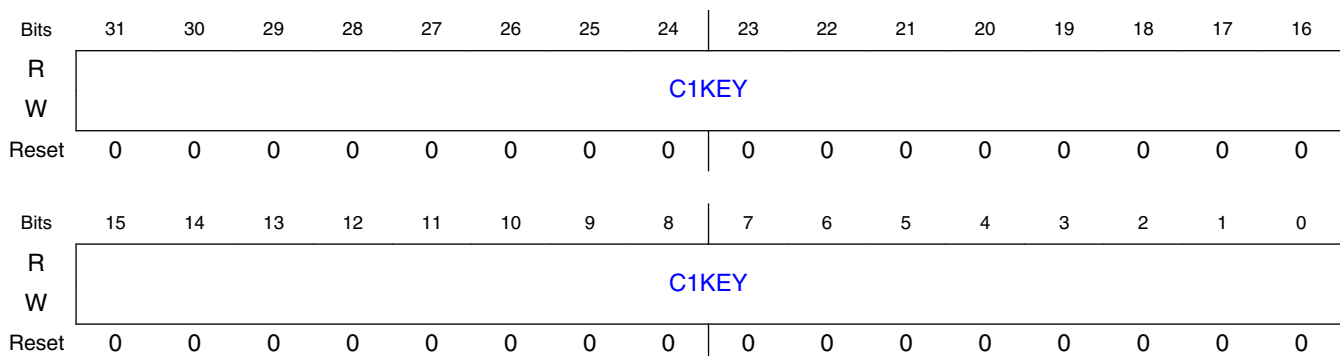
Register	Offset	Description
CaC1KRb	$8_0200h + (a \times 1_0000h) + (b \times 4h)$	Accessible only when RQDa and DENa are asserted in DECORR.

14.294.2 Function

The Class 1 Key Register normally holds the left-aligned key for the Class 1 CHAs. The MSB is in offset 200h. The Class 1 Key Register is 256 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Key Register is accessible only as full-word reads or writes to eight 32-bit registers. Although the Class 1 Key Register is only 32 bytes long, via the KEY command it is possible to load a key larger than 32 bytes. In this case part of the Class 1 Context Register is used as an "Extended Key Register". The first 32 bytes are loaded into the Class 1 Key Register, then, starting with the most-significant end, as many 8-byte chunks of the Class 1 Context Register as required are allocated to the Extended Key Register and are used to hold the remaining key bytes. The Extended Key Register bytes cannot be overwritten and will return 0 when read. The remaining bytes are still available for context data. Clearing the Class 1 Key Register will also clear the Extended Key Register bytes in the Class 1 Context Register. The other bytes in the Class 1 Context Register will not be cleared. Note that clearing the Class 1 Context Register will not clear the Extended Key Register bytes.

The Class 1 Key Register can be written via a MOVE Command, a MATH Command, a LOAD Command or a KEY Command. Before the value in the Class 1 Key Register can be used in a cryptographic operation, the size of the key must be written into the Class 1 Key Size Register. Once the Class 1 Key Size Register has been written, the Class 1 Key Register cannot be written again until the Class 1 Key Size Register has been cleared. Writing the Class 1 Key Register via a KEY Command automatically writes the Class 1 Key Size Register, but if the Class 1 Key Register is written using a MOVE, MATH or LOAD Command the Class 1 Key Size Register must be written via a separate command after the Class 1 Key Register has been written. But until the Class 1 Key Size Register has been written the Class 1 Key Register remains writable via STORE/SEQ STORE, MATH or MOVE commands and readable via LOAD/SEQ LOAD, MATH or MOVE commands. If the Class 1 Key Size Register and the Class 1 Key Register have been cleared via the Clear Written Register, the Class 1 Key Register becomes writable and readable again. This allows the Class 1 Key Register to be used for temporary storage if it is not currently needed to hold a cryptographic key. Even when the Class 1 Key Register holds a key (i.e. the Class 1 Key Size Register has been written) it may still be possible to store the key in memory in encrypted form. The FIFO STORE Command can be used to store an encrypted copy of this key (i.e. a Black Key), unless storing the key has been prohibited via the NWB bit in the KEY Command. The encrypted key can later be loaded into the Class 1 Key Register via the KEY Command by setting the ENC bit to indicate that this is a Black (i.e. encrypted) Key. The Black Key will automatically be decrypted before it is loaded into the Class 1 Key Register. A Black Key can be loaded as long as the Key Encryption Key (KEK) has not been changed (as a consequence of a security violation or a POR). Note that the Class 1 Key register is cleared when any key (including Class 2 Keys) is encrypted or decrypted, so if a Black Key is to be loaded into or stored from the Class 2 Key Register, that must be done prior to loading a key into the Class 1 Key Register. Similarly, if a key is to be stored from the Class 1 Key Register as a Black Key and also used in a cryptographic operation, the cryptographic operation should be performed first, or the key will have to be loaded a second time.

14.294.3 Diagram



14.294.4 Fields

Field	Function
31-0 C1KEY	Class 1 Key.

14.295 CCB a Class 2 Mode Register (C0C2MR - C5C2MR)

14.295.1 Offset

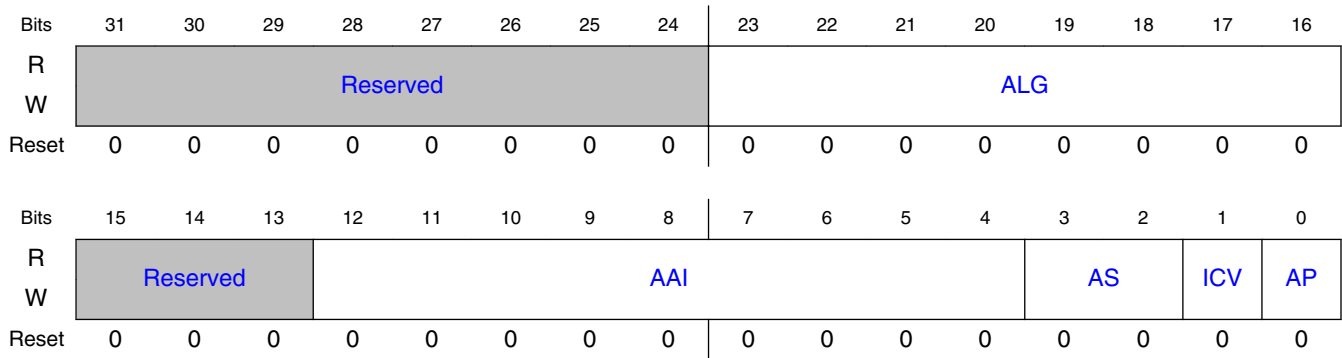
For a = 0 to 5:

Register	Offset	Description
CaC2MR	8_0404h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.295.2 Function

The Class 2 Mode Register is used to tell the Class 2 CHA which operation is being requested. The interpretation of this register is unique for each CHA. The Class 2 Mode Register is automatically written by the OPERATION Command. This register is automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

14.295.3 Diagram



14.295.4 Fields

Field	Function																					
31-24 —	Reserved. Must be 0.																					
23-16 ALG	<p>Algorithm. This field specifies which algorithm has been requested for an OPERATION command.</p> <ul style="list-style-type: none"> 00010000b - AES, when used as a Class 2 CHA 01000000b - MD5 01000001b - SHA-1 01000010b - SHA-224 01000011b - SHA-256 01000100b - SHA-384 01000101b - SHA-512 01000110b - SHA-512/224 01000111b - SHA-512/256 10010000b - CRC 10100000b - SNOW 3G f9 11000000b - ZUC Authentication 																					
15-13 —	Reserved. Must be 0.																					
12-4 AAI	<p>Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. A detailed list of additional modes can be found below.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Value</th> <th>Description</th> <th>Valid with ALG</th> </tr> </thead> <tbody> <tr> <td>001h</td> <td>IEEE 802</td> <td>CRC</td> </tr> <tr> <td>002h</td> <td>IETF 3385</td> <td>CRC</td> </tr> <tr> <td>004h</td> <td>CUST_POLY</td> <td>CRC</td> </tr> <tr> <td>010h</td> <td>DIS</td> <td>CRC</td> </tr> <tr> <td>020h</td> <td>DOS</td> <td>CRC</td> </tr> <tr> <td>040h</td> <td>DOC</td> <td>CRC</td> </tr> </tbody> </table>	Value	Description	Valid with ALG	001h	IEEE 802	CRC	002h	IETF 3385	CRC	004h	CUST_POLY	CRC	010h	DIS	CRC	020h	DOS	CRC	040h	DOC	CRC
Value	Description	Valid with ALG																				
001h	IEEE 802	CRC																				
002h	IETF 3385	CRC																				
004h	CUST_POLY	CRC																				
010h	DIS	CRC																				
020h	DOS	CRC																				
040h	DOC	CRC																				

Table continues on the next page...

CCB a Class 2 Key Size Register (C0C2KSR - C5C2KSR)

Field	Function		
	Value	Description	Valid with ALG
	001h	HMAC	MD5, SHA-*
	002h	SMAC	MD5, SHA-1
	004h	IPAD OPAD Generation	MD5, SHA-*
	0C8h	f9	SNOW 3G
	0C8h	Authentication	ZUC
	060h	CMAC	AES
	070h	XCBC-MAC	AES
	Others	Reserved	
3-2 AS	Algorithm State. This field defines the state of the algorithm that is being executed. Not every algorithm uses this field. Check the individual algorithm sections to see if this field is used. 00b - Update. 01b - Initialize. 10b - Finalize. 11b - Initialize/Finalize.		
1 ICV	ICV Checking. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.		
0 AP	Authenticate / Protect. 0b - Authenticate 1b - Protect.		

14.296 CCB a Class 2 Key Size Register (C0C2KSR - C5C2KSR)

14.296.1 Offset

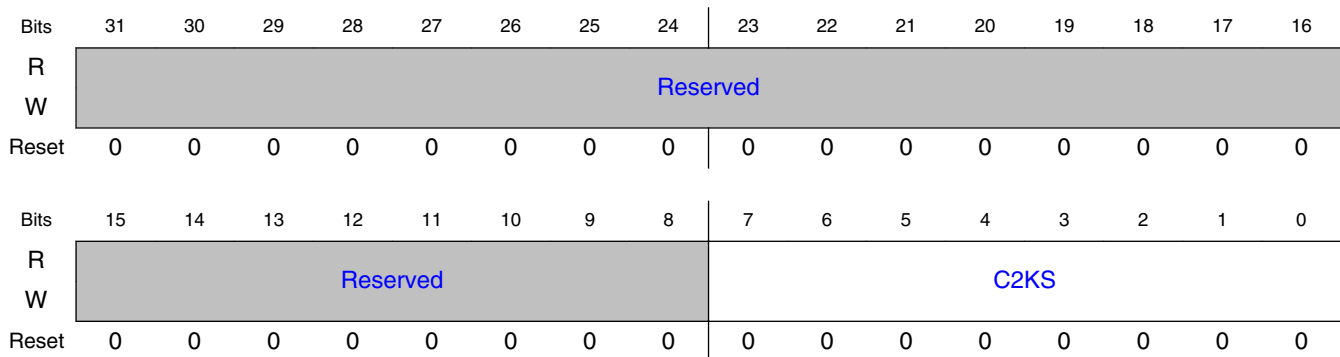
For a = 0 to 5:

Register	Offset	Description
CaC2KSR	8_040Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.296.2 Function

The Class 2 Key Size Register is used to tell the Class 2 CHA the size of the key that was loaded into the Class 2 Key Register. The Class 2 Key Size Register must be written after the key is written into the Class 2 Key Register. Writing to the Class 2 Key Size Register will prevent the user from modifying the Class 2 Key Register. The Class 2 Key Size Register is automatically written by the Key Command. This register is cleared when Trusted Descriptors are checked or re-signed.

14.296.3 Diagram



14.296.4 Fields

Field	Function
31-8 —	Reserved
7-0 C2KS	Class 2 key size in bytes.

14.297 CCB a Class 2 Data Size Register (C0C2DSR - C5C2DSR)

14.297.1 Offset

For a = 0 to 5:

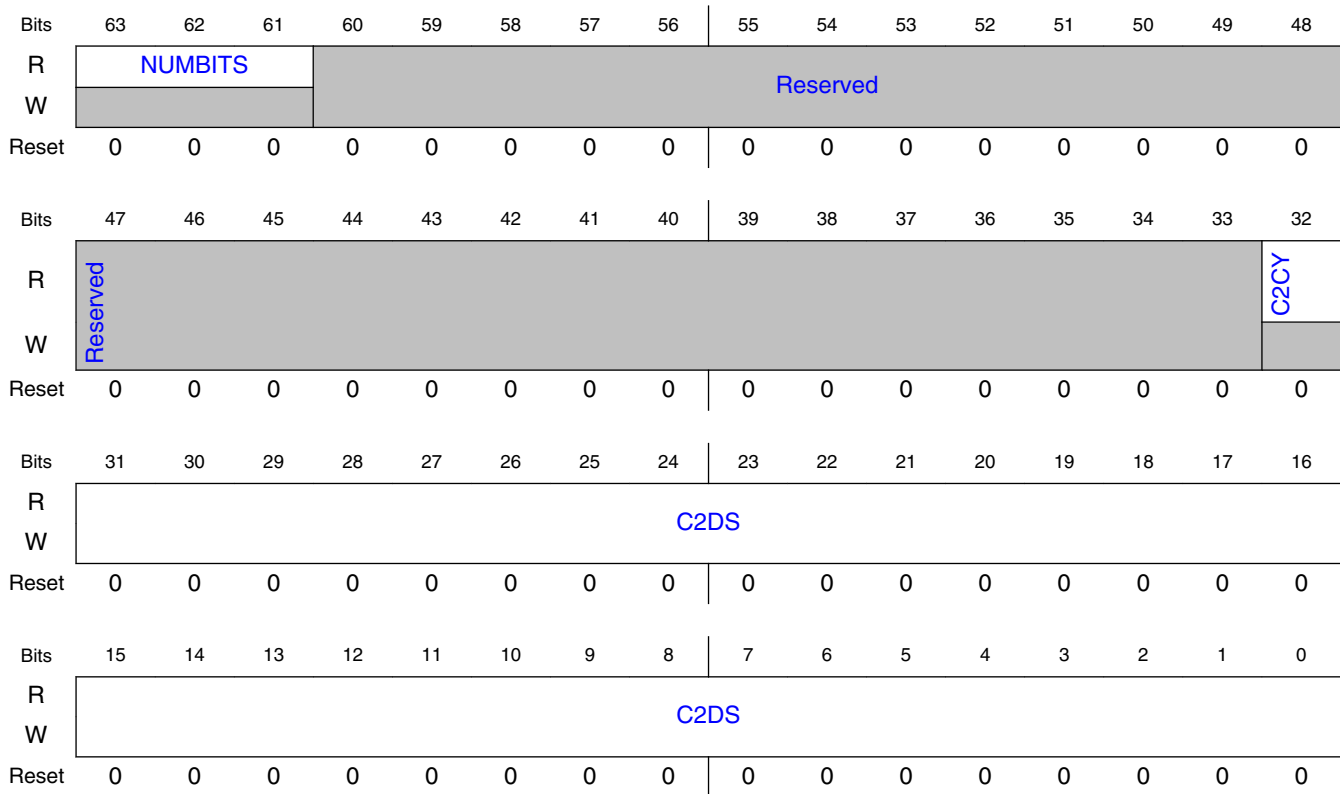
CCB a Class 2 Data Size Register (C0C2DSR - C5C2DSR)

Register	Offset	Description
CaC2DSR	8_0410h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.297.2 Function

The Class 2 Data Size Register is used to tell the Class 2 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C2CY and C2DS fields to form a data size that is measured in bits. Note that writing to the C2DS field in this register causes the written value to be added to the previous value in that field. That is, if the C2DS field currently has the value 14, writing 2 to the least-significant half of the Class 2 Data Size register (i.e. the C2DS field) will result in a value of 16 in the C2DS field. Although there is a C2CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C2CY and C2DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic information FIFO entries are enabled. This register is reset when checking the signature over, or re-signing, Trusted Descriptors. Since the Class 2 Data Size Register holds more than 32 bits, it is accessed from the IP bus as two 32-bit registers.

14.297.3 Diagram



14.297.4 Fields

Field	Function
63-61 NUMBITS	Class 2 Data Size Number of bits. For bit-oriented operations, this value is appended to the C2CY and C2DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value (C2CY C2DS NUMBITS). Note that if NUMBITS is nonzero, C2DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value.
60-33 —	Reserved
32 C2CY	Class 2 Data Size Carry. Although this field is not writable, it will be set if a write to C2DS causes a carry out of the msb of C2DS.
31-0 C2DS	Class 2 Data Size in Bytes. This is the number of whole bytes of data that will be consumed by the Class 2 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero.

14.298 CCB a Class 2 ICV Size Register (C0C2ICVSZR - C5C2ICVSZR)

14.298.1 Offset

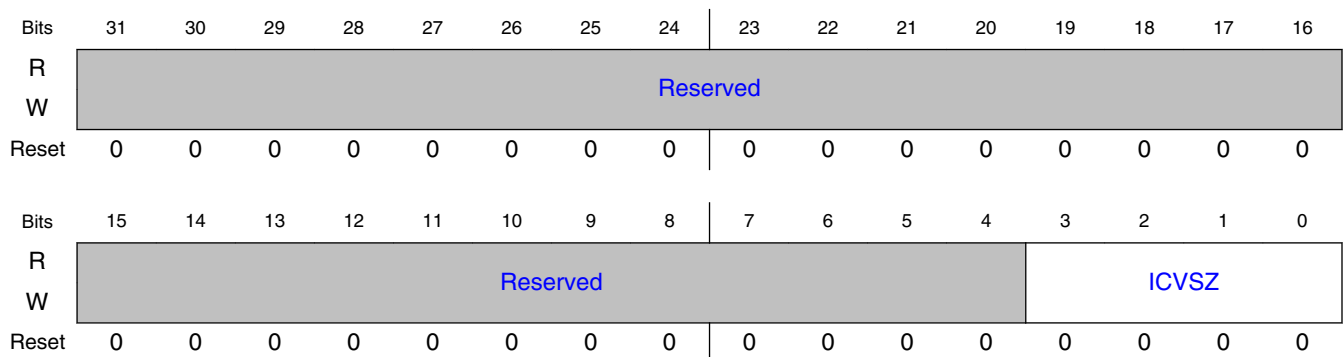
For a = 0 to 5:

Register	Offset	Description
CaC2ICVSZR	8_041Ch + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.298.2 Function

The Class 2 ICV Size Register indicates how much of the last block of ICV is valid when performing MDHA integrity check operations (e.g. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 and MD5). For AES Class 2 operations the Class 2 ICV Size Register indicates the size of the ICV. Writing to this register causes the written value to be added to the previous value in the register. This register is automatically written by FIFO LOAD commands. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

14.298.3 Diagram



14.298.4 Fields

Field	Function
31-4 —	Reserved
3-0 ICVSZ	Class 2 ICV size (mod 8) in bytes. For MDHA, writing 0 to this field will be interpreted as an ICV size of 8 bytes. For AESA, writing 0, 1, 2 or 3 to this field will be interpreted as an ICV size of 16 bytes.

14.299 CCB a Class 2 Context Register Word b (C0C2CTXR0 - C5C2CTXR17)

14.299.1 Offset

For a = 0 to 5; b = 0 to 17:

Register	Offset	Description
CaC2CTXRb	$8_0500h + (a \times 1_0000h) + (b \times 4h)$	Accessible only when RQDa and DENa are asserted in DECORR.

14.299.2 Function

The Class 2 Context Register holds the context for the Class 2 CHAs. This register is 576 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Context Register is accessible only as full-word reads or writes to eighteen 32-bit registers. The MSB is located at offset 500h with respect to the register page. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

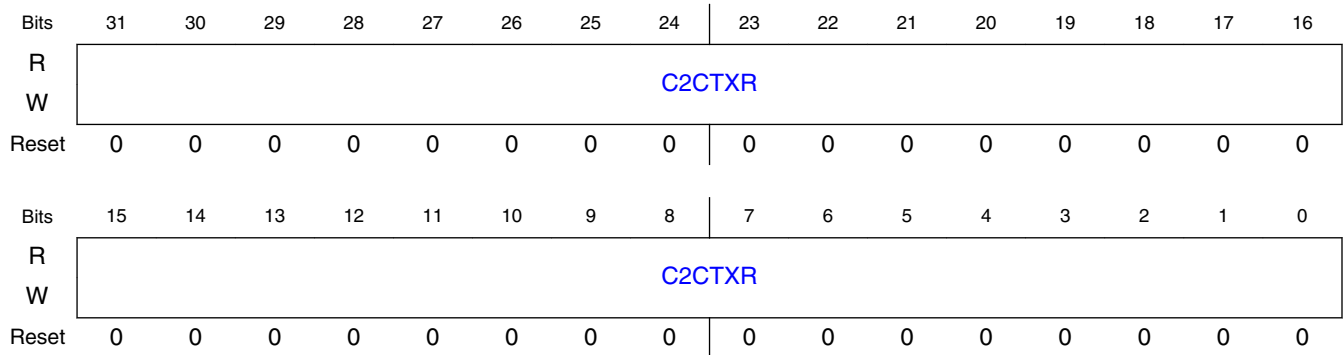
The bit assignments for this register are dependent on the algorithm. See the appropriate section for the Context Register format used by that algorithm.

- CRC: Section [CRCA Context Register](#)
- Kasumi f9: The Kasumi f9 authentication algorithm is implemented in the same CHA as the Kasumi f8 encryption algorithm. This is considered a Class 1 CHA. See Section [KFHA use of the Context Register](#)
- MD5: Section [MDHA use of the Context Register](#)

CCB a Class 2 Key Register Word b (C0C2KEYR0 - C5C2KEYR31)

- SHA-*: Section [MDHA use of the Context Register](#)
- SNOW 3G f9: Section [SNOW 3G f9 use of the Context Register](#)
- ZUC Authentication: Section [ZUCA use of the Context Register](#)

14.299.3 Diagram



14.299.4 Fields

Field	Function
31-0 C2CTXR	Class 2 Context.

14.300 CCB a Class 2 Key Register Word b (C0C2KEYR0 - C5C2KEYR31)

14.300.1 Offset

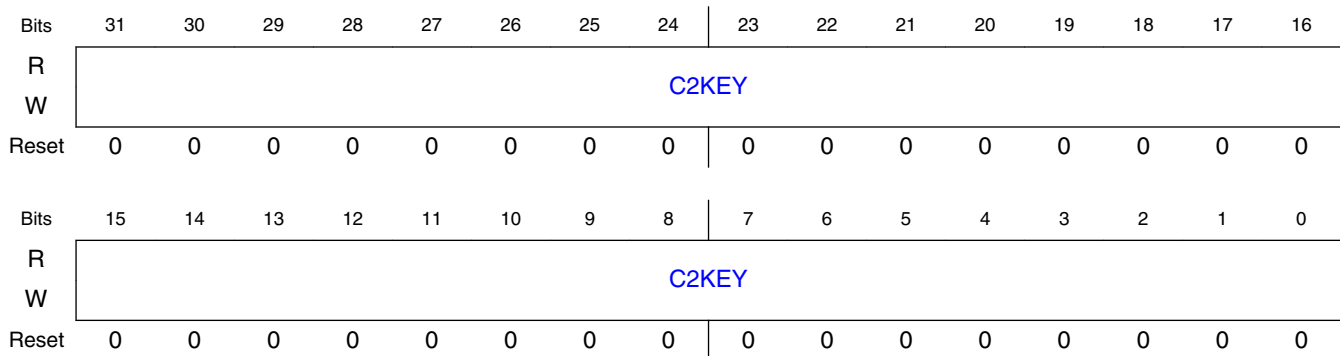
For a = 0 to 5; b = 0 to 31:

Register	Offset	Description
CaC2KEYRb	$8_0600h + (a \times 1_0000h) + (b \times 4h)$	Accessible only when RQDa and DENa are asserted in DECORR.

14.300.2 Function

The Class 2 Key Register holds the key for the Class 2 CHAs. For non-split keys, the key is left-aligned in the C2KEYR. Split keys are the IPAD and OPAD for certain MDHA operations. Note that the size of a split key is the sum of the sizes of the IPAD and the OPAD. The IPAD is left-aligned within the C2KEYR, and the OPAD is left-aligned starting at the mid-point of the C2KEYR. This register is 1024 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Key Register is accessible only as full-word reads or writes to thirty-two 32-bit registers. The MSB is located at offset 600h with respect to the start of the register page. This register is automatically written by KEY commands. The recommended practice is to write the Class 2 Key Register prior to writing any of the other Class 2 registers. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

14.300.3 Diagram



14.300.4 Fields

Field	Function
31-0 C2KEY	Class 2 Key.

14.301 CCB a FIFO Status (C0FIFOSTA - C5FIFOSTA)

14.301.1 Offset

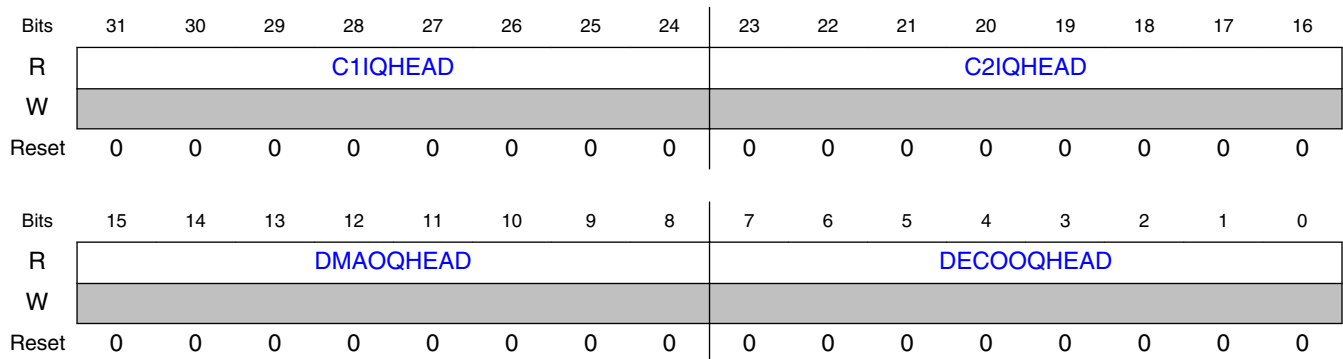
For a = 0 to 5:

Register	Offset	Description
CaFIFOSTA	8_07C0h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.301.2 Function

The CCB FIFO Status Register is used during debug to facilitate reading the CCB FIFOs. Software must keep track of the data written to the Input Data FIFO (CCB a Input Data FIFO (C0IFIFO - C5IFIFO)), but the data within the Output Data FIFO (CCB a Output Data FIFO (C0OFIFO - C5OFIFO)) can be read out. Both the Class 1 Alignment Block and the Class 2 Alignment Block (see Alignment blocks) draw data from the Input Data FIFO, and both the DMA and the DECO Alignment Block draw data from the Output Data FIFO. Reading the CaFIFOSTA register returns the current heads of the Alignment Block and DMA queues within these two FIFOs. Note that the values in this register will change as descriptors are executed, so the register should be read when the DECO is not actively executing a descriptor.

14.301.3 Diagram



14.301.4 Fields

Field	Function
31-24 C1IQHEAD	This is the current head of the Class 1 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 1 Alignment Block.
23-16 C2IQHEAD	This is the current head of the Class 2 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 2 Alignment Block.
15-8 DMAOQHEAD	This is the current head of the DMA queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DMA controller.
7-0 DECOOQHEAD	This is the current head of the DECO Alignment Block queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DECO Alignment Block. This is used during "out snooping" operations, i.e. when data is passed first through a Class 1 CHA and the results pushed into the OFIFO, and from there the results are sent through a Class 2 CHA.

14.302 CCB a iNformation FIFO When STYPE Is Not 10 (CONFIFO - C5NFIFO)

14.302.1 Offset

For a = 0 to 5:

Register	Offset	Description
CaNFIFO	8_07D0h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

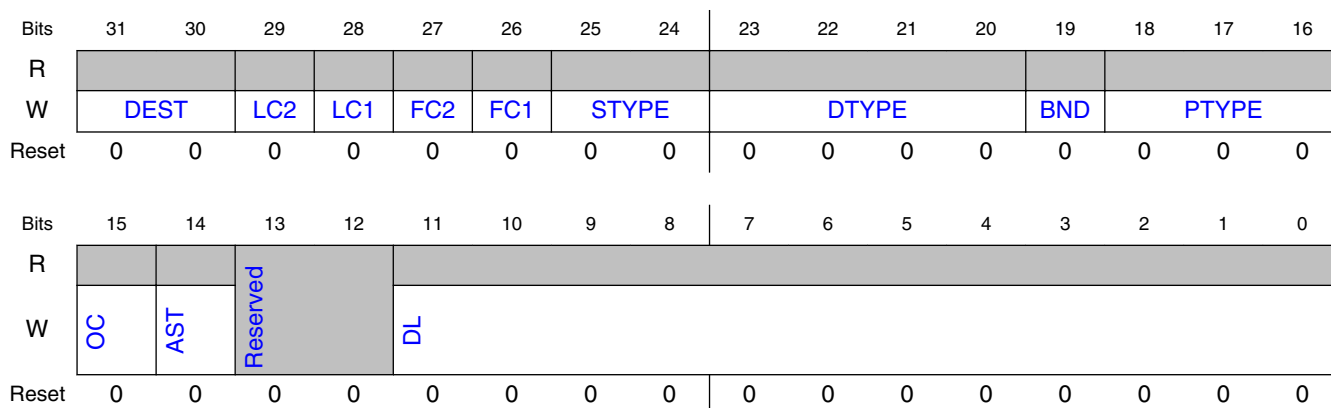
14.302.2 Function

The iNformation FIFO (Input Information FIFO) is used to control the movement of data from any of four sources to any of the three alignment blocks (see [Alignment blocks](#)). The four sources are the Input Data FIFO, the Output Data FIFO, the CCB Padding Block and the Auxiliary Data FIFO. Note that the only way to get data out of any of these sources other than via the Output Data FIFO is to use an iNformation FIFO entry.

The depth of the iNformation FIFO is four entries. During normal operation, SEC will not cause the iNformation FIFO to overflow. Care must be taken to avoid overflowing the iNformation FIFO when writing to it directly as this can cause SEC to hang. This register can be automatically written by the FIFO LOAD and MOVE commands. (If data is written to the Input Data FIFO with the LOAD Command, or some other way that does not automatically generate an info FIFO entry, the user is responsible for writing to the iNformation FIFO. See LOAD Command destination codes 78h and 7Ah in [LOAD commands](#).)

A single address is used to write to the iNformation FIFO for any particular DECO. The format of non-padding iNformation FIFO entries (STYPE ≠ 10) is shown below. The format of padding iNformation FIFO entries (STYPE = 10) is shown in [CCB a iNformation FIFO When STYPE Is 10 \(C0NFIFO_2 - C5NFIFO_2\)](#).

14.302.3 Diagram



14.302.4 Fields

Field	Function
31-30 DEST	Destination. This specifies if the current entry defines data for the Class 1 CHA and/or Class 2 CHA. It can also be used to remove data from the FIFOs that are not needed. 00b - DECO Alignment Block. If DTYPE is Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPE other than Eh (drop) or Fh (message) is used with the DECO Alignment Block destination. 01b - Class 1. 10b - Class 2. 11b - Both Class 1 and Class 2.
29 LC2	Last Class 2. This bit should be set when the data defined in the current iNformation FIFO entry is the last data going to the CHA or the last data prior to receiving ICV data going to the Class 2 CHA, as well as following the ICV data. When LC2 = 1 the alignment block will be emptied as well.

Table continues on the next page...

Field	Function																														
28 LC1	Last Class 1. This bit should be set when the data defined in the current information FIFO entry is the last data for the Class 1 CHA. When LC1 =1 a flush will be done and the alignment block will be emptied as well.																														
27 FC2	Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted. This bit can be set only via a LOAD Command and is only to be used when a MOVE from the Class 2 Alignment Block is to be done and the MOVE Command was executed when automatic information FIFO entries were disabled. In such cases, setting the LC2 bit could result in unpredictable behavior and the FC2 bit should be used.																														
26 FC1	Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.																														
25-24 STYPE	<p>Source Type. This field defines the source of the data for the Alignment Block(s). (This is the register format description when STYPE ≠ 10. The register uses a different format when STYPE = 10. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2).) For STYPE ≠ 10, there are two interpretations of the STYPE field, depending on the setting of the AST bit:</p> <table border="1"> <thead> <tr> <th>AST=0</th> <th>AST=1</th> </tr> </thead> <tbody> <tr> <td>STYPE = 00 : Input Data FIFO</td> <td>STYPE = 00 : Auxiliary Data FIFO ⁻¹</td> </tr> <tr> <td>STYPE = 01 : Output Data FIFO</td> <td></td> </tr> <tr> <td colspan="2">STYPE = 10 : Padding Block. The register format is different for this STYPE. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2).</td> </tr> <tr> <td>STYPE = 11 : Out snooping ⁻¹</td> <td>STYPE = 11 : Outsnooping from Auxiliary Data FIFO ⁻¹</td> </tr> </tbody> </table> <p>1. The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang.</p> <p>2. When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.</p> <p>3. This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.</p>	AST=0	AST=1	STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO ⁻¹	STYPE = 01 : Output Data FIFO		STYPE = 10 : Padding Block. The register format is different for this STYPE. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2) .		STYPE = 11 : Out snooping ⁻¹	STYPE = 11 : Outsnooping from Auxiliary Data FIFO ⁻¹																				
AST=0	AST=1																														
STYPE = 00 : Input Data FIFO	STYPE = 00 : Auxiliary Data FIFO ⁻¹																														
STYPE = 01 : Output Data FIFO																															
STYPE = 10 : Padding Block. The register format is different for this STYPE. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2) .																															
STYPE = 11 : Out snooping ⁻¹	STYPE = 11 : Outsnooping from Auxiliary Data FIFO ⁻¹																														
23-20 DTYPE	<p>Data Type. This field defines the type of data that is going through the Input Data FIFO. This is used by the CHA to determine what type of processing needs to be done on the data. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.</p> <table border="1"> <thead> <tr> <th>DTYPE</th> <th>Data type for PKHA</th> <th>Data type for other CHAs</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>PKHA A0</td> <td>Reserved</td> </tr> <tr> <td>1h</td> <td>PKHA A1</td> <td>AAD (AES/AES GCM)</td> </tr> <tr> <td>2h</td> <td>PKHA A2</td> <td>IV for (AES/AES GCM)</td> </tr> <tr> <td>3h</td> <td>PKHA A3</td> <td>SAD (AES/A)</td> </tr> <tr> <td>4h</td> <td>PKHA B0</td> <td>Reserved</td> </tr> <tr> <td>5h</td> <td>PKHA B1</td> <td>Reserved</td> </tr> <tr> <td>6h</td> <td>PKHA B2</td> <td>Reserved</td> </tr> <tr> <td>7h</td> <td>PKHA B3</td> <td>Reserved</td> </tr> <tr> <td>8h</td> <td>PKHA N</td> <td>Reserved</td> </tr> </tbody> </table>	DTYPE	Data type for PKHA	Data type for other CHAs	0h	PKHA A0	Reserved	1h	PKHA A1	AAD (AES/AES GCM)	2h	PKHA A2	IV for (AES/AES GCM)	3h	PKHA A3	SAD (AES/A)	4h	PKHA B0	Reserved	5h	PKHA B1	Reserved	6h	PKHA B2	Reserved	7h	PKHA B3	Reserved	8h	PKHA N	Reserved
DTYPE	Data type for PKHA	Data type for other CHAs																													
0h	PKHA A0	Reserved																													
1h	PKHA A1	AAD (AES/AES GCM)																													
2h	PKHA A2	IV for (AES/AES GCM)																													
3h	PKHA A3	SAD (AES/A)																													
4h	PKHA B0	Reserved																													
5h	PKHA B1	Reserved																													
6h	PKHA B2	Reserved																													
7h	PKHA B3	Reserved																													
8h	PKHA N	Reserved																													

Table continues on the next page...

CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C5NFIFO)

Field	Function		
	9h	PKHA E	Reserved
	Ah	Reserved	ICV
	Ch	PKHA A	Reserved
	Dh	PKHA B	Reserved
	Eh	Reserved	DECO Ignore (i.e. Skip)
	Fh	Reserved	Message Data
	Other values are reserved.		
<ol style="list-style-type: none"> The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang. When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO. This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO. 			
19 BND	Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE ≠ 10 (Padding Block).		
18-16 PTYPE	Pad Type. This field is ignored if STYPE ≠ 10 (Padding Block). 000b - All Zero. 001b - Random with nonzero bytes. 010b - Incremented (Starting with 01h). 011b - Random. 100b - All Zero with last byte containing the number of 0 bytes. 101b - Random with nonzero bytes with last byte 0. 110b - N bytes of padding all containing the number N -1. 111b - Random with nonzero bytes with last byte N.		
15 OC	OFIFO Continuation - This bit causes the final word to not be popped from the Output Data FIFO.		
14 AST	Additional Source Types. This bit selects between two meanings of the STYPE field. See the description of the STYPE field.		
13-12 —	Reserved		
11-0 DL	Data Length. The number of bytes that will be passed to a CHA. A maximum of 12 bits is supported. This means for larger chunks of data multiple entries in the iNformation FIFO will be required.		

- The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang.
- When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.
- This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.

14.303 CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C5NFIFO_2)

14.303.1 Offset

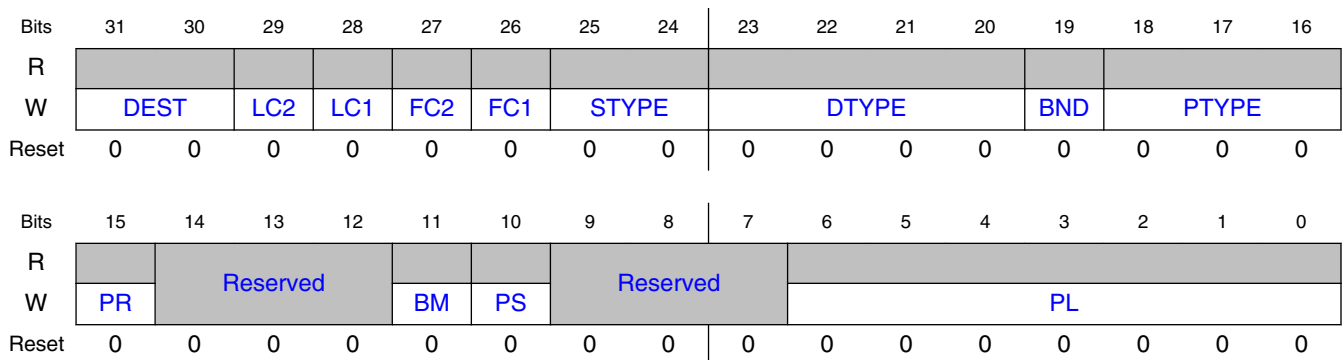
For a = 0 to 5:

Register	Offset	Description
CaNFIFO_2	8_07D0h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.303.2 Function

The format of padding iNformation FIFO entries (STYPE = 10) is shown below. The format of non-padding iNformation FIFO entries (STYPE <> 10) is shown in [CCB a iNformation FIFO When STYPE Is Not 10 \(C0NFIFO - C5NFIFO\)](#).

14.303.3 Diagram



14.303.4 Fields

Field	Function
31-30 DEST	Destination. This specifies if the current entry defines data for the Class 1 CHA and/or Class 2 CHA. It can also be used to remove data from the FIFOs that are not needed.

Table continues on the next page...

CCB aInformation FIFO When STYPE Is 10 (CONFIFO_2 - C5NFIFO_2)

Field	Function																																				
	<p>00b - DECO Alignment Block. If DTYPE is Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPE other than Eh (drop) or Fh (message) is used with the DECO Alignment Block destination.</p> <p>01b - Class 1. 10b - Class 2. 11b - Both Class 1 and Class 2.</p>																																				
29 LC2	Last Class 2. This bit should be set when the data defined in the current iNformation FIFO entry is the last data going to the CHA or the last data prior to receiving ICV data going to the Class 2 CHA, as well as following the ICV data. When LC2 =1 the alignment block will be emptied as well.																																				
28 LC1	Last Class 1. This bit should be set when the data defined in the current iNformation FIFO entry is the last data for the Class 1 CHA. When LC1 =1 a flush will be done and the alignment block will be emptied as well.																																				
27 FC2	<p>Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted.</p> <p>This bit can only be set via a LOAD Command and is only to be used when a MOVE from the Class 2 Alignment Block is to be done and the MOVE Command was executed when automatic information FIFO entries were disabled. In such cases, setting the LC2 bit could result in unpredictable behavior and the FC2 bit should be used.</p>																																				
26 FC1	Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.																																				
25-24 STYPE	<p>Source Type. This field defines the source of the data for the Alignment Block(s). This is the register format description when STYPE = 10 (Padding Block). For STYPE ≠ 10, see CCB aInformation FIFO When STYPE Is Not 10 (CONFIFO - C5NFIFO).</p> <p>* When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.</p> <p>** The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO must be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO will hang.</p>																																				
23-20 DTYPE	<p>Data Type. This field defines the type of data that is going through the Input Data FIFO. This is used by the CHA to determine what type of processing needs to be done on the data. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.</p> <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>DTYPE</th> <th>Data type for PKHA</th> <th>Data type for other CHAs</th> </tr> </thead> <tbody> <tr> <td>0h</td> <td>PKHA A0</td> <td>Reserved</td> </tr> <tr> <td>1h</td> <td>PKHA A1</td> <td>AAD (AESA AES GCM)</td> </tr> <tr> <td>2h</td> <td>PKHA A2</td> <td>IV for (AESA AES GCM)</td> </tr> <tr> <td>3h</td> <td>PKHA A3</td> <td>SAD (AESA)</td> </tr> <tr> <td>4h</td> <td>PKHA B0</td> <td>Reserved</td> </tr> <tr> <td>5h</td> <td>PKHA B1</td> <td>Reserved</td> </tr> <tr> <td>6h</td> <td>PKHA B2</td> <td>Reserved</td> </tr> <tr> <td>7h</td> <td>PKHA B3</td> <td>Reserved</td> </tr> <tr> <td>8h</td> <td>PKHA N</td> <td>Reserved</td> </tr> <tr> <td>9h</td> <td>PKHA E</td> <td>Reserved</td> </tr> <tr> <td>Ah</td> <td>Reserved</td> <td>ICV</td> </tr> </tbody> </table>	DTYPE	Data type for PKHA	Data type for other CHAs	0h	PKHA A0	Reserved	1h	PKHA A1	AAD (AESA AES GCM)	2h	PKHA A2	IV for (AESA AES GCM)	3h	PKHA A3	SAD (AESA)	4h	PKHA B0	Reserved	5h	PKHA B1	Reserved	6h	PKHA B2	Reserved	7h	PKHA B3	Reserved	8h	PKHA N	Reserved	9h	PKHA E	Reserved	Ah	Reserved	ICV
DTYPE	Data type for PKHA	Data type for other CHAs																																			
0h	PKHA A0	Reserved																																			
1h	PKHA A1	AAD (AESA AES GCM)																																			
2h	PKHA A2	IV for (AESA AES GCM)																																			
3h	PKHA A3	SAD (AESA)																																			
4h	PKHA B0	Reserved																																			
5h	PKHA B1	Reserved																																			
6h	PKHA B2	Reserved																																			
7h	PKHA B3	Reserved																																			
8h	PKHA N	Reserved																																			
9h	PKHA E	Reserved																																			
Ah	Reserved	ICV																																			

Table continues on the next page...

Field	Function		
	Ch	PKHA A	Reserved
	Dh	PKHA B	Reserved
	Eh	Reserved	DECO Ignore (i.e. Skip)
	Fh	Reserved	Message Data
	Other values are reserved.		
19 BND	Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE ≠ Padding Block.		
18-16 PTYPE	Pad Type. This field defines the type of padding that should be performed when the STYPE = Padding Block. This field is ignored if BND = 0 or STYPE ≠ Padding Block. 000b - All Zero. 001b - Random with nonzero bytes. 010b - Incremented (Starting with 01h). 011b - Random. 100b - All Zero with last byte containing the number of 0 bytes. 101b - Random with nonzero bytes with last byte 0. 110b - N bytes of padding all containing the number N -1. 111b - Random with nonzero bytes with last byte N.		
15 PR	Prediction Resistance - If PTYPE specifies random data, setting PR=1 causes the RNG to supply random data with prediction resistance (i.e. reseeds the PRNG from the TRNG).		
14-12 —	Reserved. Must be 0.		
11 BM	Boundary Minus 1. When this bit is set with boundary padding, then boundary padding to a 4, 8 or 16-byte boundary minus 1 byte will be executed. For example, if a 16-byte boundary is selected with BM=1, padding will be done such that only 15 of the 16 bytes are used, leaving the 16th byte available for the user to fill.		
10 PS	Pad Snoop. When this bit is set then the Class 2 CHA will snoop the padding data from the Output Data FIFO rather than getting it from the padding block. When snooping, the Class 1 Alignment Block receives data from the Input FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.		
9-7 —	Reserved. Must be 0.		
6-0 PL	Pad Length. The number of bytes needed to pad the current data. If boundary padding is selected then this should be set to 4, 8 or 16 bytes. PL includes the length byte or zero byte for all zero last N, random last N and random last 0 padding types.		

14.304 CCB a Input Data FIFO (C0IFIFO - C5IFIFO)

14.304.1 Offset

For a = 0 to 5:

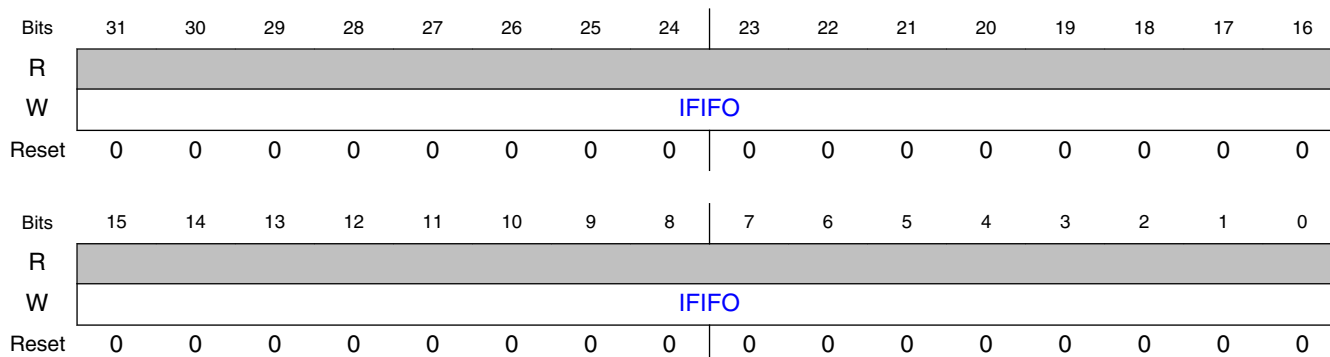
CCB a Input Data FIFO (C0IFIFO - C5IFIFO)

Register	Offset	Description
CaIFIFO	8_07E0h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.304.2 Function

There is one copy of this FIFO per DECO. Data to be processed by the various CHAs is first pushed into the Input Data FIFO for the appropriate DECO. Note that although the Input Data FIFO is 64-bits wide, a single 32-bit wide location is used to write data to the IFIFO. All data written to this location via the IP bus should be in big endian format. Like the other DECO/CCB registers, the Input Data FIFO supports byte enables, allowing one to four bytes to be written to the IFIFO from the IP bus, or one to eight bytes via a descriptor. Although data is normally pushed in multiples of 8 bits, there is a special mechanism that allows a 4-bit value to be pushed into the Input Data FIFO (see “Input Data FIFO Nibble Shift Register”, value 76h, in [LOAD commands](#)). The IFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation SEC will never overflow the Input Data FIFO. Care must be used to not overflow the Input Data FIFO when writing to it directly as results will be unpredictable. FIFO LOAD, FIFO STORE, LOAD, KEY, and MOVE commands can all automatically write to this register.

14.304.3 Diagram



14.304.4 Fields

Field	Function
31-0 IFIFO	Input Data FIFO.

14.305 CCB a Output Data FIFO (C0OFIFO - C5OFIFO)

14.305.1 Offset

For a = 0 to 5:

Register	Offset	Description
CaOFIFO	8_07F0h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.305.2 Function

There is one copy of this FIFO per DECO. Data that is output from the various CHAs is pushed into the Output Data FIFO for the appropriate DECO. The OFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation, SEC will never overflow the Output Data FIFO. KEY, MOVE, MATH, and FIFO STORE commands will all read from the Output Data FIFO. Normally data is pushed in multiples of 8 bits, but there is a special mechanism that allows a 4-bit value to be pushed into the Output Data FIFO (see Output Data FIFO Nibble Shift Register, value 77h, in [LOAD commands](#)).

There are several commands that can result in data being pushed into the Output Data FIFO:

- The OPERATION Command can cause a Class 1 CHA to put data into the Output Data FIFO.
- The KEY Command uses the Output Data FIFO when it decrypts keys. Since the Output Data FIFO must be empty and all transactions must have completed before the KEY Command will start, there will be no collision between a CHA push and an ODFNSR push to the Output Data FIFO.
- The (SEQ) FIFO STORE Command, when encrypting keys, also pushes data into the Output Data FIFO.
- A LOAD IMMEDIATE can push data directly into the Output Data FIFO. DECO will automatically stall a LOAD IMMEDIATE if necessary to prevent a collision with a push from the ODFNSR.

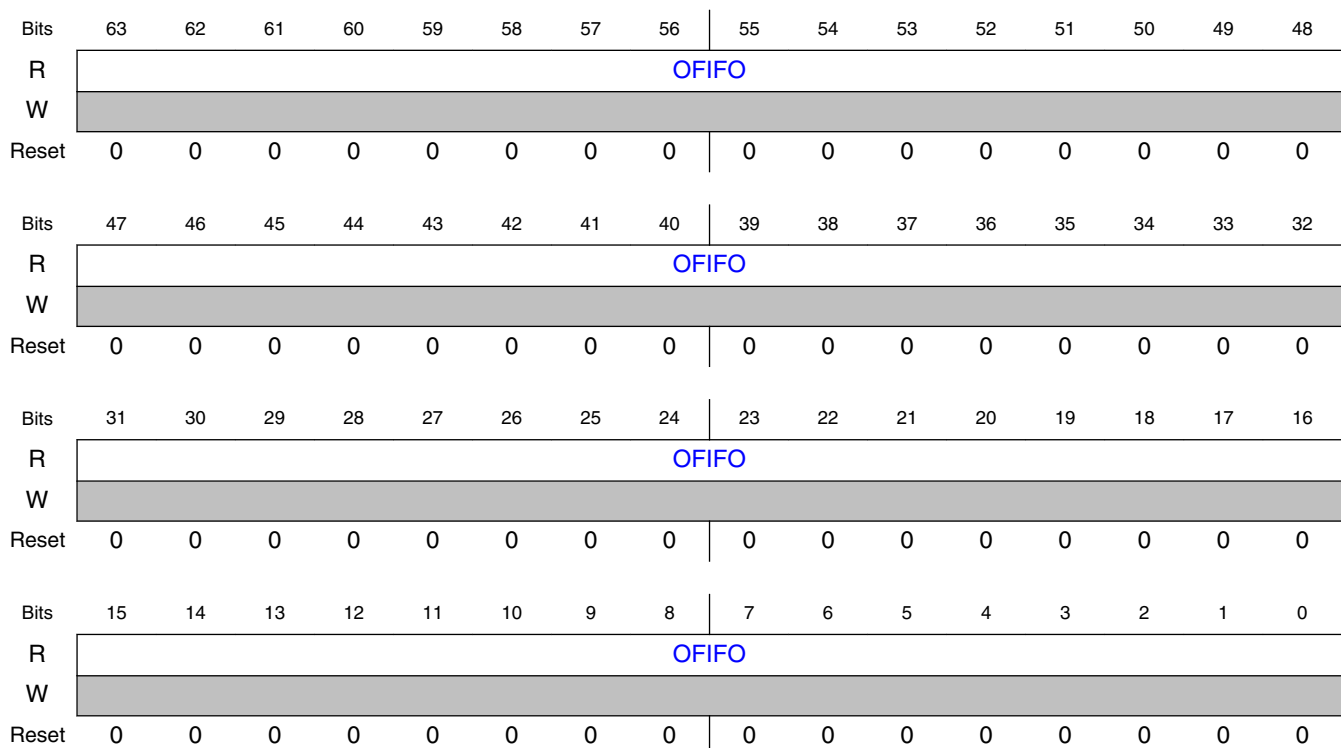
CCB a Output Data FIFO (C00FIFO - C50FIFO)

- A LOAD IMMEDIATE to the CHA Control Register can be used to "unload" or various PKHA registers into the Output Data FIFO.
- The (SEQ) FIFO STORE Command, when generating random data, also pushes data into the Output Data FIFO.

Automatic DECO stalling is accomplished as follows. Once the ODFNSR is written, the stalls described above will continue until the ODFNSR is cleared. That means that the Class 1 CHA has to assert its done signal and the ODFNSR has to have pushed its final value into the Output Data FIFO. **WARNING:** If the DECO is stalling while waiting for the ODFNSR to empty, and there is no already executed command (such as a FIFO STORE or MOVE) that will drain the Output Data FIFO sufficiently to allow the ODFNSR to empty, the DECO will hang.

Internally the Output Data FIFO is 64-bits wide, but since the IP bus is 32-bits wide, the Output Data FIFO is read via the IP bus using 32-bit word reads. The most-significant half of the 64-bit word is read from address $BASE+x7F4$, and the least significant half is read from address $BASE+x7F0$. All data read from the OFIFO is little endian.

14.305.3 Diagram



14.305.4 Fields

Field	Function
63-0 OFIFO	Output FIFO

14.306 DECOa Job Queue Control Register, most-significant half (D0JQCR_MS - D5JQCR_MS)

14.306.1 Offset

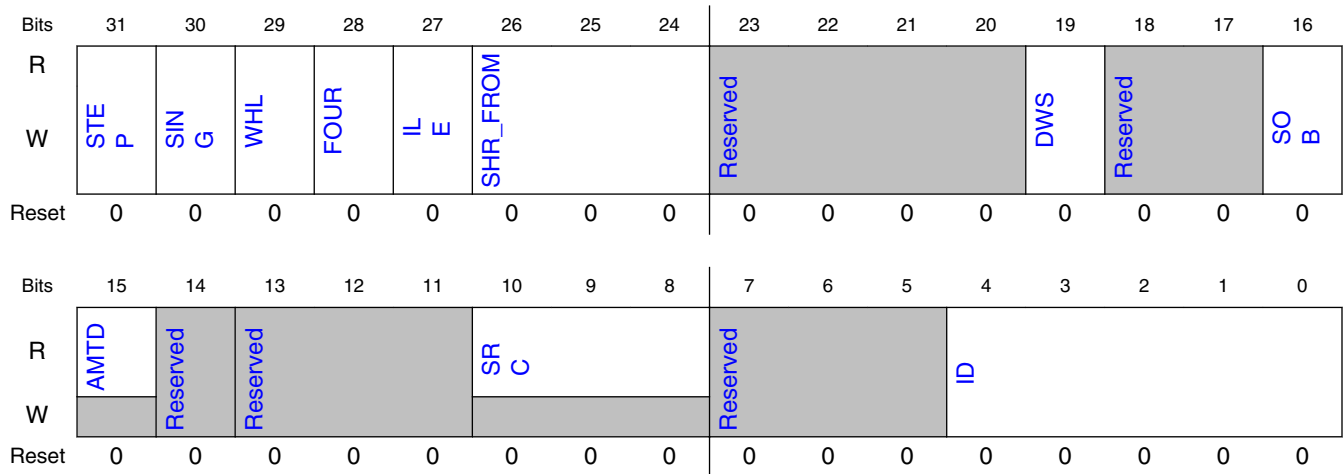
For a = 0 to 5:

Register	Offset	Description
DaJQCR_MS	8_0800h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.306.2 Function

This register tells the Descriptor Controller about the current Descriptor. There is one copy of this register per DECO. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see [Register-based service interface](#)), this register can be read or written from the IP Register bus. Writing to the most-significant half of this register causes the Descriptor Controller to start processing. Note that at least the first burst of the Descriptor (including the Job Descriptor Header and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

14.306.3 Diagram



14.306.4 Fields

Field	Function
31 STEP	Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO.
30 SING	Single Step Mode. This tells DECO to execute this descriptor, including jumps to non-local destinations, in single step mode. Only used by the processor that has control of DECO.
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the job queue controller. This bit is set for certain Job Descriptors that are internally generated by SEC.
28 FOUR	Four Words. job queue controller is passing at least 4 words of the Descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. 0b - No byte-swapping is performed for data transferred to or from the Descriptor Buffer. 1b - Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.
26-24 SHR_FROM	Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software).
23-20 —	Reserved
19 DWS	Double word swap. Causes/allows dword swapping of addresses, and MOVE and MATH immediate values.
18-17	Reserved

Table continues on the next page...

Field	Function
—	
16 SOB	Shared Descriptor or input frame burst. If set, the whole shared descriptor and/or first bursts of QI input frame data were passed to DECO with the Job Descriptor. When descriptors are executed under direct software control, this bit simply indicates that the Shared Descriptor has been loaded.
15 AMTD	Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRaICID Register.
14 —	Reserved
13-11 —	Reserved
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS) the DMA should use for bus transactions. It is illegal for the SRC field to have a value other than that of a Job Ring when running descriptors via the IP bus (i.e. under the direct control of software). 000b - Job Ring 0 001b - Job Ring 1 010b - Job Ring 2 011b - Job Ring 3 100b - RTIC 101b - Queue Manager Interface 110b - AI 111b - Reserved
7-5 —	Reserved
4-0 ID	Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed.

14.307 DECOa Job Queue Control Register, least-significant half (D0JQCR_LS - D5JQCR_LS)

14.307.1 Offset

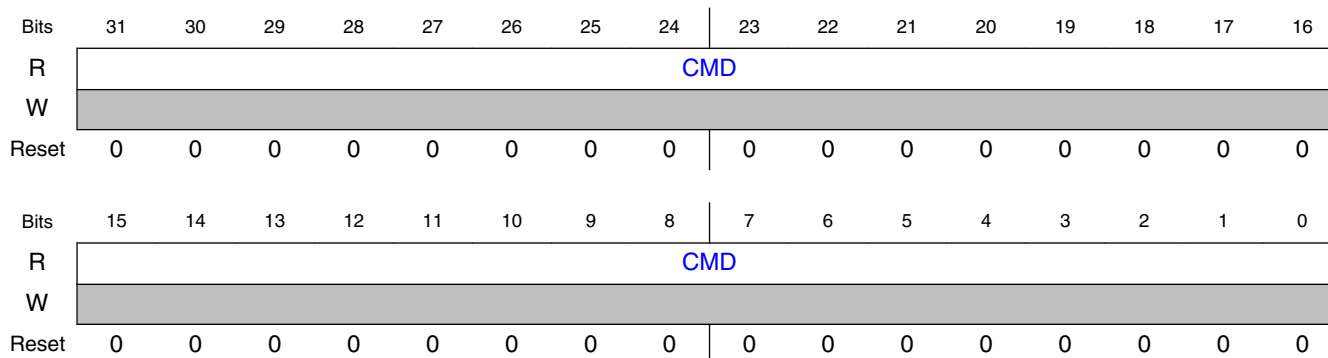
For a = 0 to 5:

Register	Offset	Description
DaJQCR_LS	8_0804h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.307.2 Function

This register tells the Descriptor Controller about the current Descriptor. There is one copy of this register per DECO. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see [Register-based service interface](#)), this register can be read or written from the IP Register bus. Writing to the most-significant half of this register causes the Descriptor Controller to start processing. Note that at least the first burst of the Descriptor (including the Job Descriptor Header and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

14.307.3 Diagram



14.307.4 Fields

Field	Function
31-0 CMD	Command. In single-step mode, reading CMD returns the first word of the command that this DECO will execute next. This value is also readable via the STORE Command, but the value read is unpredictable.

14.308 DECOa Descriptor Address Register (D0DAR - D5DAR)

14.308.1 Offset

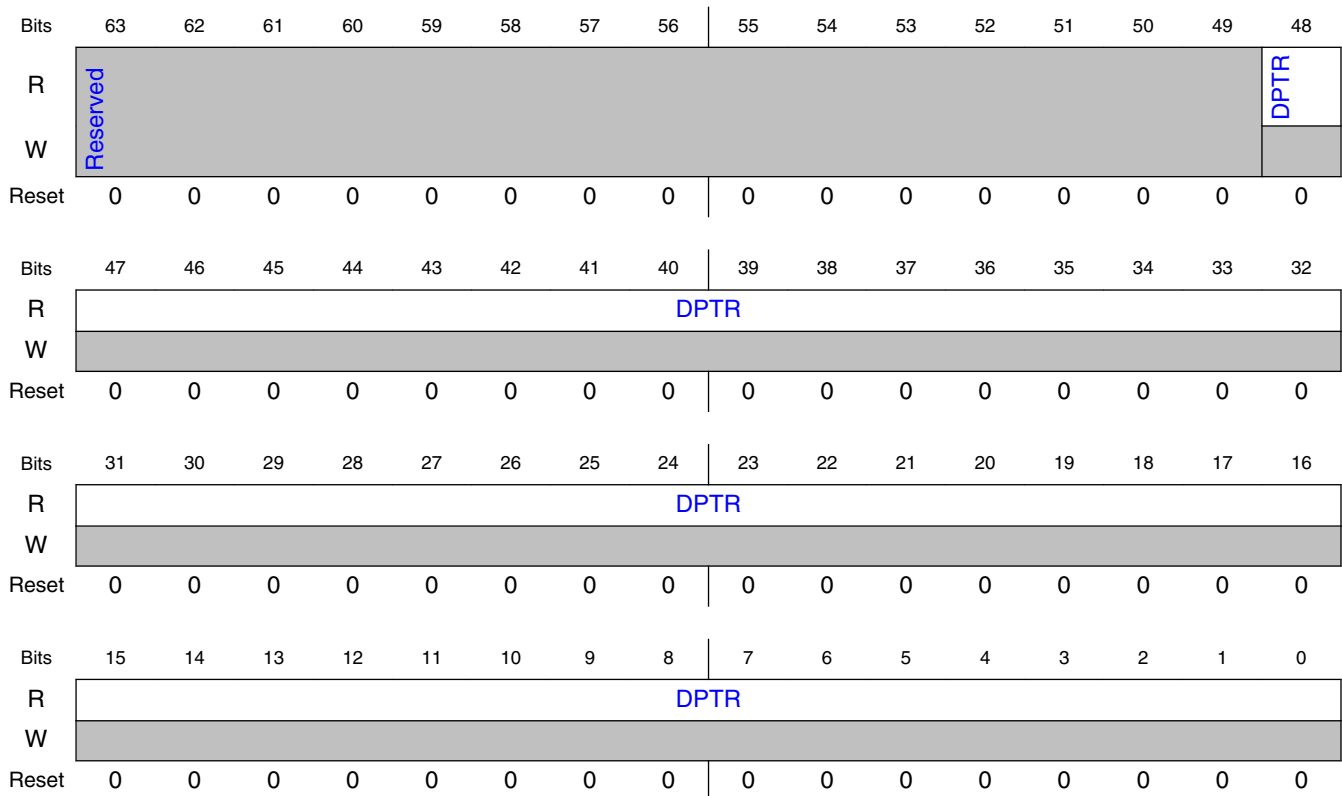
For a = 0 to 5:

Register	Offset	Description
DaDAR	8_0808h + (a × 1_0000h)	For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) . Accessible only when RQDa and DENa are asserted in DECORR.

14.308.2 Function

This DECO register holds the address of the currently executing Job Descriptor. When using DECO in single-step mode (see [Register-based service interface](#)), this register must be written prior to the Job Queue Control Register.

14.308.3 Diagram



14.308.4 Fields

Field	Function
63-49 —	Reserved
48-0 DPTR	Descriptor Pointer. Memory address of the Descriptor. Needed for write-back purposes.

14.309 DECOa Operation Status Register, most-significant half (D0OPSTA_MS - D5OPSTA_MS)

14.309.1 Offset

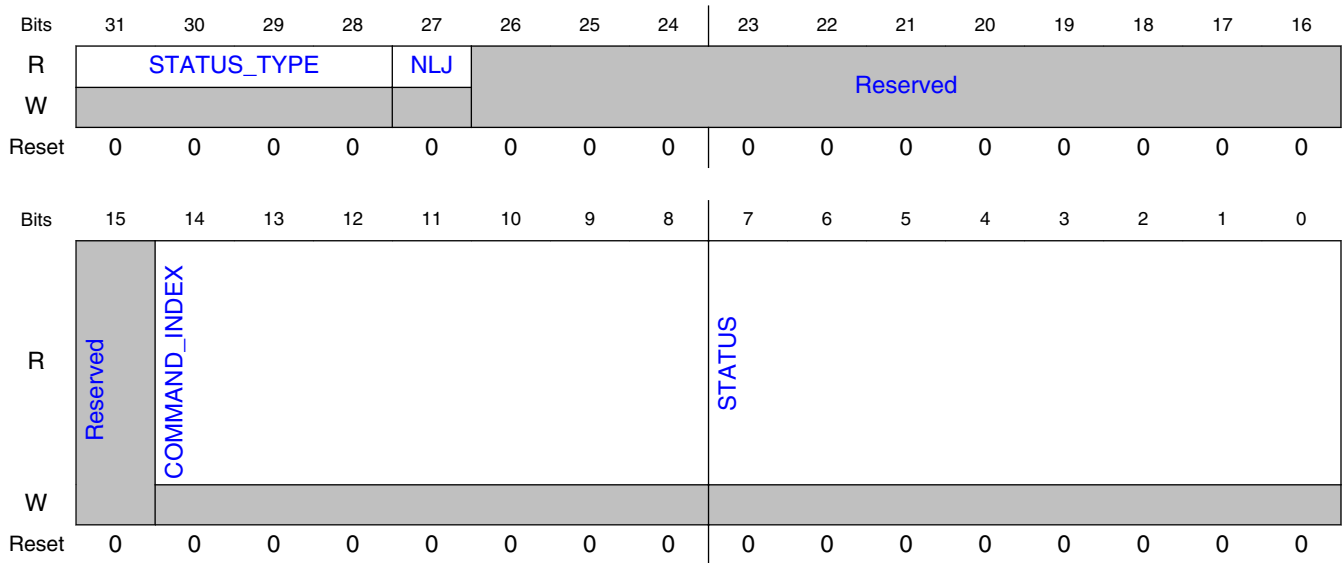
For a = 0 to 5:

Register	Offset	Description
DaOPSTA_MS	8_0810h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.309.2 Function

The DECO Operation Status Register is used to show DECO status following descriptor processing. This includes error conditions (if any), index of the next command within the descriptor, and condition flags set by Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

14.309.3 Diagram



14.309.4 Fields

Field	Function
31-28 STATUS_TYPE	Status Type. The type of status that is being reported for the just-completed command, defined as follows: 0000b - no error 0001b - DMA error 0010b - CCB error 0011b - Jump Halt User Status 0100b - DECO error 0101b - Reserved 0110b - Reserved 0111b - Jump Halt Condition Code
27 NLJ	Non-Local Jump. A jump was non-local. This includes non-local JUMP Commands and SEQ IN PTR RJD jumps and SEQ IN PTR INL jumps.
26-15 —	Reserved. Always 0.
14-8 COMMAND_IN DEX	Command index: A pointer to a 32-bit word within the descriptor. If single stepping, this is the index of the next command to be executed. If not single stepping, this is the index of the command that is now executing. In the case of an error that is not a command problem, it is approximately the index of the command where the error occurred. If the error was due to a command problem, it is the index of the current command. A command problem is an error that is detectable by DECO as it executes the command (e.g. an illegal command type). Something that isn't a command problem is an error that occurs after the command has completed executing (e.g. illegal CHA modes, DMA errors, ICV check failures).

Table continues on the next page...

DECOa Operation Status Register, least-significant half (D0OPSTA_LS - D5OPSTA_LS)

Field	Function
7-0 STATUS	If ERR_TYP indicates no error, this field contains PKHA/Math Status, as defined below. If there was an error, this field contains Error Status, defined as in the Job Ring Output Status Register DESC_ERROR field (Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3)).
7 PIZ	Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity.
6 GCD	GCD is One. The greatest common divisor of two numbers (i.e., the two numbers are relatively prime)
5 PRM	Public Key is Prime. The given number is probably prime (i.e., it passes the Miller-Rabin primality test)
4	Reserved
3 MN	Math N. The result is negative. Can only be set by add and subtract functions, 0 otherwise
2 MZ	Math Z. The result of a math operation is zero.
1 MC	Math C. The math operation resulted in a carry or borrow.
0 MNV	Math NV. Used for signed compares. This is a combination of the sign and overflow bits (i.e., Math N XOR Math C)

14.310 DECOa Operation Status Register, least-significant half (D0OPSTA_LS - D5OPSTA_LS)

14.310.1 Offset

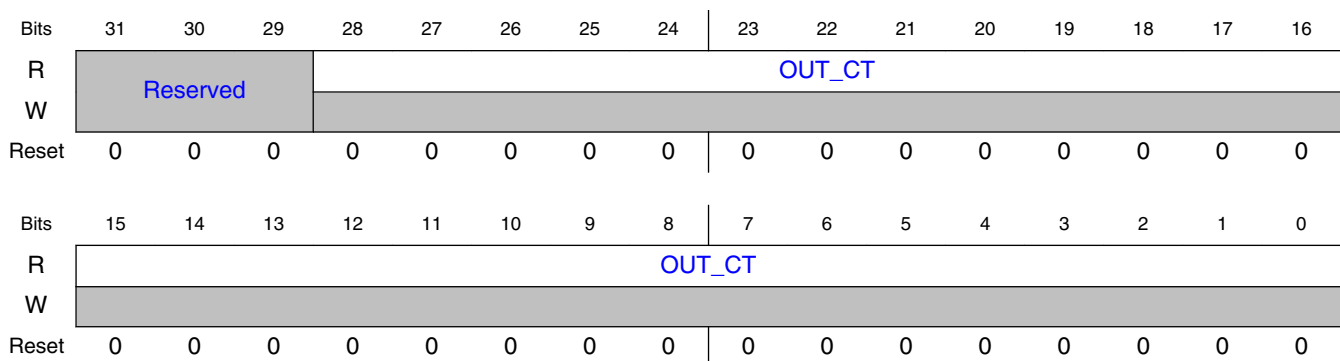
For a = 0 to 5:

Register	Offset	Description
DaOPSTA_LS	8_0814h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.310.2 Function

The DECO Operation Status Register is used to show DECO status following descriptor processing. This includes error conditions (if any), index of the next command within the descriptor, and condition flags set by Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

14.310.3 Diagram



14.310.4 Fields

Field	Function
31-29 —	Reserved
28-0 OUT_CT	Output Count. Number of bytes written to sequential out pointer.

14.311 DECOa Checksum Register (D0CKSUMR - D5CKSUMR)

14.311.1 Offset

For a = 0 to 5:

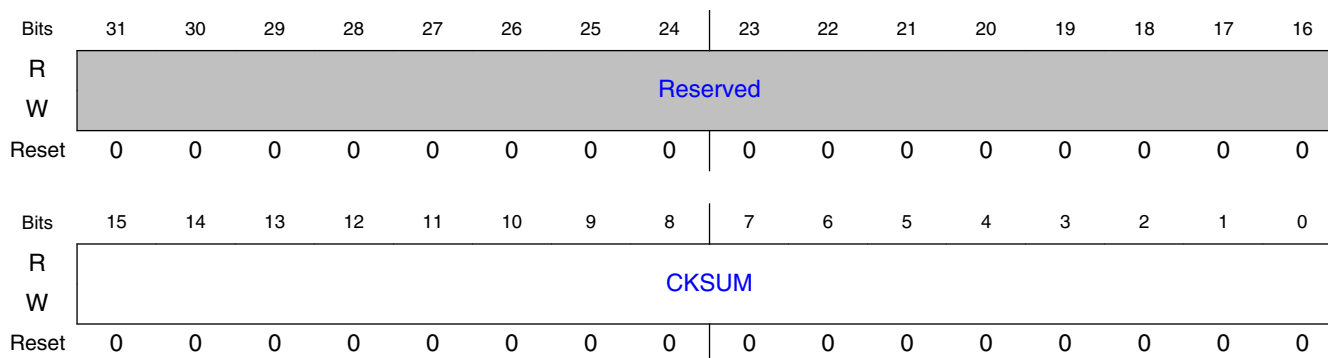
DECOa Checksum Register (D0CKSUMR - D5CKSUMR)

Register	Offset	Description
DaCKSUMR	8_0818h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.311.2 Function

Checksum. This register shows the checksum value computed by DECO. This value is intended to assist in implementing the IPSEC protocol.

14.311.3 Diagram



14.311.4 Fields

Field	Function
31-16 —	Reserved
15-0 CKSUM	Checksum. All byte data written to the output frame via SEQ STORE or SEQ FIFO STORE is included in the checksum, with the following exceptions. If a type 31h SEQ FIFO STORE is executed, the previous checksum value (if any) due to non-type 31h SEQ FIFO STORE commands is discarded and the running checksum starts with the fresh type 31h or type 3Eh data. With one exception as noted below, all byte data stored with SEQ STORE or SEQ FIFO STORE 3Eh (meta data) is included in the running checksum until a SEQ FIFO STORE is executed that is not type 31h or 3Eh. After that no additional data is included in the checksum until another type 31h SEQ FIFO STORE is executed. At that point the running checksum is not cleared, and the running checksum includes all subsequent byte data written via a type 31h or type 3Eh SEQ FIFO STORE command. Commands that store the CKSUM register are an exception to the general rules above. When the CKSUM register is stored via a STORE or SEQ STORE command, the register value will not be included in the running checksum. If DECO was computing a checksum of type 31h or type 3Eh data when the CKSUM register was stored, following the store of the CKSUM register no further data will be included in the running checksum until another SEQ FIFO STORE of type 0x31 is executed. But if at the time that the CKSUM register was stored DECO was computing a checksum of data other than type 31h or type 3Eh data, all byte data stored via SEQ STOREs and SEQ

Field	Function
	FIFO STOREs will be included in the running checksum until a type 0x31 SEQ FIFO STORE is executed, at which point the running checksum will be cleared and a checksum of type 31h or type 3Eh data will begin.

14.312 DECOa Control and Output ICID Status Register (D0COICIDSR - D5COICIDSR)

14.312.1 Offset

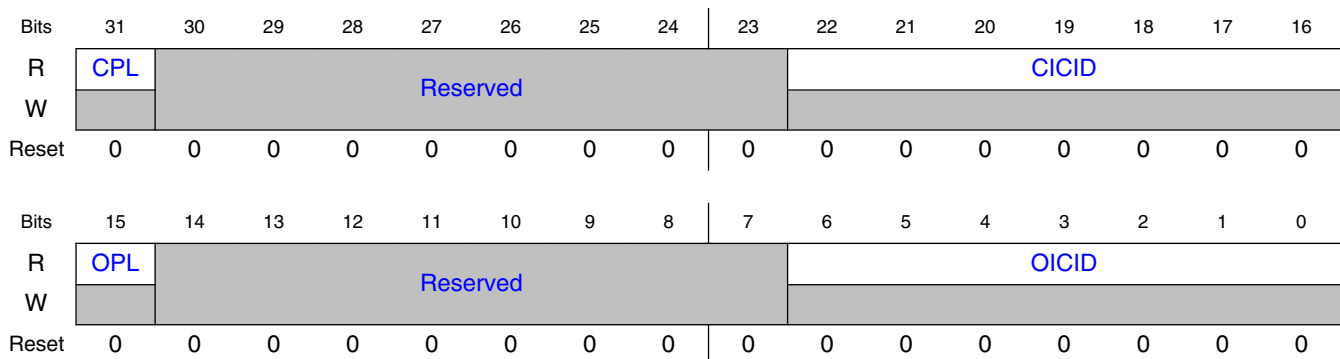
For a = 0 to 5:

Register	Offset	Description
DaCOICIDSR	8_0820h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.312.2 Function

This register shows the Control and Output ICID values used by the DECO when processing the current Descriptor. This register is written by the job queue controller.

14.312.3 Diagram



14.312.4 Fields

Field	Function
31 CPL	Control Privilege Level. This is the privilege level value asserted during an control isolation context DMA transaction.
30-23 —	Reserved
22-16 CICID	Control ICID. This is the ICID value used by the DMA when processing control operations during execution of the current Descriptor.
15 OPL	Output Privilege Level. This is the privilege level value asserted during an output isolation context DMA transaction.
14-7 —	Reserved
6-0 OICID	Output ICID. This is the ICID value used by the DMA when processing output operations during execution of the current Descriptor.

14.313 DECOa SDID and Input ICID Status Register (D0SIICIDSR - D5SIICIDSR)

14.313.1 Offset

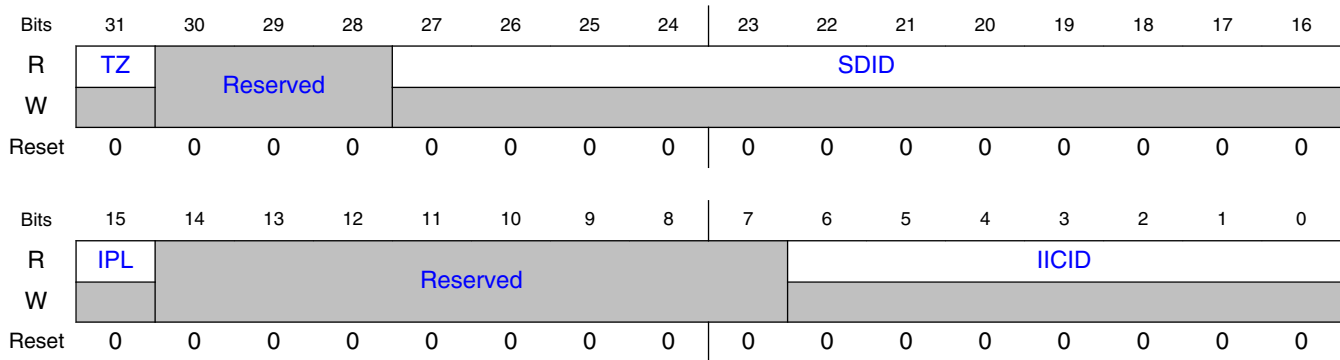
For a = 0 to 5:

Register	Offset	Description
DaSIICIDSR	8_0824h + (a × 1_0000h)	Accessible only when RQDa and DENa are asserted in DECORR.

14.313.2 Function

This register shows the SDID and Input ICID values used by the DECO when processing the current Descriptor. This register is written by the job queue controller.

14.313.3 Diagram



14.313.4 Fields

Field	Function
31 TZ	TrustZone. This is the TrustZone value (where SecureWorld = 1) used by the DECO when processing the current Descriptor.
30-28 —	Reserved
27-16 SDID	Security Domain Identifier. This is the SDID value used by the DECO when processing the current Descriptor.
15 IPL	Input Privilege Level. This is the privilege level value asserted during an input isolation context DMA transaction.
14-7 —	Reserved
6-0 IICID	This is the ICID value used by the DMA when processing input operations during execution of the current Descriptor.

14.314 DECOa Math Register b_MS (D0MTH0_MS - D5MTH7_MS)

14.314.1 Offset

For a = 0 to 5; b = 0 to 7:

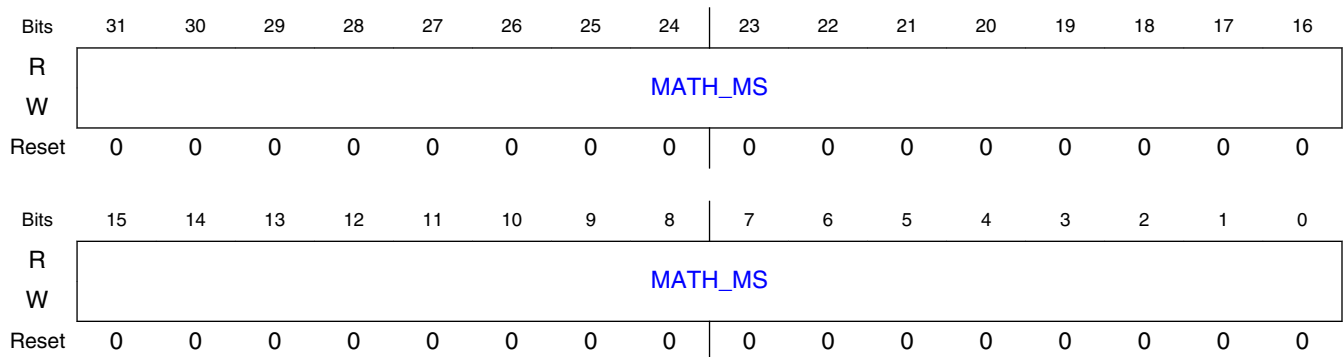
DECOa Math Register b_LS (D0MTH0_LS - D5MTH7_LS)

Register	Offset
DaMTHb_MS	8_0840h + (a × 1_0000h) + (b × 8h)

14.314.2 Function

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 8 64-bit registers per DECO. Data is moved into these registers via LOAD, MATH and MOVE commands.

14.314.3 Diagram



14.314.4 Fields

Field	Function
31-0 MATH_MS	MATH register, most-significant 32 bits.

14.315 DECOa Math Register b_LS (D0MTH0_LS - D5MTH7_LS)

14.315.1 Offset

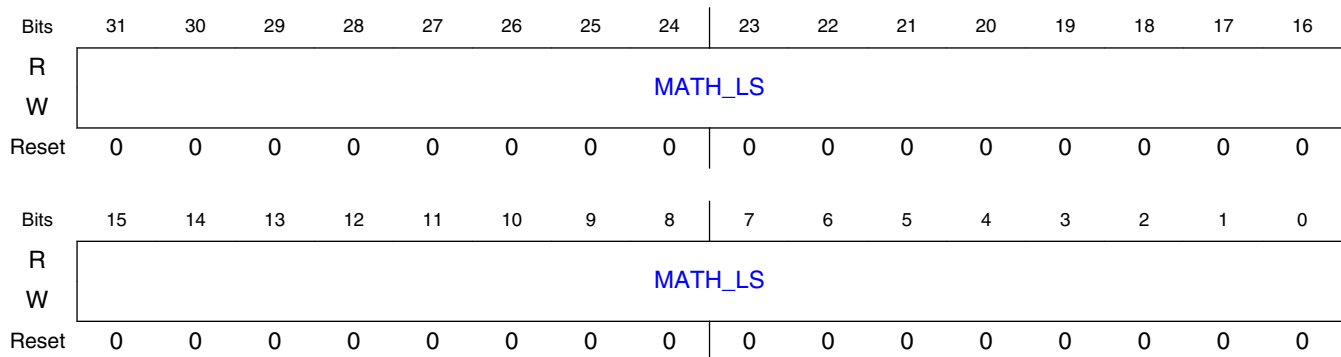
For a = 0 to 5; b = 0 to 7:

Register	Offset
DaMTHb_LS	$8_0844h + (a \times 1_0000h) + (b \times 8h)$

14.315.2 Function

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 8 64-bit registers per DECO. Data is moved into these registers via LOAD, MATH and MOVE commands.

14.315.3 Diagram



14.315.4 Fields

Field	Function
31-0 MATH_LS	MATH register, least-significant 32 bits.

14.316 DECOa Gather Table Register b (D0GTR0 - D5GTR3)

14.316.1 Offset

For a = 0 to 5; b = 0 to 3:

DECOa Gather Table Register b (D0GTR0 - D5GTR3)

Register	Offset
DaGTRb	$8_0880h + (a \times 1_0000h) + (b \times 10h)$

14.316.2 Function

The Gather Table Registers and Scatter Table Registers hold the current Gather Table (i.e. input data) and Scatter Table (i.e. output data) entries that are being used by the DECO. (See [Scatter/gather tables \(SGTs\)](#).) SEC will fetch up to four entries at a time and cache these entries in four Gather Table registers and four Scatter Table registers per DECO. Each register is 128 bits in length, so the data in each register is accessible over the 32-bit register bus as four 32-bit words.

14.316.3 Diagram

Bits	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.316.4 Fields

Field	Function
127-0 SGTE	Scatter Gather Table Entry. This field holds one Scatter Gather Table Entry. See the Scatter/gather tables (SGTs) section for a description of the Scatter Gather Table Entry format.

14.317 DECOa Scatter Table Register b (D0STR0 - D5STR3)

14.317.1 Offset

For a = 0 to 5; b = 0 to 3:

Register	Offset
DaSTRb	$8_0900h + (a \times 1_0000h) + (b \times 10h)$

14.317.2 Function

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See [Scatter/gather tables \(SGTs\)](#).) SEC will fetch up to four entries at a time and cache these entries in four Gather Table registers and four Scatter Table registers per DECO. Each register is 128 bits in length, so the data in each register is accessible over the 32-bit register bus as four 32-bit words.

14.317.3 Diagram

Bits	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SGTE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

14.317.4 Fields

Field	Function
127-0 SGTE	Scatter Gather Table Entry. This field holds one Scatter Gather Table Entry. See the Scatter/gather tables (SGTs) section for a description of the Scatter Gather Table Entry format.

14.318 DECOa Descriptor Buffer Word b (D0DESB0 - D5DESB63)

14.318.1 Offset

For a = 0 to 5; b = 0 to 63:

Register	Offset
DaDESBb	$8_0A00h + (a \times 1_0000h) + (b \times 4h)$

14.318.2 Function

The Descriptor Buffer is used by the DECO to buffer a Descriptor that has been fetched from memory. The Descriptor Buffer consists of 64 32-bit registers in consecutive addresses, beginning at the addresses shown above. For performance reasons, DECO doesn't execute the commands directly from the Descriptor Buffer. Instead, DECO executes commands from a four-word pipeline. Since commands vary in length from one to four words, up to three words in addition to the current command may also be resident in the pipeline. (They won't be executed if the job terminates or the pipeline is flushed as described below.) As a result, operations that modify the Descriptor Buffer may not have an immediate effect on the next few commands that execute. To avoid anomalous behavior when overwriting the portion of the Descriptor Buffer containing the start of the currently executing command or the following two or three words, any commands in the pipeline that the programmer intends to execute should be completely contained within the pipeline.

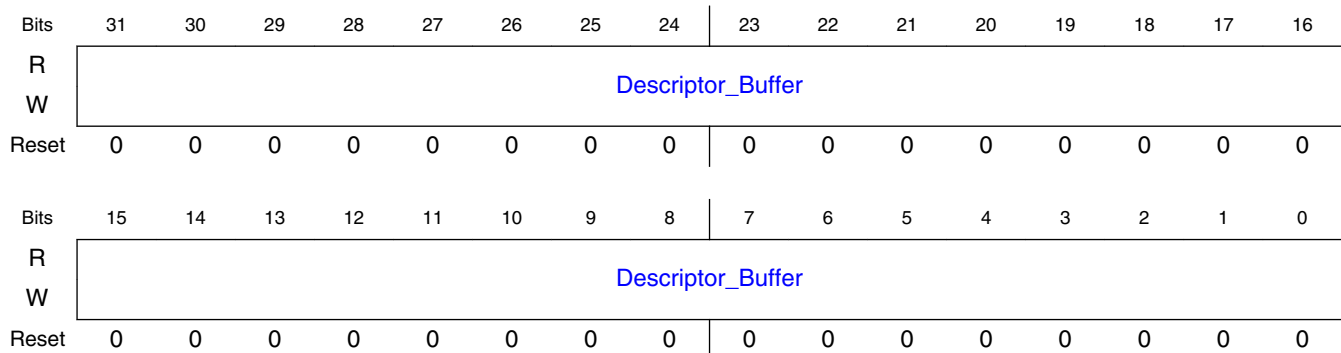
There are several ways to flush the pipeline to ensure that recently loaded commands are executed rather than the pipeline-resident commands:

- Execute a JUMP command with a negative offset

- Use the JOB HEADER or SHARED HEADER commands to do an absolute jump.
- JUMP forward more than 3 words.

Note that the Descriptor Buffer is cleared between unrelated descriptors; that is, if two successive descriptors to execute in the same DECO do not share the same shared descriptor.

14.318.3 Diagram



14.318.4 Fields

Field	Function
31-0 Descriptor_Buffer	Descriptor_Buffer

14.319 DECOa Debug Job (D0DJR - D5DJR)

14.319.1 Offset

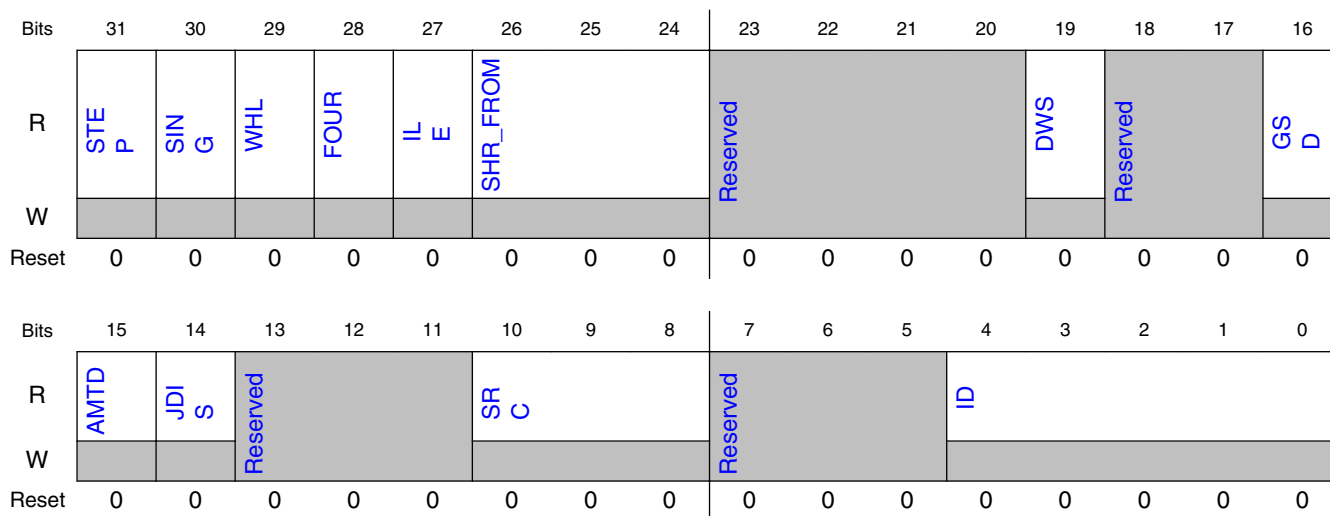
For a = 0 to 5:

Register	Offset	Description
DaDJR	8_0E00h + (a × 1_0000h)	For DECOa.

14.319.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)). Note that the DECOa Debug Job has the same format as the most-significant half of the DECO Job Queue Control Register. Note that this register is read-only.

14.319.3 Diagram



14.319.4 Fields

Field	Function
31 STEP	Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO.
30 SING	Single Step Mode. This tells DECO to execute this descriptor, including jumps to non-local destinations, in single step mode. Only used by the processor that has control of DECO.

Table continues on the next page...

Field	Function
29 WHL	Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the job queue controller (or by the processor that has control of DECO). This bit is set for certain Job Descriptors that are internally generated by SEC.
28 FOUR	Four Words. The job queue controller (or the processor that has control of DECO) has passed at least 4 words of the Descriptor to DECO.
27 ILE	Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors. 0b - No byte-swapping is performed for data transferred to or from the Descriptor Buffer. 1b - Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers.
26-24 SHR_FROM	Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software).
23-20 —	Reserved
19 DWS	Double Word Swap. Double word swapping was set.
18-17 —	Reserved
16 GSD	Got Shared Descriptor. A Shared Descriptor was already available in a DECO so the DECO didn't need to fetch the Shared Descriptor from memory.
15 AMTD	Allow Make Trusted Descriptor. If this bit is a 1, then a Job Descriptor whose HEADER command has TDES=11b (candidate trusted descriptor) is allowed to execute. The AMTD bit will be 1 only if the Job Descriptor was run from a Job Ring with the AMTD bit set to 1 in the Job Ring's JRaICID Register.
14 JDIS	Job Descriptor ICID Select. Determines whether the SEQ ICID or the Non-SEQ ICID is asserted when reading the Job Descriptor from memory. 0b - Non-SEQ ICID 1b - SEQ ICID
13-11 —	Reserved
10-8 SRC	Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS)) the DMA should use for bus transactions. When running descriptors via the IP bus (i.e. under the direct control of software), the job queue controller automatically sets this field to indicate a Job Ring source. 000b - Job Ring 0 001b - Job Ring 1 010b - Job Ring 2 011b - Job Ring 3 100b - RTIC 101b - Queue Manager Interface 110b - AI 111b - Reserved
7-5 —	Reserved

Table continues on the next page...

DECOa Debug DECO (D0DDR - D5DDR)

Field	Function
4-0 ID	Job ID. Unique tag given to each job by its source (see SRC field). Used to tell the source that the job has completed.

14.320 DECOa Debug DECO (D0DDR - D5DDR)

14.320.1 Offset

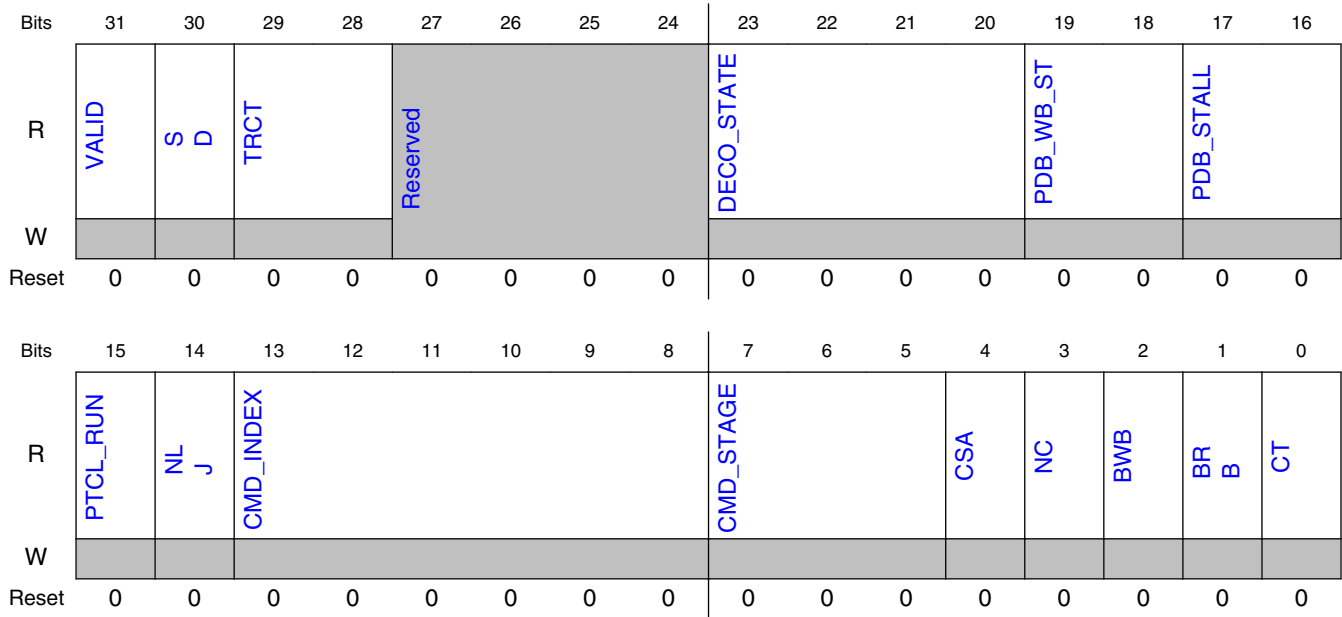
For a = 0 to 5:

Register	Offset	Description
DaDDR	8_0E04h + (a × 1_0000h)	For DECOa.

14.320.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)). Note that this register is read-only.

14.320.3 Diagram



14.320.4 Fields

Field	Function
31 VALID	Valid. If VALID=1, there is currently a job descriptor running in this DECO. The descriptor has been loaded and has started executing and is still executing.
30 SD	Shared Descriptor. The job descriptor that is running in this DECO has received a shared descriptor from another job descriptor. That is, some other job descriptor used this shared descriptor (in the same DECO or a different DECO), and this job descriptor is using the shared descriptor without having to load it from memory. In the case of SERIAL or WAIT sharing, then the keys were shared as well. If the SC bit was on, then the context was also shared.
29-28 TRCT	DMA Transaction Count. This indicates how many outstanding external DMA transactions are pending. This is the total of reads and writes. DECO is limited to, at most, two such transactions.
27-24 —	Reserved
23-20 DECO_STATE	DECO State. The current state of DECO's main state machine.
19-18 PDB_WB_ST	PDB Writeback State. Lower two bits of the state machine that tracks the state of PDB writebacks.
17-16 PDB_STALL	PDB Stall State. The state of the machine that tracks the stalling of PDB writebacks. Used in conjunction with PDB_WB_ST. Used only if there is more than one DECO.
15 PTCL_RUN	Protocol running. PTCL_RUN=1 indicates that a protocol is running in this DECO.

Table continues on the next page...

DECOa Debug Job Pointer (D0DJP - D5DJP)

Field	Function
14 NLJ	Took Non-local JUMP. If NLJ=1 the original job descriptor running in this DECO has caused another job descriptor to be executed. This is true for JUMP NON-LOCAL, SEQ IN PTR INLINE, and SEQ IN PTR RJD.
13-8 CMD_INDEX	Command Index. If this DECO is currently executing a command, CMD_INDEX points to that command within the descriptor buffer.
7-5 CMD_STAGE	Command Stage. Each command executes in a number of steps, or stages. There are 8 possible stages. CMD_STAGE indicates which stage DECO has reached in the process of executing a command.
4 CSA	Command Stage Aux. A refinement of the CMD_STAGE stages. Some stages may be split into two substages, and CSA will indicate which of those two substages DECO has reached.
3 NC	No Command. This DECO is not currently executing a command. This can be because the descriptor isn't executing or DECO is doing a JUMP of some sort.
2 BWB	Burster Write Busy. The WRITE machine in the Burster is busy. This means that the WRITE machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. STORE, SEQ STORE, FIFO STORE, and SEQ FIFO STORE commands use the WRITE machine. The WRITE machine is also used to update the Shared Descriptor HEADER when propagating DNR and by the Trusted State Machine to store a computed signature.
1 BRB	Burster Read Busy. The READ machine in the Burster is busy. This means that the READ machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. LOAD, SEQ LOAD, FIFO LOAD, SEQ FIFO LOAD, and the KEY command all use the READ machine. The read to satisfy RIF in the Shared Descriptor HEADER also uses the READ machine. The SEQ FIFO STORE command can also use the READ machine when handling meta data. Jumping non-locally via any method will also use the READ machine. Commands that reference Scatter/Gather Tables will also cause the READ machine to be used to read the entries in the tables.
0 CT	Checking Trusted. This DECO is currently generating the signature of a Trusted Descriptor. This may be to sign, re-sign, or check the signature.

14.321 DECOa Debug Job Pointer (D0DJP - D5DJP)

14.321.1 Offset

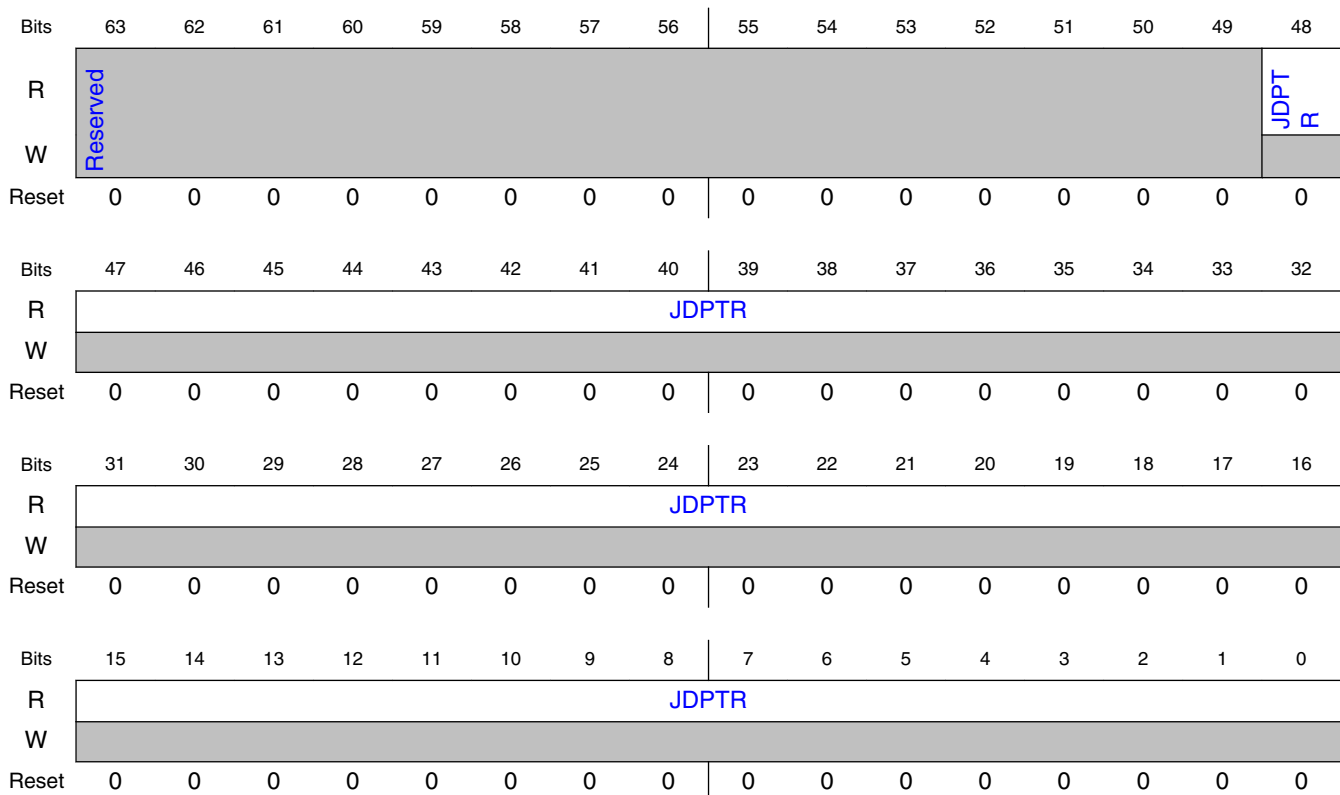
For a = 0 to 5:

Register	Offset	Description
DaDJP	8_0E08h + (a × 1_0000h)	For DECOa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.321.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)). Note that this register is read-only.

14.321.3 Diagram



14.321.4 Fields

Field	Function
63-49	Reserved

Table continues on the next page...

DECOa Debug Shared Pointer (D0SDP - D5SDP)

Field	Function
—	
48-0 JDPTR	Job Descriptor Pointer.

14.322 DECOa Debug Shared Pointer (D0SDP - D5SDP)

14.322.1 Offset

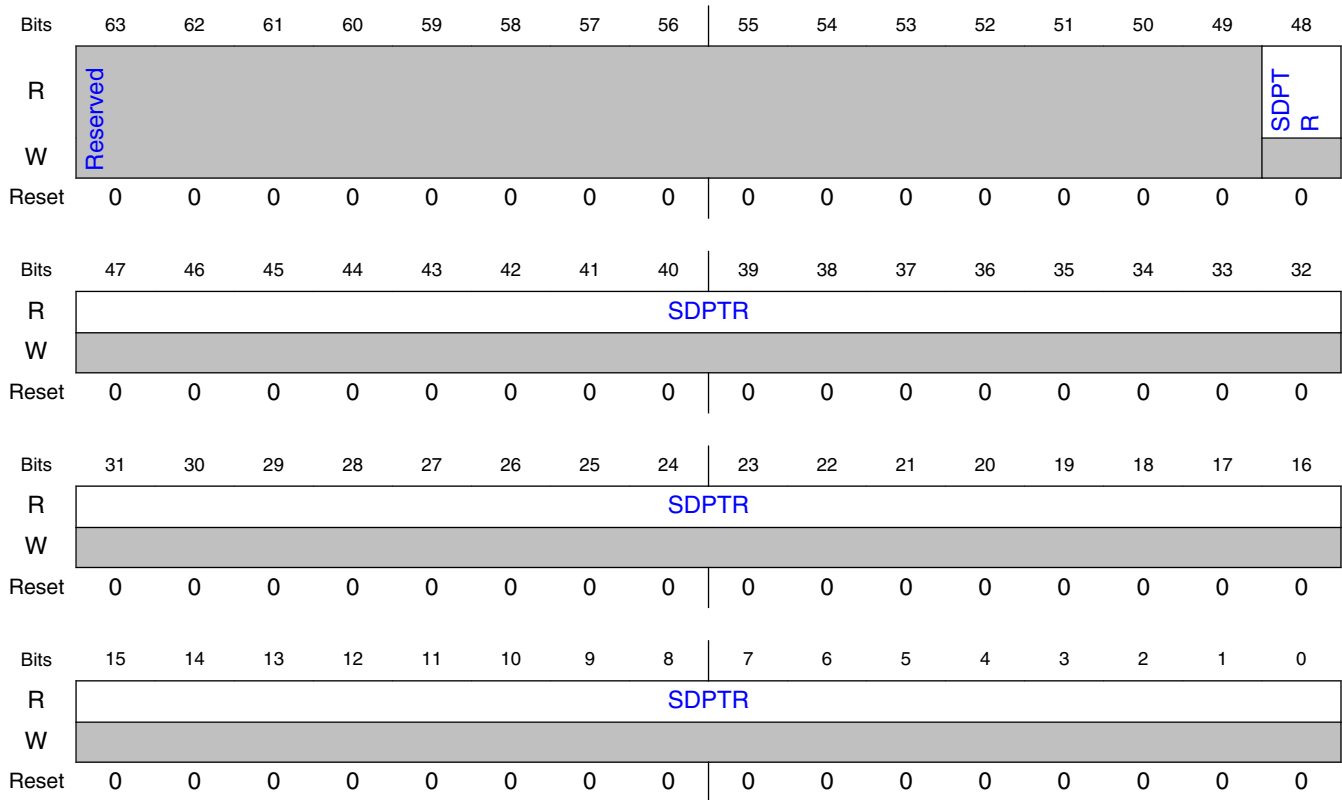
For a = 0 to 5:

Register	Offset	Description
DaSDP	$8_0E10h + (a \times 1_0000h)$	For DECOa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R) .

14.322.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)). Note that this register is read-only.

14.322.3 Diagram



14.322.4 Fields

Field	Function
63-49 —	Reserved
48-0 SDPTR	Shared Descriptor Pointer.

14.323 DECOa Debug_ICID, most-significant half (D0DIR_MS - D5DIR_MS)

14.323.1 Offset

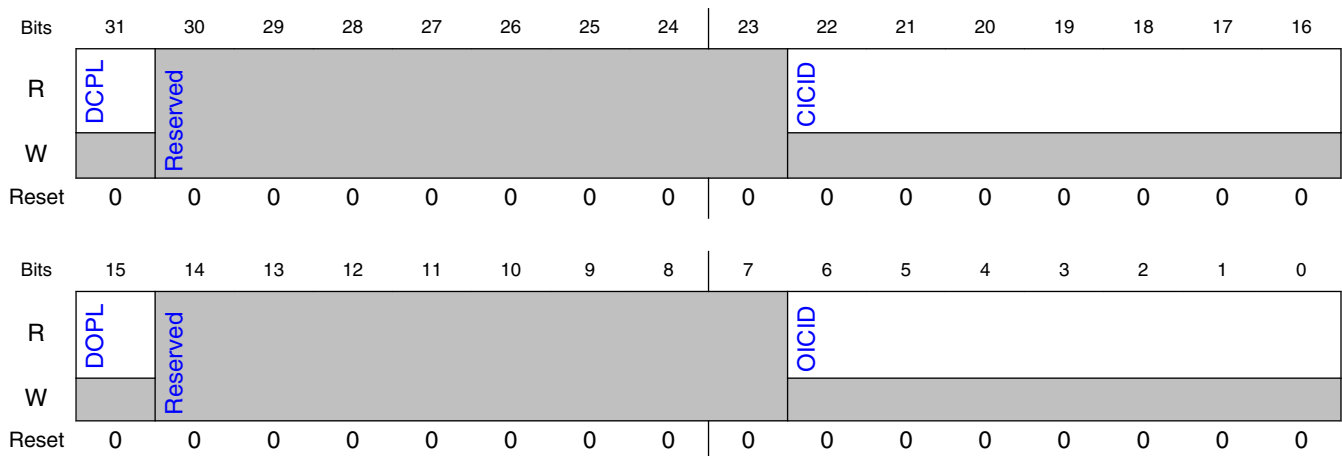
For a = 0 to 5:

Register	Offset	Description
DaDIR_MS	8_0E18h + (a × 1_0000h)	For DECOa.

14.323.2 Function

The DECOa Debug Job, DECOa Debug_DBG, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. This register is read-only. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)).

14.323.3 Diagram



14.323.4 Fields

Field	Function
31	DECO Control Privilege Level. This is the PL value that is asserted during DMA transactions related to control operations.

Table continues on the next page...

Field	Function
DCPL	
30-23 —	Reserved
22-16 CICID	DECO Control ICID. This is the ICID value that is asserted during DMA transactions related to control operations.
15 DOPL	DECO Output Privilege Level. This is the PL value that is asserted during DMA transactions related to data output operations.
14-7 —	Reserved
6-0 OICID	DECO Output ICID. This ICID value is asserted during DMA transactions related to data output operations.

14.324 DECOa Debug ICID, least-significant half (D0DIR_LS - D5DIR_LS)

14.324.1 Offset

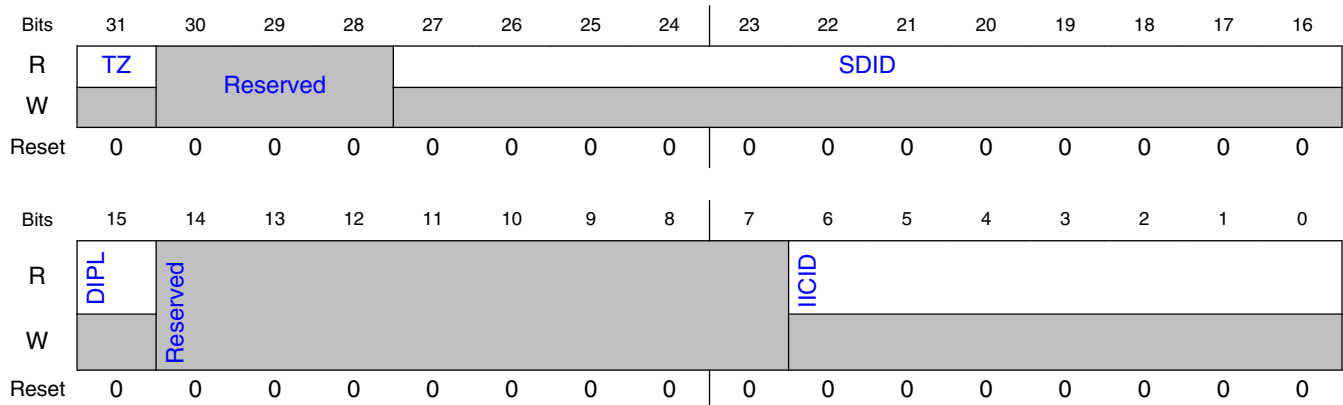
For a = 0 to 5:

Register	Offset	Description
DaDIR_LS	8_0E1Ch + (a × 1_0000h)	For DECOa.

14.324.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Note that this register is read-only. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see [Register-based service interface](#)).

14.324.3 Diagram



14.324.4 Fields

Field	Function
31 TZ	DECO TrustZone. This is the TrustZone SecureWorld indicator that will be asserted during a DMA transaction or used for Black Keys, Blobs and Trusted Descriptors. SecureWorld is indicated by TZ = 1.
30-28 —	Reserved
27-16 SDID	DECO SDID. This SDID value is used for Black Keys, Blobs and Trusted Descriptors.
15 DIPL	DECO Input Privilege Level. This is the PL value that is asserted during DMA transactions related to data input operations.
14-7 —	Reserved
6-0 IICID	DECO Input ICID. This ICID value is asserted during DMA transactions related to data input operations.

14.325 Sequence Output Length Register (SOL0 - SOL5)

14.325.1 Offset

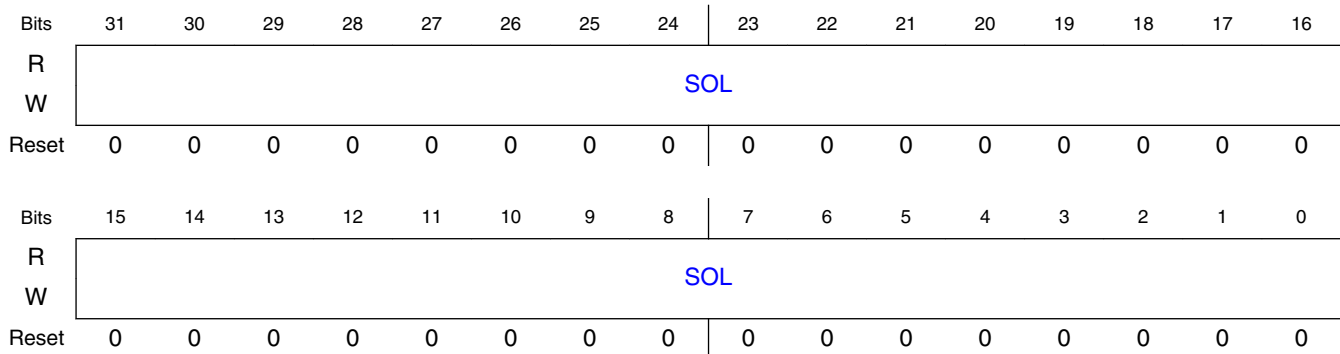
For a = 0 to 5:

Register	Offset	Description
SOLa	8_0E20h + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.325.2 Function

The Sequence Out Length Register is used to specify the amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). See [SEQ vs non-SEQ commands](#) for a discussion of sequences. See [Using sequences for fixed and variable length data](#) for a discussion of the use of the SOL register in Output Sequences. The SEQ OUT PTR command can be used to load the SOL register. The SOL Register can be read or written via the MATH Command (see SRC0 and DEST fields in [MATH and MATHI Commands](#)). When the DECO is under direct control of software (see [Register-based service interface](#)) this register is accessible at the addresses shown above.

14.325.3 Diagram



14.325.4 Fields

Field	Function
31-0	Output Sequence Length. This value is used in output data sequences.
SOL	SOL can also be used as a general purpose math register.

14.326 Variable Sequence Output Length Register (VSOL0 - VSOL5)

14.326.1 Offset

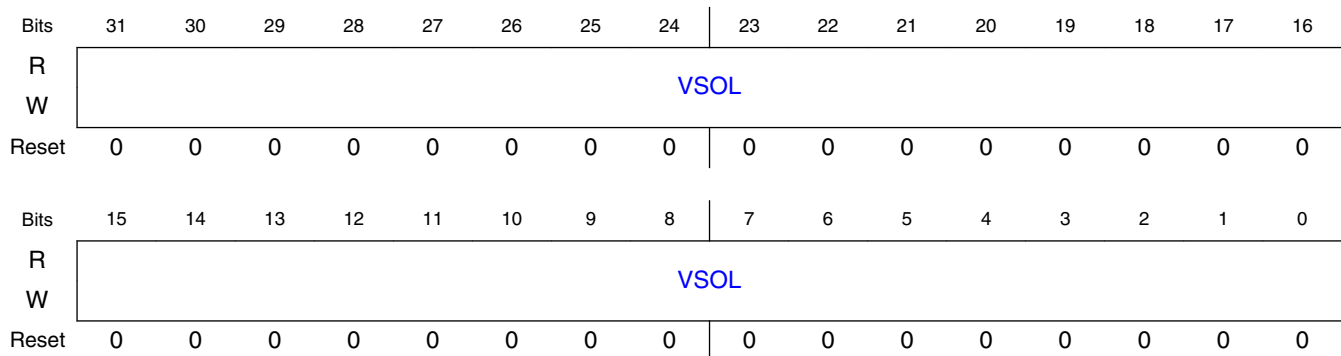
For a = 0 to 5:

Register	Offset	Description
VSOLa	$8_0E24h + (a \times 1_0000h)$	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.326.2 Function

The Variable Sequence Out Length Register is used to specify a variable amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). See [SEQ vs non-SEQ commands](#) for a discussion of sequences. See [Using sequences for fixed and variable length data](#) for a discussion of the use of the VSOL register in Output Sequences. The VSOL Register can be read or written via the MATH Command (see SRC0, SRC1 and DEST fields in [MATH and MATHI Commands](#)). When the DECO is under direct control of software (see [Register-based service interface](#) this register is accessible at the addresses shown above. Note that VSOL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSOL register, located at offset E34h.

14.326.3 Diagram



14.326.4 Fields

Field	Function
31-0 VSOL	This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See UVSOL register.

14.327 Sequence Input Length Register (SIL0 - SIL5)

14.327.1 Offset

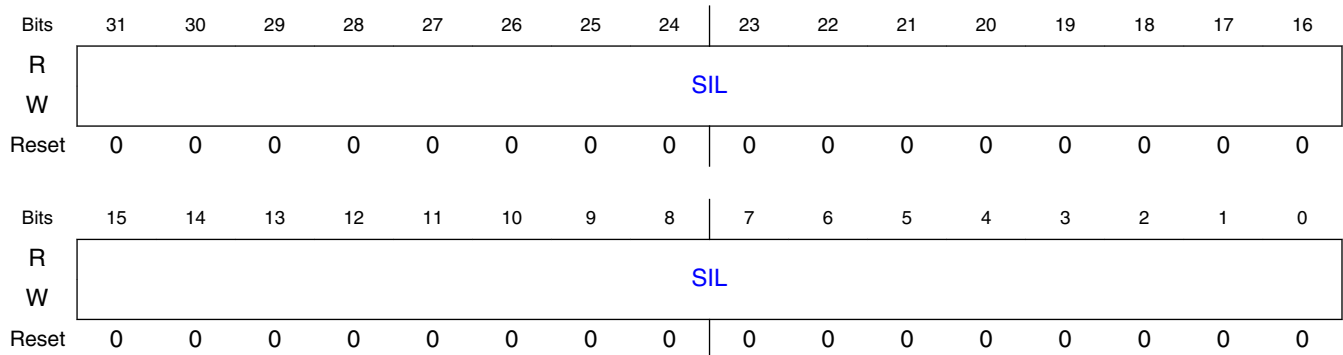
For a = 0 to 5:

Register	Offset	Description
SILa	8_0E28h + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.327.2 Function

The Sequence In Length Register is used to specify the amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). See Section [SEQ vs non-SEQ commands](#) for a discussion of sequences. See [Using sequences for fixed and variable length data](#) for a discussion of the use of the SIL register in Input Sequences. The SIL Register can be read or written via the MATH Command (see SRC0 and DEST fields in [MATH and MATHI Commands](#)). When the DECO is under direct control of software (see [Register-based service interface](#) this register is accessible at the addresses shown above. This register can also be loaded by the SEQ IN PTR command.

14.327.3 Diagram



14.327.4 Fields

Field	Function
31-0 SIL	This value is used in input data sequences. SIL can also be used as a general purpose math register.

14.328 Variable Sequence Input Length Register (VSIL0 - VSIL5)

14.328.1 Offset

For a = 0 to 5:

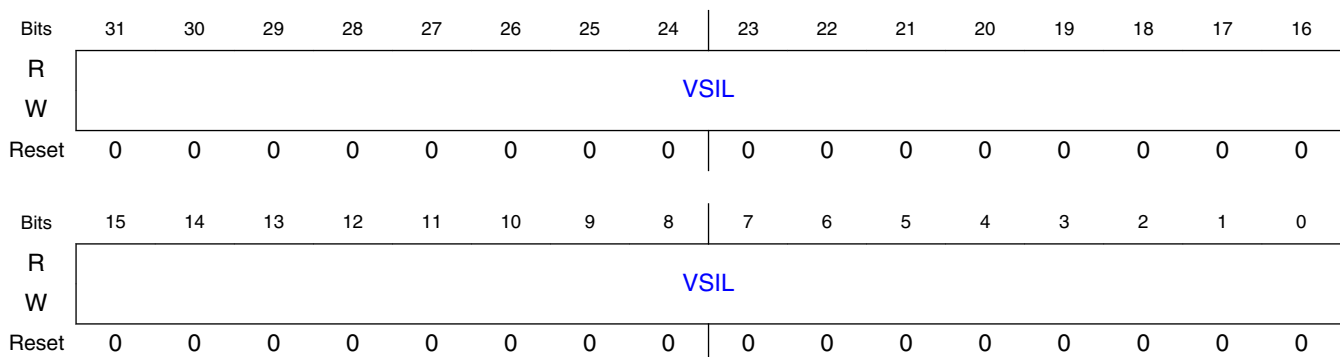
Register	Offset	Description
VSILa	8_0E2Ch + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.328.2 Function

The Variable Sequence In Length Register is used to specify a variable amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). See Section [SEQ vs non-SEQ commands](#) for a discussion of

sequences. See [Using sequences for fixed and variable length data](#) for a discussion of the use of the VSIL register in Input Sequences. This register is also loaded when RIF is executed for a Shared Descriptor. The VSIL Register can be read or written via the MATH Command (see SRC0, SRC1, and DEST fields in [MATH and MATHI Commands](#)). When the DECO is under direct control of software (see [Register-based service interface](#)) this register is accessible at the addresses shown above. Note that VSIL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSIL register, located at offset E38h.

14.328.3 Diagram



14.328.4 Fields

Field	Function
31-0 VSIL	This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See UVSIL register.

14.329 Protocol Override Register (D0POVRD - D5POVRD)

14.329.1 Offset

For a = 0 to 5:

Protocol Override Register (D0POVRD - D5POVRD)

Register	Offset	Description
DaPOVRD	8_0E30h + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

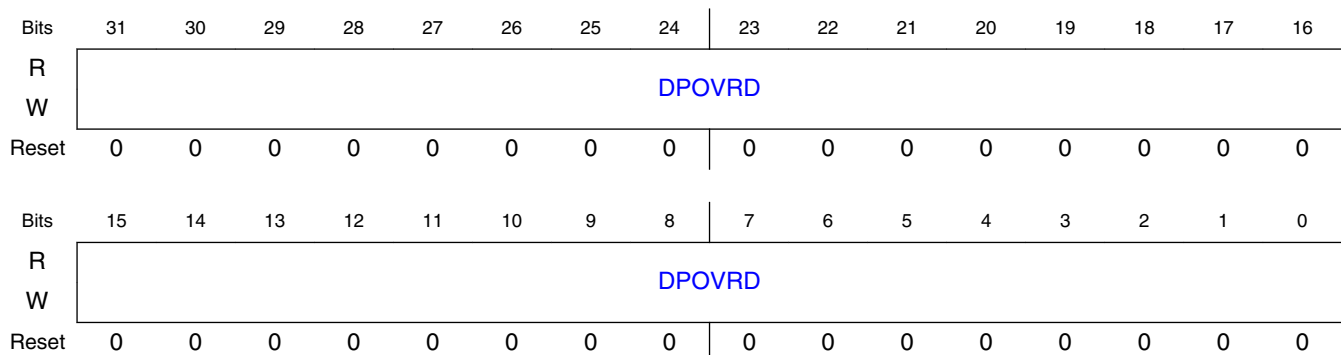
14.329.2 Function

The DECO Protocol Override Register is used to override the PDB options for certain built-in network protocols. can be read or written via the MATH Command (see SRC0, SRC1 and DST fields in [MATH and MATHI Commands](#)) and it can be written via a LOAD IMMEDIATE command (see DST value 07h, Class=11 in [LOAD commands](#)). Note that DPOVRD can also be used as a general purpose math register. The format of the register is specific to the protocol (see table below). When the DECO is under direct control of software (see [Register-based service interface](#)) this register is accessible at the addresses shown above.

Table 14-8. DECO Protocol Override Register - Formats for Various Networking Protocols

Protocol	Format Diagram
IPsec ESP Encapsulation	See Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register
IPsec ESP Decapsulation	See Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register
SSL 3.0 / TLS 1.0 Encapsulation	See Overriding the PDB for SSL, TLS, and DTLS Encapsulation
TLS 1.1 / 1.2 Encapsulation	See Overriding the PDB for SSL, TLS, and DTLS Encapsulation
DTLS Encapsulation	See Overriding the PDB for SSL, TLS, and DTLS Encapsulation

14.329.3 Diagram



14.329.4 Fields

Field	Function
31-0 DPOVRD	This register can be written to override various PDB settings. The format used for the DPOVRD register depends on the particular protocol operation that is executed. Table 14-8 contains links to the different format diagrams.

14.330 Variable Sequence Output Length Register; Upper 32 bits (UVSOL0 - UVSOL5)

14.330.1 Offset

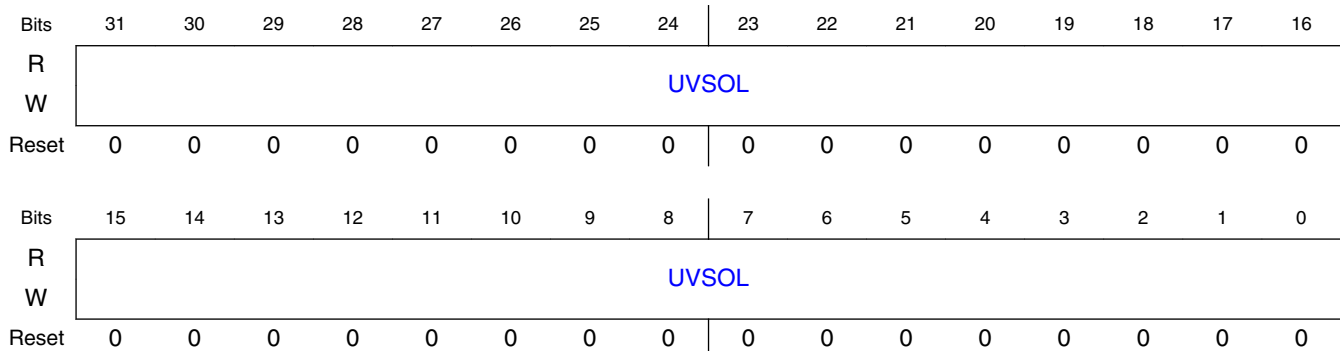
For a = 0 to 5:

Register	Offset	Description
UVSOLa	8_0E34h + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.330.2 Function

VSOL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSOL register, located at offset E24h, and the most-significant 32 bits are accessible as the UVSOL register, located at offset E34h.

14.330.3 Diagram



14.330.4 Fields

Field	Function
31-0 UVSOL	This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See VSOL register. In some older versions of SEC the UVSOL register did not exist, i.e. the VSOL register was only 32 bits. In those older versions when VSOL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVOL/VSOL and then 0 bits will be shifted in from the left.

14.331 Variable Sequence Input Length Register; Upper 32 bits (UVSIL0 - UVSIL5)

14.331.1 Offset

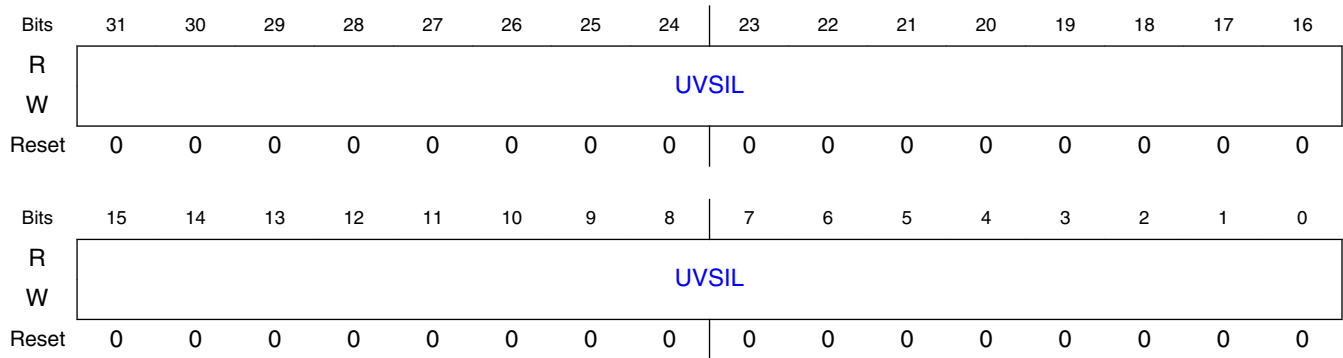
For a = 0 to 5:

Register	Offset	Description
UVSILa	8_0E38h + (a × 1_0000h)	For DECOa. Accessible only when RQDa and DENa are asserted in DECORR.

14.331.2 Function

VSIL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSIL register, located at offset E2Ch, and the most-significant 32 bits are accessible as the UVSIL register, located at offset E38h.

14.331.3 Diagram



14.331.4 Fields

Field	Function
31-0 UVSIL	This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See VSIL register. In some older versions of SEC the UVSIL register did not exist, i.e. the VSIL register was only 32 bits. In those older versions when VSIL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVIL/VSIL and then 0 bits will be shifted in from the left.

Appendix A

Revision History

Because this is the initial publication of the document, there are no changes.



Appendix B

Acronyms and abbreviations

Table B-1. Acronyms and abbreviated terms

Term	Meaning
AAD	Additional Authenticated Data
AES	Advanced Encryption Standard - 128-bit block encryption algorithm, using a 128, 192 or 256-bit key.
AXI	AMBA Advanced eXtensible Interface (AXI) Protocol Specification. Defined by Arm Ltd.
BMan	Buffer Manager A companion block to SEC that manages the buffers that supply data and receive results through SEC's Queue Manager Interface
CBC	Cipher Block Chaining An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
CCB	Cryptographic Control Block A logic module within SEC
CCM	Counter with CBC-MAC Mode An authenticated encryption mode of operation. (Also known as CBC-MAC for CTR mode.)
CFB	Cipher FeedBack An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
CHA	Cryptographic Hardware Accelerator One of the hardware accelerators used in SEC
CRJD	Control Replacement Job Descriptor
CSP	Critical Security Parameter Security related information (such as secret and private cryptographic keys or authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. (See FIPS140-2)
CTR	Counter mode An encryption mode of operation used with AES
DECO	Descriptor Controller A logic module within SEC
DEK	Data Encryption Key.
DES	Data Encryption Standard 64-bit block encryption algorithm, using a 64-bit key.
3DES	Triple DES 64-bit block encryption algorithm, using a 128 or 196-bit key.

Table continues on the next page...

Table B-1. Acronyms and abbreviated terms (continued)

Term	Meaning
DPAA1/2	Datapath Acceleration Architecture, version 1 and version 2
DRBG	Deterministic Random Bit Generator A deterministic algorithm that generates a sequence of numbers whose values are statistically random. Sometimes called "PRNG" (pseudorandom number generator).
ECB	Electronic Code Book An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
FQ	Frame Queue
HAB	High Assurance Boot software
HMAC	A hashing mode of operation used to implement a Message Authentication Code
ICID	Isolation Context Identifier
IPAD	Inner padding defined for HMAC
ICV	Integrity Check Value A checksum or message digest that allows detection of errors or changes in data.
IJD	Inline Job Descriptor
IV	Initialization Vector A value used to initialize some encryption modes of operation
JD	Job Descriptor
JDKEK	Job Descriptor Key Encryption Key
JQC	Job Queue Controller The hardware that schedules jobs received from the Job Rings, Queue Manager Interface and RTIC
JR	Job Ring
MD5	A message digest algorithm returning a 128-bit hash value
MDHA	Message Digest Hardware Accelerator (hashing accelerator block)
NS	Non-secure indication NS = 0 is secure. This signal is generated by the TrustZone feature implemented in some Arm processors. The Central Security Unit (CSU) generates this signal for other bus masters.
NVTK	Non-volatile Test Key
OFB	Output FeedBack An encryption mode of operation. This is one of the official modes of operation specified for DES and AES.
OPAD	Outer padding defined for HMAC
OTPMK	One-time-programmable Master Key
PKHA	Public Key Hardware Accelerator (ECC, RSA, DH, DSA)
POR	Power On Reset.
PRNG	Pseudo Random Number Generator A deterministic algorithm that generates a sequence of numbers whose values are statistically random. See DRBG.
PSP	Public Security Parameter Security-related public information whose modification can compromise the security of a cryptographic module.
QMan	Queue Manager

Table continues on the next page...

Table B-1. Acronyms and abbreviated terms (continued)

Term	Meaning
	A companion block to SEC that organizes data streams into queues of PDUs that are processed through SEC's Queue Manager Interface
QI	Queue (Manager) Interface A logic block within SEC that interacts with the BMan and QMan blocks external to SEC
RNG	Random Number Generator A hardware module within SEC that generates random numbers based on the interaction of two free running ring oscillators and uses these random numbers to seed a DRBG.
RJD	Replacement Job Descriptor
RTIC	Run-Time Integrity Checker A logic block within SEC that generates a tamper alarm if the integrity of selected memory areas have been compromised
SD	Shared Descriptors
SEC	Security Engine (Also known as Cryptographic Acceleration and Assurance Module)
SecMon	Security Monitor A companion block to SEC that detects security violations and maintains security state.
SHA-1	A message digest algorithm defined in FIPS 180-2 returning a 160-bit hash value.
SHA-224	A message digest algorithm defined in FIPS 180-2 returning a 224-bit hash value.
SHA-256	A message digest algorithm defined in FIPS 180-2 returning a 256-bit hash value.
SHA-384	A message digest algorithm defined in FIPS 180-2 returning a 384-bit hash value.
SHA-512	A message digest algorithm defined in FIPS 180-2 returning a 512-bit hash value.
SSP	Sensitive Security Parameter Data whose integrity must be protected
SWRST	Software Reset Register resets caused by writing 1 to the SWRST field in the MCFGR register.
TD	Trusted Descriptor
TDSK	Trusted Descriptor Signing Key
TDKEK	Trusted Descriptor Key Encryption Key
TRK	Trusted Root Key
ZMK	Zeroizable Master Key



Appendix C

Glossary

Table C-1. Glossary of terms

Term	Description
Alleged RC4	A stream cipher that is compatible with RC4.
Black blob	A blob whose input data when exporting and whose output when importing was assumed to be a black key. When exporting a black blob, the input data is first decrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB command was in a trusted descriptor) before being encrypted with the blob key. When importing a Black Blob, the data blob is first decrypted with the Blob Key before being encrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB Command was in a trusted descriptor). (See <i>Red Blob</i> .)
Black key	A key that has been encrypted using either the JDKEK or the TDKEK. (See <i>Red Key</i> .)
Blob	As used in this Block Guide the term 'blob' refers to a cryptographically protected data object consisting of a Blob Key encrypted with a Blob Key Encryption Key, a Data Blob encrypted with a Blob Key, and the MAC Tag resulting from the AES-CCM encryption of the Data Blob.
Blob key	The 256-bit random number used for AES-CCM encryption of the data portion of a blob.
Blob key encryption key	The Blob Key Encryption Key (BKEK) is a 256-bit key used when encrypting cryptographic Blobs exported from memory. It is intended for use in protecting the confidentiality and integrity of this data. The BKEK is derived from the Master Key or Non-volatile Test Key, a constant embedded in the SEC Descriptor that initiated the Blob operation, the Security mode and the Blob type. (See <i>Master Key</i> .)
Critical security parameter	A critical security parameter (CSP) is security-related information (e.g., secret and private cryptographic keys, and authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)]
Data encryption key	A data encryption key is a key that can be referenced in a descriptor as a cryptographic key and that is not one of other keys defined in this glossary. Some examples are: a symmetric key used for encryption or decryption of session data, a private key used for signing data, a public key used for verifying a signature, a private or public key used in a key establishment operation, an HMAC key.
Decrypt key	A decrypt key is used for decrypting data to yield plaintext (unencrypted data). Some cryptographic algorithms (e.g. AES) successively modify the cryptographic key during the steps of the cryptographic operation; therefore the decrypt form of the key is different from the encrypt form of the key.
Descriptor	A descriptor is a sequence of commands that causes SEC to perform cryptographic functions. There are three types of descriptors: job descriptors, shared descriptors, and trusted descriptors. Shared descriptors and trusted descriptors are actually special forms of job descriptors. Note that this usage of the term 'descriptor' is not related to the term 'frame descriptor'.
Fail mode	SEC clears its CSP registers (e.g. key registers) upon entrance to <i>Fail Mode</i> . SEC enters Fail Mode when the SecMon's security state machine enters its Fail State. This could be due to the detection of tampering, scan or JTAG testing or due to the failure of a security module.
Frame descriptor	SEC inputs and outputs data over its Queue Interface in data structures called 'frame descriptors'. Note that the term 'frame descriptor' is not related to the terms 'job descriptor', 'trusted descriptor', or 'shared descriptor'.

Table continues on the next page...

Table C-1. Glossary of terms (continued)

Term	Description
Hash	A hash is the message digest resulting from a hashing operation, such as SHA-1, SHA-256 or MD5. A cryptographic hashing operation is a collision-resistant one-way function that yields a fixed-length bit string from a variable length input. A function is collision-resistant if it is difficult to find two input strings that yield the same Message Digest. A function is one-way if it is computationally infeasible to calculate the input, given only the Message Digest.
Job descriptor	The term 'job descriptor' means a descriptor that is not a shared descriptor or a trusted descriptor. Unlike a shared descriptor, a job descriptor can reference another descriptor, and unlike a trusted descriptor, a job descriptor is not signed.
Job descriptor key encryption key	The Job Descriptor Key Encryption Key (JDKEK) is a 256-bit key used to protect the confidentiality of Data Encryption Keys (DEK) referenced by job descriptors. A new JDKEK value is generated by the SEC's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during job descriptor processing. (See <i>Trusted Descriptor Key Encryption Key</i> .)
Key encryption key	A Key Encryption Key (KEK) is a cryptographic key used to encrypt other cryptographic keys. SEC supports various KEKs that are used in different circumstances. (See <i>JDKEK, TDKEK</i>)
Link table	A link table is also referred to by the term "Scatter/Gather Table".
Master key	The master key is a 256-bit secret value that SEC receives from the SecMon. (See <i>Non-volatile Test Key, OTP Master Key, Zeroizable Master Key, and Blob Key Encryption Key</i> .)
Message digest	A message digest (also called a hash) is a fixed-size string that is the result of computing a cryptographic one-way function of some input data.
Non-volatile test key	<p>The Non-volatile Test Key (NVTK) is a 256-bit key hardwired into SEC. When SEC is in the Non-Secure Mode SEC will use the NVTK to derive Blob key encryption keys, rather than using the secret Master Key. The NVTK value is public knowledge, and is the same in every SOC. It is used for known-answer tests when testing the SEC cryptographic hardware.</p> <p>The NVTK is simply an all 0s key, used when the OTPMK is not available. It allows software to work without proper key access. The NVTK is in every version of CAAM, and is the same in every version of CAAM. There is no key register to read, as the NVTK is built into the hardware.</p>
Non-secure mode	SEC's Non-secure Mode is intended to allow SEC to be tested without compromising the security of sensitive data. In this mode a known version of the BKEK (based on the Non-volatile Test Key) is used for exporting and importing Blobs. Therefore any Blobs exported while in Secure Mode or Trusted Mode cannot be successfully imported while in Non-secure Mode.
OTP master key	The OTP Master Key (OTPMK) is a 256-bit secret value stored in one-time-programmable storage on the SOC. The value is generally written to the one-time-programmable storage while the SOC is in the factory. The OTPMK bits are protected with a lock that, when set, prevents modifying the value. In some configurations SecMon will use the OTPMK to derive the value of the master key that the SecMon supplies over a private bus to SEC. Its value cannot otherwise be read, sensed or scanned.
Processor	A processor is a bus master capable of executing software.
Public security parameter	A public security parameter (PSP) is security-related public information whose modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)] The Trusted Root Key is a PSP.
Read safe	A read-safe transaction reads a full aligned burst of data, even if not all of the data is needed.
Red blob	A blob whose data input when exporting is assumed to be not encrypted, and whose data output when importing is not encrypted. (See <i>Black Blob</i>)
Red key	A key that is not encrypted. (See <i>Black Key</i>)
Replay	Replay is a type of security attack in which old data is presented by a hacker as if it were new data. For instance, a hacker could replace a new Blob that shows that a software license has expired with an old Blob that indicates that the license is still valid. The term "replay" is sometimes also used to refer to a denial of service attack based upon flooding the system with the same message over and

Table continues on the next page...

Table C-1. Glossary of terms (continued)

Term	Description
	over. If this message is encrypted or cryptographically authenticated, then the attacker may not be able to generate new messages and instead would "replay" a legitimate message that the attacker had snooped from the network.
Secure mode	Secure Mode is the normal operating mode of SEC. The Security State Machine within the SecMon determines when SEC is operating in Secure Mode.
Security monitor	Security Monitor (SecMon) is a companion logic block to SEC. It implements a security alarm and maintains a security state machine.
Sensitive data	Sensitive data is data that should be protected against unauthorized disclosure.
Sensitive security parameter	The term 'sensitive security parameters' (SSP) encompasses critical security parameters and public security parameters. [from FIPS PUB 140-3 (DRAFT)]
SEQ	Sequence. For most memory referencing descriptor commands SEC implements an auto-incrementing addressing mode using sequence input address and sequence output address registers. This is intended to facilitate the processing of cryptographic networking protocols.
Shared descriptor	A shared descriptor is a special type of job descriptor that can be executed only when it is referenced by another descriptor. Shared descriptors are intended to contain data, such as keys and sequence numbers, that are shared by two or more other descriptors.
Trusted descriptor	A trusted descriptor is a special type of job descriptor that has some additional access privileges and some additional security protections. Trusted descriptors are protected from modification by means of a signature over the descriptor. SEC verifies the signature before executing the trusted descriptor, and aborts execution if the signature is incorrect. (See <i>Trusted Descriptor Signing Key</i>)
Trusted descriptor signing key	The <i>Trusted Descriptor Signing Key</i> (TDSK) is a 256-bit key used in HMAC-SHA-256 to sign and verify the signature over trusted descriptors. A new TDSK value is generated by the SEC RNG at each POR, and is used throughout the current power-on cycle. SEC will allow TDSK to be used to sign a new trusted descriptor only if the descriptor is submitted via a Job Ring that has AMTD set in its JRaCID register. Otherwise, SEC will use TDSK only to verify the signature over a trusted descriptor, or to update the signature on an existing trusted descriptor that has modified itself during its execution.
Trusted descriptor key encryption key	The Trusted Descriptor Key Encryption Key (TDKEK) is a 256-bit key that can be used to protect the confidentiality of Data Encryption Keys (DEKs) referenced by trusted descriptors. A new TDKEK value is generated by the SEC's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during trusted descriptor processing. (See <i>Job Descriptor Key Encryption Key</i>)
Trusted mode	Trusted Mode is a special operating mode of SEC. The Security State Machine within the SecMon determines when SEC is operating in Trusted Mode. This mode is implemented so that trusted boot-time software, or a hypervisor or TrustZone Secure World software can store data in and retrieve data from Trusted Mode Blobs that are not accessible to software running while SEC is in Secure Mode or Non-Secure Mode.
Trusted root key	The Trusted Root Key is a public signature key used by HAB to verify the signature over the Command Sequence File. The key could be RSA (probably 2048 bits) or ECC-DSA (probably 511 bits). The integrity and authenticity of this key is protected by placing a SHA-256 hash of this key in fuses on the SOC. The fuses are located in a bank with a lock fuse that, when set, prevents any changes to the hash value.
Word	A word of memory or a one-word register contains 32 bits.
Write safe	A write-safe transaction writes 0s to addresses past the targeted locations up to the next 8, 16, 32 or 64-byte address boundary, depending upon the offset within the cacheline.
Zeroizable master key	The Zeroizable Master Key (ZMK) is a 256-bit key stored in a register in the low-power domain of SecMon. In some configurations and security states SecMon will use the ZMK to derive the value of the Master Key that SecMon supplies over the snvs_master_key signal to SEC. Its value cannot otherwise be read or scanned. The value can be generated by the SEC RNG, and can be loaded automatically by hardware. The value can be zeroized when a tamper event is detected. (See <i>Master Key</i> .)

Table continues on the next page...

Table C-1. Glossary of terms (continued)

Term	Description
Zeroize	A set of data storage locations is <i>zeroized</i> by overwriting the storage locations with a value (not necessarily 0) that is independent of the previous content of the storage locations.

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μ Vision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

Document Number LS2088ASECRM
Revision 0, 04/2018

