M•CORE™

MMC2001 Device Driver Reference Manual Revision 1.0

Freescale Semiconductor, Inc.

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

The M•CORE name and logotype are trademarks of Motorola, Inc.

© Motorola, Inc. 1999

Freescale Semiconductor, Inc.

Table of Contents

Section 1 Introduction

1.1 Overview 15
1.2 Service Categories 16
1.2.1 Level 1 Services 16
1.2.2 Level 2 Services 17
1.3 Interrupt Integration. 18
1.3.1 Interrupt Service Entry 18
1.3.2 Interrupt Status Communication 19
1.4 References 20

Section 2 Glossary

2.1 Terminology 21

Section 3 Using the Device Driver Library

3.1 Library and Interface Files 25
3.2 Library Macros 26
3.3 Error Checking 27
3.4 Memory-Mapped Peripheral Device Blocks 28
3.5 Direct Register Access 28
3.6 Usage Tutorial. 29
3.7 Standard Data Type Definitions 33

Section 4 EdgePort_A Level 1

4.1 Overview 35
4.2 EdgePort_A Level 1 Device Driver Functions. 35
4.3 Data Type Definitions 35
4.3.1 Abstract Data Type Definitions 35
4.3.2 Enumerated Data Type Definitions. 36
4.4 API Definitions 36
4.4.1 EdgePort_A_GetRegister 37
4.4.2 EdgePort_A_SetRegister 39

Section 5 EIM_A Level 1

Freescale Semiconductor, Inc.

5.1 Overview 41
5.2 EIM_A Level 1 Device Driver Functions 41
5.3 Data Type Definitions 41
5.3.1 Abstract Data Type Definitions 42
5.3.2 Enumerated Data Type Definitions 42
5.4 API Definitions 42
5.4.1 EIM_A_GetRegister 43
5.4.2 EIM_A_SetRegister 45

Section 6 INTC_A Level 1

6.1 Overview 47
6.2 INTC_A Level 1 Device Driver Functions 47
6.3 Data Type Definitions 48
6.3.1 Abstract Data Type Definitions 48
6.3.2 Enumerated Data Type Definitions 49
6.4 API Definitions 49
6.4.1 INTC_A_Init 50
6.4.2 INTC_A_IntEnable 52
6.4.3 INTC_A_IntDisable 54
6.4.4 INTC_A_SetISF 56
6.4.5 INTC_A_SetSSF 58
6.4.6 INTC_A_GetRegister 60
6.4.7 INTC_A_SetRegister 62

Section 7 ISPI_A Level 1

7.1 Overview 65
7.2 ISPI_A Level 1 Device Driver Functions 65
7.3 Data Type Definitions 66
7.3.1 Abstract Data Type Definitions 66
7.3.2 Enumerated Data Type Definitions 66
7.4 API Definitions 68
7.4.1 ISPI_A_Init 68
7.4.2 ISPI_A_ManualEnable 71
7.4.3 ISPI_A_IntervalEnable 73
7.4.4 ISPI_A_SlaveEnable 75
7.4.5 ISPI_A_Disable 77

Freescale Semiconductor, Inc.

7.4.6 ISPI_A_Transmit 79
7.4.7 ISPI_A_Receive 81
7.4.8 ISPI_A_GPControl 83
7.4.9 ISPI_A_Loopback 85
7.4.10 ISPI_A_ClearOverrun 87
7.4.11 ISPI_A_GetInterruptStatus 89
7.4.12 ISPI_A_GetRegister 91
7.4.13 ISPI_A_SetRegister 93

Section 8 KPP_A Level 1

8.1 Overview 95
8.2 KPP_A Level 1 Device Driver Functions 95
8.3 Data Type Definitions 95
8.3.1 Abstract Data Type Definitions 95
8.3.2 Enumerated Data Type Definitions 96
8.4 API Definitions 96
8.4.1 KPP_A_Init 97
8.4.2 KPP_A_KeyControl 99
8.4.3 KPP_A_GetStatus 101
8.4.4 KPP_A_KeyColumnScan 103
8.4.5 KPP_A_GetRegister 106
8.4.6 KPP_A_SetRegister 108

Section 9 LCD_A Level 1

9.1 Overview 111
9.2 LCD_A Level 1 Device Driver Functions 112
9.3 Data Type Definitions 113
9.3.1 Abstract Data Type Definitions 113
9.3.2 Enumerated Data Type Definitions 113
9.3.3 Initialization Defaults Macros 115
9.4 API Definitions 115
9.4.1 LCD_A_Init 116
9.4.2 LCD_A_ClrDisplay 119
9.4.3 LCD_A_ReturnHome 121
9.4.4 LCD_A_SetEntryMode 123
9.4.5 LCD_A_DisplayControl 125

Freescale Semiconductor, Inc.

9.4.6 LCD_A_Shift 127
9.4.7 LCD_A_WaitNotBusy 129
9.4.8 LCD_A_SetDDRAMAddr 131
9.4.9 LCD_A_Write 133
9.4.10 LCD_A_Read 135
9.4.11 LCD_A_CGWrite 137
9.4.12 LCD_A_GetRegister 140
9.4.13 LCD_A_SetRegister 142

Section 10 PWM_A Level 1

10.1 Overview 145
10.2 PWM_A Level 1 Device Driver Functions 145
10.3 Data Type Definitions 146
10.3.1 Abstract Data Type Definitions 146
10.3.2 Enumerated Data Type Definitions 146
10.4 API Definitions 147
10.4.1 PWM_A_Init 148
10.4.2 PWM_A_Start 150
10.4.3 PWM_A_Stop 152
10.4.4 PWM_A_UpdateOutput 154
10.4.5 PWM_A_GetStatus 156
10.4.6 PWM_A_GetIRQ 158
10.4.7 PWM_A_GetRegister 160
10.4.8 PWM_A_SetRegister 162

Section 11 TRM_A Level 1

11.1 Overview 165
11.2 TRM_A Level 1 Device Driver Functions 165
11.3 Data Type Definitions 166
11.3.1 Abstract Data Type Definitions 166
11.3.2 Enumerated Data Type Definitions 166
11.4 API Definitions 167
11.4.1 TRM_A_InitRSCR 167
11.4.2 TRM_A_GetTODAlarmStatus 169
11.4.3 TRM_A_ClearTODAlarmInterrupt 171
11.4.4 TRM_A_ControlTODAlarmEnable 173

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.4.5 TRM_A_ControlTODAlarmInterrupt 175
11.4.6 TRM_A_InitWD. 177
11.4.7 TRM_A_ServiceWD 179
11.4.8 TRM_A_InitPIT 181
11.4.9 TRM_A_SetPITModulus. 183
11.4.10 TRM_A_ClearPITInterrupt 185
11.4.11 TRM_A_ControlPITEnable 187
11.4.12 TRM_A_ControlPITInterrupt. 189
11.4.13 TRM_A_GetPITStatus 191
11.4.14 TRM_A_GetRegister 193
11.4.15 TRM_A_SetRegister. 195

Section 12 UART_A Level 1

12.1 Overview. 197
12.2 UART_A Level 1 Device Driver Functions 197
12.3 Data Type Definitions 198
12.3.1 Abstract Data Type Definitions 198
12.3.2 Enumerated Data Type Definitions. 198
12.3.3 Initialization Default Macros 200
12.4 API Definitions 201
12.4.1 UART_A_Init. 201
12.4.2 UART_A_Enable 204
12.4.3 UART_A_Disable 206
12.4.4 UART_A_IntEnable 208
12.4.5 UART_A_IntDisable 210
12.4.6 UART_A_SetDivider. 212
12.4.7 UART_A_Receive 214
12.4.8 UART_A_Transmit 216
12.4.9 UART_A_ReadPin 218
12.4.10 UART_A_WritePin 220
12.4.11 UART_A_Infrared. 222
12.4.12 UART_A_SendBreak 224
12.4.13 UART_A_ParityError 226
12.4.14 UART_A_Loopback 228
12.4.15 UART_A_IrLoopback 230
12.4.16 UART_A_GetStatus 232

Freescale Semiconductor, Inc.

12.4.17 UART_A_GetRegister 234
12.4.18 UART_A_SetRegister. 236

Section 13 UART_A Level 2

13.1 Overview. 239
13.2 UART_A Level 2 Device Driver Functions 239
13.3 Data Type Definitions 239
13.3.1 Abstract Data Type Definitions. 240
13.3.2 Initialization Default Macros 240
13.4 API Definitions 241
13.4.1 BRT_A_Init 242
13.4.2 BRT_A_BufInit 245
13.4.3 BRT_A_BufReset 247
13.4.4 BRT_A_BufStatus 249
13.4.5 BRT_A_Enable. 251
13.4.6 BRT_A_Disable 253
13.4.7 BRT_A_IntEnable. 255
13.4.8 BRT_A_IntDisable 257
13.4.9 BRT_A_SetBaudRate. 259
13.4.10 BRT_A_SetThreshold. 261
13.4.11 BRT_A_Receive 263
13.4.12 BRT_A_Transmit 265
13.4.13 BRT_A_ReadPin 267
13.4.14 BRT_A_WritePin 269
13.4.15 BRT_A_Infrared 271
13.4.16 BRT_A_SendBreak 273
13.4.17 BRT_A_ParityError. 275
13.4.18 BRT_A_Loopback 277
13.4.19 BRT_A_IrLoopback 279
13.4.20 BRT_A_GetStatus 281
13.4.21 BRT_A_RX_ISF 283
13.4.22 BRT_A_TX_ISF 285
13.4.23 BRT_A_GetRegister. 287
13.4.24 BRT_A_SetRegister 289

Appendix A Return Codes

Freescale Semiconductor, Inc.

A.1 Return Codes List 291

Index

Record of Changes

Freescale Semiconductor, Inc.

Table of Contents

Freescale Semiconductor, Inc.

List of Figures

Figure 1-1 Device Driver Library System Relationships 15
Figure 1-2 Level 1 Service Relationships 16
Figure 1-3 Level 2 Service Relationships 18
Figure 1-4 Code and Data Structure Layout 20
Figure 3-1 Module Interface File Distribution. 25

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Figures

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Table 2-1 Terminology 21
Table 3-1 MMC2001 Peripheral Device Register Addresses. 28
Table A-1 Return Codes 291

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 1 Introduction

The purpose of this manual is to describe a library of software drivers for peripheral devices on the M•CORE MMC2001.

NOTE: *Device driver names are indicated by the device name with an underscore followed by a capitalized letter.*

1.1 Overview

The M•CORE™ device driver application programming interface (API) and library consists of a set of executable components, interfaces, and documentation to support initialization and control of peripheral devices integrated with the M•CORE processor, specifically those devices found on M•CORE development system boards and single-chip products. Driver modules are designed to be used either in a stand-alone application or integrated into a system using a real-time operating system (RTOS).

In an actual system, the device driver code resides between the hardware, an optional RTOS layer, and the application code. **Figure 1-1** illustrates the relationship of the device driver code with other system components.

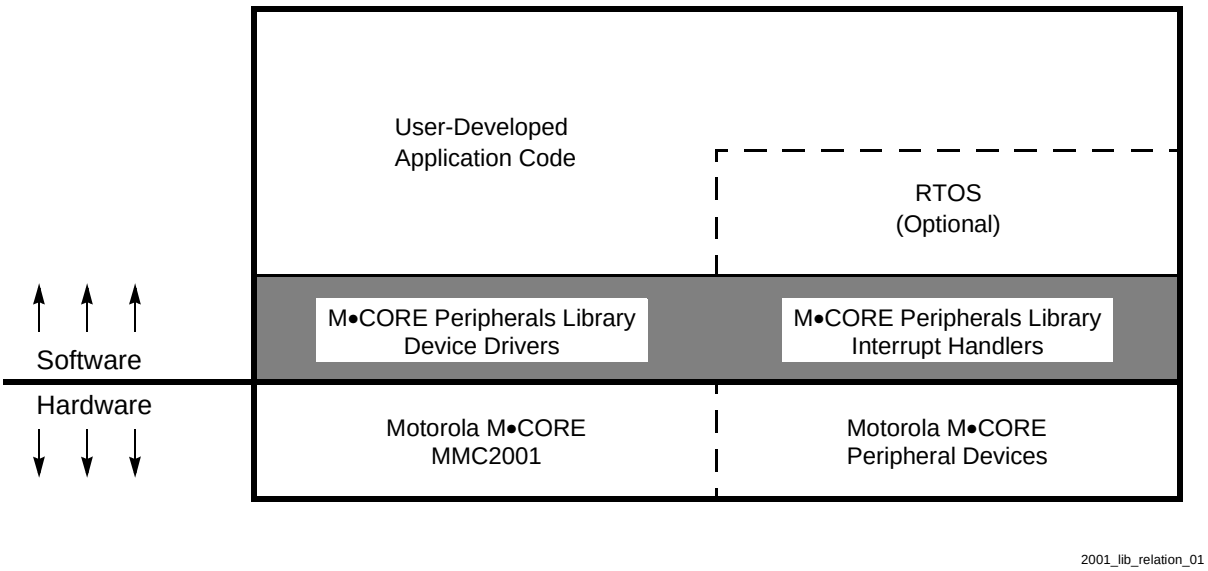

Figure 1-1 Device Driver Library System Relationships

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

1.2 Service Categories

The device driver library module for a given device may be viewed as a *service* in support of that device. Device services can be categorized by the degree of device abstraction which the service presents and the amount of interrupt processing support it provides.

Library functions access device information through a *device handle*. The device handle is always the first parameter in any library call, but what the handle refers to will vary depending on the service level of the call.

1.2.1 Level 1 Services

Level 1 services are at the lowest level; they interact directly with the hardware and return immediately to the caller (possibly with indication that the requested action could not be taken due to the device being busy). The device handle in a level 1 call is always the base address of the device register block.

Being essentially at the level of the raw hardware, a level 1 service has minimal interrupt support, although a higher level service employing interrupts may be built upon a level 1 service. The main benefit of a level 1 service is that it is symbolic; there are no hardware register names or structures to remember, and the useful capability of parameter checking is also available. All M•CORE device driver library modules provide an interface to level 1 services.

Figure 1-2 shows level 1 service relationships.

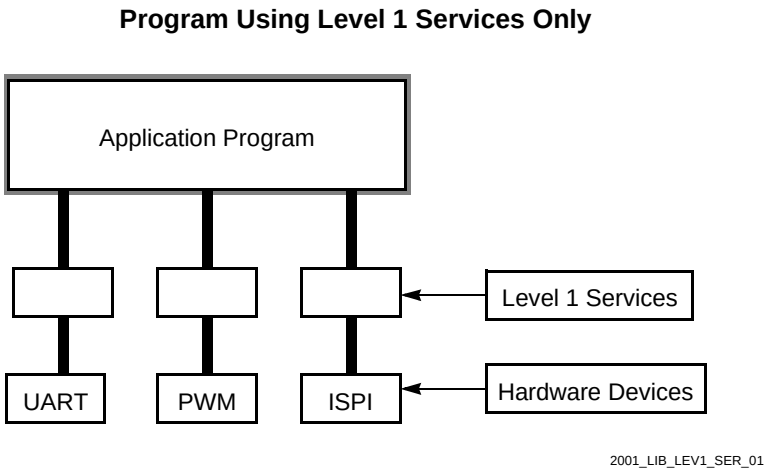

Figure 1-2 Level 1 Service Relationships

Freescale Semiconductor, Inc.

1.2.2 Level 2 Services

A level 2 service presents the user with a more abstract model of a given device than does a level 1 service. It is generally built upon a level 1 service, although it may make use of optimizations unavailable to the application programmer (such as in-lining of level 1 functions). An application program that uses a level 2 service should call only level 2 functions (although some of these may simply pass control through to lower-level functions).

The device handle in a level 2 call is the address of a *device descriptor*. The device descriptor is a user-allocated block of storage which contains, at a minimum, a pointer to the device hardware registers, and beyond that any other state information required to implement service functionality. Other possible components of a device descriptor might be:

- A device completion flag to signal the application program logic
- A completion code accompanying the flag
- A buffer for a single datum
- A queue of data
- A structured message

One device descriptor is created for each instance of a device associated with the service. An example of a level 2 service is an SCI driver implementing a buffered character queue.

Figure 1-3 shows level 2 service relationships.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

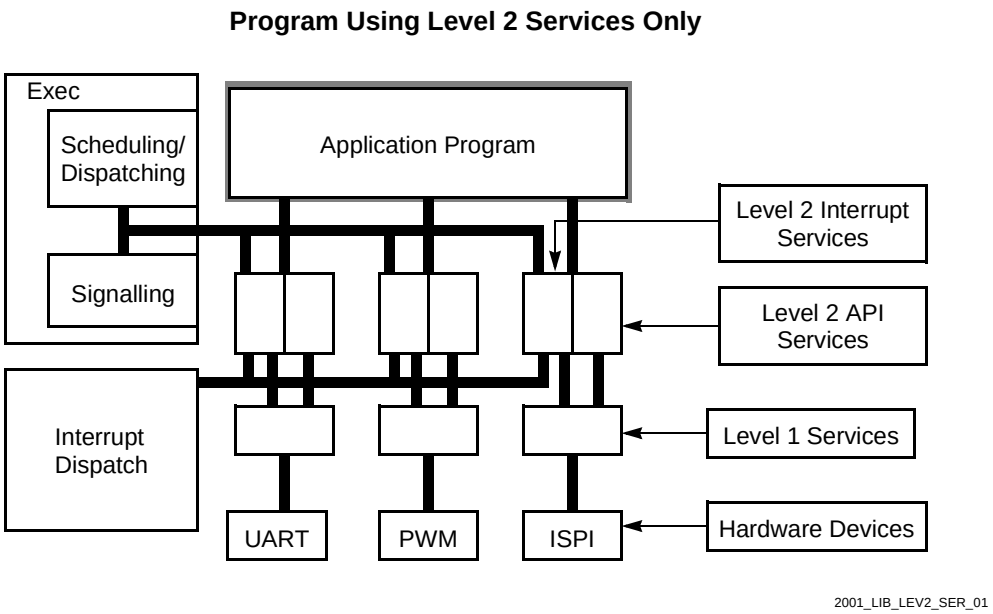

Figure 1-3 Level 2 Service Relationships

1.3 Interrupt Integration

At level 1 (the lowest level), functions are provided that do not explicitly deal with device interrupts. The API of a service at this level presents the programmer with a model of the device essentially at the hardware level, although it provides the programmer with symbolic device access and hides the details of register field layout and manipulation sequences. The programmer can choose to utilize a peripheral service exclusively at this level.

At level 2, a more abstract model is presented which hides more of the hardware characteristics of the peripheral device from the programmer's view. Level 2 is the lowest level at which a device may cause an interrupt without necessitating a response from the application program. At either level, there must be a means for establishing interrupt service entry and a method for communicating status back to the application.

1.3.1 Interrupt Service Entry

The M•CORE processor can support both vectored and autovectored interrupts. In the case of vectored interrupts, the address of a function is mapped directly to the vector space of the processor. For autovectoring, all interrupts are routed through the INT/FINT vectors in the processor vector space. The code executed as a result of interrupt processing is known as the *interrupt service routine* (ISR).

Freescale Semiconductor, Inc.

In order to preserve generality for interrupt processing among all device driver library modules, the ISR can be implemented as an *interrupt dispatch routine*. The dispatch routine calls an *interrupt service function* (ISF) which performs the actual work associated with the interrupt. The dispatch routine can take care of a number of things in preparation for calling the ISF:

1. Sorting out among individual device hardware requests coming through the same vector.
2. Specifying the relevant hardware device address or device descriptor for the given instance of a peripheral. This allows an interrupt service function to be written independently of the hardware or descriptor address, and therefore capable of servicing any number of like devices in the same system.
3. Serving as a “wrapper” for the interrupt service function: performing a normal C function call, receiving control back from the ISF when it is finished, and performing any post-processing that may be necessary.

For example, assume that there are two SCI devices on a chip, and two hardware vectors. Each of these will point to a single dispatch routine. The dispatch routine determines whether the interrupt is for the receive or transmit channel. It also knows the descriptor address (device handle) unique to the hardware device being serviced, and passes this to the appropriate interrupt service function.

1.3.2 Interrupt Status Communication

The interrupt service function may be either a user-implemented routine, or an API function capable of performing the requisite interrupt handling. When the ISF is called by the dispatcher it is still within the interrupt context, so there must be a way of communicating the ISF return status to the application program. This can be accomplished by having the dispatcher call a *service signalling function* (SSF) upon return from the ISF. The SSF is passed the return code of the interrupt service function, along with other pertinent parameters such as the device descriptor and any returned data.

Separating the roles of interrupt service function and service signalling function makes it possible to use arbitrary API functions as ISFs, and provides for customization of the signalling function. The dispatcher as interrupt service routine can be adapted to whatever interrupt mechanism is supported by the system hardware.

Figure 1-4 shows interrupt service relationships.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

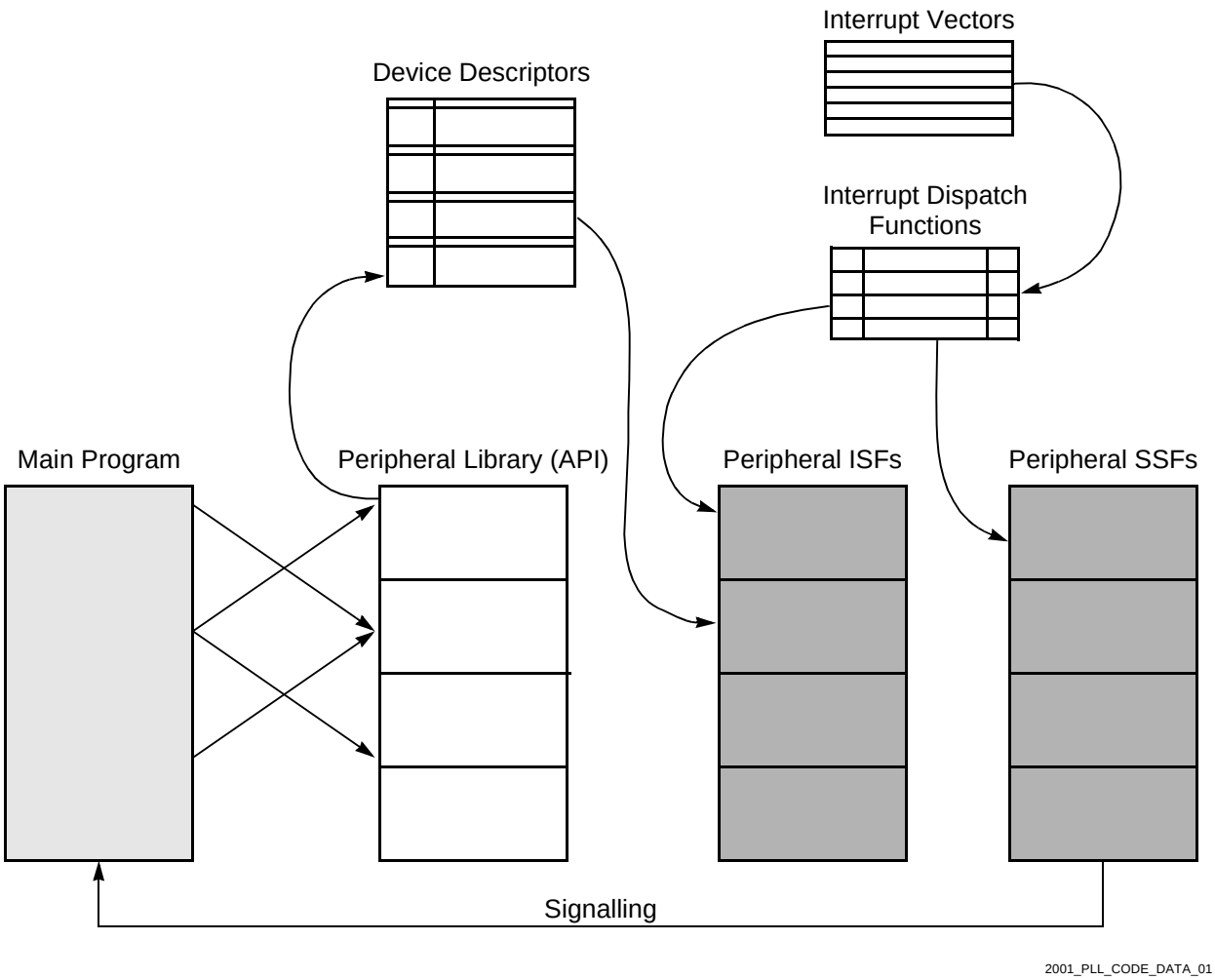

Figure 1-4 Code and Data Structure Layout

1.4 References

Refer to the following documents for more information:

1. Motorola *M•CORE Reference Manual* (MCORERM/AD)
2. Motorola *M•CORE Evaluation System User's Manual* (MCOREEVSUM/D)
3. Motorola *MMC2001 Reference Manual* (MMC2001RM/D)
4. Motorola *M•CORE Applications Binary Interface Standards Manual* (MCOREABI/D)

Freescale Semiconductor, Inc.

Section 2 Glossary

This section provides a glossary of terms used in the M•CORE MMC2001 Device Driver Reference Manual.

2.1 Terminology

Table 2-1 Terminology

Term	Definition
ABI	Applications Binary Interface
Abstract Device	A unit as seen by a programmer through a peripheral library service. The service provides an interface which hides details of the hardware unit.
API	Application Program Interface. The entire set of facilities (function parameters, return values, predefined data structures, etc.) as seen by the writer of the application program.
Application Program	The customer-created software which characterizes the application. The application program makes use of the peripheral library through the API.
CGRAM	Character Generator Random Access Memory in the LCD_A. CGRAM has user-programmable character bitmaps which can be selected for display.
CGROM	Character Generator Read Only Memory in the LCD_A. CGROM has built-in character bitmaps available for selection and display via references from DDRAM.
CMB	Computer and Memory Board
CPU	Central Processor Unit
CTS	Clear-to-Send
DDRAM	Display Data Random Access Memory in the LCD_A. DDRAM stores display data represented in 8-bit character codes.
Device Descriptor	A data structure containing the base address(es) of the hardware registers of a unit that underlies an instance of the abstract device being serviced, plus the state variables of that device (e.g., status flags, I/O buffers or pointers to them).
Device Handle	For level 1 service: the base address of the hardware registers of the unit being serviced. For level 2 service: the address of a device descriptor describing the instance of the abstract device being serviced.
Dispatch Function	Hardware interrupt vector function that determines the interrupt source and calls appropriate user-developed ISF (same as ISR).
Edge Port	External Interrupts/General-Purpose Input/Output (GPIO) Module
EEPROM	Electrically Erasable (Programmable) ROM. ROM which is erasable and rewriteable by on-chip circuitry a byte or word at a time, and perhaps also by group of bytes (“row”) or in its entirety (“array”). The write cycle is orders of magnitude slower than the read cycle and can be performed a limited number of times during the lifetime of the device.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 2-1 Terminology

Term	Definition
EIM	External Interface Module
EPC	Exception program counter
EPROM	Erasable Programmable ROM. ROM which is programmable in the field and is bulk erasable by ultraviolet radiation.
EPSR	Exception Processor Status Word
EVB	Evaluation Board
EVM	Evaluation Module. Any board-level emulator employing an M•CORE chip.
FIFO	First-In/First-Out
Flash EEPROM	Flash-Erasable EEPROM. EEPROM that can be erased only as an entire array.
FPC	Fast Interrupt Program Counter
FPSR	Fast Interrupt Processor Status Word
Frame	Whole character package in an information transfer, including data bits, start and stop bits, parity bits, etc.
GPIO	General-Purpose Input/Output
INTC	Interrupt Controller found on M•CORE-based microcontrollers.
ISF	Interrupt Service Function
ISR	Interrupt Service Routine
ISPI	Interval Mode Serial Peripheral Interface
KPP	Keypad Port. A 16-bit peripheral which can be used either for keypad matrix scanning or as general-purpose I/O.
LCD	Liquid Crystal Display
Level 1 Service	A service that provides basic register-level access to a peripheral unit, not explicitly providing for interrupts. The abstract device provided is almost identical to the actual hardware.
Level 2 Service	A service that provides a higher level of abstraction than a level 1 service, typically including interrupt service (where appropriate) to provide that abstraction. Example: a queued character I/O system built upon an SCI unit.
M•CORE	A family of Motorola microcontrollers based on the microRISC ENGINE.
MCU	Microcontroller Unit. Any of Motorola's modular family of microcontrollers.
OnCE™	On-Chip Emulation. Non-intrusive debugging port built onto all M•CORE silicon.
OTPROM	One-Time Programmable ROM. An EPROM that would normally be erasable by ultraviolet radiation, but which is packaged for cost reasons in an opaque package, making it non-erasable.
PC	Personal Computer (IBM compatible computer with a 486 or better processor)
Peripheral Library	A linkable library of software to support the programming of M•CORE peripheral devices.
PIT	Programmable Interval Timer Submodule
PSR	Processor Status Register

Freescale Semiconductor, Inc.

Table 2-1 Terminology

Term	Definition
PWM	Pulse-Width Modulation Submodule
RAM	Random Access Memory. Read/write volatile memory.
ROM	Read Only (non-volatile) Memory
RSCR	Reset Source/Chip Configuration Register
RTOS	Real-Time Operating System of generic type.
RTS	Request-to-Send
Service	A collection of functions which provide a single effective interface to a peripheral unit.
SSF	Signalling Service Function
Target	The hardware system being developed or debugged.
TOD	Time-Of-Day Submodule
TRM	Timer/Reset Module
UART	Universal Asynchronous Receiver/Transmitter
Unit	A peripheral device or submodule of a device. Example: the PWM submodule of the CTM4.
VBR	Vector Base Register
WD	Watchdog Timer Submodule

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Section 3 Using the Device Driver Library

This section describes the layout of the M•CORE MMC2001 device driver library interface files, library macro usage, error checking, direct register access, tutorial usage, and standard data type definitions.

3.1 Library and Interface Files

The M•CORE MMC2001 device driver library is distributed as a binary object library conforming to the M•CORE Applications Binary Interface (ABI) definition (refer to the *M•CORE Applications Binary Interface Standards Manual* (MCOREABI/D) for more information). It is written in ANSI C, and can be linked into any application that is compiled with a commercially available M•CORE C/C++ compiler that adheres to the ABI standard.

The library itself is contained in a file called *libplib.a*. Module-specific interface files are distributed with the library in the *inc* directory and are named after the peripheral devices to which they apply. **Figure 3-1** displays the distribution tree.

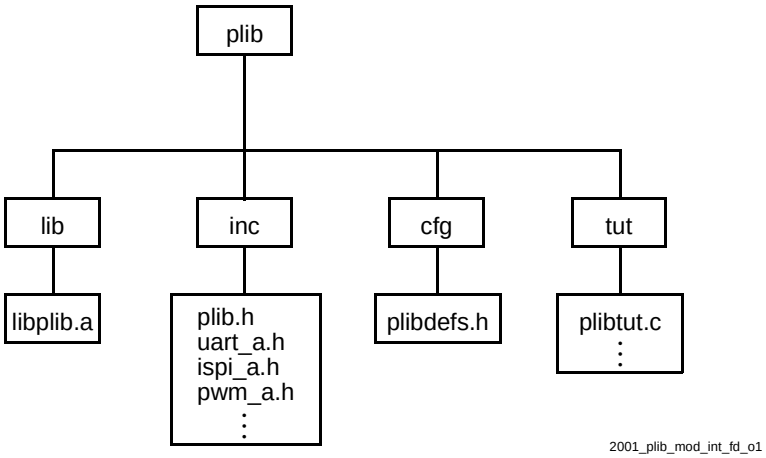

Figure 3-1 Module Interface File Distribution

The *inc* directory also contains global header files included by the module-specific interface files, such as error code definitions. The *cfg* directory holds user-modifiable files containing device register block addresses and driver configuration macros for parameter checking.

Individual device header files are set up to include any higher-level interface files they may require. However, two global header files are of particular interest. The file *inc\plib.h* is the highest-level device driver library header file and contains

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

standard type definitions for all modules. The file *cfg\plibdefs.h* contains module register addresses and preprocessor macro definitions which are user-modifiable.

NOTE: *Only files contained in the cfg subdirectory are user-modifiable; changing other header files may adversely affect library operation.*

3.2 Library Macros

The device driver library API definitions are C language preprocessor macros. This was done in order to efficiently implement selective in-line code generation and API parameter checking. The convention for naming addressable API functions is to append an underscore-lowercase F (_f) to the API name. For example, the UART initialization API is called UART_A_Init, which corresponds to a macro. The addressable function corresponding to this API definition is called UART_A_Init_f. This is useful to know when attempting to load API function addresses into a table or passing an API function pointer as a parameter. However, in most cases using the standard API definition is appropriate.

User-configurable preprocessor macros are concerned with runtime parameter checking. Parameter checking code is enabled at application compile time by setting the <module>_PARAM_CHECKING flag (defined initially in *plibdefs.h*) to a non-zero value, where <module> is the name of the peripheral device in question. Refer to the following example:

NOTE: *The PARAM_CHECKING flag may be used selectively, thereby activating parameter checking in only certain API invocations.*

NOTE: *Parameter checking by default is enabled in the file cfg\plibdefs.h.*

```
#undef ISPI_A_PARAM_CHECKING
#define ISPI_A_PARAM_CHECKING 1         /* Turn on ISPI param checking */

/* Perform parameter checking on initialization */
if (ISPI_A_Init(ISPIPtr, ISPI_BAUD_RATE_4, TRUE, HIGH,
               TOTEMPOLE, TRUE, FALSE, FALSE) != DD_ERR_NONE)
       ...

#undef ISPI_A_PARAM_CHECKING
#define ISPI_A_PARAM_CHECKING 0        /* Turn off ISPI param checking */

/* No parameter checking on enable */
if (ISPI_A_Enable(ISPIPtr, ISPI_A_CLOCK_COUNT_16) != DD_ERR_NONE)
       ...
```

Some modules support a shorthand for API functions which may have many parameters, providing defaults for most of the arguments. These default macros generally have the same name as the documented API function, except that all characters in the macro name are uppercase and _DEFAULT is appended to the

Freescale Semiconductor, Inc.

end. The only macro parameter required is the device descriptor parameter. Refer to the following example:

```
/*FH*****************************************************************/
/* Macro:       UART_A_INIT_DEFAULT                                 */
/*                                                                  */
/* Purpose:     Call Uart_A_Init with default parameters.           */
/*              Make a copy of this macro to customize defaults.    */
/*                                                                  */
/*EH*****************************************************************/
#define UART_A_INIT_DEFAULT(UARTPtr)                                \
    UART_A_Init(                                                    \
                UARTPtr,                                            \
                UART_A_DEFAULT_DIVIDER,                             \
                UART_A_DEFAULT_SIZE,                                \
                UART_A_DEFAULT_PARITY,                              \
                UART_A_DEFAULT_STOP_BITS,                           \
                UART_A_DEFAULT_RX_TRIG,                             \
                UART_A_DEFAULT_TX_TRIG,                             \
                UART_A_DEFAULT_RTS_INT,                             \
                UART_A_DEFAULT_DOZE,                                \
                UART_A_DEFAULT_FLOW,                                \
                UART_A_DEFAULT_UART_PINS,                           \
                UART_A_DEFAULT_OUTPUT_PINS                          \
               )
```

NOTE: *Default values for parameters such as word size, parity, and number of stop bits are supplied in the macro definition. These default values are defined in the corresponding module header file.*

3.3 Error Checking

All device driver library functions return a status of type ddErr_t, an enumeration defined in the global *errors.h* header file. If a function encounters no exceptional conditions during its operation, it will return a status of DD_ERR_NONE. In most cases a returned status other than DD_ERR_NONE indicates some type of error situation, and the resulting behavior of the function is undefined. An exception to this rule would be when a function returned information concerning an out-of-band condition such as a received break signal on a serial line.

Return codes are grouped by module type, so that the actual enumeration values for a given module are sequential. Return code symbols are prefixed by the module name. There is a small group of global return codes with the prefix “DD_”. Refer to **Appendix A Return Codes** for a complete listing of function return codes.

Freescale Semiconductor, Inc.

3.4 Memory-Mapped Peripheral Device Blocks

Table 3-1 shows the device handles (defined in *cfg\plibdefs.h*) for MMC2001 peripherals. All device register blocks are mapped to a C language structure as defined in the header file for the appropriate module.

Table 3-1 MMC2001 Peripheral Device Register Addresses

Identifier	Data Type	Device Handle
# define_PWS_INTC	pINTC_A_t	0x10000000
# define_PWS_TRM	pTRM_A_t	0x10001000
# define_PWS_KPP	pKPP_A_t	0x10003000
# define_PWS_EIM	pEIM_A_t	0x10004000
# define_PWS_PWM	pPWM_A_t	0x10005000
# define_PWS_PWM0	pPWM_A_t	0x10005000
# define_PWS_PWM1	pPWM_A_t	0x10005008
# define_PWS_PWM2	pPWM_A_t	0x10005010
# define_PWS_PWM3	pPWM_A_t	0x10005018
# define_PWS_PWM4	pPWM_A_t	0x10005020
# define_PWS_PWM5	pPWM_A_t	0x10005028
# define_PWS_EdgePort	pEdgePort_A_t	0x10007000
# define_PWS_ISPI	pISPI_A_t	0x10008000
# define_PWS_UART0	pUART_A_t	0x10009000
# define_PWS_UART1	pUART_A_t	0x1000A000
# define_PWS_LCD	pLCD_A_t	0x2C000000

3.5 Direct Register Access

All device driver library modules contain generic *GetRegister* and *SetRegister* routines for arbitrarily manipulating device registers. They are all written in a uniform way to make them easy to use between and among modules. However, one can access registers in an even more direct way without departing from C language syntax. The handle provided for any level 1 library routine refers to the base address of the device register block. Therefore, the handle may be used as a structure pointer to access device registers directly.

For example, the definition of the MMC2001 pulse width modulator (PWM) registers is:

```
typedef struct
   {
   volatile u2  PWMCR;      /* Control */
   u2  PWMPR;               /* Period  */
   u2  PWMWR;               /* Width   */
   volatile u2  PWMCTR;     /* Counter */
   }
PWM_A_t, *pPWM_A_t;
```


Freescale Semiconductor, Inc.

To obtain a handle to the first PWM channel, the following syntax may be used:

```
pPWM_A_t pwm0 = (pPWM_A_t)__PWS_PWM0;
```

The identifier __PWS_PWM0 is defined in the user-modifiable *cfg\plibdefs.h* file, and instantiates the address of the initial PWM channel as a manifest constant. The declaration above allocates storage for a pointer to a structure of type PWM_A_t, which reflects the definition of the PWM device registers. The pwm0 variable provides direct access to the PWM registers for this channel.

The following assignments set the PWM channel 0 period and width registers:

```
pwm0->PWMPR = 0x200;   /* set for 50% duty cycle */
pwm0->PWMWR = 0x100;
```

The code below enables the PWM to run with a clock divider of 256:

```
pwm0->PWMCR |= PWM_A_COUNTEN_MASK | PWM_A_DIV_256;
```

The following code disables the PWM channel 0 counter:

```
pwm0->PWMCR &= ~PWM_A_COUNTEN_MASK;
```

The identifiers PWM_A_COUNTENMASK and PWM_A_DIV_256 are defined, along with other useful register and field mask definitions, in the *inc\pwm_a.h* file. Analogous definitions for any given device driver library module can be found in the associated module header file.

3.6 Usage Tutorial

To use the M•CORE device driver library, write an application which incorporates calls to the library functions, compile the program and link with the API library (and any other files required to resolve external references), then download the program to a target system for execution. This tutorial makes use of the following tools:

- Diab Data C compiler
- SDS Single-Step debugger
- Motorola MMC2001 evaluation board (EVB)

Consider an application that successively writes a large number of characters from one UART (serial controller) on the MMC2001 EVB to the other. The UART must be initialized, the channels must be enabled for either transmit or receive, and there must be a mechanism for exchanging characters. The following code makes use of the UART API for these operations:

Freescale Semiconductor, Inc.

```
/* 1*/  /*  plibtut.c                                                  */
/* 2*/  /*  PLIB tutorial program                                       */
/* 3*/  /*                                                             */
/* 4*/  /*  This program assumes two UART_A channels with transmit and*/
/* 5*/  /*  receive lines appropriately connected.                     */
/* 6*/
/* 7*/  #include <stdlib.h>
/* 8*/  #include <stdio.h>
/* 9*/  #include <string.h>
/*10*/  #include "uart_a.h"
/*11*/
/*12*/  #define CLOCK_RATE      16380000    /* 16.38MHz clock */
/*13*/  #define SAMPLE_RATE     16          /* bit sampling constant */
/*14*/  #define MAX_RETRIES     50000       /* num of xmt/rcv retries */
/*15*/  #define STRESS_VALUE    100000      /* cnt in stress test */
/*16*/
/*17*/  void uart_stress_tests (void);
/*18*/  int main (void);
/*19*/
/*20*/  pUART_A_t uart0 = (pUART_A_t)__PWS_UART0;
/*21*/  pUART_A_t uart1 = (pUART_A_t)__PWS_UART1;
/*22*/  char format[]    = "%s\n";
/*23*/  char passed[]    = "... PASSED    ";
/*24*/  char failed[]    = "*** FAILED ***";
/*25*/  char succeeded[] = "... SUCCEEDED ";
/*26*/
/*27*/  void uart_stress_tests(void)
/*28*/  {
/*29*/      u4 i = 0, j, k;
/*30*/      u1 c1, c2;
/*31*/      ddErr_t status = DD_ERR_NONE;
/*32*/
/*33*/      printf ("\nUART_A Stress Tests:\n");
/*34*/
/*35*/      if (UART_A_INIT_DEFAULT (uart0) ||
/*36*/          UART_A_INIT_DEFAULT (uart1) != DD_ERR_NONE) {
/*37*/          printf ("*** Stress test initialization failed!\n");
/*38*/          return;
/*39*/      }
/*40*/
/*41*/      /* Enable transmit and receive */
/*42*/
/*43*/      if (UART_A_Enable (uart0, UART_A_TX) != DD_ERR_NONE ||
/*44*/          UART_A_Enable (uart1, UART_A_RX) != DD_ERR_NONE) {
/*45*/          printf ("*** Stress test initialization failed!\n");
/*46*/          return;
/*47*/      }
/*48*/
/*49*/      /* Write and read the data */
/*50*/
/*51*/      for (k = 1; k <= 5; k++) {
/*52*/
/*53*/          printf ("    %d character xmt/rcv test.......",
/*54*/                  STRESS_VALUE * k);
```


Freescale Semiconductor, Inc.

```
/*55*/
/*56*/          for (j = 0; j < STRESS_VALUE * k; j++) {
/*57*/              c1 = (u1)(rand() & 0xFF);
/*58*/              status = UART_A_ERR_DATA_PENDING;
/*59*/              for (i = 0;
/*60*/                   i < MAX_RETRIES &&
/*61*/                   status == UART_A_ERR_DATA_PENDING;
/*62*/                   i++)
/*63*/                  status = UART_A_Transmit (uart0, c1);
/*64*/              if (i == MAX_RETRIES || status != DD_ERR_NONE)
/*65*/                  break;
/*66*/              status = UART_A_ERR_DATA_PENDING;
/*67*/              for (i = 0;
/*68*/                   i < MAX_RETRIES &&
/*69*/                   status == UART_A_ERR_DATA_PENDING;
/*70*/                   i++)
/*71*/                  status = UART_A_Receive (uart1, &c2);
/*72*/              if (i == MAX_RETRIES || status != DD_ERR_NONE)
/*73*/                  break;
/*74*/              if (c2 != c1) {
/*75*/                  status = UART_A_ERR_RECEIVE_ERROR;
/*76*/                  break;
/*77*/              }
/*78*/          }
/*79*/          printf (format,
/*80*/                  (i < MAX_RETRIES && status == DD_ERR_NONE) ?
/*81*/                  passed : failed);
/*82*/      }
/*83*/  }
/*84*/
/*85*/
/*86*/  int main (void)
/*87*/  {
/*88*/      uart_stress_tests ();
/*89*/
/*90*/      printf ("\n");
/*91*/      exit (0);
/*92*/
/*93*/  }   /* end main */
```

In lines 20 and 21, the global variables uart0 and uart1 are assigned the base address of the register block for the MMC2001 UART0 and UART1. These variables represent the UART device handles and are always the first parameter in any UART API function. The base addresses are defined in the user-modifiable *plibdefs.h* file.

In lines 35 and 36, the default initialization macro is used to initialize both UART devices.

NOTE: *Only the device handle is required as a macro parameter; all other initialization arguments are defaulted. The default values may be customized by redefining them prior to macro invocation.*

Freescale Semiconductor, Inc.

In lines 43 and 44, UART0 is enabled as the transmitter and UART1 is enabled as the receiver.

In line 57, a random character is generated for UART0 to transmit to UART1. At line 63 the character is transmitted, and at line 71 the character is received. A check at line 74 insures that the character received is the same as the character sent. Up to 50,000 characters are transmitted this way.

The Diab compiler can compile and link this program all in one step. From the Single-Step debugger command window:

```
SingleStep> dcc -tRCE0ES -I..\inc -I..\cfg plibtut.c chario.o \
> -Wmplibtut.lnk -L..\lib -lplib -o plibtut.elf
```

In the command above, *dcc* is the front-end program for the Diab preprocessor, compiler, assembler, and linker. File *pstrike.lnk* is the linker control file which specifies RAM and ROM addresses and where various program sections (code, data, stack) are to reside in memory. The file *plibtut.c* contains the source code listed above and the command option -lplib indicates that the resulting object code is to be linked with the M•CORE device driver library. The executable is placed in a file called *plibtut.elf*.

The ELF file can be downloaded to the MMC2001 EVB through the SDS Single-Step debugger.

After downloading, the program may be run:

```
SingleStep> go

UART_A Stress Tests:
    100000 character xmt/rcv test.......... PASSED
    200000 character xmt/rcv test.......... PASSED
    300000 character xmt/rcv test.......... PASSED
    400000 character xmt/rcv test.......... PASSED
    500000 character xmt/rcv test.......... PASSED

go: info: P:\mcore\plib\uart_a\src\unit_test\plibtut.ou1 - stopped by
Breakpoint Exception
SingleStep>
```

Freescale Semiconductor, Inc.

3.7 Standard Data Type Definitions

All device driver library routines use common typedefs for scalar parameter typing. The typedefs are listed below, along with enumeration definitions which may also be common across device driver modules. These definitions can be found in the global include file *plib.h*.

```
/* Peripheral library standard scalar typedefs */
typedef enum      {FALSE, TRUE} bool;
typedef unsigned char           u1;
typedef signed char             s1;
typedef unsigned short int      u2;
typedef signed short int        s2;
typedef unsigned long int       u4;
typedef signed long int         s4;

typedef enum {
    DD_CMOS,            /* Pins are wired as CMOS output */
    DD_OPEN_DRAIN       /* Pins are wired as open-drain output */
} ddWiredOR_t;

typedef enum {
    DD_LOW,             /* Active LOW */
    DD_HIGH             /* Active HI */
} ddPinSense_t;
```

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Section 4 EdgePort_A Level 1

This section describes the functions of a EdgePort_A level 1 device driver.

4.1 Overview

The edge port module (EdgePort) and its accompanying device driver (EdgePort_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the EdgePort on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The edge port module has eight external interrupt pins. Each pin can be configured individually as a level-sensitive interrupt, an edge-detecting interrupt (rising edge, falling edge, or both), or a general-purpose I/O pin.

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the edge port module.

4.2 EdgePort_A Level 1 Device Driver Functions

The EdgePort_A level 1 device driver software provides the following functions:

- Gets any EdgePort register
- Sets any EdgePort register

4.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

4.3.1 Abstract Data Type Definitions

The following are EdgePort_A abstract data types.

4.3.1.1 EdgePort_A_t - EdgePort Register Block

The following structure definition corresponds to the EdgePort register block layout.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
typedef struct
{
    u2  EPPAR;                /* Edge Port Pin Assignment Register */
    u2  EPDDR;                /* Edge Port Data Direction Register */
    volatile u2  EPDR;        /* Edge Port Data Register */
    volatile u2  EPFR;        /* Edge Port Flag Register */
} EdgePort_A_t, *pEdgePort_A_t;
```

4.3.2 Enumerated Data Type Definitions

The following are EdgePort_A enumerated data types.

4.3.2.1 EdgePort_A_RegisterSwitch_t - EdgePort Register Switch

These values are used to select among EdgePort registers.

```
typedef enum {
   EdgePort_A_EPPAR_SWITCH, /* Select Edge Port Pin Assignment Register */
   EdgePort_A_EPDDR_SWITCH, /* Select Edge Port Data Direction Register */
    EdgePort_A_EPDR_SWITCH, /* Select Edge Port Data Register */
    EdgePort_A_EPFR_SWITCH  /* Select Edge Port Flag Register */
} EdgePort_A_RegisterSwitch_t;
```

4.4 API Definitions

The following paragraphs provide level 1 API definitions for an EdgePort_A device driver.

Freescale Semiconductor, Inc.

4.4.1 EdgePort_A_GetRegister

The following paragraphs describe the EdgePort_A_GetRegister function.

4.4.1.1 Description

EdgePort_A_GetRegister gets an EdgePort register. Call this function at any time.

4.4.1.2 Prototype

```
ddErr_t EdgePort_A_GetRegister (
                                pEdgePort_A_t                EdgePortPtr,
                                EdgePort_A_RegisterSwitch_t  EdgePort_A_RegisterSwitch,
                                u2                           *GetRegisterPtr
                                );
```

4.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
EdgePortPtr	pEdgePort_A_t	EdgePort base address associated with this driver	Any non-zero core processor data address
EdgePort_A_RegisterSwitch	EdgePort_A_RegisterSwitch_t	Selects among EdgePort registers	EdgePort_A_EPPAR_SWITCH EdgePort_A_EPDDR_SWITCH EdgePort_A_EPDR_SWITCH EdgePort_A_EPFR_SWITCH
GetRegisterPtr	u2 *	Result address for selected EdgePort register	Any non-zero core processor data address

4.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

4.4.1.5 Code Usage Example

```
/* Include the Edge Port version A header file */
#include “edgeport_a.h”
...
ddErr_t retval;
...

/* Create an Edge Port version A Device Handle */
pEdgePort_A_t edgeportptr = (pEdgePort_A_t) __PWS_EdgePort;

/* Variable to hold Edge Port Register data */
u2 EdgePort_Reg;
...

/* Get Register Edge Port version A */
retval = EdgePort_A_GetRegister(
                        edgeportptr,            /* EdgePort Handle */
                        EdgePort_A_EPDDR_SWITCH,/* Register Switch */
                        &EdgePort_Reg           /* Data Value     */
                        );

/* Perform an optional check on the EdgePort_A_GetRegister() return code
*/
if ( (retval != DD_ERR_NONE) )
{
      /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

4.4.2 EdgePort_A_SetRegister

The following paragraphs describe the EdgePort_A_SetRegister function.

4.4.2.1 Description

EdgePort_A_SetRegister sets an EdgePort register. Call this function at any time.

4.4.2.2 Prototype

```
ddErr_t EdgePort_A_SetRegister (
                                pEdgePort_A_t                EdgePortPtr,
                                EdgePort_A_RegisterSwitch_t  EdgePort_A_RegisterSwitch,
                                u2                           RegisterValue
                                );
```

4.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
EdgePortPtr	pEdgePort_A_t	EdgePort base address associated with this driver	Any non-zero core processor data address
EdgePort_A_RegisterSwitch	EdgePort_A_RegisterSwitch_t	Selects among EdgePort registers	EdgePort_A_EPPAR_SWITCH EdgePort_A_EPDDR_SWITCH EdgePort_A_EPDR_SWITCH EdgePort_A_EPFR_SWITCH
RegisterValue	u2	Data to copy into selected EdgePort register	Any 16-bit value

4.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

4.4.2.5 Code Usage Example

```
/* Include the Edge Port version A header file */
#include “edgeport_a.h”
...
ddErr_t retval;
...

/* Create an Edge Port version A Device Handle */
pEdgePort_A_t edgeportptr = (pEdgePort_A_t) __PWS_EdgePort;

/* Set Register Edge Port version A */
retval = EdgePort_A_SetRegister(
                        edgeportptr,            /* EdgePort Handle */
                        EdgePort_A_EPDDR_SWITCH,/* Register Switch */
                        (EPDDR_EPDD0_MASK +     /* Data Value */
                        EPDDR_EPDD1_MASK +
                        EPDDR_EPDD2_MASK +
                        EPDDR_EPDD3_MASK)
                        );

/* Perform an optional check on the EdgePort_A_SetRegister() return code
*/
if ( (retval != DD_ERR_NONE) )
{
      /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

Section 5 EIM_A Level 1

This section describes the functions of an EIM_A level 1 device driver.

5.1 Overview

The external interface module (EIM) and its accompanying device driver (EIM_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the EIM on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The EIM handles the interface to devices external to the MCU, including generation of chip selects for external peripherals and memory. It provides the following features:

- Four chip selects for external devices, each covering a range of 16 Mbytes
- Programmable wait-state generator for each chip select
- Selectable protection for each chip select
- Programmable data port size for each chip select
- Control for external/internal boot ROM device selection
- Bus watchdog counter for all bus cycles
- Programmable general output capability for unused chip-select outputs
- Show cycles to allow internal bus cycles to be monitored externally

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the EIM module.

5.2 EIM_A Level 1 Device Driver Functions

The EIM_A level 1 device driver software provides the following functions:

- Gets any EIM register
- Sets any EIM register

5.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3.1 Abstract Data Type Definitions

The following are EIM_A abstract data types.

5.3.1.1 EIM_A_t - EIM Register Block

The following structure definition corresponds to the EIM register block layout.

```
typedef struct
{
    u4  CS0CR;    /* CS0 Control Register */
    u4  CS1CR;    /* CS1 Control Register */
    u4  CS2CR;    /* CS2 Control Register */
    u4  CS3CR;    /* CS3 Control Register */
    u4  RESERVED0;
    u4  RESERVED1;
    u4  EIMCR;    /* EIM Configuration Register */
} EIM_A_t, *pEIM_A_t;
```

5.3.2 Enumerated Data Type Definitions

The following are EIM_A enumerated data types.

5.3.2.1 EIM_A_RegisterSwitch_t - EIM Register Switch

These values are used to select among EIM registers.

```
typedef enum {
    EIM_A_CS0CR_SWITCH,       /* Select CS0 Control Register */
    EIM_A_CS1CR_SWITCH,       /* Select CS1 Control Register */
    EIM_A_CS2CR_SWITCH,       /* Select CS2 Control Register */
    EIM_A_CS3CR_SWITCH,       /* Select CS3 Control Register */
    EIM_A_EIMCR_SWITCH        /* Select EIM Configuration Register */
} EIM_A_RegisterSwitch_t;
```

5.4 API Definitions

The following paragraphs provide level 1 API definitions for an EIM_A device driver.

Freescale Semiconductor, Inc.

5.4.1 EIM_A_GetRegister

The following paragraphs describe the EIM_A_GetRegister function.

5.4.1.1 Description

EIM_A_GetRegister gets an EIM register. Call this function at any time.

5.4.1.2 Prototype

```
ddErr_t EIM_A_GetRegister (
                    pEIM_A_t                    EIMPtr,
                    EIM_A_RegisterSwitch_t      EIM_A_RegisterSwitch,
                    u4                          *GetRegisterPtr
                    );
```

5.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
EIMPtr	pEIM_A_t	EIM base address associated with this driver	Any non-zero core processor data address
EIM_A_RegisterSwitch	EIM_A_RegisterSwitch_t	Selects among EIM registers	EIM_A_CS0CR_SWITCH EIM_A_CS1CR_SWITCH EIM_A_CS2CR_SWITCH EIM_A_CS3CR_SWITCH EIM_A_EIMCR_SWITCH
GetRegisterPtr	u4 *	Result address for selected EIM register	Any non-zero core processor data address

5.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

5.4.1.5 Code Usage Example

```
/* Include the EIM version A header file */
#include “eim_a.h”
...
ddErr_t retval;
...

/* Create an EIM version A Device Handle */
pEIM_A_t eimptr = (pEIM_A_t) __PWS_EIM;

/* Variable to hold EIM Register data */
u2 EIM_Reg;
...

/* Get Register EIM version A */
retval = EIM_A_GetRegister(
                           eimptr,              /* EIM Handle      */
                           EIM_A_CS0SR_SWITCH,  /* Register Switch */
                           &EIM_Reg             /* Data Value     */
                           );

/* Perform an optional check on the EIM_A_GetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

5.4.2 EIM_A_SetRegister

The following paragraphs describe the EIM_A_SetRegister function.

5.4.2.1 Description

EIM_A_SetRegister sets an EIM register. Call this function at any time.

5.4.2.2 Prototype

```
ddErr_t EIM_A_SetRegister (
                    pEIM_A_t                    EIMPtr,
                    EIM_A_RegisterSwitch_t      EIM_A_RegisterSwitch,
                    u4                          RegisterValue
                    );
```

5.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
EIMPtr	pEIM_A_t	EIM base address associated with this driver	Any non-zero core processor data address
EIM_A_RegisterSwitch	EIM_A_RegisterSwitch_t	Selects among EIM registers	EIM_A_CS0CR_SWITCH EIM_A_CS1CR_SWITCH EIM_A_CS2CR_SWITCH EIM_A_CS3CR_SWITCH EIM_A_EIMCR_SWITCH
RegisterValue	u4	Data to copy into selected EIM register	Any 32-bit value

5.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

5.4.2.5 Code Usage Example

```
/* Include the EIM version A header file */
#include “eim_a.h”
...
ddErr_t retval;
...

/* Create an EIM version A Device Handle */
pEIM_A_t eimptr = (pEIM_A_t) __PWS_EIM;

/* Set Register EIM version A */
retval = EIM_A_SetRegister(
                           eimptr,              /* EIM Handle */
                           EIM_A_CS0SR_SWITCH,  /* Register Switch */
                           (CSXCR_CSEN_MASK +   /* Data Value     */
                           CSXCR_WP_MASK)
                           );

/* Perform an optional check on the EIM_A_SetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

Section 6 INTC_A Level 1

This section describes the functions of an INTC_A level 1 device driver.

6.1 Overview

The interrupt controller module (INTC) and its accompanying device driver (INTC_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the INTC on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The interrupt controller performs the following functions:

- Collects requests from multiple interrupt sources and provides an interface to M•CORE interrupt control lines
- Supports up to 32 interrupt sources
- Supports two categories of interrupts: normal and fast
- Indicates pending interrupt sources via a register
- Independently enables/disables any interrupt source
- Selects normal or fast interrupt request for any interrupt source
- Provides a mechanism for software to schedule an interrupt

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the interrupt controller.

6.2 INTC_A Level 1 Device Driver Functions

INTC_A level 1 device driver software provides the following functions:

- Initializes the vector base register (VBR) and establishes dispatch routines for normal and fast interrupts
- Enables and disables selected normal and fast interrupts.
- Sets up user-specified interrupt service functions (ISF) for selected devices and interrupt types
- Sets up user-specified interrupt signaling functions (SSF) for selected devices and interrupt types
- Gets and sets selected interrupt controller registers. This permits programming the interrupt controller in a method not supported by the other INTC_A level 1 device driver functions.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

6.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

6.3.1 Abstract Data Type Definitions

The following are INTC_A abstract data types.

6.3.1.1 INTC_A_t - Interrupt Controller Register Definition

The following structure definition corresponds to the INTC controller register block layout.

```
typedef struct
{
    volatile u4  INTSRC;      /* Interrupt Source Register */
    u4  NIER;                 /* Normal Interrupt Enable Register */
    u4  FIER;                 /* Fast Interrupt Enable Register */
    volatile u4  NIPND;       /* Normal Interrupt Pending Register */
    volatile u4  FIPND;       /* Fast Interrupt Pending Register */
} INTC_A_t, *pINTC_A_t;
```

6.3.1.2 srvTbl_t, sigTbl_t, intTbl_t - Interrupt Controller Function Table Entry Definitions

The following structure definition corresponds to the INTC controller function table entry block layout.

```
typedef struct
{                             /* Interrupt Service Function entry */
    ddErr_t (*func)(void *param1, void *param2); /* function pointer */
    void *param1;             /* first parameter */
    void *param2;             /* second parameter */
    u4 mask;                  /* interrupt enable mask */
} srvTbl_t;

typedef struct
{                             /* Signaling Service Function entry */
   void (*func)(ddErr_t status, void *param1, void *param2);/* func ptr */
    void *param1;             /* first parameter */
    void *param2;             /* second parameter */
} sigTbl_t;

typedef struct
{
    srvTbl_t    ISF[INTSRC_MAX];        /* interrupt service table */
    sigTbl_t    SSF[INTSRC_MAX];        /* signaling service table */
} intTbl_t;
```


Freescale Semiconductor, Inc.

6.3.2 Enumerated Data Type Definitions

The following are INTC_A enumerated data types.

6.3.2.1 INTC_A_RegisterSwitch_t - INTC Register Selection

These values are used to get and set INTC Register selections.

```
typedef enum
{                                   /* Get and set register selections */
    INTC_A_INTSRC_SWITCH,           /* Select INTSRC */
    INTC_A_NIER_SWITCH,             /* Select NIER */
    INTC_A_FIER_SWITCH,             /* Select FIER */
    INTC_A_NIPND_SWITCH,            /* Select NIPND */
    INTC_A_FIPND_SWITCH,            /* Select FIPND */
} INTC_A_RegisterSwitch_t;
```

6.4 API Definitions

The following paragraphs provide level 1 API definitions for an INTC_A device driver.

Freescale Semiconductor, Inc.

6.4.1 INTC_A_Init

The following paragraphs describe the INTC_A_Init function.

6.4.1.1 Description

INTC_A_Init initializes the machine vector base register and interrupt service and signaling function table pointer for operation.

NOTE: *The pointer to the interrupt service and signaling function table is stored in the first available vector entry reserved for vectored interrupt use. This, along with the updating of the fast and normal interrupt vector addresses, implies that the interrupt vector table is stored in RAM.*

This function should be called once during the system software initialization. This function should not be called when system interrupts are enabled.

NOTE: *It is the caller's responsibility to allocate memory for the IntTbl_t structure prior to calling this function. The IntTbl_t structure must exist for the duration of the application program. This is done by appending "static" before the memory declaration.*

6.4.1.2 Prototype

```
ddErr_t INTC_A_Init    (
                       pINTC_A_t          INTCPtr,
                       void               *VBA,
                       intTbl_t           *TblPtr
                       );
```

6.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
VBA	void *	Vector base address to be written to machine vector base register	Any core processor data address
TblPtr	intTbl_t *	Pointer to interrupt service and signaling function table	Any non-zero core processor data address

Freescale Semiconductor, Inc.

6.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL

6.4.1.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
static intTbl_t funcs; /* interrupt and signaling service tables */
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/* Initialize interrupt controller version A */
retval = INTC_A_Init(
                  intctlr,                /* INTC Device Handle   */
                  (void *)0x30000000,     /* Vector Base Address */
                  &funcs                  /* Function Table Addr */
                  );

/* Perform an optional check on the INTC_A_Init() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

6.4.2 INTC_A_IntEnable

The following paragraphs describe the INTC_A_IntEnable function.

6.4.2.1 Description

INTC_A_IntEnable enables device and system interrupts for devices specified in the interrupt mask. This function may be used after a call to INTC_A_Init.

6.4.2.2 Prototype

```
ddErr_t INTC_A_IntEnable    (
                            pINTC_A_t       INTCPtr,
                            u4              IntMask,
                            bool            FastInt,
                            bool            PSR
                            );
```

6.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
IntMask	u4	Mask that specifies the interrupt sources to enable.	Any 32-bit value
FastInt	bool	Indicates whether IntMask and PSR refer to fast or normal interrupt sources	TRUE = enable sources as fast interrupts FALSE = enable sources as normal interrupts
PSR	bool	Indicates whether to enable system exceptions and either normal or fast interrupts, depending on FastInt	TRUE = enable system exceptions and interrupts FALSE = do not enable system exceptions and interrupts

6.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

6.4.2.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/* Enable interrupt controller version A */
retval = INTC_A_IntEnable(
                        intctlr,                /* INTC Device Handle */
                        INTSRC_TOD_ALARM_MASK,  /* Time-of-Day Alarm */
                        TRUE,                   /* Fast Interrupt     */
                        TRUE                    /* Enable in PSR      */
                        );

/* Perform an optional check on the INTC_A_IntEnable() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.4.3 INTC_A_IntDisable

The following paragraphs describe the INTC_A_IntDisable function.

6.4.3.1 Description

INTC_A_IntDisable disables device and system interrupts for devices specified in the interrupt mask. This function may be used after a call to INTC_A_Init.

6.4.3.2 Prototype

```
ddErr_t INTC_A_IntDisable   (
                            pINTC_A_t       INTCPtr,
                            u4              IntMask,
                            bool            FastInt,
                            bool            PSR
                            );
```

6.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
IntMask	u4	Mask that specifies the interrupt sources to disable	Any 32-bit value
FastInt	bool	Indicates whether IntMask and PSR refer to fast or normal interrupt sources	TRUE = disable fast interrupt sources FALSE = disable normal interrupt sources
PSR	bool	Indicates whether to disable system exceptions and either normal or fast interrupts, depending on FastInt	TRUE = disable system exceptions and interrupts FALSE = do not disable system exceptions and interrupts

6.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

6.4.3.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/* Disable interrupt controller version A */
retval = INTC_A_IntDisable(
                        intctlr,                /* INTC Device Handle */
                        INTSRC_TOD_ALARM_MASK,  /* Time-of-Day Alarm */
                        TRUE,                   /* Fast Interrupt     */
                        TRUE                    /* Disable in PSR     */
                        );

/* Perform an optional check on the INTC_A_IntDisable() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.4.4 INTC_A_SetISF

The following paragraphs describe the INTC_A_SetISF function.

6.4.4.1 Description

INTC_A_SetISF establishes an interrupt service function for a specified interrupt source. This function may be used after a call to INTC_A_Init.

6.4.4.2 Prototype

```
ddErr_t INTC_A_SetISF(
                        pINTC_A_t           INTCPtr,
                        u2                  IntSource,
                        u4                  IntMask
                        ddErr_t             (*ISFAddr)(void *param1, void *param2),
                        void                *ISFParam1,
                        void                *ISFParam2
                        );
```

6.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
IntSource	u2	Bit number corresponding to source in controller register	0 to 31
IntMask	u4	Bit mask indicating lower priority interrupts	Any 32-bit value
ISFAddr	ddErr_t (*) (void *, void *)	Pointer to interrupt service function	Any non-zero core processor text address
ISFParam1	void *	First parameter to interrupt service function	Any core processor data address
ISFParam2	void *	Second parameter to interrupt service function	Any core processor data address

Freescale Semiconductor, Inc.

6.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
INTC_A_ERR_INVALID_INTERRUPT_SOURCE	Source designator is invalid

6.4.4.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Set up interrupt service function for buffered UART receiver */
retval = INTC_A_SetISF (
            intctlr,                      /* INTC Device Handle */
            INTSRC_UART0_RECEIVE_BITNO,   /* UART0 Interrupt Source */
            INTSRC_UART0_RECEIVE_MASK,    /* UART0 Interrupt Mask */
            /* Pointer to Buffered UART Interrupt Service Function*/
            (ddErr_t(*)(void *, void *))BRT_A_RX_ISF,
            (void *)brtptr,               /* ISF Parameter */
            NULL                          /* ISF Dummy Parameter */
            );

/* Perform an optional check on the INTC_A_SetISF() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

6.4.5 INTC_A_SetSSF

The following paragraphs describe the INTC_A_SetSSF function.

6.4.5.1 Description

INTC_A_SetSSF establishes signaling service function for a specified interrupt source. This function may be used after a call to INTC_A_Init.

6.4.5.2 Prototype

```
ddErr_t INTC_A_SetSSF    (
                         pINTC_A_t    INTCPtr,
                         u2           IntSource,
                         void         (*SSFAddr)(ddErr_t, void *param1, void *param2),
                         void         *SSFParam1,
                         void         *SSFParam2
                         );
```

6.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
IntSource	u2	Bit number corresponding to source in controller register	0 to 31
SSFAddr	void (*) (ddErr_t, void *, void *)	Pointer to signaling service function	Any non-zero core processor text address
SSFParam1	void *	First parameter to interrupt service function	Any core processor data address
SSFParam2	void *	Second parameter to interrupt service function	Any core processor data address

6.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
INTC_A_ERR_INVALID_INTERRUPT_SOURCE	Source designator is invalid

Freescale Semiconductor, Inc.

6.4.5.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
volatile bool charrdy;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

static void rxsig (void)    /* UART receiver signaling function */
{
    charrdy = TRUE;
}

/* Set up signaling service function for UART receiver */
retval = INTC_A_SetSSF (
            intctlr,                        /* INTC Device Handle */
            INTSRC_UART0_RECEIVE_BITNO,     /* UART0 Interrupt Source */
            /* Pointer to Buffered UART Interrupt Service Function*/
            (void(*)(ddErr_t, void *, void *))rxsig,
            NULL,                           /* ISF Dummy Parameter */
            NULL                            /* ISF Dummy Parameter */
            );

/* Perform an optional check on the INTC_A_SetSSF() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

6.4.6 INTC_A_GetRegister

The following paragraphs describe the INTC_A_GetRegister function.

6.4.6.1 Description

INTC_A_GetRegister gets an interrupt controller register. Call this function at any time.

6.4.6.2 Prototype

```
ddErr_t INTC_A_GetRegister   (
                             pINTC_A_t                INTCPtr,
                             INTC_A_RegisterSwitch    INTCRegisterSwitch,
                             u4                       *GetRegisterPtr
                             );
```

6.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
INTCRegisterSwitch	INTC_A_RegisterSwitch_t	Selects among interrupt controller registers	INTC_A_INTSRC_SWITCH INTC_A_NIER_SWITCH INTC_A_FIER_SWITCH INTC_A_NIPND_SWITCH INTC_A_FIPND_SWITCH
GetRegisterPtr	u4 *	Result address for selected interrupt controller register	Any non-zero core processor data address

6.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

6.4.6.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
u4 intsrc;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/*Get interrupt source register value for interrupt controller version A*/
retval = INTC_A_GetRegister (
                        intctlr,                /* INTC Device Handle */
                        INTC_A_INTSRC_SWITCH,   /* Register Selector */
                        &intsrc                 /* Returned Value     */
                        );

/* Perform an optional check on the INTC_A_GetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

6.4.7 INTC_A_SetRegister

The following paragraphs describe the INTC_A_SetRegister function.

6.4.7.1 Description

INTC_A_SetRegister sets an interrupt controller register. Call this function at any time.

6.4.7.2 Prototype

```
ddErr_t INTC_A_SetRegister   (
                             pINTC_A_t                INTCPtr,
                             INTC_A_RegisterSwitch    INTCRegisterSwitch,
                             u4                       RegisterValue
                             );
```

6.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
INTCPtr	pINTC_A_t	INTC base address associated with this driver	Any non-zero core processor data address
INTCRegisterSwitch	INTC_A_RegisterSwitch_t	Selects among interrupt controller registers	INTC_A_NIER_SWITCH INTC_A_FIER_SWITCH
RegisterValue	u4	Data to copy into selected interrupt controller register	Any 32-bit value

6.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

6.4.7.5 Code Usage Example

```
/* Include the interrupt controller version A header file */
#include “intc_a.h”
...
ddErr_t retval;
...

/* Create an interrupt controller version A device handle */
pINTC_A_t intctlr = (pINTC_A_t)__PWS_INTCTLR;

/* Enable normal interrupts for PWM 0 channel in interrupt controller
version A */
retval = INTC_A_SetRegister (
            brtptr,                             /* INTC Device Handle*/
            INTC_A_NIER_SWITCH,                 /* Register Selector */
            brtptr->NIER | INTSRC_PWM0_MASK     /* Value to Set */
            );

/* Perform an optional check on the INTC_A_SetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Section 7 ISPI_A Level 1

This section describes the functions of an ISPI_A level 1 device driver.

7.1 Overview

The interval mode serial peripheral interface module (ISPI) and its accompanying device driver (ISPI_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the ISPI on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The ISPI provides a high-speed synchronous serial interface to communicate to external devices such as A/D converters and non-volatile RAMs. The ISPI provides the control and clock for data transfers and can be configured as either a master or a slave device. In addition, the ISPI includes a timer that delays the initiation of a serial transfer for a programmable period.

The ISPI provides three operating modes:

- Manual mode — A traditional SPI master operation mode.
- Interval mode — Similar to manual mode, except that it includes a programmable timer to support timed transfers.
- Slave mode — The ISPI operates as a traditional slave SPI; the clock becomes an input, and the transfer is controlled entirely by the external master clock.

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the ISPI.

7.2 ISPI_A Level 1 Device Driver Functions

The ISPI_A level 1 device driver software provides the following functions:

- Initializes, enables, and disables the ISPI
- Transmits and receives ISPI data
- Gets the ISPI interrupt status
- Drives the ISPI general-purpose output pin
- Enables and disables ISPI loopback mode
- Clears ISPI overrun conditions
- Gets and sets all ISPI registers

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

7.3.1 Abstract Data Type Definitions

The following are ISPI_A abstract data types.

7.3.1.1 ISPI_A_t - ISPI Register Definitions

The following structure definition corresponds to the ISPI register block layout.

```
typedef struct
{
    u2  SPDR;           /* ISPI Send/Receive Data Register */
    u2  SPCR;           /* ISPI Control Register */
    u2  SPICR;          /* ISPI Interval Control Register */
    u2  SPSR;           /* ISPI Status Register */
    u2  reserved[2044];   /* reserved */
} ISPI_A_t, *pISPI_A_t;
```

7.3.2 Enumerated Data Type Definitions

The following are ISPI_A enumerated data types.

7.3.2.1 ISPI_A_BaudRate_t - ISPI Baud Rate Type

These values are used to set the baud rate field in the ISPI control register (SPCR).

```
typedef enum {
    ISPI_A_BAUD_RATE_0,        /* System Clock Divided by 8    */
    ISPI_A_BAUD_RATE_1,        /* System Clock Divided by 16   */
    ISPI_A_BAUD_RATE_2,        /* System Clock Divided by 32   */
    ISPI_A_BAUD_RATE_3,        /* System Clock Divided by 64   */
    ISPI_A_BAUD_RATE_4,        /* System Clock Divided by 128  */
    ISPI_A_BAUD_RATE_5,        /* System Clock Divided by 256  */
    ISPI_A_BAUD_RATE_6,        /* System Clock Divided by 512  */
    ISPI_A_BAUD_RATE_7         /* System Clock Divided by 1024 */
} ISPI_A_BaudRate_t;
```

Freescale Semiconductor, Inc.

7.3.2.2 DD_PinSense_t - Device Driver Pin Sense Type

These values are used to control device driver pin sense.

```
typedef enum {
LOW                                          /* Active Low */
HIGH                                         /* Active High */
} DD_PinSense_t;
```

7.3.2.3 ISPI_A_OutputDrive_t - ISPI Output Drive Type

These values are used to set the drive type field in the SPCR.

```
typedef enum {
      ISPI_TOTEM_POLE,              /* Outputs are totem-pole */
      ISPI_OPEN_DRAIN               /* Outputs are open-drain */
} ISPI_A_OutputDrive_t;
```

7.3.2.4 ISPI_A_ClockCount_t - ISPI Clock Count Selection

These values are used to set the clock count field in the SPCR.

```
typedef enum {
ISPI_A_CLOCK_COUNT_DISABLE,         /* Disable ISPI    */
ISPI_A_CLOCK_COUNT_2,               /* 2-bit Transfer  */
ISPI_A_CLOCK_COUNT_3,               /* 3-bit Transfer  */
ISPI_A_CLOCK_COUNT_4,               /* 4-bit Transfer  */
ISPI_A_CLOCK_COUNT_5,               /* 5-bit Transfer  */
ISPI_A_CLOCK_COUNT_6,               /* 6-bit Transfer  */
ISPI_A_CLOCK_COUNT_7,               /* 7-bit Transfer  */
ISPI_A_CLOCK_COUNT_8,               /* 8-bit Transfer  */
ISPI_A_CLOCK_COUNT_9,               /* 9-bit Transfer  */
ISPI_A_CLOCK_COUNT_10,              /* 10-bit Transfer */
ISPI_A_CLOCK_COUNT_11,              /* 11-bit Transfer */
ISPI_A_CLOCK_COUNT_12,              /* 12-bit Transfer */
ISPI_A_CLOCK_COUNT_13,              /* 13-bit Transfer */
ISPI_A_CLOCK_COUNT_14,              /* 14-bit Transfer */
ISPI_A_CLOCK_COUNT_15,              /* 15-bit Transfer */
ISPI_A_CLOCK_COUNT_16               /* 16-bit Transfer */
} ISPI_A_ClockCount_t;
```

7.3.2.5 ISPI_A_RegisterSwitch_t - ISPI Register Selection

These values are used by the ISPI_A_GetRegister function to determine which ISPI register should be read.

```
typedef enum {
      ISPI_A_SPCR_SWITCH,                           /* Select SPCR   */
      ISPI_A_SPICR_SWITCH,                          /* Select SPICR  */
      ISPI_A_SPSR_SWiTCH,                           /* Select SPSR   */
      ISPI_A_SPDR_SWITCH                            /* Select SPDR   */
} ISPI_A_RegisterSwitch_t;
```


Freescale Semiconductor, Inc.

7.4 API Definitions

The following paragraphs provide level 1 API definitions for an ISPI_A device driver.

7.4.1 ISPI_A_Init

The following paragraphs describe the ISPI_A_Init function.

7.4.1.1 Description

ISPI_A_Init initializes the ISPI for operation. This function should be called once during the system software initialization. For the ISPI to be fully functional, a polling or interrupt-driven mechanism must be used. When the ISPI is enabled, this function should not be called; when the ISPI is disabled, this function can be called.

7.4.1.2 Prototype

```
ddErr_t ISPI_A_Init     (
                        pISPI_A_t                   ISPIPtr,
                        ISPI_A_BaudRate_t           BaudRate,
                        bool                        DozeModeResponse,
                        ddPinSense_t                ISPI_PinSense,
                        ISPI_A_OutputDrive_t        DriveType,
                        bool                        InterruptRequestEnable,
                        bool                        OppositeClockPhase,
                        bool                        InvertedClockPolarity
                        );
```

Freescale Semiconductor, Inc.

7.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
BaudRate	ISPI_A_BaudRate_t	Determines the baud rate for the ISPI bit clock	ISPI_A_BAUD_RATE_0 through ISPI_A_BAUD_RATE_7
DozeModeResponse	bool	Determines whether ISPI is affected by CPU doze instruction	TRUE = doze when CPU dozes FALSE = do not doze when CPU dozes
ISPIPinSense	ddPinSense_t	Controls the sense of the SPI_EN pin	HIGH = SPI_EN pin is active high LOW = SPI_EN pin is active low
DriveType	ISPI_A_OutputDrive_t	Controls the configuration of the SPI_CLK, SPI_EN, and SPI_MOSI output buffers in master mode	TOTEMPOLE = outputs are totem-pole OPENDRAIN = outputs are open-drain
InterruptRequestEnable	bool	Controls the interrupt request output signal	TRUE = interrupts are enabled FALSE = interrupts are disabled
OppositeClockPhase	bool	Controls the phase shift of the SPI_CLK	TRUE = opposite phase FALSE = normal phase
InvertedClockPolarity	bool	Controls the polarity of the SPI_CLK	TRUE = inverted polarity FALSE = normal polarity

7.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_BAUD_RATE	Baud rate is out of range
DD_ERR_PIN_SENSE	Pin sense is not valid
ISPI_A_INVALID_DRIVE_TYPE	Drive type invalid

Freescale Semiconductor, Inc.

7.4.1.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

/* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* Init ISPI version A */
    retval = ISPI_A_Init(
                        ispiptr,                /* ISPI Device Handle */
                        ISPI_A_BAUD_RATE_7,     /* Baud Rate          */
                        FALSE,                  /* Doze Mode          */
                        DD_LOW,                 /* Pin Sense          */
                        ISPI_TOTEM_POLE,        /* Drive Type         */
                        TRUE,                   /* Interrupt Enable   */
                        FALSE,                  /* Clock Phase        */
                        FALSE                   /* Clock Polarity     */
                        );

    /* Perform an optional check on the ISPI_A_Init() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.2 ISPI_A_ManualEnable

The following paragraphs describe the ISPI_A_ManualEnable function.

7.4.2.1 Description

ISPI_A_ManualEnable enables the ISPI in manual mode. This function may be used after a call to ISPI_A_Init or ISPI_A_Disable.

7.4.2.2 Prototype

```
ddErr_t ISPI_A_ManualEnable  (
                              pISPI_A_t            ISPIPtr,
                              ISPI_A_ClockCount_t  ClockCount
                              );
```

7.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ClockCount	ISPI_A_ClockCount_t	Selects the length of the ISPI transfer and controls the justification of data	ISPI_A_CLOCK_COUNT_DISABLE, ISPI_A_CLOCK_COUNT_2, through ISPI_A_CLOCK_COUNT_16

7.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
ISPI_A_ERR_INVALID_CLOCK_COUNT	Clock count invalid

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.2.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

  /* Create an ISPI version A Device Handle */
  pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

  /* Manual Enable ISPI version A */
  retval = ISPI_A_ManualEnable(
                        ispiptr,             /* ISPI Device Handle */
                        ISPI_A_CLOCK_COUNT_8 /* Clock Count */
                        );

  /* Perform an optional check on the ISPI_A_ManualEnable() return code */
  if ( (retval != DD_ERR_NONE) )
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

7.4.3 ISPI_A_IntervalEnable

The following paragraphs describe the ISPI_A_IntervalEnable function.

7.4.3.1 Description

ISPI_A_IntervalEnable enables the ISPI in interval mode. This function may be used after a call to ISPI_A_Init or ISPI_A_Disable.

7.4.3.2 Prototype

```
ddErr_t ISPI_A_IntervalEnable  (
                               pISPI_A_t                ISPIPtr,
                               ISPI_A_ClockCount_t      ClockCount,
                               u2                       IntervalCount
                               );
```

7.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ClockCount	ISPI_A_Clock Count_t	Selects the length of the ISPI transfer and controls the justification of data	ISPI_A_CLOCK_COUNT_DISABLE, ISPI_A_CLOCK_COUNT_2, through ISPI_A_CLOCK_COUNT_16
IntervalCount	u2	Selects the length of the ISPI interval timer period	0 through 0x1fff (13 bits)

7.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_CLOCK_COUNT	Clock count invalid
ISPI_A_ ERR_INVALID_INTERVAL_COUNT	Interval count invalid

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.3.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

  /* Create an ISPI version A Device Handle */
  pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

  /* Interval Enable ISPI version A */
  retval = ISPI_A_IntervalEnable(
                  ispiptr,                /* ISPI Device Handle */
                  ISPI_A_CLOCK_COUNT_8,   /* Clock Count */
                  0x1fff                  /* max Interval Count value */
                  );

/* Perform an optional check on the ISPI_A_IntervalEnable() return code */
  if ( (retval != DD_ERR_NONE) )
  {
     /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

7.4.4 ISPI_A_SlaveEnable

The following paragraphs describe the ISPI_A_SlaveEnable function.

7.4.4.1 Description

ISPI_A_SlaveEnable enables the ISPI in slave mode. This function may be used after a call to ISPI_A_Init or ISPI_A_Disable.

7.4.4.2 Prototype

```
ddErr_t ISPI_A_SlaveEnable    (
                              pISPI_A_t              ISPIPtr,
                              ISPI_A_ClockCount_t    ClockCount
                              );
```

7.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ClockCount	ISPI_A_ClockCount_t	Selects the length of the ISPI transfer and controls the justification of data	ISPI_A_CLOCK_COUNT_DISABLE, ISPI_A_CLOCK_COUNT_2, through ISPI_A_CLOCK_COUNT_16

7.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
ISPI_A_ERR_INVALID_CLOCK_COUNT	Clock count invalid

Freescale Semiconductor, Inc.

7.4.4.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

  /* Slave Enable ISPI version A */
  retval = ISPI_A_SlaveEnable(
                        ispiptr,                /* ISPI Device Handle */
                        ISPI_A_CLOCK_COUNT_8    /* Clock Count */
                        );

   /* Perform an optional check on the ISPI_A_SlaveEnable() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.5 ISPI_A_Disable

The following paragraphs describe the ISPI_A_Disable function.

7.4.5.1 Description

ISPI_A_Disable disables the ISPI from master, interval, or slave mode. This function may be used after a call to ISPI_A_ManualEnable, ISPI_A_IntervalEnable, or ISPI_A_SlaveEnable.

7.4.5.2 Prototype

```
ddErr_t ISPI_A_Disable      (
                            pISPI_A_t          ISPIPtr
                            );
```

7.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address

7.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.5.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* Disable ISPI version A */
    retval = ISPI_A_Disable(
                            ispiptr     /* ISPI Device Handle */
                            );

    /* Perform an optional check on the ISPI_A_Disable() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.6 ISPI_A_Transmit

The following paragraphs describe the ISPI_A_Transmit function.

7.4.6.1 Description

ISPI_A_Transmit enables the ISPI to transmit data. This function may be used after a call to ISPI_ManualEnable, ISPI_A_IntervalEnable, ISPI_A_SlaveEnable, ISPI_A_Receive, or ISPI_A_Transmit.

7.4.6.2 Prototype

```
ddErr_t ISPI_A_Transmit     (
                            pISPI_A_t       ISPIPtr,
                            u2              TransmitData
                            );
```

7.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
TransmitData	u2	Data bits to be transmitted to external device	Two to 16 bits of information as specified in the SPCR clock count field

7.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.6.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* Transmit ISPI version A */
    retval = ISPI_A_Transmit(
                             ispiptr,    /* ISPI Device Handle */
                             0xff        /* TransmitData */
                             );

    /* Perform an optional check on the ISPI_A_Transmit() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.7 ISPI_A_Receive

The following paragraphs describe the ISPI_A_Receive function.

7.4.7.1 Description

The ISPI_A_Receive function enables the ISPI to receive data. This interrupt service function must be installed in the interrupt dispatch function or used after an ISPI interrupt status has occurred.

7.4.7.2 Prototype

```
ddErr_t ISPI_A_InterruptReceive     (
                                    pISPI_A_t           ISPIPtr,
                                    u2                  *ReceiveDataPtr
                                    );
```

7.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ReceiveDataPtr	u2 *	Address of data bits received from the shift register	Any non-zero core processor data address

7.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL
DD_ERR_NO_INTERRUPT	Interrupt service request (SPSR bit 14) is not present
ISPI_A_ERR_OVERRUN	Overrun condition (SPSR bit 15) is set
ISPI_A_ERR_OVERRUN_AND_NO_INTERRUPT	No interrupt request and overrun detected

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4.7.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
u2 receive_data;
...

      /* Create an ISPI version A Device Handle */
      pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

      /* Interrupt Receive ISPI version A */
      retval = ISPI_A_InterruptReceive(
                              ispiptr,          /* ISPI Device Handle */
                              &receive_data     /* Pointer for data */
                              );

/*Perform an optional check on the ISPI_A_InterruptReceive() return code*/
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.8 ISPI_A_GPControl

The following paragraphs describe the ISPI_A_GPControl function.

7.4.8.1 Description

The ISPI_A_GPControl function drives the ISPI general-purpose output pin high or low. Call this function at any time.

7.4.8.2 Prototype

```
ddErr_t ISPI_A_GPControl    (
                            pISPI_A_t          ISPIPtr,
                            ddPinsense_t       GPControl
                            );
```

7.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
GPControl	ddPinsense_t	Controls the data on the SPI_GP pin	High = SPI_GP pin is driven high Low = SPI_GP pin is driven low

7.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4.8.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* GP Control ISPI version A */
    retval = ISPI_A_GPControl(
                              ispiptr, /* ISPI Device Handle */
                              DD_LOW   /* SPI_GP pin is driven low */
                              );

    /* Perform an optional check on the ISPI_A_GPControl() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.9 ISPI_A_Loopback

The following paragraphs describe the ISPI_A_Loopback function.

7.4.9.1 Description

The ISPI_A_Loopback function enables or disables ISPI loopback. Call this function at any time.

7.4.9.2 Prototype

```
ddErr_t ISPI_A_Loopback     (
                            pISPI_A_t       ISPIPtr,
                            bool            LoopbackEnable
                            );
```

7.4.9.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
LoopbackEnable	bool	Enable/disable ISPI Loopback	TRUE = Enable ISPI Loopback FALSE = Disable ISPI Loopback

7.4.9.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4.9.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* Loopback ISPI version A */
    retval = ISPI_A_Loopback(
                             ispiptr, /* ISPI Device Handle */
                             TRUE /* Enable ISPI Loopback */
                             );

    /* Perform an optional check on the ISPI_A_Loopback() return code */
    if ( (retval != DD_ERR_NONE) )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

7.4.10 ISPI_A_ClearOverrun

The following paragraphs describe the ISPI_A_ClearOverrun function.

7.4.10.1 Description

The ISPI_A_ClearOverrun function clears an ISPI overrun condition. Call this function at any time.

7.4.10.2 Prototype

```
ddErr_t ISPI_A_ClearOverrun   (
                              pISPI_A_t          ISPIPtr
                              );
```

7.4.10.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address

7.4.10.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.10.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
u2 receive_data;
...

    /* Create an ISPI version A Device Handle */
    pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

    /* Interrupt Receive ISPI version A */
    retval = ISPI_A_InterruptReceive(
                            ispiptr,          /* ISPI Device Handle */
                            &receive_data     /* Pointer for data   */
                            );

    /* Perform check on return code   */
    if (retval == ISPI_ERR_OVERRUN)
    {
        /* ClearOverrun ISPI version A */
        retval = ISPI_A_ClearOverrun(
                                ispiptr     /* ISPI Device Handle */
                                );

        /* Perform check on return code */
        if ( (retval != DD_ERR_NONE) )
        {
             /* do application-specific error status reporting */
        }
    }
```


Freescale Semiconductor, Inc.

7.4.11 ISPI_A_GetInterruptStatus

The following paragraphs describe the ISPI_A_GetInterruptStatus function.

7.4.11.1 Description

The ISPI_A_GetInterruptStatus function gets the ISPI interrupt status. Call this function at any time.

7.4.11.2 Prototype

```
ddErr_t ISPI_A_GetInterruptStatus    (
                                     pISPI_A_t        ISPIPtr,
                                     bool             *ResultPtr
                                     );
```

7.4.11.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ResultPtr	bool *	Address of the result value	Any non-zero core processor data address

7.4.11.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.11.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

/* Create an ISPI version A Device Handle */
pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

/* Variable to hold ISPI Interrupt Status */
bool ISPI_IntStat = FALSE;
...

/* Create a loop to poll the ISPI Interrupt status */
while (ISPI_IntStat == FALSE)
{
    /* Get Register ISPI version A */
    retval = ISPI_A_GetInterruptStatus(
                  ispiptr,                /* ISPI Handle */
                  &ISPI_Intstat           /* ISPI Interrupt Status*/
                  );

/* Perform an optional check on the ISPI_A_GetInterruptStatus() return
code */
    if ( (retval != DD_ERR_NONE) )
    {
        /* do application-specific error status reporting */
    }
}
```

Freescale Semiconductor, Inc.

7.4.12 ISPI_A_GetRegister

The following paragraphs describe the ISPI_A_GetRegister function.

7.4.12.1 Description

The ISPI_A_GetRegister function gets an ISPI register. Call this function at any time.

7.4.12.2 Prototype

```
ddErr_t ISPI_A_GetRegister    (
                              pISPI_A_t                  ISPIPtr,
                              ISPI_A_RegisterSwitch_t    ISPIRegisterSwitch,
                              u2                         *GetRegisterPtr
                              );
```

7.4.12.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ISPIRegisterSwitch	ISPI_A_RegisterSwitch_t	Selects among ISPI registers	ISPI_A_SPCR_SWITCH ISPI_A_SPICR_SWITCH ISPI_A_SPSR_SWITCH ISPI_A_SPDR_SWITCH
GetRegisterPtr	u2 *	Address of copy of selected ISPI register	Any non-zero core processor data address

7.4.12.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

7.4.12.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

/* Create an ISPI version A Device Handle */
pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

/* Variable to hold ISPI Register data */
u2 ISPI_Reg;

/* Get Register ISPI version A */
retval = ISPI_A_GetRegister(
                            ispiptr,            /* ISPI Handle */
                            ISPI_A_SPSR_SWITCH, /* Register Switch */
                            &ISPI_Reg           /* ISPI Register Value */
                            );

/* Perform an optional check on the ISPI_A_GetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

7.4.13 ISPI_A_SetRegister

The following paragraphs describe the ISPI_A_SetRegister function.

7.4.13.1 Description

The ISPI_A_SetRegister function sets an ISPI register. Call this function at any time.

7.4.13.2 Prototype

```
ddErr_t ISPI_A_SetRegister      (
                                pISPI_A_t                   ISPIPtr,
                                ISPI_A_RegisterSwitch_t     ISPIRegisterSwitch,
                                u2                          RegisterValue
                                );
```

7.4.13.3 Arguments

Formal Name	Type	Description	Range/Valid
ISPIPtr	pISPI_A_t	ISPI base address associated with this driver	Any non-zero core processor data address
ISPIRegisterSwitch	ISPI_A_RegisterSwitch_t	Selects among ISPI registers	ISPI_A_SPCR_SWITCH ISPI_A_SPICR_SWITCH ISPI_A_SPSR_SWITCH ISPI_A_SPDR_SWITCH
RegisterValue	u2	Data to be copied to the selected ISPI register	Any 16-bit value

7.4.13.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.13.5 Code Usage Example

```
/* Include the ISPI version A header file */
#include “ispi_a.h”
...
ddErr_t retval;
...

/* Create an ISPI version A Device Handle */
pISPI_A_t ispiptr = (pISPI_A_t) __PWS_ISPI;

/* Set Register ISPI version A */
retval = ISPI_A_SetRegister(
                            ispiptr,              /* ISPI Handle */
                            ISPI_A_SPSR_SWITCH,   /* Register Switch */
                            SPSR_OVR_MASK         /* Data Value */
                            );

/* Perform an optional check on the ISPI_A_SetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

Section 8 KPP_A Level 1

This section describes the functions of a KPP_A level 1 device driver.

8.1 Overview

The keypad port (KPP) and its accompanying device driver (KPP_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the KPP on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The keypad port can be used either for keypad matrix scanning or as general-purpose I/O. Sixteen pins are dedicated to the keypad port. Keypads of any configuration up to eight rows and eight columns are supported through software configuration of the peripheral pins. Any pins not used for the keypad are available for general-purpose input/output. The registers are configured such that the pins can be treated as an I/O port up to 16 bits wide.

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the keypad port.

8.2 KPP_A Level 1 Device Driver Functions

KPP_A level 1 device driver software provides the following functions:

- Initializes the KPP for keypad matrix scanning
- Controls KPP synchronizers, status, and interrupts
- Gets Key Press and Key Release status
- Identifies Key Press row and column
- Gets and sets any KPP register. This permits programming any of the 16 KPP pins for general-purpose I/O, or using the KPP in a method not supported by the other KPP_A level 1 device driver functions.

8.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

8.3.1 Abstract Data Type Definitions

The following are KPP_A abstract data types.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.3.1.1 KPP_A_t - KPP Register Definitions

The following structure definition corresponds to the KPP register block layout.

```
typedef struct
{
    u2  KPCR;                   /* KPP Control Register */
    volatile u2  KPSR;          /* KPP Status Register */
    u2  KDDR;                   /* KPP Data Direction Register */
    volatile u2  KPDR;          /* KPP Data Register */
} KPP_A_t, *pKPP_A_t;
```

8.3.2 Enumerated Data Type Definitions

The following are KPP_A enumerated data types.

8.3.2.1 KPP_A_RegisterSwitch_t - KPP Register Selection

These values are used to select among KPP registers.

```
typedef enum {
      KPP_A_KPCR_SWITCH,                  /* Select KPCR */
      KPP_A_KPSR_SWITCH,                  /* Select KPSR */
      KPP_A_KDDR_SWITCH,                  /* Select KDDR */
      KPP_A_KPDR_SWITCH                   /* Select KPDR */
} KPP_A_RegisterSwitch_t;
```

8.4 API Definitions

The following paragraphs provide level 1 API definitions for a KPP_A device driver.

Freescale Semiconductor, Inc.

8.4.1 KPP_A_Init

The following paragraphs describe the KPP_A_Init function.

8.4.1.1 Description

KPP_A_Init initializes the KPP for operation. Call this function once during the system initialization.

8.4.1.2 Prototype

```
ddErr_t     KPP_A_Init    (
                          pKPP_A_t      KPPPtr,
                          u1            ColumnMask,
                          u1            RowMask
                          );
```

8.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
ColumnMask	u1	Mask that specifies the column bits that will be configured for use in the keypad matrix	Any non-zero 8-bit value
RowMask	u1	Mask that specifies the row bits that will be configured for use in the keypad matrix	Any non-zero 8-bit value

8.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
KPP_A_ERR_ZERO_ROWS	Rows parameter is zero
KPP_A_ERR_ZERO_COLUMNS	Columns parameter is zero

Freescale Semiconductor, Inc.

8.4.1.5 Code Usage Example

```
/* Include the KPP version A header file */
#include "kpp_a.h"
...
ddErr_t retval;
...

/* Create a KPP version A Device Handle */
pKPP_A_t kppptr = (pKPP_A_t) __PWS_KEYPAD;

/* Variables to hold column and row configuration */
u1 columnmask = PARAMETER_COLUMN_3_MASK +   /* Select columns 0-3 of KPP */
                PARAMETER_COLUMN_2_MASK +
                PARAMETER_COLUMN_1_MASK +
                PARAMETER_COLUMN_0_MASK;

u1 rowmask = PARAMETER_ROW_3_MASK +         /* Select rows 0-3 of KPP */
             PARAMETER_ROW_2_MASK +
             PARAMETER_ROW_1_MASK +
             PARAMETER_ROW_0_MASK;
...

/* Initialize KPP version A */
retval = KPP_A_Init(
                    kppptr,       /* KPP Handle */
                    columnmask,   /* Column configuration */
                    rowmask       /* Row configuration */
                    );

/* Perform an optional check on the KPP_A_Init() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

8.4.2 KPP_A_KeyControl

The following paragraphs describe the KPP_A_KeyControl function.

8.4.2.1 Description

KPP_A_KeyControl directs the keypad to detect and generate a key depress and/or key release condition. This function may be used after a call to KPP_A_Init.

8.4.2.2 Prototype

```
ddErr_t     KPP_A_KeyControl    (
                                pKPP_A_t            KPPPtr,
                                u1                  ColumnMask,
                                bool                KeyReleaseInterruptEnable,
                                bool                KeyDepressInterrupt
                                );
```

8.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
ColumnMask	u1	Mask that specifies the column bits that will be configured for use in the keypad matrix	Any non-zero 8-bit value
KeyReleaseInterrupt	bool	Controls the Key Release Interrupt Enable	TRUE = enable Key Release Interrupt FALSE = disable Key Release Interrupt
KeyDepressInterrupt	bool	Controls the Key Depress Interrupt Enable	TRUE = enable Key Depress Interrupt FALSE = disable Key Depress Interrupt

8.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
KPP_A_ERR_ZERO_COLUMNS	Columns parameter is zero

Freescale Semiconductor, Inc.

8.4.2.5 Code Usage Example

```
/* Include the KPP version A header file */
#include "kpp_a.h"
...
ddErr_t retval;
...

/* Create a KPP version A Device Handle */
pKPP_A_t kppptr = (pKPP_A_t) __PWS_KEYPAD;

/* Variables to hold column and row configuration */
u1 columnmask = PARAMETER_COLUMN_3_MASK +   /* Select columns 0-3 of KPP */
                PARAMETER_COLUMN_2_MASK +
                PARAMETER_COLUMN_1_MASK +
                PARAMETER_COLUMN_0_MASK;

/* Variable to hold KPP Register data */
u2 KPP_Reg;
...
/* Set up KPP version A KeyControl */
retval = KPP_A_KeyControl(
                          kppptr,       /* KPP Handle */
                          columnmask,   /* Column configuration */
                          FALSE,        /* Disable KRIE */
                          TRUE          /* Enable KDIE */
                         );

/* Perform an optional check on the KPP_A_KeyControl() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```


Freescale Semiconductor, Inc.

8.4.3 KPP_A_GetStatus

The following paragraphs describe the KPP_A_GetStatus function.

8.4.3.1 Description

KPP_A_GetStatus returns the Keypad Key Release and Keypad Key Depress status. This function may be used after a call to KPP_A_KeyControl.

8.4.3.2 Prototype

```
ddErr_t     KPP_A_GetStatus     (
                                pKPP_A_t        KPPPtr,
                                bool            *ReleaseResultPtr,
                                bool            *DepressResultPtr
                                );
```

8.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
ReleaseResultPtr	bool *	Result address for Keypad Key Release status	Any non-zero core processor data address
DepressResultPtr	bool *	Result address for Keypad Key Depress status	Any non-zero core processor data address

8.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

8.4.3.5 Code Usage Example

```
/* Include the KPP version A header file */
#include "kpp_a.h"
...
ddErr_t retval;
...

/* Create a KPP version A Device Handle */
pKPP_A_t kppptr = (pKPP_A_t) __PWS_KEYPAD;

/* Variable to hold KPP Release/Depress Status */
bool DepStatus = FALSE;
bool RelStatus = FALSE;

/* Get Status KPP version A */
while (DepStatus == FALSE)
{
    retval = KPP_A_GetStatus(
                             kppptr,
                             &RelStatus,
                             &DepStatus
                            );

    /* Perform an optional check on the KPP_A_GetStatus() return code */
    if ( (retval != DD_ERR_NONE) )
    {
        /* do application-specific error status reporting */
    }
}
```

Freescale Semiconductor, Inc.

8.4.4 KPP_A_KeyColumnScan

The following paragraphs describe the KPP_A_KeyColumnScan function.

8.4.4.1 Description

KPP_A_KeyColumnScan identifies the number of key rows pressed into one column. This function may be used after a call to KPP_A_KeyControl. This function should be called for each column on the keypad matrix.

In an interrupt-driven mechanism, this function may be called from a KPP interrupt service routine.

In a poll-driven mechanism, this function may be called after a KPP_A_GetStatus has determined a key press has occurred.

8.4.4.2 Prototype

```
ddErr_t     KPP_A_KeyColumnScan     (
                                    pKPP_A_t        KPPPtr,
                                    u1              Column,
                                    u1              ColumnMask,
                                    u1              RowMask,
                                    u1              *RowResultPtr
                                    );
```

Freescale Semiconductor, Inc.

8.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
Column	u1	Specifies the one column bit that will be scanned	PARAMETER_COLUMN_7_MASK PARAMETER_COLUMN_6_MASK PARAMETER_COLUMN_5_MASK PARAMETER_COLUMN_4_MASK PARAMETER_COLUMN_3_MASK PARAMETER_COLUMN_2_MASK PARAMETER_COLUMN_1_MASK PARAMETER_COLUMN_0_MASK
ColumnMask	u1	Mask that specifies the column bit that are configured for use in the keypad matrix	Any non-zero 8-bit value
RowMask	u1	Mask that specifies the row bits in the keypad matrix that will be scanned	Any non-zero 8-bit value
RowResultPtr	u1 *	Result address for number of rows pressed	Any non-zero core processor data address

8.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result Address is NULL
KPP_A_ERR_INVALID_COLUMN	Column parameter not one column
KPP_A_ERR_ZERO_ROWS	Rows parameter is zero
KPP_A_ERR_ZERO_COLUMNS	Columns parameter is zero

Freescale Semiconductor, Inc.

8.4.4.5 Code Usage Example

```
/* include the kpp version a header file */
#include "kpp_a.h"
...
dderr_t retval;
...

/* create a kpp version a device handle */
pkpp_a_t kppptr = (pkpp_a_t) __pws_keypad;

/* variables to hold column and row configuration */
columnmask = parameter_column_3_mask +   /* columns 0-3 of kpp */
             parameter_column_2_mask +
             parameter_column_1_mask +
             parameter_column_0_mask;

rowmask =    parameter_row_3_mask +    /* rows 0-3 of kpp */
             parameter_row_2_mask +
             parameter_row_1_mask +
             parameter_row_0_mask;

/* array to hold results for each column */
u1 rowresult[4];

/* columns to scan */
u1 key_columns[] = {
                    parameter_column_0_mask,
                    parameter_column_1_mask,
                    parameter_column_2_mask,
                    parameter_column_3_mask
                   };
int i;

for (i=0; i<4; i++)
{
    retval = kpp_a_keycolumnscan(
                                 kppptr,
                                 key_columns[i],
                                 columnmask,
                                 rowmask,
                                 &rowresult[i]
                                 );

    /* perform an optional check on the kpp_a_keycontrol() return code */
    if ( (retval != dd_err_none) )
    {
        /* do application-specific error status reporting */
    }
}
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

8.4.5 KPP_A_GetRegister

The following paragraphs describe the KPP_A_GetRegister function.

8.4.5.1 Description

KPP_A_GetRegister gets a KPP register. Call this function at any time.

8.4.5.2 Prototype

```
ddErr_t     KPP_A_GetRegister   (
                                pKPP_A_t                KPPPtr,
                                KPP_A_RegisterSwitch_t  KPPRegisterSwitch,
                                u2                      *GetRegisterPtr
                                );
```

8.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
KPPRegisterSwitch	KPP_A_RegisterSwitch_t	Selects among KPP registers	KPP_A_KPCR_SWITCH KPP_A_KPSR_SWITCH KPP_A_KDDR_SWITCH KPP_A_KPDR_SWITCH
GetRegisterPtr	u2 *	Result address for selected KPP register	Any non-zero core processor data address

8.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result Address is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

8.4.5.5 Code Usage Example

```
/* Include the KPP version A header file */
#include "kpp_a.h"
...
ddErr_t retval;
...

/* Create a KPP version A Device Handle */
pKPP_A_t kppptr = (pKPP_A_t) __PWS_KEYPAD;

/* Variable to hold KPP Register data */
u2 KPP_Reg;
...

/* Get Register KPP version A */
retval = KPP_A_GetRegister(
                           kppptr,              /* KPP Handle */
                           KPP_A_KPSR_SWITCH,   /* Register Switch */
                           &KPP_Reg             /* Data Value */
                           );

/* Perform an optional check on the KPP_A_GetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.4.6 KPP_A_SetRegister

The following paragraphs describe the KPP_A_SetRegister function.

8.4.6.1 Description

KPP_A_SetRegister sets a KPP register. Call this function at any time.

8.4.6.2 Prototype

```
ddErr_t     KPP_A_SetRegister   (
                                pKPP_A_t                KPPPtr,
                                KPP_A_RegisterSwitch_t  KPPRegisterSwitch,
                                u2                      RegisterValue
                                );
```

8.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
KPPPtr	pKPP_A_t	KPP base address associated with this driver	Any non-zero core processor data address
KPPRegisterSwitch	KPP_A_RegisterSwitch_t	Selects among KPP registers	KPP_A_KPCR_SWITCH KPP_A_KPSR_SWITCH KPP_A_KDDR_SWITCH KPP_A_KPDR_SWITCH
RegisterValue	u2	Data to copy into selected KPP register	Any 16-bit value

8.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

8.4.6.5 Code Usage Example

```
/* Include the KPP version A header file */
#include "kpp_a.h"
...
ddErr_t retval;
...

/* Create a KPP version A Device Handle */
pKPP_A_t kppptr = (pKPP_A_t) __PWS_KEYPAD;

/* Set Register KPP version A */
retval = KPP_A_SetRegister(
                           kppptr,              /* KPP Handle */
                           KPP_A_KPSR_SWITCH,   /* Register Switch */
                           KPSR_KDIE_MASK       /* Data Value */
                           );

/* Perform an optional check on the KPP_A_SetRegister() return code */
if ( (retval != DD_ERR_NONE) )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Section 9 LCD_A Level 1

This section describes the functions of an LCD_A level 1 device driver.

9.1 Overview

The KS0066U character type dot matrix liquid crystal display (LCD) controller device and its accompanying device driver (LCD_A) reside on the MMCMPFB1200 platform board (PFB). The MMCMPFB1200 is a development system for the MMC2001. Level 1 service is designed to allow the programmer to deal with the KS0066U on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

KS0066U registers are directly memory addressable from the perspective of the MCU. The KS0066U is wired to the LCD peripheral and provides the electrical signaling required for the imaging of a set of characters. The KS0066U does not generate interrupts. It is necessary to query the part before issuing commands, reads, or writes to ensure that it is not in a busy state.

The KS0066U has various selectable display modes for which the LCD_A driver will provide access. It is cursor-oriented in the sense that there is a visible or invisible cursor position within the display window. The cursor position determines where the next character will be written. The cursor can be moved to a specific location within the display window. The cursor can either advance, decrement, or remain in place after a character is written to a position. The display can be shifted left or right by one position. The display can be programmed to shift left or right as each character is written. If two lines are being displayed, both lines are shifted.

The KS0066U contains an internal ROM-based character set with 208 5 x 8 dot character patterns and 32 5 x 10 dot character patterns. The character set includes the standard alphanumerics. The numeric values of the characters follows the standard ASCII convention. These values are used as zero-based offsets into character generator read-only memory (CGROM). A character is displayed by writing its corresponding character generator random access memory (CGRAM) address to the KS0066U, which then copies the bitmap at that position into display data random access memory (DDRAM) at a position corresponding to the current location of the cursor.

The first character stored in the CGROM is the “space” character at offset 0x20. The KS0066U allows the programmer to store up to eight custom character bitmap patterns in internal RAM at offsets 0x0 through 0x7 which would normally correspond to ASCII non-printing control characters. The contents of DDRAM can be read as well as written. This feature makes it possible both to display characters and to query the device to see which characters are currently being displayed.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Refer to Hitachi's HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver) for more information on the LCD.

9.2 LCD_A Level 1 Device Driver Functions

LCD_A level 1 device driver software provides the following functions:

- Sets the interface data length for an 8-bit or 4-bit microprocessor
- Sets the line display mode to one line or two lines
- Sets the character display mode to 5x8 or 5x10
- Turns the display on or off
- Turns the cursor on or off
- Sets the cursor mode to blinking or non-blinking
- Clears the display
- Sets the direction the cursor will move (left or right) after a character is written to or read from the display
- Turns on or off shift mode which can shift the display left or right before each character is written (direction is determined by the direction the cursor is set to move)
- Moves the cursor to the Home position
- Shifts the cursor one character position (left or right) from its current location in the display
- Shifts the display by one character position to the left or right
- Writes the DDRAM address into the address counter to specify reading from and writing to DDRAM
- Reads the busy flag and the address counter contents
- Writes a character to the display at the current cursor position
- Reads the character value at the current cursor position from the display
- Writes one or more custom character bitmaps into the CGRAM (up to eight max)
- Sets the control and writedata registers
- Gets the contents of the status and readdata registers

Freescale Semiconductor, Inc.

9.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

9.3.1 Abstract Data Type Definitions

The following are LCD_A abstract data types.

9.3.1.1 LCD_A_t - LCD Register Definitions

The following structure definition corresponds to the LCD register block layout.

NOTE: *While the KS0066U registers are 8-bit, the driver registers are mapped to 16-bits for hardware reasons.*

```
typedef struct {
      u2 RESERVED0;
      union {
            volatile u2 LCR;        /* LCD Control Register */
            volatile u2 LSR;        /* LCD Status Register */
      } cmd;
      u2 RESERVED1;
      union {
            volatile u2 LDRR;       /* LCD Data Read Register */
            volatile u2 LDWR;       /* LCD Data Write Register */
      } data;
} LCD_A_t, *pLCD_A_t;
```

9.3.2 Enumerated Data Type Definitions

The following are LCD_A enumerated data types.

9.3.2.1 LCD_A_RegisterSwitch_t - LCD Register Selection

These values are used to select among LCD registers.

```
typedef enum {
    LCD_A_LCR_SWITCH,         /* Select CONTROL Register */
    LCD_A_LSR_SWITCH,         /* Select STATUS Register */
    LCD_A_LDRR_SWITCH,        /* Select READDATA Register */
    LCD_A_LDWR_SWITCH         /* Select WRITEDATA Register */
} LCD_A_RegisterSwitch_t;
```

9.3.2.2 LCD_A_DataLength_t - LCD Data Length Selection

These values are used to select 4 or 8-bit microcontroller operation.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

```
typedef enum {
      FOUR_BIT_DATA_LENGTH  = 0x00, /* 4-bit microcontroller operation */
      EIGHT_BIT_DATA_LENGTH = 0x10  /* 8-bit microcontroller operation */
} LCD_A_DataLength_t;
```

9.3.2.3 LCD_A_DisplayLines_t - Line Number Selection

These values are used to select a one line or a two line display.

```
typedef enum {
      ONE_LINE = 0x00,        /* 1 line display */
      TWO_LINES = 0x08        /* 2 line display */
} LCD_A_DisplayLines_t;
```

9.3.2.4 LCD_A_CharFont_t - Character Font Selection

These values are used to select character font size.

```
typedef enum {
      FONT_5X8_DOTS = 0x00,   /* 5x8 dot character font */
      FONT_5X10_DOTS = 0x04   /* 5x10 dot character font */
} LCD_A_CharFont_t;
```

9.3.2.5 LCD_A_IncMode_t - Increment Mode Selection

These values are used to select increment or decrement mode.

```
typedef enum {
      DECREMENT = 0x00,       /* Decrement after write/read from DDRAM */
      INCREMENT = 0x02        /* Increment after write/read from DDRAM */
} LCD_A_IncMode_t;
```

9.3.2.6 LCD_A_Component_t - Cursor or Display Selection

These values are used to select the cursor or the display.

```
typedef enum {
      CURSOR = 0x00,          /* Select cursor */
      DISPLAY = 0x08          /* Select display */
} LCD_A_Component_t;
```

Freescale Semiconductor, Inc.

9.3.2.7 LCD_A_CustomChar_t - CGRAM Character Pattern

These values are used to select CGRAM character patterns.

```
typedef enum {
      CHAR_PAT_0,       /* 0 */
      CHAR_PAT_1,       /* 1 */
      CHAR_PAT_2,       /* 2 */
      CHAR_PAT_3,       /* 3 */
      CHAR_PAT_4,       /* 4 */
      CHAR_PAT_5,       /* 5 */
      CHAR_PAT_6,       /* 6 */
      CHAR_PAT_7        /* 7 */
} LCD_A_CustomChar_t;
```

9.3.3 Initialization Defaults Macros

The following macros may be used as LCD_A_Init parameter values. In particular, the macro LCD_A_INIT_DEFAULT may be used in place of a call to LCD_A_Init. The only explicit argument to LCD_A_INIT_DEFAULT is the base address of the LCD to be initialized.

```
#define LCD_A_DEFAULT_DATA_LENGTH      EIGHT_BIT_DATA_LENGTH/* 8-bit */
#define LCD_A_DEFAULT_DISPLAY_LINES    TWO_LINES      /*2 lines */
#define LCD_A_DEFAULT_CHAR_FONT        FONT_5X8_DOTS  /* 5X8 dots */
#define LCD_A_DEFAULT_INC_MODE         INCREMENT      /*Increment mode*/
#define LCD_A_DEFAULT_DISPLAY_ON       TRUE     /*Turn display on*/
#define LCD_A_DEFAULT_CURSOR_ON        TRUE     /*Turn cursor on*/
#define LCD_A_DEFAULT_BLINK_ON         TRUE     /*Turn blinking on*/
#define LCD_A_DEFAULT_SHIFTMODE_ON     FALSE    /*Turn shifting off*/

#define LCD_A_INIT_DEFAULT(LCDPtr) \
    LCD_A_Init(LCDPtr, \
                  LCD_A_DEFAULT_DATA_LENGTH, \
                  LCD_A_DEFAULT_DISPLAY_LINES, \
                  LCD_A_DEFAULT_CHAR_FONT, \
                  LCD_A_DEFAULT_INC_MODE, \
                  LCD_A_DEFAULT_DISPLAY_ON, \
                  LCD_A_DEFAULT_CURSOR_ON, \
                  LCD_A_DEFAULT_BLINK_ON, \
                  LCD_A_DEFAULT_SHIFTMODE_ON)
```

9.4 API Definitions

The following paragraphs provide level 1 API definitions for an LCD_A device driver.

Freescale Semiconductor, Inc.

9.4.1 LCD_A_Init

The following paragraphs describe the LCD_A_Init function.

9.4.1.1 Description

LCD_A_Init initializes the LCD for operation. Call this function once during the system initialization. On power up, the LCD has the following settings:

- Display is cleared
- Data length is set to 8-bit interface
- Number of lines displayed is one
- Character font used is 5x8 dots
- Display and cursor are turned off
- Non-blinking mode is set
- Increment mode is set to increment
- Shift mode is off

9.4.1.2 Prototype

```
ddErr_t     LCD_A_Init    (
                          pLCD_A_t                LCDPtr,
                          LCD_A_DataLength_t      DataLenValue,
                          LCD_A_DisplayLines_t    NumDispLinesValue,
                          LCD_A_CharFont_t        CharFontValue,
                          LCD_A_IncMode_t         IncrementValue,
                          bool                    Display_ON,
                          bool                    Cursor_ON,
                          bool                    Blink_ON,
                          bool                    ShiftMode_ON
                          );
```

Freescale Semiconductor, Inc.

9.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
DataLenValue	LCD_A_DataLength_t	Selects the data length interface for the microcontroller	FOUR_BIT_DATA_LENGTH EIGHT_BIT_DATA_LENGTH
NumDispLinesValue	LCD_A_DisplayLines_t	Selects the number of lines for display	ONE_LINE TWO_LINES
CharFontValue	LCD_A_CharFont_t	Selects character font	FONT_5x8_DOTS FONT_5x10_DOTS
IncrementValue	LCD_A_IncMode_t	Selects direction of cursor or display when an update occurs	DECREMENT INCREMENT
Display_ON	bool	Selects operation mode of display	TRUE = Display is on FALSE = Display is off
Cursor_ON	bool	Selects operation mode of cursor	TRUE = Cursor is on FALSE = Cursor is off
Blink_ON	bool	Selects blinking mode of cursor	TRUE = Blinking is on FALSE = Blinking is off
ShiftMode_ON	bool	Turns shift mode on or off	TRUE = Shift mode is on FALSE = Shift mode is off

9.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
LCD_A_ERR_INVALID_DATA_LENGTH	Data length must be 8 or 4-bit
LCD_A_ERR_INVALID_LINE_VALUE	Number of lines is 1 or 2
LCD_A_ERR_INVALID_CHAR_FONT	Character font is invalid
LCD_A_ERR_INVALID_INC_MODE	Inc mode is invalid
LCD_A_ERR_INVALID_SETUP	5 x 10 font not valid with 2-line

Freescale Semiconductor, Inc.

9.4.1.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Init LCD version A */
retval = LCD_A_Init(
                  lcdptr,                /* LCD Device Base Address */
                  EIGHT_BIT_DATA_LENGTH, /* Interface Data Length */
                  ONE_LINE,             /* Number of lines to display */
                  FONT_5X10_DOTS,        /* Character Font */
                  INCREMENT,             /* Increment Mode */
                  TRUE,                  /* Display On/Off Switch */
                  TRUE,                  /* Cursor On/Off Switch */
                  FALSE,                 /* Blink On/Off Switch */
                  FALSE                  /* Shift Mode On/Off Switch */
                  );


/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.2 LCD_A_ClrDisplay

The following paragraphs describe the LCD_A_ClrDisplay function.

9.4.2.1 Description

LCD_A_ClrDisplay clears the display. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

The following occur when the display is cleared:

- Current DDRAM address is set to zero (which returns the cursor to Home)
- Display returns to its original status if it was shifted
- Sets the increment mode to increment

9.4.2.2 Prototype

```
ddErr_t LCD_A_ClrDisplay    (
                            pLCD_A_t        LCDPtr
                            );
```

9.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address

9.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.2.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Clear Display LCD version A */
retval = LCD_A_ClearDisplay(
                        lcdptr      /* LCD Device Base Address */
                        );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.3 LCD_A_ReturnHome

The following paragraphs describe the LCD_A_ReturnHome function.

9.4.3.1 Description

LCD_A_ReturnHome returns the cursor to the Home position on the display. Home is the character position at the left edge of the display (in the first line if two lines are displayed). It is assumed that the LCD has been initialized through a previous call to LCD_A_Init. This function also returns the display to its original status if it was shifted.

9.4.3.2 Prototype

```
ddErr_t LCD_A_ReturnHome    (
                            pLCD_A_t        LCDPtr
                            );
```

9.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address

9.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.3.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Return Home LCD version A */
retval = LCD_A_ReturnHome(
                        lcdptr              /* LCD Device Base Address */
                        );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.4 LCD_A_SetEntryMode

The following paragraphs describe the LCD_A_SetEntryMode function.

9.4.4.1 Description

LCD_A_SetEntryMode selects increment mode and turns display shift mode on or off. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init. This function also returns the display to its original status if it was shifted.

9.4.4.2 Prototype

```
ddErr_t LCD_A_SetEntryMode  (
                            pLCD_A_t            LCDPtr,
                            LCD_A_IncMode_t     IncrementValue,
                            bool                ShiftMode_ON
                            );
```

9.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
IncrementValue	LCD_A_IncMode_t	Address counter contents decremented or incremented by one	DECREMENT INCREMENT
ShiftMode_ON	bool	Sets display shift mode on or off. If on, the display shifts by one character position before a character is written to the DDRAM	TRUE = enable shift mode FALSE = disable shift mode

9.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
LCD_A_ERR_INVALID_INC_MODE	Inc mode is invalid

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.4.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Set Entry Mode LCD version A */
retval = LCD_A_SetEntryMode(
                            lcdptr,        /* LCD Device Base Address */
                            INCREMENT,     /* Increment Mode */
                            TRUE           /* Shift Mode On/Off Switch */
                            );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.5 LCD_A_DisplayControl

The following paragraphs describe the LCD_A_DisplayControl function.

9.4.5.1 Description

LCD_A_DisplayControl turns the display on or off, turns the cursor on or off, and sets the cursor mode to blinking or non-blinking.

NOTE: *Blinking can occur whether the cursor is on or not. Turning the display off does not clear the display. The data remains in DDRAM and is visible the next time the display is turned on.*

It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

9.4.5.2 Prototype

```
ddErr_t LCD_A_DisplayControl  (
                              pLCD_A_t        LCDPtr,
                              bool            Display_ON,
                              bool            Cursor_ON,
                              bool            Blink_ON
                              );
```

9.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
Display_ON	bool	Turns display on or off	TRUE = Display on FALSE = Display off
Cursor_ON	bool	Turns cursor on or off	TRUE = Cursor on FALSE = Cursor off
Blink_ON	bool	Turns blinking on or off	TRUE = Blinking on FALSE = Blinking off

9.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.5.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Display Control LCD version A */
retval = LCD_A_DisplayControl(
                              lcdptr,     /* LCD Device Base Address */
                              TRUE,       /* Display On/Off Switch */
                              TRUE,       /* Cursor On/Off Switch */
                              TRUE        /* Blink On/Off Switch */
                              );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.6 LCD_A_Shift

The following paragraphs describe the LCD_A_Shift function.

9.4.6.1 Description

LCD_A_Shift shifts the cursor or display by one character position to the right or left. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

9.4.6.2 Prototype

```
ddErr_t LCD_A_Shift    (
                       pLCD_A_t              LCDPtr,
                       LCD_A_Component_t     Selected,
                       bool                  MoveRight
                       );
```

9.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
Selected	LCD_A_Component_t	Cursor or display selected for shift	CURSOR DISPLAY
MoveRight	bool	Direction of shift	TRUE = Shift to right FALSE = Shift to left

9.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
LCD_A_ERR_INVALID_COMPONENT	Component selection is invalid

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.6.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Shift LCD version A */
retval = LCD_A_Shift(
                     lcdptr,        /* LCD Device Base Address */
                     DISPLAY,       /* Cursor/Display Switch */
                     TRUE           /* Move Left/Right Switch */
                     );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.7 LCD_A_WaitNotBusy

The following paragraphs describe the LCD_A_WaitNotBusy function.

9.4.7.1 Description

LCD_A_WaitNotBusy waits for the busy flag to clear. It returns when the busy flag is cleared or a time out has occurred while waiting for the busy flag to clear. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

NOTE: *Along with the busy flag, the KS0066U status register contains the display address counter. Attempting to read the address counter value immediately after the busy flag indicates "not busy" may be unreliable. There is a delay from the time the busy flag is cleared to the time that the address counter is updated for functions that change the address counter. This means that one read of the address counter may report the value before it is updated. The next read may report the value of the address counter after it has been updated.*

9.4.7.2 Prototype

```
ddErr_t LCD_A_WaitNotBusy (
                                    pLCD_A_t                    LCDPtr,
                                    );
```

9.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address

9.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
LCD_A_ERR_TIME_OUT_STATUS	Time-out of internal LCD_A

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

9.4.7.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Wait Not Busy LCD version A */
retval = LCD_A_WaitNotBusy(
                           lcdptr         /* LCD Device Base Address */
                           );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.8 LCD_A_SetDDRAMAddr

The following paragraphs describe the LCD_A_SetDDRAMAddr function.

9.4.8.1 Description

LCD_A_SetDDRAMAddr sets the DDRAM address for writing to or reading from the display. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

NOTE: *If two-line display is selected, DDRAM addresses between 0x28 and 0x39 are not valid. No check is performed to ensure that the DDAddr parameter is outside this range. Any data written to these addresses will be lost.*

9.4.8.2 Prototype

```
ddErr_t LCD_A_SetDDRAMAddr (
                                    pLCD_A_t                    LCDPtr,
                                    u1                          DDAddr
                                    );
```

9.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
DDAddr	u1	DDRAM address supplied by the user	0x00 to 0x27 (on display line one of two lines) 0x40 to 0x67 (on display line two of two lines) 0x00 to 0x4F (on display line one of one line)

9.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
LCD_A_ERR_INVALID_DDRAM_ADDR	DDRAM address > than 0x67

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

9.4.8.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;
...
/* Set DD RAM Address LCD version A */
retval = LCD_A_SetDDRAMAddr(
                            lcdptr,        /* LCD Device Base Address */
                            0x05           /* DD RAM Address */
                            );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.9 LCD_A_Write

The following paragraphs describe the LCD_A_Write function.

9.4.9.1 Description

LCD_A_Write writes the user-specified character to the display at the current cursor position. It is assumed that the LCD has been initialized through a previous call to LCD_A_Init. If a DDRAM address is not currently in the address counter contents, SetDDRAMAddr should be called first.

The position on the display at which the character is written is determined by the address counter contents. This address is affected by the following functions:

- LCD_A_Init
- LCD_A_CirDisplay
- LCD_A_ReturnHome
- LCD_A_SetDDRAMAddr
- LCD_A_Write
- LCD_A_Read
- LCD_A_CGWrite

The address counter is updated according to the increment mode setting after a write occurs and the busy flag is cleared. This update to the address counter makes successive writes possible.

9.4.9.2 Prototype

```
ddErr_t LCD_A_Write   (
                      pLCD_A_t        LCDPtr,
                      u1              DataVal
                      );
```

9.4.9.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
DataVal	u1	Data value	Any 8-bit value

Freescale Semiconductor, Inc.

9.4.9.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

9.4.9.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;
...
/* Write LCD version A */
retval = LCD_A_Write(
                        lcdptr,     /* LCD Device Base Address */
                        0x41        /* Data Value */
                        );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.10 LCD_A_Read

The following paragraphs describe the LCD_A_Read function.

9.4.10.1 Description

LCD_A_Read reads the characters in DDRAM at the current cursor position into address specified by the user. This function may be called after SetDDRAMAddr has been called.

The position on the display at which the character is read is determined by the address counter contents. This address is affected by the following functions:

- LCD_A_Init
- LCD_A_CirDisplay
- LCD_A_ReturnHome
- LCD_A_SetDDRAMAddr
- LCD_A_Write
- LCD_A_Read
- LCD_A_CGWrite

The address counter is updated according to the increment mode setting after a write occurs and the busy flag is cleared. This update to the address counter makes successive writes possible.

9.4.10.2 Prototype

```
ddErr_t LCD_A_Read   (
                     pLCD_A_t        LCDPtr,
                     u1              *AddrPtr
                     );
```

9.4.10.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
AddrPtr	u1 *	Result address for character	Any non-zero core processor data address

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

9.4.10.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL

9.4.10.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Create a variable for the LCD Character */
u1 DataValue;
...
/* Read LCD version A */
retval = LCD_A_Read(
                    lcdptr,          /* LCD Device Base Address */
                    &DataValue       /* LCD Result Character */
                    );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

9.4.11 LCD_A_CGWrite

The following paragraphs describe the LCD_A_CGWrite function.

9.4.11.1 Description

LCD_A_CGWrite writes user-specified data to the CGRAM. It reads the number of bytes at the custom character address based on the character font chosen. For example, if a setting of 5 x 8 dots is chosen, eight bytes are used from the custom character array. The address counter is set for the CGRAM. The function returns an 8-bit code for accessing the custom character through the DDRAM.

It is assumed that the LCD has been initialized through a previous call to LCD_A_Init.

9.4.11.2 Prototype

```
ddErr_t LCD_A_CGWrite    (
                         pLCD_A_t                LCDPtr,
                         u1                      *CustomCharArr,
                         LCD_A_CharFont_t        FontValue,
                         LCD_A_CustomChar_t      CharPattern,
                         u1                      *CharAccessCode
                         );
```

Freescale Semiconductor, Inc.

9.4.11.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
CustomCharArr	u1 *	The character bitmap is an array A[i,j] of dots mapped onto a vector of bytes. Each byte corresponds to a row of dots in the bitmap. The five least significant bit positions of each byte represent the columns of the bitmap. The number of rows in the bitmap is determined by the FontValue parameter.	Any non-zero core processor data address
FontValue	LCD_A_CharFont_t	Specifies which character font the LCD is initialized to use. This indicates the size of the CustomCharArr. Given CustomCharArr[j], if FontValue is FONT_5x8_DOTS, then j is eight. When FontValue is FONT_5x10_DOTS, j is ten.	FONT_5x8_DOTS FONT_5x10_DOTS
CharPattern	LCD_A_CustomChar_t	Specifies which CGRAM character pattern to write. This determines where in CGRAM the bitmap is written and which DDRAM code corresponds to this character pattern.	For character font 5 x 8 dots: CharPat0 = 0 to CharPat7 = 7 For character font 5 x 10 dots: CharPat0 = 0 to CharPat3 = 3
CharAccessCode	u1 *	8-bit character code for accessing custom characters through DDRAM	Any non-zero core processor data address

9.4.11.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
LCD_A_ERR_INVALID_CHAR_FONT	Character font is invalid
LCD_A_ERR_INVALID_CHAR_PAT	Invalid character pattern
LCD_A_ERR_INVALID_BITMAP	Data byte > 0x20 invalid

Freescale Semiconductor, Inc.

9.4.11.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Declare needed variables */
LCD_A_CustomChar_t CharPattern = CHAR_PAT_3;
u1 DataValue;
u1 CharArr[LCD_A_F5X10_ARR_MAX];
...
/* Initialize the LCD Character Array to Write a “P” */
CharArr[0] = 0x00;
CharArr[1] = 0x0F;
CharArr[2] = 0x09;
CharArr[3] = 0x09;
CharArr[4] = 0x0F;
CharArr[5] = 0x08;
CharArr[6] = 0x08;
CharArr[7] = 0x08;
CharArr[8] = 0x08;
CharArr[9] = 0x00;
...
/* CG Write LCD version A */
retval = LCD_A_CGWrite(
                       lcdptr,             /* LCD Device Base Address */
                       CharArr,            /* Character Bitmap */
                       FONT_5X10_DOTS,     /* Font Size */
                       CharPattern,        /* CG RAM Charcater Pattern */
                       &DataValue          /* Character Code */
                       );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

9.4.12 LCD_A_GetRegister

The following paragraphs describe the LCD_A_GetRegister function.

9.4.12.1 Description

LCD_A_GetRegister gets an LCD register. Call this function at any time.

NOTE: *Along with the busy flag, the KS0066U status register contains the display address counter. Attempting to read the address counter value immediately after the busy flag indicates "not busy" may be unreliable. There is a delay from the time the busy flag is cleared to the time that the address counter is updated for functions that change the address counter. This means that one read of the address counter may report the value before it is updated. The next read may report the value of the address counter after it has been updated.*

9.4.12.2 Prototype

```
ddErr_t LCD_A_GetRegister   (
                            pLCD_A_t                LCDPtr,
                            LCD_A_RegisterSwitch_t  LCDRegisterSwitch,
                            u1                      *GetRegisterPtr
                            );
```

9.4.12.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
LCDRegisterSwitch	LCD_A_RegisterSwitch_t	Selects among a subset of LCD registers	LCD_A_LSR_SWITCH LCD_A_LDRR_SWITCH
GetRegisterPtr	u1 *	Result address for selected LCD register	Any non-zero core processor data address

9.4.12.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

9.4.12.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

/* Create a variable for the LCD Character */
u1 RegisterValue;
...
/* Get Register LCD version A */
retval = LCD_A_GetRegister(
                        lcdptr,           /* LCD Device Base Address */
                        LCD_A_LCR_SWITCH, /* LCD Register Value */
                        &RegisterValue    /* LCD Register Result */
                        );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

9.4.13 LCD_A_SetRegister

The following paragraphs describe the LCD_A_SetRegister function.

9.4.13.1 Description

LCD_A_SetRegister sets either the control or writedata LCD register. Call this function at any time.

9.4.13.2 Prototype

```
ddErr_t LCD_A_SetRegister   (
                            pLCD_A_t                LCDPtr,
                            LCD_A_RegisterSwitch_t  LCDRegisterSwitch,
                            u1                      RegisterValue
                            );
```

9.4.13.3 Arguments

Formal Name	Type	Description	Range/Valid
LCDPtr	pLCD_A_t	LCD base address associated with this driver	Any non-zero core processor data address
LCDRegisterSwitch	LCD_A_RegisterSwitch_t	Selects among a subset of LCD registers	LCD_A_LSR_SWITCH LCD_A_LDRR_SWITCH
RegisterValue	u1	Data to copy into selected LCD register	Any 8-bit value

9.4.13.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

9.4.13.5 Code Usage Example

```
/* Include the LCD version A header file */
#include "lcd_a.h"
...
ddErr_t retval;
...
/* Create an LCD version A Device Handle */
pLCD_A_t lcdptr = (pLCD_A_t) __PWS_LCD;

...
/* Set Register LCD version A */
retval = LCD_A_SetRegister(
                        lcdptr,           /* LCD Device Base Address */
                        LCD_A_LCR_SWITCH, /* LCD Register Switch */
                        0x38              /* Register Value */
                        );

/* Perform an optional check on the return code */
if ( (retval != DD_ERR_NONE) )
{
/* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Section 10 PWM_A Level 1

This section describes the functions of a PWM_A level 1 device driver.

10.1 Overview

The pulse-width modulator (PWM) and its accompanying device driver (PWM_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the PWM on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

Each PWM channel consists of a free-running 10-bit counter monitored by two comparators connected to the width and period registers. The period register determines the period of the pulse train. When a match occurs, it resets the free-running counter and asserts (begins) the pulse on the output pin. The width register determines the width of the pulses. When a match occurs, it negates the pulse to the output pin.

The period and width registers are double-buffered; that is, the values written by the program into these registers are ordinarily strobed into working copies of the registers at the end of a pulse period, thus providing glitch-free transitions in period and width. This strobing action can be overridden in favor of immediate loads by means of a bit in the control register.

The counting rate can be selected from among eight prescaler taps, dividing the system clock by between four and 65536.

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the PWM.

10.2 PWM_A Level 1 Device Driver Functions

The PWM_A level 1 device driver software provides the following functions:

- Initializes the PWM
- Starts and stops the PWM
- Updates the pulse-width parameters
- Returns the values of the interrupt request flag, data flag (pin level), and counter value in the supplied result structure
- Returns the state of the interrupt request flag in the PWM control register (PWMCR)
- Get and set PWM registers

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

10.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

10.3.1 Abstract Data Type Definitions

The following are PWM_A abstract data types.

10.3.1.1 PWM_A_t - PWM Register Block

The following structure definition corresponds to the PWM register block layout.

```
/* register structure */
typedef struct {
      volatile u2  PWMCR;     /* Control */
      u2  PWMPR;              /* Period  */
      u2  PWMWR;              /* Width   */
      volatile u2  PWMCTR;    /* Counter */
      } PWM_A_t;
```

10.3.1.2 PWM_A_status_t - PWM Status Information

The following structure definition is used for returning PWM status information.

```
/* returned status structure */
typedef struct {
   bool IRQ;     /* whether PWM_A period has completed */
   bool data;    /* Output pin state */
   u2 counter;   /* current value of period counter, in PCLK clock ticks */
   } PWM_A_status_t;
```

10.3.2 Enumerated Data Type Definitions

The following are PWM_A enumerated data types.

Freescale Semiconductor, Inc.

10.3.2.1 PWM_A_ClockSel_t - PWM Clock Select

These values are used to set the clock divider tap selection.

```
/* for PWM_A_Init arg. clockSel */
typedef enum {
    PWM_A_DIV_4,
    PWM_A_DIV_8,
    PWM_A_DIV_16,
    PWM_A_DIV_64,
    PWM_A_DIV_256,
    PWM_A_DIV_2048,
    PWM_A_DIV_16384,
    PWM_A_DIV_65536
} PWM_A_ClockSel_t;
```

10.3.2.2 PWM_A_RegisterSwitch_t - PWM Register Switch

These values are used to select among PWM registers.

```
typedef enum
{                                   /* Get and set register selections  */
    PWM_A_PWMCR_SWITCH,              /* Select PWMCR                     */
    PWM_A_PWMPR_SWITCH,              /* Select PWMPR                     */
    PWM_A_PWMWR_SWITCH,              /* Select PWMWR                     */
    PWM_A_PWMCTR_SWITCH,             /* Select PWMCTR                    */
} PWM_A_RegisterSwitch_t;
```

10.4 API Definitions

The following paragraphs provide level 1 API definitions for a PWM_A device driver.

Freescale Semiconductor, Inc.

10.4.1 PWM_A_Init

The following paragraphs describe the PWM_A_Init function.

10.4.1.1 Description

PWM_A_Init initializes the PWM for operation. Call this function before any operations with the PWM_A are attempted.

10.4.1.2 Prototype

```
ddErr_t     PWM_A_Init   (
                         pPWM_A_t          PWMPtr,
                         PWM_A_ClockSel_t  ClockSel,
                         bool              IEnab,
                         bool              PolarityLo,
                         bool              Doze
                         );
```

10.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
ClockSel	PWM_A_ClockSel_t	Clock divider tap selection	PWM_A_DIV_4 PWM_A_DIV_8 PWM_A_DIV_16 PWM_A_DIV_64 PWM_A_DIV_256 PWM_A_DIV_2048 PWM_A_DIV_16384 PWM_A_DIV_65536
IEnab	bool	Interrupt Enable	TRUE...FALSE
PolarityLo	bool	Determines PWM polarity	TRUE = negative going pulse FALSE = positive going pulse
Doze	bool	Determines whether PWM is affected by DOZE mode	TRUE = disabled in DOZE mode FALSE = unaffected by DOZE mode

Freescale Semiconductor, Inc.

10.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
PWM_A_ERR_CLOCKSEL	ClockSel out of range

10.4.1.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

    /* Create a PWM version A Device Handle */
    pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

    /* Init PWM version A */
    retval = PWM_A_Init(
                        pwmptr,            /* PWM Device Handle */
                        PWM_A_DIV_8,       /* Clock Divider Tap */
                        TRUE,              /* Interrupt Enable */
                        FALSE,             /* Pulse Polarity */
                        FALSE              /* Doze Mode */
                        );

    /* Perform an optional check on the PWM_A_Init() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

10.4.2 PWM_A_Start

The following paragraphs describe the PWM_A_Start function.

10.4.2.1 Description

PWM_A_Start starts the PWM. If the PWM has already started, this function has no effect. Call this function at any time.

10.4.2.2 Prototype

```
ddErr_t     PWM_A_Start     (
                            pPWM_A_t    PWMPtr
                            );
```

10.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address

10.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

10.4.2.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

  /* Create a PWM version A Device Handle */
  pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

  /* Start PWM version A */
  retval = PWM_A_Start(
                      pwmptr                 /* PWM Device Handle */
                      );

  /* Perform an optional check on the PWM_A_Start() return code */
  if ( retval != DD_ERR_NONE )
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

10.4.3 PWM_A_Stop

The following paragraphs describe the PWM_A_Stop function.

10.4.3.1 Description

PWM_A_Stop stops the PWM. If the PWM has already stopped, this function has no effect. Call this function at any time.

10.4.3.2 Prototype

```
ddErr_t        PWM_A_Stop        (
                                 pPWM_A_t      PWMPtr
                                 );
```

10.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address

10.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

10.4.3.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

  /* Create a PWM version A Device Handle */
  pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

  /* Stop PWM version A */
  retval = PWM_A_Stop(
                      pwmptr,        /* PWM Device Handle */
                      );

  /* Perform an optional check on the PWM_A_Stop() return code */
  if ( retval != DD_ERR_NONE )
  {
     /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

10.4.4 PWM_A_UpdateOutput

The following paragraphs describe the PWM_A_UpdateOutput function.

10.4.4.1 Description

PWM_A_UpdateOutput updates the PWM period and pulse width from the parameters provided. The user should specify whether the parameters should take immediate effect, or wait until the end of the current pulse period. Call this function at any time.

10.4.4.2 Prototype

```
ddErr_t     PWM_A_UpdateOutput     (
                                   pPWM_A_t    PWMPtr,
                                   bool        Load,
                                   u2          Period,
                                   u2          Width
                                   );
```

10.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
Load	bool	Determines whether new clock period is started	TRUE = force new period FALSE = do not force new period
Period	u2	Period of repetition in PCLK ticks	10-bit unsigned int
Width	u2	Duration of pulse in PCLK ticks	10-bit unsigned int

10.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
PWM_A_ERR_PERIOD	Period out of range
PWM_A_ERR_WIDTH	Width out of range

Freescale Semiconductor, Inc.

10.4.4.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

    /* Create a PWM version A Device Handle */
    pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

  /* Update PWM version A period and pulse width */
  retval = PWM_A_UpdateOutput(
                              pwmptr,         /* PWM Device Handle */
                              FALSE,          /* Positive Pulse    */
                              0x100,          /* 256 Tick Period   */
                              0x080           /* 128 Tick Duration */
                              );

   /* Perform an optional check on the PWM_A_UpdateOutput() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

10.4.5 PWM_A_GetStatus

The following paragraphs describe the PWM_A_GetStatus function.

10.4.5.1 Description

PWM_A_GetStatus returns the state of the PWM, including whether the PWM period has completed, the output pin level, and the current value of the period counter. Call this function at any time.

10.4.5.2 Prototype

```
ddErr_t     PWM_A_GetStatus     (
                                pPWM_A_t            PWMPtr,
                                PWM_A_Status_t      *Status
                                );
```

10.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
Status	PWM_A_Status_t *	Address of a structure which receives the status information	Any non-zero core processor data address

10.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
PWM_A_ERR_STATUS	Status param is NULL

Freescale Semiconductor, Inc.

10.4.5.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

    /* Create a PWM version A Device Handle */
    pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

    /* Allocate a PWM version A Status Block */
    PWM_A_status_t status;

    /* Get PWM version A status */
    retval = PWM_A_GetStatus(
                            pwmptr,       /* PWM Device Handle */
                            &status       /* Status Block Addr */
                            );

    /* Perform an optional check on the PWM_A_GetStatus() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

10.4.6 PWM_A_GetIRQ

The following paragraphs describe the PWM_A_GetIRQ function.

10.4.6.1 Description

PWM_A_GetIRQ returns the value of the IRQ flag in the PWM control register (PWMCR), indicating that a PWM period has been completed. The flag in the PWMCR is cleared. Call this function at any time.

10.4.6.2 Prototype

```
ddErr_t     PWM_A_GetIRQ    (
                            pPWM_A_t    PWMPtr,
                            bool        *IRQHiPtr
                            );
```

10.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
IRQHiPtr	bool *	Determines whether the IRQ pin is asserted or not	TRUE = IRQ is high (asserted) FALSE = IRQ is low (negated)

10.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
PWM_A_ERR_IRQHIPTR	IRQHiPtr is NULL

Freescale Semiconductor, Inc.

10.4.6.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
bool irqHigh;           /* IRQ high flag */
...

    /* Create a PWM version A Device Handle */
    pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM;

    /* Get PWM version A IRQ status */
    retval = PWM_A_GetIRQ(
                         pwmptr,    /* PWM Device Handle  */
                         &irqHigh   /* IRQ Is High        */
                         );

    /* Perform an optional check on the PWM_A_GetIRQ() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

10.4.7 PWM_A_GetRegister

The following paragraphs describe the PWM_A_GetRegister function.

10.4.7.1 Description

PWM_A_GetRegister gets a PWM register. Call this function at any time.

10.4.7.2 Prototype

```
ddErr_t PWM_A_GetRegister (
                    pPWM_A_t                    PWMPtr,
                    PWM_A_RegisterSwitch_t      PWM_A_RegisterSwitch,
                    u4                          *GetRegisterPtr
                    );
```

10.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
PWM_A_RegisterSwitch	PWM_A_RegisterSwitch_t	Selects among PWM registers	PWM_A_PWMCR_SWITCH PWM_A_PWMPR_SWITCH PWM_A_PWMWR_SWITCH
GetRegisterPtr	u4 *	Result address for selected PWM register	Any non-zero core processor data address

10.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

10.4.7.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
u2 count;
...

/* Create a PWM version A device handle */
pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM0;

/* Get counter register value for PWM version A */
retval = PWM_A_GetRegister(
                           pwmptr,                /* PWM Device Handle */
                           PWM_A_PWMCTR_SWITCH,   /* Register Selector */
                           &count                 /* Returned Value    */
                           );

/* Perform an optional check on the PWM_A_GetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

10.4.8 PWM_A_SetRegister

The following paragraphs describe the PWM_A_SetRegister function.

10.4.8.1 Description

PWM_A_SetRegister sets a PWM register. Call this function at any time.

10.4.8.2 Prototype

```
ddErr_t PWM_A_SetRegister (
                    pPWM_A_t                    PWMPtr,
                    PWM_A_RegisterSwitch_t      PWM_A_RegisterSwitch,
                    u4                          RegisterValue
                    );
```

10.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
PWMPtr	pPWM_A_t	PWM base address associated with this driver	Any non-zero core processor data address
PWM_A_RegisterSwitch	PWM_A_RegisterSwitch_t	Selects among PWM registers	PWM_A_PWMCR_SWITCH PWM_A_PWMPR_SWITCH PWM_A_PWMWR_SWITCH PWM_A_PWMCTR_SWITCH
RegisterValue	u4	Data to copy into selected PWM register	Any 32-bit value

10.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

10.4.8.5 Code Usage Example

```
/* Include the PWM version A header file */
#include "pwm_a.h"
...
ddErr_t retval;
...

/* Create a PWM version A device handle */
pPWM_A_t pwmptr = (pPWM_A_t) __PWS_PWM0;

/* Set load for new period in control register for PWM version A */
retval = PWM_A_SetRegister(
                  pwmptr,                       /* PWM Device Handle*/
                  PWM_A_PWMCR_SWITCH,           /* Register Selector*/
                  pwmptr->PWMCR | PWM_A_LOAD_MASK /* Value to Set */
                  );

/* Perform an optional check on the PWM_A_SetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Section 11 TRM_A Level 1

This section describes the functions of a TRM_A level 1 device driver.

11.1 Overview

The timer/reset module (TRM) and its accompanying device driver (TRM_A) are used on the M•CORE MMC2001. The TRM implements several time-related functions. Level 1 service is designed to allow the programmer to deal with the TRM on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The TRM consists of the following:

- Reset Source/Chip Configuration — This submodule controls reset operation. The reset source/chip configuration register (RSCR) gives the state of the reset sources and serves to control the CLKOUT pin.
- Time-Of-Day Timer (TOD) — This submodule consists of a free-running 40-bit counter clocked by a 256-Hz signal derived from the low-frequency reference clock oscillator on the chip.
- Watchdog Timer (WD) — This submodule consists of a 6-bit modulus down-counter clocked by a 2-Hz signal derived from the low-frequency reference clock oscillator on the chip.
- Programmable Interval Timer (PIT) — This submodule is a 16-bit down counter clocked by an 8192 Hz signal derived from the low-frequency reference clock oscillator on the chip.

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the TRM.

11.2 TRM_A Level 1 Device Driver Functions

The TRM_A level 1 device driver software provides the following functions:

- Initializes the reset source/chip configuration (RSCR) register
- Enables/disables TOD alarm and alarm interrupt, gets status of and clears TOD alarm IRQ flag
- Initializes the WD and performs service sequence
- Initializes and enables/disables the PIT, sets a new modulus in the PIT data register (ITDR), enables/disables interrupts, gets status, and clears IRQ flag status
- Gets and sets TRM registers

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.3 Data Type Definitions

The following paragraphs show abstract data type and enumerated data type definitions.

11.3.1 Abstract Data Type Definitions

The following are TRM_A abstract data types.

11.3.1.1 TRM_A_t - TRM Register Structure

The following structure definition corresponds to the TRM register block layout.

```
/* register structure */
typedef volatile struct {
   u4  RSCR;      /* Reset source / Chip configuration */
   u4  TODCSR;    /* TOD control/status                */
   u4  TODSR;     /* TOD seconds                       */
   u4  TODFR;     /* TOD fraction                      */
   u4  TODSAR;    /* TOD seconds alarm                 */
   u4  TODFAR;    /* TOD fraction alarm                */
   u4
       reserved;
   u4  WCR;       /* Watchdog control                  */
   u4  WSR;       /* Watchdog service                  */
   u4  ITCSR;     /* PIT control/status                */
   u4  ITDR;      /* PIT data                          */
   u4  ITADR;     /* PIT alternate data                */
   } TRM_A_t, *pTRM_A_t;
```

11.3.2 Enumerated Data Type Definitions

The following are TRM_A enumerated data type definitions.

11.3.2.1 TRM_A_Register_t - TRM Register Switch

These values are used to select among TRM registers.

```
typedef enum{
      TRM_A_RSCR_REGISTER,     /* Reset Source/Chip Configuration */
      TRM_A_TODCSR_REGISTER,   /* TOD Control/Status Register */
      TRM_A_TODSR_REGISTER,    /* TOD Seconds Register */
      TRM_A_TODFR_REGISTER,    /* TOD Fraction Register */
      TRM_A_TODSAR_REGISTER,   /* TOD Seconds Alarm Register */
      TRM_A_TODFAR_REGISTER,   /* TOD Fraction Alarm Register */
      TRM_A_WCR_REGISTER,      /* Watchdog Control Register */
      TRM_A_WSR_REGISTER,      /* Watchdog Service Register */
      TRM_A_ITCSR_REGISTER,    /* PIT Control/Status Register */
      TRM_A_ITDR_REGISTER,     /* PIT Data Register */
      TRM_A_ITADR_REGISTER     /* PIT Alternate Data Register */
      } TRM_A_Register_t;
```


Freescale Semiconductor, Inc.

11.4 API Definitions

The following paragraphs provide level 1 API definitions for a TRM_A device driver.

11.4.1 TRM_A_InitRSCR

The following paragraphs describe the TRM_A_InitRSCR function.

11.4.1.1 Description

TRM_A_InitRSCR initializes the reset source/chip configuration (RSCR) register. This allows the user to specify the value written to both of the software-modifiable bits in the RSCR. This function may be called at any time.

11.4.1.2 Prototype

```
ddErr_t TRM_A_InitRSCR      (
                            pTRM_A_t        TRMPtr,
                            bool            LowRefClk,
                            bool            ClockOutEnabled
                            );
```

11.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
LowRefClk	bool	Determines whether CLKOUT source is LOW_REFCLK or HIGH_REFCLK	TRUE = CLKOUT source is LOW_REFCLK FALSE = CLKOUT source is HIGH_REFCLK
ClockOutEnabled	bool	Determines whether CLKOUT is enabled or disabled	TRUE = CLKOUT is enabled FALSE = CLKOUT is disabled

11.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.1.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Init RSCR TRM version A */
  retval = TRM_A_InitRSCR(
                              trmptr,     /* TRM Device Handle */
                              FALSE,      /* CKOS boolean */
                              FALSE       /* CKOE bollean */
                              );

  /* Perform an optional check on the TRM_A_InitRSCR() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.2 TRM_A_GetTODAlarmStatus

The following paragraphs describe the TRM_A_GetTODAlarmStatus function.

11.4.2.1 Description

TRM_A_GetTODAlarmStatus gets the status of the TOD alarm interrupt flag. This function may be called to determine if a TOD interrupt has occurred.

11.4.2.2 Prototype

```
ddErr_t TRM_A_GetTODAlarmStatus   (
                                  pTRM_A_t        TRMPtr,
                                  bool            *ResultPtr
                                  );
```

11.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
ResultPtr	bool *	Address of the return value	Any non-zero core processor data address

11.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.2.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Create a variable to hold the TOD Alarm Status */
  bool results;

  /* Get Time-of-Day Alarm Status TRM version A */
  retval = TRM_A_GetTODAlarmStatus(
                              trmptr,     /* TRM Device Handle */
                              &results    /* Pointer to variable */
                              );

  /* Perform an optional check on the TRM_A_GetTODAlarmStatus() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.3 TRM_A_ClearTODAlarmInterrupt

The following paragraphs describe the TRM_A_ClearTODAlarmInterrupt function.

11.4.3.1 Description

TRM_A_ClearTODAlarmInterrupt clears TOD alarm IRQ flag bit. This function may be called after a TOD interrupt occurs.

11.4.3.2 Prototype

```
ddErr_t TRM_A_ClearTODAlarmInterrupt    (
                                        pTRM_A_t        TRMPtr,
                                        );
```

11.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address

11.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.3.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Clear Time-of-Day Alarm Interrupt TRM version A */
  retval = TRM_A_ClearTODAlarmInterrupt(
                                    trmptr      /* TRM Device Handle */
                                    );

  /* Perform an optional check on the TRM_A_ClearTODAlarmInterrupt() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.4 TRM_A_ControlTODAlarmEnable

The following paragraphs describe the TRM_A_ControlTODAlarmEnable function.

11.4.4.1 Description

TRM_A_ControlTODAlarmEnable enables or disables the TOD alarm. This function may be called after TRM_A_SetRegister has been called.

11.4.4.2 Prototype

```
ddErr_t TRM_A_ControlTODAlarmEnable    (
                                        pTRM_A_t        TRMPtr,
                                        bool            TODEnable
                                        );
```

11.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
TODEnable	bool	Enables/disables TOD alarm switch	TRUE = enable TOD alarm switch FALSE = disable TOD alarm switch

11.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.4.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Control Time-of-Day Alarm Enable TRM version A */
  retval = TRM_A_ControlTODAlarmEnable(
                                  trmptr,   /* TRM Device Handle */
                                  TRUE      /* TOD Enable */
                                  );

  /* Perform an optional check on the TRM_A_ControlTODAlarmEnable() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.5 TRM_A_ControlTODAlarmInterrupt

The following paragraphs describe the TRM_A_ControlTODAlarmInterrupt function.

11.4.5.1 Description

TRM_A_ControlTODAlarmInterrupt enables or disables the TOD alarm interrupt. This function may be called after TRM_A_SetRegister has been called.

11.4.5.2 Prototype

```
ddErr_t TRM_A_ControlTODAlarmInterrupt    (
                                          pTRM_A_t        TRMPtr,
                                          bool            InterruptEnable
                                          );
```

11.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
InterruptEnable	bool	Enables/disables TOD alarm interrupt switch	TRUE = enable TOD FALSE = disable TOD

11.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.5.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Control Time-of-Day Alarm Interrupt TRM version A */
  retval = TRM_A_ControlTODAlarmInterrupt(
                              trmptr,     /* TRM Device Handle */
                              TRUE        /* TOD Interrupt Enable */
                              );

  /* Perform an optional check on the TRM_A_ControlTODAlarmInterrupt() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.6 TRM_A_InitWD

The following paragraphs describe the TRM_A_InitWD function.

11.4.6.1 Description

TRM_A_InitWD initializes the watchdog timer subsystem. The first write of a one to a watchdog control register (WCR) bit — stop enable, WD enable, doze enable, or debug enable — after system reset “freezes” that control bit to a one. The WCR time-out field can be written multiple times but only the lower six bits of the time-out value are meaningful.

11.4.6.2 Prototype

```
ddErr_t TRM_A_InitWD        (
                            pTRM_A_t        TRMPtr,
                            u1              Timeout,
                            bool            StopEnable,
                            bool            WDEnable,
                            bool            DebugEnable,
                            bool            DozeEnable
                            );
```

11.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
Time-out	u1	6-bit time-out value in units of 0.5 seconds	0 to 63
StopEnable	bool	Determines whether the WD operates in stop mode	TRUE = stops in STOP mode FALSE = does not stop in STOP mode
WDEnable	bool	Determines whether the WD is enabled or not	TRUE = enables WD FALSE = does not enable WD
DebugEnable	bool	Determines whether the WD stops in DEBUG mode	TRUE = stops in DEBUG mode FALSE = does not stop in DEBUG mode
DozeEnable	bool	Determines whether the WD stops in DOZE mode	TRUE = stops in DOZE mode FALSE = does not stop in DOZE mode

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
TRM_A_BAD_TIMEOUT_VAL	Watchdog time-out out of range

11.4.6.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Initialize Watchdog TRM version A */
  retval = TRM_A_InitWD(
                  trmptr,     /* TRM Device Handle */
                  63,         /* Watchdog Timeout value */
                  FALSE,      /* Watchdog not affected in Stop mode */
                  TRUE,       /* Watchdog Enabled */
                  FALSE,      /* Watchdog not affected in Debug mdoe */
                  FALSE       /* Watchdog disabled in Doze Mode */
                  );

  /* Perform an optional check on the TRM_A_InitWD() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.7 TRM_A_ServiceWD

The following paragraphs describe the TRM_A_ServiceWD function.

11.4.7.1 Description

TRM_A_ServiceWD performs the required watchdog service sequence of writing 0x5555 followed by 0xAAAA to the watchdog service register (WSR). Writing this service sequence keeps the watchdog from causing a reset. This function may be called as often as the time-out period selected during the initialization call.

11.4.7.2 Prototype

```
ddErr_t TRM_A_ServiceWD        (
                               pTRM_A_t        TRMPtr
                               );
```

11.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address

11.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

11.4.7.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Service Watchdog TRM version A */
  retval = TRM_A_ServiceWD(
                              trmptr      /* TRM Device Handle */
                              );

  /* Perform an optional check on the TRM_A_ServiceWD() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.8 TRM_A_InitPIT

The following paragraphs describe the TRM_A_InitPIT function.

11.4.8.1 Description

TRM_A_InitPIT initializes the programmable interval timer (PIT).

11.4.8.2 Prototype

```
ddErr_t TRM_A_InitPIT        (
                             pTRM_A_t        TRMPtr,
                             bool            StopEnable,
                             bool            DozeEnable,
                             bool            DebugEnable,
                             bool            ReloadEnable
                             );
```

11.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
StopEnable	bool	Determines whether the PIT operates in STOP mode	TRUE = operates in STOP mode FALSE = does not operate in STOP mode
DozeEnable	bool	Determines whether the PIT operates in DOZE mode	TRUE = operates in DOZE mode FALSE = does not operate in DOZE mode
DebugEnable	bool	Determines whether the PIT operates in DEBUG mode	TRUE = operates in DEBUG mode FALSE = does not operate in DEBUG mode
ReloadEnable	bool	Determines whether the PIT rolls over to a programmed interval or at 0xFFFF	TRUE = counter rolls over to programmed interval FALSE = counter rolls over to 0xFFFF

11.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.8.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Initialize Interval Timer TRM version A */
  retval = TRM_A_InitPIT(
                  trmptr,     /* TRM Device Handle */
                  FALSE,      /* PIT not affected in Stop mode */
                  FALSE,      /* PIT not affected in Doze mode */
                  FALSE,      /* PIT not affected in Debug mode */
                  TRUE        /* PIT is reloaded from Modulus Latch */
                  );

  /* Perform an optional check on the TRM_A_InitPIT() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.9 TRM_A_SetPITModulus

The following paragraphs describe the TRM_A_SetPITModulus function.

11.4.9.1 Description

TRM_A_SetPITModulus sets the PIT modulus (the number of states through which the PIT counts) in the PIT data register (ITDR). This function may be called after TRM_A_InitPIT.

11.4.9.2 Prototype

```
ddErr_t TRM_A_SetPITModulus     (
                                 pTRM_A_t        TRMPtr,
                                 u2              Modulus,
                                 bool            CounterOverwriteEnable
                                 );
```

11.4.9.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
Modulus	u2	PIT modulus value	16-bit unsigned
CounterOverwriteEnable	bool	Determines whether the modulus latch is transparent or a holding register	TRUE = Modulus latch is transparent FALSE = Modulus latch is a holding register

11.4.9.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.9.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Set Interval Timer Modulus TRM version A */
  retval = TRM_A_SetPITModulus(
                               trmptr,   /* TRM Device Handle */
                               0x1000,   /* PIT Data Register Value */
                               FALSE    /* Modulus Latch is Transparent */
                               );

  /* Perform an optional check on the TRM_A_SetPITModulus() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.10 TRM_A_ClearPITInterrupt

The following paragraphs describe the TRM_A_ClearPITInterrupt function.

11.4.10.1 Description

TRM_A_ClearPITInterrupt clears the interval timer interrupt flag. This function may be called after TRM_A_InitPIT.

11.4.10.2 Prototype

```
ddErr_t TRM_A_ClearPITInterrupt     (
                                    pTRM_A_t            TRMPtr
                                    );
```

11.4.10.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address

11.4.10.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.10.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Clear Interval Timer Interrupt TRM version A */
  retval = TRM_A_ClearPITInterrupt(
                                   trmptr      /* TRM Device Handle */
                                   );

  /* Perform an optional check on the TRM_A_ClearPITInterrupt() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.11 TRM_A_ControlPITEnable

The following paragraphs describe the TRM_A_ControlPITEnable function.

11.4.11.1 Description

TRM_A_ControlPITEnable enables or disables the PIT. This function may be called after TRM_A_InitPIT.

11.4.11.2 Prototype

```
ddErr_t TRM_A_ControlPITEnable    (
                                  pTRM_A_t        TRMPtr,
                                  bool            PITEnable
                                  );
```

11.4.11.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
PITEnable	bool	Enables/disables the PIT switch	TRUE = enable PIT switch FALSE = disable PIT switch

11.4.11.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

11.4.11.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Control Interval Timer Enable TRM version A */
  retval = TRM_A_ControlPITEnable(
                                    trmptr,   /* TRM Device Handle */
                                    TRUE      /* PIT Enable */
                                    );

  /* Perform an optional check on the TRM_A_ControlPITEnable() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.12 TRM_A_ControlPITInterrupt

The following paragraphs describe the TRM_A_ControlPITInterrupt function.

11.4.12.1 Description

TRM_A_ControlPITInterrupt enables or disables the PIT interrupt. This function may be called after TRM_A_InitPIT and TRM_A_SetPITModulus.

11.4.12.2 Prototype

```
ddErr_t TRM_A_ControlPITInterrupt   (
                                     pTRM_A_t        TRMPtr,
                                     bool            InterruptEnable
                                     );
```

11.4.12.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
InterruptEnable	bool	Enables/disables the PIT interrupt switch	TRUE = enable PIT interrupt switch FALSE = disable PIT interrupt switch

11.4.12.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.12.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Control Interval Timer Interrupt TRM version A */
  retval = TRM_A_ControlPITInterrupt(
                              trmptr,     /* TRM Device Handle */
                              TRUE        /* PIT Interrupt Enable */
                              );

  /* Perform an optional check on the TRM_A_ControlPITInterrupt() */
  /* return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.13 TRM_A_GetPITStatus

The following paragraphs describe the TRM_A_GetPITStatus function.

11.4.13.1 Description

TRM_A_GetPITStatus gets the current interrupt status of the PIT submodule. This function may be called to determine if a PIT interrupt has occurred.

11.4.13.2 Prototype

```
ddErr_t TRM_A_GetPITStatus      (
                                 pTRM_A_t        TRMPtr,
                                 bool            *ResultPtr
                                 );
```

11.4.13.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
ResultPtr	bool *	Address of the return value	Any non-zero core processor data address

11.4.13.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
TRM_A_BAD_RESULT_ADDR	Result pointer was NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.13.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Create a variable to hold the PIT Interrupt Status */
  bool results;

  /* Get Interval Timer Interrupt Status TRM version A */
  retval = TRM_A_GetPITStatus(
                              trmptr,     /* TRM Device Handle */
                              &results    /* Pointer to variable */
                              );

  /* Perform an optional check on the TRM_A_GetPITStatus() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.14 TRM_A_GetRegister

The following paragraphs describe the TRM_A_GetRegister function.

11.4.14.1 Description

TRM_A_GetRegister copies the contents of the user-specified TRM register to the user-specified variable. This function may be called any time.

11.4.14.2 Prototype

```
ddErr_t TRM_A_GetRegister   (
                            pTRM_A_t            TRMPtr,
                            u4                  *ResultPtr,
                            TRM_A_Register_t    RegisterSwitch
                            );
```

11.4.14.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
ResultPtr	u4 *	Address of the return value	Any non-zero core processor data address
RegisterSwitch	TRM_A_Register_t	Switch to select among TRM registers	Any member of TRM_A_Register_t

11.4.14.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_BAD_RESULT_ADDR	Result address is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

11.4.14.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Create a variable to hold the TRM Register value */
  u4 trmregister;

  /* Get Register TRM version A */
  retval = TRM_A_GetRegister(
                        trmptr,                 /* TRM Device Handle */
                        &trmregister            /* Pointer to variable */
                        TRM_A_ITADR_REGISTER    /* TRM Register Switch */
                        );

  /* Perform an optional check on the TRM_A_GetRegister() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

11.4.15 TRM_A_SetRegister

The following paragraphs describe the TRM_A_SetRegister function.

11.4.15.1 Description

TRM_A_SetRegister allows a 32-bit value to be written to any user-specified TRM register, except the PIT alternate data register (ITADR), which is read only. To determine the appropriate bit to write to any TRM register, refer to the *MMC2001 Reference Manual* (MMC2001RM/D). This function can be called any time.

11.4.15.2 Prototype

```
ddErr_t TRM_A_SetRegister   (
                            pTRM_A_t            TRMPtr,
                            u4                  RegisterValue,
                            TRM_A_Register_t    RegisterSwitch
                            );
```

11.4.15.3 Arguments

Formal Name	Type	Description	Range/Valid
TRMPtr	pTRM_A_t	TRM base address associated with this driver	Any non-zero core processor data address
RegisterValue	u4	Register value	32-bit unsigned
RegisterSwitch	TRM_A_Register_t	Switch to select among TRM registers	Any member of TRM_A_Register_t, except TRM_A_ITADR_REGISTER

11.4.15.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

11.4.15.5 Code Usage Example

```
/* Include the TRM version A header file */
#include "trm_a.h"
...
ddErr_t retval;
...

  /* Create a TRM version A Device Handle */
  pTRM_A_t trmptr = (pTRM_A_t) __PWS_TIMER;

  /* Set Register TRM version A */
  retval = TRM_A_SetRegister(
                       trmptr,                 /* TRM Device Handle */
                       0x1000,                 /* Data Value */
                       TRM_A_ITDR_REGISTER     /* TRM Register Switch */
                       );

  /* Perform an optional check on the TRM_A_SetRegister() return code */
  if (retval != DD_ERR_NONE)
  {
       /* do application-specific error status reporting */
  }
```

Freescale Semiconductor, Inc.

Section 12 UART_A Level 1

This section describes the functions of a UART_A level 1 device driver.

12.1 Overview

The universal asynchronous receiver/transmitter (UART) and its accompanying device driver (UART_A) are used on the M•CORE MMC2001. Level 1 service is designed to allow the programmer to deal with the UART on a near-register level, but without concern for exact positions of registers and fields, or detailed sequences of register operations.

The UART module provides asynchronous serial communication with external devices such as modems and other computers. It consists of two independent and almost identical channels, differing only in the request-to-send (RTS) and clear-to-send (CTS) functions of UART0. Features of the UART module include:

- Full-duplex operation
- Direct support of Infrared Data Association mechanism
- 16-entry first in, first out (FIFO) buffers for transmit and receive
- 16x bit clock generator for bit rates of 300 bits per second (bps) to 115.2 Kbps
- RTS interrupt with wake up from stop capability
- Low-power modes

Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the UART.

12.2 UART_A Level 1 Device Driver Functions

The UART_A level 1 device driver software provides the following functions:

- Initializes, enables, and disables UART operations
- Enables and disables selected UART interrupts
- Transmits and receives UART data
- Enables and disables UART loopback mode
- Transmits break sequences
- Obtains UART receiver status
- Obtains RTS/CTS status, if applicable

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

- Returns status information to the application program about the result of requested operations.
- Gets and sets all UART registers

12.3 Data Type Definitions

The following paragraphs show abstract data type definitions, enumerated data type definitions, and initialization default macros.

12.3.1 Abstract Data Type Definitions

The following are UART_A abstract data types.

12.3.1.1 UART_A_t - UART Register Definitions

The following structure definition corresponds to the UART register block layout.

```
typedef struct
{
    volatile u2  URX;          /* UART Receive Register */
    u2  reserved0;             /* reserved */
    u2  reserved1[30];         /* reserved */
    volatile    u2  UTX;       /* UART Transmit Register */
    u2  reserved2;             /* reserved */
    u2  reserved3[30];         /* reserved */
    volatile    u2  UCR1;      /* UART Control Register 1 */
    u2  UCR2;                  /* UART Control Register 2 */
    u2  UBRGR;                 /* UART Bit Rate Generator Register */
    u2  USR;                   /* UART Status Register */
    u2  UTS;                   /* UART Test Register */
    u2  UPCR;                  /* UART Port Control Register */
    u2  UDDR;                  /* UART Data Direction Register */
    u2  UPDR;                  /* UART Port Data Register */
    u2  reserved4[1976];       /* reserved */
} UART_A_t, *pUART_A_t;
```

12.3.2 Enumerated Data Type Definitions

The following are UART_A enumerated data types.

Freescale Semiconductor, Inc.

12.3.2.1 UART_A_Size_t - UART Data Frame Type

These values are used to set the data word size in UART control register 2 (UCR2).

```
typedef enum
{                               /* Data word sizes in frame */
      UART_A_DATA_7,            /* 7-bit data frame */
      UART_A_DATA_8             /* 8-bit data frame */
} UART_A_Size_t;
```

12.3.2.2 UART_A_Parity_t - UART Parity Selection

These values are used to establish the parity setting in UCR2.

```
typedef enum
{                               /* Parity settings */
      UART_A_PARITY_NONE,       /* No parity checking */
      UART_A_PARITY_ODD,        /* Odd parity */
      UART_A_PARITY_EVEN        /* Even parity */
} UART_A_Parity_t;
```

12.3.2.3 UART_A_Trig_t - UART FIFO Trigger Level

These values are used to set the FIFO trigger levels in UART control register 1 (UCR1).

```
typedef enum
{                         /* FIFO threshold triggers */
      UART_A_TRIG_1,      /* One character FIFO interrupt trigger */
      UART_A_TRIG_4,      /* Four character FIFO interrupt trigger */
      UART_A_TRIG_8,      /* Eight character FIFO interrupt trigger */
      UART_A_TRIG_14      /* Fourteen character FIFO interrupt trigger */
} UART_A_Trig_t;
```

12.3.2.4 UART_A_TxRx_t - UART Transmitter/Receiver Selection

These values are used to select between transmitter and receiver operations.

```
typedef enum
{                               /* Transmitter/receiver selection */
      UART_A_TXRX_NONE,         /* Neither transmitter nor receiver */
      UART_A_TX,                /* UART transmitter only */
      UART_A_RX,                /* UART receiver only */
      UART_A_TXRX,              /* UART transmitter and receiver */
      UART_A_MODULE             /* UART module */
} UART_A_TxRx_t;
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

12.3.2.5 UART_A_RegisterSwitch_t - UART Register Selection

```
typedef enum
{                                /* Get and set register selections  */
    UART_A_URX_SWITCH,           /* Select URX */
    UART_A_UTX_SWITCH = 32,      /* Select UTX */
    UART_A_UCR1_SWITCH = 64,     /* Select UCR1 */
    UART_A_UCR2_SWITCH,          /* Select UCR2 */
    UART_A_UBRGR_SWITCH,         /* Select UBRGR */
    UART_A_USR_SWITCH,           /* Select USR */
    UART_A_UTS_SWITCH,           /* Select UTS */
    UART_A_UPCR_SWITCH,          /* Select UPCR */
    UART_A_UDDR_SWITCH,          /* Select UDDR */
    UART_A_UPDR_SWITCH           /* Select UPDR */
} UART_A_RegisterSwitch_t;
```

12.3.3 Initialization Default Macros

The following macros may be used as UART_A_Init parameter values. In particular, the macro UART_A_INIT_DEFAULT may be used in place of a call to UART_A_Init. The only explicit argument to UART_A_INIT_DEFAULT is the base address of the UART to be initialized.

```
#define UART_A_DEFAULT_SYS_CLOCK    16380000          /* 16.38 MHz  */
#define UART_A_DEFAULT_BAUD_RATE    115200            /* 115.2K bps */
#define UART_A_DEFAULT_DIVIDER      8                 /* 115.2K bps */
#define UART_A_DEFAULT_SIZE         UART_DATA_8       /* 8-bit data */
#define UART_A_DEFAULT_PARITY       UART_PARITY_NONE  /* no parity  */
#define UART_A_DEFAULT_STOP_BITS    1                 /* 1 stop bit */
#define UART_A_DEFAULT_RX_TRIG      UART_TRIG_8 /* 8 char threshold */
#define UART_A_DEFAULT_TX_TRIG      UART_TRIG_8 /* 8 char threshold */
#define UART_A_DEFAULT_RTS_INT      FALSE       /* RTS int disabled */
#define UART_A_DEFAULT_DOZE         TRUE        /* DOZE enabled */
#define UART_A_DEFAULT_FLOW         FALSE       /* handshake disabled */
#define UART_A_DEFAULT_UART_PINS    (UART_RXD | UART_TXD)
#define UART_A_DEFAULT_OUTPUT_PINS  (UART_RTS | UART_CTS)

#define UART_A_INIT_DEFAULT(UARTPtr)        \
    UART_A_Init(UARTPtr,                    \
                UART_A_DEFAULT_DIVIDER,     \
                UART_A_DEFAULT_SIZE,        \
                UART_A_DEFAULT_PARITY,      \
                UART_A_DEFAULT_STOP_BITS,   \
                UART_A_DEFAULT_RX_TRIG,     \
                UART_A_DEFAULT_TX_TRIG,     \
                UART_A_DEFAULT_RTS_INT,     \
                UART_A_DEFAULT_DOZE,        \
                UART_A_DEFAULT_FLOW,        \
                UART_A_DEFAULT_UART_PINS,   \
                UART_A_DEFAULT_OUTPUT_PINS)
```


Freescale Semiconductor, Inc.

12.4 API Definitions

The following paragraphs provide level 1 API definitions for a UART_A device driver.

12.4.1 UART_A_Init

The following paragraphs describe the UART_A_Init function.

12.4.1.1 Description

UART_A_Init initializes the UART for operation. Execute this function once at system reset. Do not call this function when the UART is enabled; however, it can be called when the UART is disabled.

12.4.1.2 Prototype

```
ddErr_t UART_A_Init   (
                      pUART_A_t          UARTPtr,
                      u2                 Divider,
                      UART_A_Size_t      Size,
                      UART_A_Parity_t    Parity,
                      u1                 StopBits,
                      UART_A_Trig_t      RxTrig,
                      UART_A_Trig_t      TxTrig,
                      bool               RTSInt,
                      bool               Doze,
                      bool               Flow,
                      u1                 UARTPins,
                      u1                 OutputPins
                      );
```

Freescale Semiconductor, Inc.

12.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Divider	u2	Clock divider is derived from the following equation: *Divider = System Clock / Nominal Rate / 16* where: System Clock is the clock frequency in MHz Nominal Rate is the standard baud rate value (e.g. 110, 1200, 2400, 9600, 19200 etc.)	0 to 4095
Size	UART_A_Size_t	Format is based on the number of data bits in the frame	UART_A_DATA_7 UART_A_DATA_8
Parity	UART_A_Parity_t	Parity type	UART_A_PARITY_NONE, UART_A_PARITY_ODD, UART_A_PARITY_EVEN
StopBits	u1	Number of stop bits per frame	One or two
RxTrig	UART_A_Trig_t	FIFO threshold for receiver interrupts	UART_A_TRIG_1 UART_A_TRIG_4 UART_A_TRIG_8 UART_A_TRIG_14
TxTrig	UART_A_Trig_t	FIFO threshold for transmitter interrupts	UART_A_TRIG_1 UART_A_TRIG_4 UART_A_TRIG_8 UART_A_TRIG_14
RTSInt	bool	Indicates whether to enable RTS delta interrupts	TRUE = enable FALSE = do not enable
Doze	bool	Determines whether the UART is affected by CPU doze instruction	TRUE = doze when CPU dozes FALSE = do not doze
Flow	bool	Determines whether to enable UART hardware flow control	TRUE = enable FALSE = do not enable
UARTPins	u1	Determines which UART pins are to be used by the UART module. The remaining pins may be used for general-purpose I/O.	UART_A_RXD UART_A_TXD UART_A_RTS UART_A_CTS
OutputPins	u1	Determines which general-purpose I/O pins are to be used as outputs. Remaining pins will default to inputs.	UART_A_RXD UART_A_TXD UART_A_RTS UART_A_CTS

Freescale Semiconductor, Inc.

12.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_CLOCK_DIVIDER	Clock divider parameter is out of range
DD_ERR_INVALID_SIZE	Size parameter is out of range
UART_A_ERR_INVALID_PARITY	Bad parity selection parameter
UART_A_ERR_INVALID_STOP_BITS	Stop bits not one or two
UART_A_ERR_INVALID_TRIGGER	FIFO trigger value not within range
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

12.4.1.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Init UART version A */
    retval = UART_A_Init(
                        uartptr,            /* UART Device Handle */
                        8,                  /* Clock Divider      */
                        UART_A_DATA_8,      /* Data Size          */
                        UART_A_PARITY_NONE, /* Parity             */
                        1,                  /* Stop Bits          */
                        UART_A_TRIG_8,      /* Rx FIFO Trigger    */
                        UART_A_TRIG_8,      /* Tx FIFO Trigger    */
                        FALSE,              /* RTS Interrupt      */
                        TRUE,               /* Doze Mode          */
                        FALSE,              /* Flow Control       */
                        (UART_A_RXD_MASK |  /* UART Pins          */
                        UART_A_TXD_MASK),
                        (UART_A_RTS_MASK |  /* Output Pins        */
                        UART_A_CTS_MASK)
                        );

    /* Perform an optional check on the UART_A_Init() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```


For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

12.4.2 UART_A_Enable

The following paragraphs describe the UART_A_Enable function.

12.4.2.1 Description

UART_A_Enable enables the UART transmitter, receiver, or both simultaneously. It can also activate the UART without enabling the transmitter or receiver. This function may be called any time after UART_A_Init.

12.4.2.2 Prototype

```
ddErr_t UART_A_Enable      (
                           pUART_A_t          UARTPtr,
                           UART_A_TxRx_t      TxRx
                           );
```

12.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to enable the UART transmitter, receiver, both, or neither (e.g. only enable UART module).	UART_A_TX UART_A_ RX UART_A_TXRX UART_A_MODULE

12.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_ERR_INVALID_TXRX	Transmit/receive selection is out of range

Freescale Semiconductor, Inc.

12.4.2.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Enable UART version A */
    retval = UART_A_Enable(
                            uartptr,         /* UART Device Handle */
                            UART_A_TXRX      /* Transmit & Receive */
                            );

    /* Perform an optional check on the UART_A_Enable() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.3 UART_A_Disable

The following paragraphs describe the UART_A_Disable function.

12.4.3.1 Description

UART_A_Disable disables the UART transmitter, receiver, or both simultaneously. It can also disable the entire UART. This function may be called any time after UART_A_Init.

12.4.3.2 Prototype

```
ddErr_t UART_A_Disable      (
                            pUART_A_t          UARTPtr,
                            UART_A_TxRx_t      TxRx
                            );
```

12.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to disable the UART transmitter, receiver, both, or the entire UART module.	UART_A_TX UART_A_ RX UART_A_TXRX UART_A_MODULE

12.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_ERR_INVALID_TXRX	Transmit/receive selection is out of range

Freescale Semiconductor, Inc.

12.4.3.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Disable UART version A */
    retval = UART_A_Disable(
                        uartptr,         /* UART Device Handle */
                        UART_A_TX        /* Transmit Only      */
                        );

    /* Perform an optional check on the UART_A_Disable() return code */
    if ( retval != DD_ERR_NONE )
    {
        /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.4 UART_A_IntEnable

The following paragraphs describe the UART_A_IntEnable function.

12.4.4.1 Description

UART_A_IntEnable enables interrupts for the UART transmitter, receiver, or both simultaneously. It can also enable RTS delta interrupts. This function may be called any time after UART_A_Init.

12.4.4.2 Prototype

```
ddErr_t UART_A_IntEnable    (
                            pUART_A_t          UARTPtr,
                            UART_A_TxRx_t      TxRx,
                            bool               RTSInt
                            );
```

12.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to enable interrupts for the UART transmitter, receiver, both, or neither.	UART_A_TX UART_A_ RX UART_A_TXRX UART_A_TXRX_NONE
RTSInt	bool	Indicates whether to enable the RTS delta interrupt.	TRUE = enable FALSE = do not enable

12.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_ERR_INVALID_TXRX	Transmit/receive selection is out of range

Freescale Semiconductor, Inc.

12.4.4.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Interrupt Enable UART version A */
    retval = UART_A_IntEnable(
                              uartptr,       /* UART Device Handle */
                              UART_A_RX,     /* Receive Only       */
                              FALSE          /* No RTS Interrupt   */
                              );

    /* Perform an optional check on the UART_A_IntEnable() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.5 UART_A_IntDisable

The following paragraphs describe the UART_A_IntDisable function.

12.4.5.1 Description

UART_A_IntDisable disables interrupts for the UART transmitter, receiver, or both simultaneously. It can also disable RTS delta interrupts. This function may be called any time after UART_A_Init.

12.4.5.2 Prototype

```
ddErr_t UART_A_IntDisable    (
                             pUART_A_t          UARTPtr,
                             UART_A_TxRx_t      TxRx,
                             bool               RTSInt
                             );
```

12.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to disable interrupts for the UART transmitter, receiver, both, or neither	UART_A_TX UART_A_ RX UART_A_TXRX UART_A_TXRX_NONE
RTSInt	bool	Indicates whether to disable the RTS delta interrupt	TRUE = disable FALSE = do not disable

12.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_ERR_INVALID_TXRX	Transmit/receive selection is out of range

Freescale Semiconductor, Inc.

12.4.5.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Disable UART version A */
    retval = UART_A_IntDisable(
                              uartptr,    /* UART Device Handle */
                              UART_A_TX,  /* Transmit Only */
                              TRUE        /* Disable RTS Interrupt */
                              );

   /* Perform an optional check on the UART_A_IntDisable() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.6 UART_A_SetDivider

The following paragraphs describe the UART_A_SetDivider function.

12.4.6.1 Description

UART_A_SetDivider sets the UART clock divider. This function may be called any time after UART_A_Init.

12.4.6.2 Prototype

```
ddErr_t UART_A_SetDivider   (
                            pUART_A_t       UARTPtr,
                            u2              Divider
                            );
```

12.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Divider	u2	Clock divider is derived from the following equation: *Divider = System Clock / Nominal Rate / 16* where: System Clock is the clock frequency in MHz Nominal Rate is the standard baud rate value (e.g. 110, 1200, 2400, 9600, 19200 etc.)	0 to 4095

12.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_CLOCK_DIVIDER	Clock divider parameter is out of range

Freescale Semiconductor, Inc.

12.4.6.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Set UART version A Clock Divider */
    retval = UART_A_SetDivider(
                              uartptr,        /* UART Device Handle */
                              8               /* New Clock Divider  */
                              );

   /* Perform an optional check on the UART_A_SetDivider() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.7 UART_A_Receive

The following paragraphs describe the UART_A_Receive function.

12.4.7.1 Description

UART_A_Receive returns the contents of the UART receive register. This function may be called any time after UART_A_Enable.

12.4.7.2 Prototype

```
ddErr_t UART_A_Receive    (
                          pUART_A_t        UARTPtr,
                          u1               *Datap
                          );
```

12.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Datap	u1 *	Pointer to buffer for received data	Any non-zero core processor data address

12.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL
UART_ERR_DATA_PENDING	Data not available
UART_ERR_OVERRUN_ERROR	Previously received datum has not been read
UART_ERR_BREAK_DETECT	Break sequence detected
UART_ERR_FRAMING_ERROR	Framing error detected
UART_ERR_PARITY_ERROR	Parity error detected

Freescale Semiconductor, Inc.

12.4.7.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
u1 receive_data;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Receive UART version A */
    retval = UART_A_Receive(
                            uartptr,         /* UART Device Handle */
                            &receive_data    /* Pointer for Data   */
                            );

    /* Perform an optional check on the UART_A_Receive() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.8 UART_A_Transmit

The following paragraphs describe the UART_A_Transmit function.

12.4.8.1 Description

UART_A_Transmit writes data to the UART transmit register. If previously transmitted data has not been sent, UART_A_Transmit returns a pending data error code. This function may be called any time after UART_A_Enable.

12.4.8.2 Prototype

```
ddErr_t UART_A_Transmit    (
                           pUART_A_t      UARTPtr,
                           u1             Data
                           );
```

12.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Data	u1	Data to be transmitted	0 to 127 if 7-bit data frame, otherwise 0 to 255

12.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_ERR_INVALID_DATA_VALUE	Data value outside frame size
UART_ERR_DATA_PENDING	Data not available

Freescale Semiconductor, Inc.

12.4.8.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Transmit UART version A */
    retval = UART_A_Transmit(
                             uartptr,        /* UART Device Handle */
                             0xff            /* Transmit Data      */
                             );

    /* Perform an optional check on the UART_A_Transmit() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.9 UART_A_ReadPin

The following paragraphs describe the UART_A_ReadPin function.

12.4.9.1 Description

UART_A_ReadPin returns the state of the GPIO pin. This function may be called any time after UART_A_Init.

12.4.9.2 Prototype

```
ddErr_t UART_A_ReadPin    (
                          pUART_A_t        UARTPtr,
                          u1               Pin,
                          bool             *Statep
                          );
```

12.4.9.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Pin	u1	Ordinal number of pin to read	UART_A_RXD_BITNO UART_ A_TXD_BITNO UART_ A_RTS_BITNO UART_ A_CTS_BITNO
Statep	bool *	Pointer to buffer for retrieved pin state	Any non-zero core processor data address

12.4.9.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

Freescale Semiconductor, Inc.

12.4.9.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
bool pin_state;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Read UART version A Pin */
    retval = UART_A_ReadPin(
                            uartptr,          /* UART Device Handle */
                            UART_A_RTS_BITNO, /* Read RTS Pin */
                            &pin_state        /* Ptr to Pin State */
                            );

    /* Perform an optional check on the UART_A_ReadPin() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.10 UART_A_WritePin

The following paragraphs describe the UART_A_WritePin function.

12.4.10.1 Description

UART_A_WritePin writes a state to the GPIO pin. This function may be called any time after UART_A_Init.

12.4.10.2 Prototype

```
ddErr_t UART_A_WritePin    (
                           pUART_A_t      UARTPtr,
                           u1             Pin,
                           bool           State
                           );
```

12.4.10.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Pin	u1	Ordinal number of pin to write	UART_A_RXD_BITNO UART_ A_TXD_BITNO UART_ A_RTS_BITNO UART_ A_CTS_BITNO
State	bool	Pin state to write	TRUE = 1 FALSE = 0

12.4.10.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

Freescale Semiconductor, Inc.

12.4.10.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Write UART version A Pin */
    retval = UART_A_WritePin(
                             uartptr,          /* UART Device Handle  */
                             UART_A_CTS_BITNO, /* Write CTS Pin       */
                             TRUE              /* Pin State to Write */
                             );

    /* Perform an optional check on the UART_A_WritePin() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.11 UART_A_Infrared

The following paragraphs describe the UART_A_Infrared function.

12.4.11.1 Description

UART_A_Infrared enables or disables the infrared interface. This function may be called any time after UART_A_Init.

12.4.11.2 Prototype

```
ddErr_t UART_A_Infrared     (
                            pUART_A_t       UARTPtr,
                            bool            Enable
                            );
```

12.4.11.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Enable	bool	Indicates infrared interface is enabled or disabled	TRUE = infrared enabled FALSE = infrared disabled

12.4.11.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_LOOPBACK_ENABLED	Serial loopback is enabled

Freescale Semiconductor, Inc.

12.4.11.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* UART version A Infrared Interface */
    retval = UART_A_Infrared(
                             uartptr,    /* UART Device Handle */
                             TRUE        /* Enable Infrared Interface */
                             );

    /* Perform an optional check on the UART_A_Infrared() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.12 UART_A_SendBreak

The following paragraphs describe the UART_A_SendBreak function.

12.4.12.1 Description

UART_A_SendBreak sends one break frame before recommencing data transmission. This function may be called any time after UART_A_Init.

12.4.12.2 Prototype

```
ddErr_t UART_A_SendBreak    (
                            pUART_A_t          UARTPtr,
                            );
```

12.4.12.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address

12.4.12.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

12.4.12.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* UART version A Transmit Break Sequence */
    retval = UART_A_SendBreak(
                              uartptr     /* UART Device Handle */
                              );

    /* Perform an optional check on the UART_A_SendBreak() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.13 UART_A_ParityError

The following paragraphs describe the UART_A_ParityError function.

12.4.13.1 Description

UART_A_ParityError forces the transmitter to generate a parity error if parity is enabled. This function may be called any time after UART_A_Init.

12.4.13.2 Prototype

```
ddErr_t UART_A_ParityError    (
                              pUART_A_t        UARTPtr,
                              bool             Enable
                              );
```

12.4.13.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Enable	bool	Indicates whether parity error generation is enabled or disabled	TRUE = parity error generation enabled FALSE = parity error generation disabled

12.4.13.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_PARITY	Parity checking is disabled

Freescale Semiconductor, Inc.

12.4.13.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* UART version A Parity Error Generation */
    retval = UART_A_ParityError(
                              uartptr,/* UART Device Handle */
                              TRUE  /* Parity Error Generation */
                              );

   /* Perform an optional check on the UART_A_ParityError() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.14 UART_A_Loopback

The following paragraphs describe the UART_A_Loopback function.

12.4.14.1 Description

UART_A_Loopback enables or disables the feedback path on the data serial shifter. This function may be called any time after UART_A_Init.

12.4.14.2 Prototype

```
ddErr_t UART_A_Loopback    (
                           pUART_A_t        UARTPtr,
                           bool             Enable
                           );
```

12.4.14.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Enable	bool	Indicates whether looping and feedback path along data serial shifter is enabled or disabled	TRUE = loopback enabled FALSE = loopback disabled

12.4.14.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_IR_ENABLED	Infrared interface is enabled

Freescale Semiconductor, Inc.

12.4.14.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Loopback UART version A */
    retval = UART_A_Loopback(
                             uartptr,    /* UART Device Handle   */
                             TRUE        /* Enable UART Loopback */
                             );

    /* Perform an optional check on the UART_A_Loopback() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.15 UART_A_IrLoopback

The following paragraphs describe the UART_A_IrLoopback function.

12.4.15.1 Description

UART_A_IrLoopback enables or disables the feedback path on the infrared interface. This function may be called any time after UART_A_Init.

12.4.15.2 Prototype

```
ddErr_t UART_A_IrLoopback   (
                            pUART_A_t       UARTPtr,
                            bool            Enable
                            );
```

12.4.15.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Enable	bool	Indicates whether looping from transmitter to receiver in the infrared interface is enabled or disabled	TRUE = loopback enabled FALSE = loopback disabled

12.4.15.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_IR_DISABLED	Infrared interface is disabled

Freescale Semiconductor, Inc.

12.4.15.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Infrared Loopback UART version A */
    retval = UART_A_IrLoopback(
                              uartptr,   /* UART Device Handle     */
                              TRUE       /* UART Infrared Loopback */
                              );

   /* Perform an optional check on the UART_A_IrLoopback() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.16 UART_A_GetStatus

The following paragraphs describe the UART_A_GetStatus function.

12.4.16.1 Description

UART_A_GetStatus returns the contents of the UART status register (USR). If Receiver = true, it returns the upper eight bits of the UART receive data register (URX). This function may be called any time after UART_A_Init.

12.4.16.2 Prototype

```
ddErr_t UART_A_GetStatus    (
                            pUART_A_t       UARTPtr,
                            u2              *Statusp,
                            bool            Receiver
                            );
```

12.4.16.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
Statusp	u2 *	Address of location to store status word	Any non-zero core processor data address
Receiver	bool	Determines whether to return general UART status or receiver status	TRUE = return receiver status FALSE = return general status

12.4.16.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL

Freescale Semiconductor, Inc.

12.4.16.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
u2 status;
...

    /* Create a UART version A Device Handle */
    pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

    /* Get Status UART version A */
    retval = UART_A_GetStatus(
                              uartptr,    /* UART Device Handle */
                              &status,    /* Status Pointer */
                              TRUE        /* Return Receiver Status */
                              );

    /* Perform an optional check on the UART_A_GetStatus() return code */
    if ( retval != DD_ERR_NONE )
    {
         /* do application-specific error status reporting */
    }
```

Freescale Semiconductor, Inc.

12.4.17 UART_A_GetRegister

The following paragraphs describe the UART_A_GetRegister function.

12.4.17.1 Description

UART_A_GetRegister acquires a UART register. This function may be called any time after UART_A_Init.

12.4.17.2 Prototype

```
ddErr_t UART_A_GetStatus    (
                            pUART_A_t                UARTPtr,
                            UART_A_RegisterSwitch    UARTRegisterSwitch,
                            u2                       *GetRegisterPtr
                            );
```

12.4.17.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
UARTRegisterSwitch	UART_A_RegisterSwitch	Select among UART registers	UART_A_URX_SWITCH UART_A_UCR1_SWITCH UART_A_UCR2_SWITCH UART_A_UBRGR_SWITCH UART_A_USR_SWITCH UART_A_UTS_SWITCH UART_A_UPCR_SWITCH UART_A_UDDR_SWITCH UART_A_UPDR_SWITCH
GetRegisterPtr	u2 *	Address of copy of selected UART register	Any non-zero core processor data address

12.4.17.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

12.4.17.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
u2 received;
...

/* Create a UART version A device handle */
pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

/* Get receive register value for UART version A */
retval = UART_A_GetRegister(
                        uartptr,                /* UART Device Handle */
                        UART_A_URX_SWITCH,      /* Register Selector */
                        &received               /* Returned Value */
                        );

/* Perform an optional check on the UART_A_GetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

12.4.18 UART_A_SetRegister

The following paragraphs describe the UART_A_SetRegister function.

12.4.18.1 Description

UART_A_SetRegister sets a UART register. This function may be called any time after UART_A_Init.

12.4.18.2 Prototype

```
ddErr_t UART_A_GetStatus    (
                            pUART_A_t                UARTPtr,
                            UART_A_RegisterSwitch    UARTRegisterSwitch,
                            u2                       RegisterValue
                            );
```

12.4.18.3 Arguments

Formal Name	Type	Description	Range/Valid
UARTPtr	pUART_A_t	UART base address associated with this driver	Any non-zero core processor data address
UARTRegisterSwitch	UART_A_RegisterSwitch	Select among UART registers	UART_A_UTX_SWITCH UART_A_UCR1_SWITCH UART_A_UCR2_SWITCH UART_A_UBRGR_SWITCH UART_A_UTS_SWITCH UART_A_UPCR_SWITCH UART_A_UDDR_SWITCH UART_A_UPDR_SWITCH
RegisterValue	u2	Data to be copied to the selected UART register	Any non-zero core processor data address

12.4.18.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

Freescale Semiconductor, Inc.

12.4.18.5 Code Usage Example

```
/* Include the UART version A header file */
#include "uart_a.h"
...
ddErr_t retval;
...

/* Create a UART version A device handle */
pUART_A_t uartptr = (pUART_A_t) __PWS_UART0;

/* Set loopback enable in test register for UART version A */
retval = UART_A_SetRegister(
                  uartptr,                      /* UART Device Handle*/
                  UART_A_UTS_SWITCH,            /* Register Selector */
                  uartptr->UTS | UTS_LOOP_MASK  /* Value to Set */
                  );

/* Perform an optional check on the UART_A_SetRegister() return code */
if ( retval != DD_ERR_NONE )
{
```

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Section 13 UART_A Level 2

This section describes the functions of a UART_A level 2 device driver.

13.1 Overview

The universal asynchronous receiver/transmitter (UART) and its accompanying device driver (BRT_A) are used on the M•CORE MMC2001. Level 2 service indirectly addresses module registers through a device descriptor block accessed via a handle. The device descriptor block contains auxiliary information for implementing higher level operations such as buffering or queuing.

The UART module provides asynchronous serial communication with external devices such as modems and other computers. Refer to the Motorola *MMC2001 Reference Manual* (MMC2001RM/D) for more information on the UART.

13.2 UART_A Level 2 Device Driver Functions

The UART_A level 2 device driver software provides the following functions:

- Initializes, enables, and disables UART operations
- Enables and disables selected UART interrupts
- Buffered transmit and receive of UART data
- Enables and disables UART loopback mode
- Transmits break sequences
- Obtains UART receiver status
- Obtains RTS/CTS status, if applicable
- Returns status information to the application program about the result of requested operations.
- Handles UART transmit and receive interrupts
- Gets and sets all UART registers

13.3 Data Type Definitions

The following paragraphs show abstract data type definitions and initialization default macros.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.3.1 Abstract Data Type Definitions

The following are BRT_A abstract data types.

13.3.1.1 BRT_A_Buf_t - UART Buffer Definition

```
typedef struct
 {
    u1 *TxBuffer;             /* pointer to transmit buffer */
    u1 *RxBuffer;             /* pointer to receive buffer */
    u4 TxBuflen;              /* length of transmit buffer */
    u4 RxBuflen;              /* length of receive buffer */
    u4 TxThresh;              /* transmit buffer threshold */
    u4 RxThresh;              /* receive buffer threshold */
    volatile   u4 TxFront;    /* offset to start of transmit buffer */
    volatile   u4 RxFront;    /* offset to start of receive buffer */
    volatile   u4 TxRear;     /* offset to end of transmit buffer */
    volatile   u4 RxRear;     /* offset to end of receive buffer */
    volatile   u4 TxCount;    /* count of chars in transmit buffer */
    volatile   u4 RxCount;    /* count of chars in receive buffer */
brtTxThresh;                  /* transmit buffer threshold */
    u4 RxbrtRxThresh;         /* receive buffer threshold */
} BRT_A_Buf_t, *pBRT_A_Buf_t;
```

13.3.1.2 BRT_A_t - Buffered UART Device Descriptor

```
/typedef struct
 {
      pUART_A_t         UART;       /* UART register map base address */
      BRT_A_Buf_t       Buf;        /* UART circular buffer info */
      u4                Clock;      /* System clock frequency */
      u4                Flags;      /* Descriptor flags */
} BRT_A_t, *pBRT_A_t;
```

13.3.2 Initialization Default Macros

The following macros may be used as BRT_A_Init parameter values. In particular, the macro BRT_A_INIT_DEFAULT may be used in place of a call to BRT_A_Init. The only explicit argument to BRT_A_INIT_DEFAULT is the base address of the UART to be initialized.

```
#define UART_A_DEFAULT_SYS_CLOCK    32768000    /* 32.768 MHz */
#define UART_A_DEFAULT_BAUD_RATE    115200      /* 115.2K bps */
#define UART_A_DEFAULT_DIVIDER      8           /* 115.2K bps */
#define UART_A_DEFAULT_SIZE         UART_DATA_8 /* 8-bit data */
#define UART_A_DEFAULT_PARITY       UART_PARITY_NONE  /* no parity */
#define UART_A_DEFAULT_STOP_BITS    1           /* 1 stop bit */
#define UART_A_DEFAULT_RX_TRIG      UART_TRIG_8 /* 8 char threshold */
#define UART_A_DEFAULT_TX_TRIG      UART_TRIG_8 /* 8 char threshold */
#define UART_A_DEFAULT_RTS_INT      FALSE       /* RTS int disabled */
```


Freescale Semiconductor, Inc.

```
#define UART_A_DEFAULT_DOZE         TRUE        /* DOZE enabled */
#define UART_A_DEFAULT_FLOW         FALSE       /* handshake disabled */
#define UART_A_DEFAULT_UART_PINS    (UART_RXD | UART_TXD)
#define UART_A_DEFAULT_OUTPUT_PINS  (UART_RTS | UART_CTS)

#define BRT_A_INIT_DEFAULT(BRTPtr)                                      \
    BRT_A_Init(                                                        \
               BRTPtr,                                                  \
               UART_A_DEFAULT_SYS_CLOCK,                                \
               UART_A_DEFAULT_BAUD_RATE,                                \
               UART_A_DEFAULT_SIZE,                                     \
               UART_A_DEFAULT_PARITY,                                   \
               UART_A_DEFAULT_STOP_BITS,                                \
               UART_A_DEFAULT_RX_TRIG,                                  \
               UART_A_DEFAULT_TX_TRIG,                                  \
               UART_A_DEFAULT_RTS_INT,                                  \
               UART_A_DEFAULT_DOZE,                                     \
               UART_A_DEFAULT_FLOW,                                     \
               UART_A_DEFAULT_UART_PINS,                                \
               UART_A_DEFAULT_OUTPUT_PINS                               \
              )
```

13.4 API Definitions

The following paragraphs provide level 2 API definitions for a UART_A device driver.

Freescale Semiconductor, Inc.

13.4.1 BRT_A_Init

The following paragraphs describe the BRT_A_Init function.

13.4.1.1 Description

BRT_A_Init initializes the UART for operation. This function should be executed once at system reset. When the UART is enabled, this function should not be called; however, it can be called when the UART is disabled.

The nominal baud rate is bounded by the system clock frequency. For example, the lowest baud rate possible with a 32 MHz system clock is 501. The lower-level UART routines will return an error condition if the supplied baud rate falls outside the system clock bounds.

NOTE: *It is the caller's responsibility to allocate memory for the BRT_A_t structure and initialize the UART register address in the UART field of the device descriptor block. Refer to* ***13.4.1.5 Code Usage Example*** *for more information.*

13.4.1.2 Prototype

```
ddErr_t BRT_A_Init    (
                      pBRT_A_t            BRTPtr,
                      u4                  SysClock,
                      u4                  BaudRate,
                      UART_A_Size_t       Size,
                      UART_A_Parity_t     Parity,
                      u1                  StopBits,
                      UART_A_Trig_t       RxTrig,
                      UART_A_Trig_t       TxTrig,
                      bool                RTSInt,
                      bool                Doze,
                      bool                Flow,
                      u1                  UARTPins,
                      u1                  OutputPins
                      );
```


Freescale Semiconductor, Inc.

13.4.1.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART driver base address associated with this driver	Any non-zero core processor data address
SysClock	u4	System clock rate	N/A
BaudRate	u4	Nominal baud rate	N/A
Size	UART_A_Size_t	Format is based on the number of data bits in the frame	UART_A_DATA_7 UART_A_DATA_8
Parity	UART_A_Parity_t	Parity type	UART_A_PARITY_NONE UART_A_PARITY_ODD UART_A_PARITY_EVEN
StopBits	u1	Number of stop bits per frame	1 or 2
RxTrig	UART_A_Trig_t	FIFO threshold for receiver interrupts	UART_A_TRIG_1 UART_A_TRIG_4 UART_A_TRIG_8 UART_A_TRIG_14
TxTrig	UART_A_Trig_t	FIFO threshold for transmitter interrupts	UART_A_TRIG_1 UART_A_TRIG_4 UART_A_TRIG_8 UART_A_TRIG_14
RTSInt	bool	Indicates whether to enable RTS delta interrupts	TRUE = enable FALSE = do not enable
Doze	bool	Determines whether the UART is affected by CPU doze instruction	TRUE = doze when CPU dozes FALSE = do not doze
Flow	bool	Determines whether to enable UART hardware flow control	TRUE = enable FALSE = do not enable
UARTPins	u1	Determines which UART pins are to be used by the UART module. The remaining pins may be used for general-purpose I/O.	UART_A_RXD UART_A_TXD UART_A_RTS UART_A_CTS
OutputPins	u1	Determines which general-purpose I/O pins are to be used as outputs. Remaining pins will default to inputs.	UART_A_RXD UART_A_TXD UART_A_RTS UART_A_CTS

Freescale Semiconductor, Inc.

13.4.1.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_DIVIDE_BY_ZERO	Divide-by-zero error
DD_ERR_INVALID_CLOCK_DIVIDER	Computed clock divider is out of range
DD_ERR_INVALID_SIZE	Size parameter is out of range
UART_A_ERR_INVALID_PARITY	Bad parity selection parameter
UART_A_ERR_INVALID_STOP_BITS	Stop bits not 1 or 2
UART_A_ERR_INVALID_TRIGGER	FIFO trigger value not within range
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

13.4.1.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Initialize buffered UART version A */
retval = BRT_A_Init(
                     brtptr,       /* Buffered UART Device Handle   */
                     32768000,              /* System Clock in MHz */
                     19200,                 /* Canonical baud rate */
                     UART_A_DATA_8,         /* Data Size            */
                     UART_A_PARITY_NONE,    /* Parity               */
                     1,                     /* Stop Bits            */
                     UART_A_TRIG_8,         /* Rx FIFO Trigger      */
                     UART_A_TRIG_8,         /* Tx FIFO Trigger      */
                     FALSE,                 /* RTS Interrupt        */
                     TRUE,                  /* Doze Mode            */
                     FALSE,                 /* Flow Control         */
                     (UART_A_RXD_MASK |     /* UART Pins            */
                      UART_A_TXD_MASK),
                     (UART_A_RTS_MASK |     /* Output Pins          */
                      UART_A_CTS_MASK)
                    );

/* Perform an optional check on the BRT_A_Init() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```


Freescale Semiconductor, Inc.

13.4.2 BRT_A_BufInit

The following paragraphs describe the BRT_A_BufInit function.

13.4.2.1 Description

BRT_A_Init initializes circular buffers for interrupt-driven UART operation. This function may be called any time after BRT_A_Init. It must be called prior to enabling the transmitter or receiver whenever interrupts are enabled and BRT_A_ISF is used as the interrupt service mechanism.

13.4.2.2 Prototype

```
ddErr_t BRT_A_BufInit  (
                        pBRT_A_t        BRTPtr,
                        u1              *TxBuf,
                        u4              TxLen,
                        u4              TxThresh,
                        u1              *RxBuf,
                        u4              RxLen,
                        u4              RxThresh
                        );
```

13.4.2.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxBuf	u1 *	Pointer to transmit buffer	Any valid core processor data address
TxLen	u4	Length of transmit buffer	Must be a power of two
TxThresh	u4	Transmit buffer threshold	N/A
RxBuf	u1 *	Pointer to receive buffer	Any valid core processor data address
RxLen	u4	Length of receive buffer	Must be a power of two
RxThresh	u4	Receive buffer threshold	N/A

13.4.2.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_BUFFER_LENGTH	Length of buffer is not a power of two

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.2.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
#define IOBUFSIZE   1024

ddErr_t retval;
char rxbuf[IOBUFSIZE];
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Initialize buffers for buffered UART version A */
retval = BRT_A_BufInit(
                        brtptr,     /* Buffered UART Device Handle */
                        NULL,       /* no transmit buffering        */
                        0,          /* transmit buffer size         */
                        0,          /* transmit buffer threshold    */
                        txbuf,      /* buffered receive             */
                        IOBUFSIZE,  /* receive buffer size          */
                        IOBUFSIZE   /* receive buffer threshold     */
                        );

/* Perform an optional check on the BRT_A_BufInit() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.3 BRT_A_BufReset

The following paragraphs describe the BRT_A_BufReset function.

13.4.3.1 Description

BRT_A_BufReset resets the specified buffer pointer, effectively clearing the buffer contents. This function may be called any time after BRT_A_BufInit.

13.4.3.2 Prototype

```
ddErr_t BRT_A_BufReset      (
                            pBRT_A_t            BRTPtr,
                            UART_A_TxRx_t       TxRx
                            );
```

13.4.3.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to clear the UART transmit buffer, receive buffer, or both buffers	UART_A_TX UART_A_RX UART_A_TXRX

13.4.3.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

Freescale Semiconductor, Inc.

13.4.3.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Reset buffers for buffered UART version A */
retval = BRT_A_BufReset(
                  brtptr,     /* Buffered UART Device Handle      */
                  UART_A_TXRX /* Reset Transmit & Receive Buffers */
                  );

/* Perform an optional check on the BRT_A_BufReset() return code*/
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.4 BRT_A_BufStatus

The following paragraphs describe the BRT_A_BufStatus function.

13.4.4.1 Description

BRT_A_BufStatus returns the current count of characters in both the transmit and receive buffers. This function may be called any time after BRT_A_BufInit.

13.4.4.2 Prototype

```
ddErr_t BRT_A_BufStatus     (
                            pBRT_A_t        BRTPtr,
                            u4              *TxStatus
                            u4              *RxStatus
                            );
```

13.4.4.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxStatus	u4 *	Location to store transmit buffer status	Any non-zero core processor data address
RxStatus	u4 *	Location to store receive buffer status	Any non-zero core processor data address

13.4.4.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL

Freescale Semiconductor, Inc.

13.4.4.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
u4 txstat, rxstat;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Get buffer status for buffered UART version A */
retval = BRT_A_BufStatus(
                        brtptr,     /* Buffered UART Device Handle */
                        &txstat,    /* Transmit Status Location */
                        &rxstat     /* Receive Status Location */
                        );

/* Perform an optional check on the BRT_A_BufStatus() return code*/
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.5 BRT_A_Enable

The following paragraphs describe the BRT_A_Enable function.

13.4.5.1 Description

BRT_A_Enable enables the UART transmitter, receiver, or both simultaneously. It can also enable the UART without activating the transmitter or the receiver. This function may be called any time after BRT_A_Init.

13.4.5.2 Prototype

```
ddErr_t BRT_A_Enable        (
                            pBRT_A_t            BRTPtr,
                            UART_A_TxRx_t       TxRx
                            );
```

13.4.5.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to enable the UART transmitter, receiver, both, or the entire UART module	UART_A_TX UART_A_RX UART_A_TXRX UART_A_MODULE

13.4.5.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

Freescale Semiconductor, Inc.

13.4.5.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Enable buffered UART version A */
retval = BRT_A_Enable(
                        brtptr,     /* Buffered UART Device Handle */
                        UART_A_TXRX /* Transmit & Receive          */
                        );

/* Perform an optional check on the BRT_A_Enable() return code*/
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.6 BRT_A_Disable

The following paragraphs describe the BRT_A_Disable function.

13.4.6.1 Description

BRT_A_Disable disables the UART_A transmitter, receiver, or both simultaneously. It can also disable the entire UART_A module. This function may be called any time after BRT_A_Init.

13.4.6.2 Prototype

```
ddErr_t BRT_A_Disable      (
                           pBRT_A_t          BRTPtr,
                           UART_A_TxRx_t     TxRx
                           );
```

13.4.6.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to disable the UART transmitter, receiver, both, or the entire UART module	UART_A_TX UART_A_RX UART_A_TXRX UART_A_MODULE

13.4.6.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

Freescale Semiconductor, Inc.

13.4.6.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Disable buffered UART version A */
retval = BRT_A_Disable(
                        brtptr,     /* Buffered UART Device Handle */
                        UART_A_TX   /* Transmit Only                */
                        );

/* Perform an optional check on the BRT_A_Disable() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.7 BRT_A_IntEnable

The following paragraphs describe the BRT_A_IntEnable function.

13.4.7.1 Description

BRT_A_IntEnable enables interrupts for the UART transmitter, receiver, or both simultaneously. It can also enable RTS delta interrupts. This function may be called any time after BRT_A_Init.

13.4.7.2 Prototype

```
ddErr_t BRT_A_IntEnable     (
                            pBRT_A_t          BRTPtr,
                            UART_A_TxRx_t     TxRx,
                            bool              RTSint
                            );
```

13.4.7.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to enable the UART transmitter, receiver, both, or none	UART_A_TX UART_A_RX UART_A_TXRX UART_A_NONE
RTSint	bool	Indicates whether to enable the RTS delta interrupt	TRUE = enable FALSE = do not enable

13.4.7.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.7.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Interrupt enable buffered UART version A */
retval = BRT_A_IntEnable(
                        brtptr,     /* Buffered UART Device Handle */
                        UART_A_RX,  /* Receive Only                */
                        FALSE       /* No RTS Interrupt            */
                        );

/* Perform an optional check on the BRT_A_IntEnable() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.8 BRT_A_IntDisable

The following paragraphs describe the BRT_A_IntDisable function.

13.4.8.1 Description

BRT_A_IntDisable disables interrupts for the UART transmitter, receiver, or both simultaneously. It can also disable RTS delta interrupts. This function may be called any time after BRT_A_Init.

13.4.8.2 Prototype

```
ddErr_t BRT_A_IntDisable    (
                            pBRT_A_t        BRTPtr,
                            UART_A_TxRx_t   TxRx,
                            bool            RTSint
                            );
```

13.4.8.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to disable the UART transmitter, receiver, both, or none	UART_A_TX UART_A_RX UART_A_TXRX UART_A_NONE
RTSint	bool	Indicates whether to disable the RTS delta interrupt	TRUE = enable FALSE = do not enable

13.4.8.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.8.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Disable buffered UART version A interrupts */
retval = BRT_A_IntDisable(
                        brtptr,     /* Buffered UART Device Handle */
                        UART_A_TX,  /* Transmit Only               */
                        TRUE        /* Disable RTS Interrupt       */
                        );

/* Perform an optional check on the BRT_A_IntDisable() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.9 BRT_A_SetBaudRate

The following paragraphs describe the BRT_A_SetBaudRate function.

13.4.9.1 Description

BRT_A_SetBaudRate sets the UART baud rate. The nominal baud rate is bounded by the system clock frequency. For example, the lowest baud rate possible with a 32-MHz system clock is 501. The lower-level UART routines return an error condition if the supplied baud rate falls outside the system clock bounds.

This function may be called any time after BRT_A_Init.

NOTE: *It is assumed that all appropriate UART_A initializations have been performed through a previous call to BRT_A_Init. It is assumed that the caller has validated the configuration information prior to function invocation.*

13.4.9.2 Prototype

```
ddErr_t BRT_A_SetBaudRate   (
                            pBRT_A_t        BRTPtr,
                            u4              SysClock,
                            u4              BaudRate
                            );
```

13.4.9.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
SysClock	u4	System clock rate	N/A
BaudRate	u4	Nominal baud rate	N/A

13.4.9.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_DIVIDE_BY_ZERO	Divide-by-zero error
DD_ERR_INVALID_CLOCK_DIVIDER	Computed clock divider is out of range

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.9.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Set buffered UART version A baud rate */
retval = BRT_A_SetBaudRate(
                        brtptr,     /* Buffered UART Device Handle */
                        32768000,   /* System Clock in MHz          */
                        9600        /* New Baud Rate                */
                        );

/* Perform an optional check on the BRT_A_SetBaudRate() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.10 BRT_A_SetThreshold

The following paragraphs describe the BRT_A_SetThreshold function.

13.4.10.1 Description

BRT_A_SetThreshold sets the transmit and receive buffer thresholds. This function may be called any time after BRT_A_BufInit.

13.4.10.2 Prototype

```
ddErr_t BRT_A_SetThreshold  (
                            pBRT_A_t          BRTPtr,
                            UART_A_TxRx_t     TxRx,
                            u4                Threshold
                            );
```

13.4.10.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
TxRx	UART_A_TxRx_t	Indicates whether to set transmit, receive, or both buffer thresholds	UART_A_TX UART_A_RX UART_A_TXRX
Threshold	u4	Buffer threshold value	N/A

13.4.10.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_TXRX	Transmit/Receive selection is out of range

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.10.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Set buffered UART version A transmit threshold */
retval = BRT_A_SetThreshold(
                        brtptr,     /* Buffered UART Device Handle */
                        UART_A_TX,  /* Transmit Only               */
                        512         /* Transmit Buffer Threshold   */
                        );

/* Perform an optional check on the BRT_A_SetThreshold() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.11 BRT_A_Receive

The following paragraphs describe the BRT_A_Receive function.

13.4.11.1 Description

BRT_A_Receive returns the count characters from the UART receive register, or from the receive buffer if receive interrupts are enabled. This function may be called any time after BRT_A_Init.

13.4.11.2 Prototype

```
ddErr_t BRT_A_Receive    (
                         pBRT_A_t        BRTPtr,
                         u1              *Datap,
                         u4              *Count,
                         s4              Time-out
                         );
```

13.4.11.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Datap	u1 *	Pointer to buffer for received data	Any non-zero core processor data address
Count	u4 *	Pointer to number of characters to receive. The actual number of characters received is returned here.	N/A
Time-out	s4	Time in microseconds to busy/wait until count characters are received. If zero, then do not wait. If negative, then wait forever.	N/A

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.4.11.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
UART_A_ERR_DATA_PENDING	Data not available
UART_A_ERR_BREAK_DETECT	Break sequence detected
UART_A_ERR_OVERRUN_ERROR	Previously received datum has not been read
UART_A_ERR_FRAMING_ERROR	Framing error detected
UART_A_ERR_PARITY_ERROR	Parity error detected

13.4.11.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
#define IOBUFSIZE   1024

ddErr_t retval;
char rxbuf[IOBUFSIZE];
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Receive buffered UART version A */
retval = BRT_A_Receive(
                        brtptr,     /* Buffered UART Device Handle */
                        rxbuf,      /* Pointer for Data            */
                        IOBUFSIZE,  /* Amount to Read              */
                        -1          /* No Time-out                  */
                        );

/* Perform an optional check on the BRT_A_Receive() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```


Freescale Semiconductor, Inc.

13.4.12 BRT_A_Transmit

The following paragraphs describe the BRT_A_Transmit function.

13.4.12.1 Description

BRT_A_Transmit writes data to the UART data register, or the transmit buffer if transmit interrupts are enabled. This function may be called any time after BRT_A_Init.

13.4.12.2 Prototype

```
ddErr_t BRT_A_Transmit    (
                          pBRT_A_t        BRTPtr,
                          u1              *Datap,
                          u4              *Count,
                          s4              Time-out
                          );
```

13.4.12.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Datap	u1 *	Pointer to buffer for transmitted data	Any non-zero core processor data address
Count	u4 *	Pointer to number of characters to transmit. The actual number of characters transmitted is returned here.	N/A
Time-out	s4	Time in microseconds to busy/wait until count characters are transmitted. If zero, then do not wait. If negative, then wait forever.	N/A

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.4.12.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
UART_A_ERR_INVALID_DATA_VALUE	Data value outside frame size
UART_A_ERR_DATA_PENDING	Data not available

13.4.12.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
char txbuf[] = “Transmit this string”;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Transmit buffered UART version A */
retval = BRT_A_Transmit(
                        brtptr,     /* Buffered UART Device Handle */
                        txbuf,      /* Transmit Data               */
                        strlen (txbuf),/* Data Length                */
                        5000        /* Time-out After 5 Milliseconds */
                        );

/* Perform an optional check on the BRT_A_Transmit() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.13 BRT_A_ReadPin

The following paragraphs describe the BRT_A_ReadPin function.

13.4.13.1 Description

BRT_A_ReadPin returns the state of the GPIO pin. This function may be called any time after BRT_A_Init.

13.4.13.2 Prototype

```
ddErr_t BRT_A_ReadPin     (
                          pBRT_A_t        BRTPtr,
                          u1              Pin,
                          bool            *Statep
                          );
```

13.4.13.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Pin	u1	Ordinal number of pin to read	UART_A_RXD_BITNO UART_A_TXD_BITNO UART_A_RTS_BITNO UART_A_CTS_BITNO
Statep	bool *	Pointer to buffer for retrieved pin	Any non-zero core processor data address

13.4.13.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

Freescale Semiconductor, Inc.

13.4.13.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
bool pin_state;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Read buffered UART version A pin */
retval = BRT_A_ReadPin(
                  brtptr,           /* Buffered UART Device Handle */
                  UART_A_RTS_BITNO, /* Read RTS Pin */
                  &pin_state        /* Ptr to Pin State */
                  );

/* Perform an optional check on the BRT_A_ReadPin() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.14 BRT_A_WritePin

The following paragraphs describe the BRT_A_WritePin function.

13.4.14.1 Description

BRT_A_WritePin writes a state to the GPIO pin. This function may be called any time after BRT_A_Init.

13.4.14.2 Prototype

```
ddErr_t BRT_A_WritePin    (
                          pBRT_A_t      BRTPtr,
                          u1            Pin,
                          bool          State
                          );
```

13.4.14.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Pin	u1	Ordinal number of pin to write	UART_A_RXD_BITNO UART_A_TXD_BITNO UART_A_RTS_BITNO UART_A_CTS_BITNO
State	bool	Pin state to write	TRUE = 1 FALSE = 0

13.4.14.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_PIN	GPIO operation on non-existent pin

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.14.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Write buffered UART version A pin */
retval = BRT_A_WritePin(
                  brtptr,           /* Buffered UART Device Handle */
                  UART_A_CTS_BITNO, /* Write CTS Pin */
                  TRUE              /* Pin State to Write */
                  );

/* Perform an optional check on the BRT_A_WritePin() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.15 BRT_A_Infrared

The following paragraphs describe the BRT_A_Infrared function.

13.4.15.1 Description

BRT_A_Infrared enables or disables the infrared interface. This function may be called any time after BRT_A_Init.

13.4.15.2 Prototype

```
ddErr_t BRT_A_Infrared      (
                            pBRT_A_t        BRTPtr,
                            bool            Enable
                            );
```

13.4.15.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Enable	bool	Indicates infrared interface is enabled or disabled	TRUE = infrared enabled FALSE = infrared disabled

13.4.15.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.4.15.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* buffered UART version A infrared interface */
retval = BRT_A_Infrared(
                        brtptr,     /* Buffered UART Device Handle */
                        TRUE        /* Enable Infrared Interface    */
                        );

/* Perform an optional check on the BRT_A_Infrared() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.16 BRT_A_SendBreak

The following paragraphs describe the BRT_A_SendBreak function.

13.4.16.1 Description

BRT_A_SendBreak sends one or two break frames before recommencing data transmission. This function may be called any time after BRT_A_Init.

13.4.16.2 Prototype

```
ddErr_t BRT_A_SendBreak     (
                            pBRT_A_t            BRTPtr
                            );
```

13.4.16.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address

13.4.16.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.16.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* buffered UART version A Transmit Break Sequence */
retval = BRT_A_SendBreak(
                        brtptr      /* Buffered UART Device Handle   */
                        );

/* Perform an optional check on the BRT_A_SendBreak() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.17 BRT_A_ParityError

The following paragraphs describe the BRT_A_ParityError function.

13.4.17.1 Description

BRT_A_ParityError forces the transmitter to generate a parity error on the next transmission if parity is enabled. This function may be called any time after BRT_A_Init.

13.4.17.2 Prototype

```
ddErr_t BRT_A_ParityError    (
                             pBRT_A_t        BRTPtr,
                             bool            Enable
                             );
```

13.4.17.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Enable	bool	Indicates whether parity error generation should be turned on or off	TRUE = parity error generation enabled FALSE = parity error generation disabled

13.4.17.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_INVALID_PARITY	Bad parity selection parameter

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.4.17.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* buffered UART version A parity error generation */
retval = BRT_A_ParityError(
                        brtptr,     /* Buffered UART Device Handle */
                        TRUE        /* Parity Error Generation      */
                        );

/* Perform an optional check on the BRT_A_ParityError() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.18 BRT_A_Loopback

The following paragraphs describe the BRT_A_Loopback function.

13.4.18.1 Description

BRT_A_Loopback enables or disables the feedback path on the data serial shifter. This function may be called any time after BRT_A_Init.

13.4.18.2 Prototype

```
ddErr_t BRT_A_Loopback      (
                            pBRT_A_t        BRTPtr,
                            bool            Enable
                            );
```

13.4.18.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Enable	bool	Indicates whether looping and feedback path along data serial shifter is enabled or disabled	TRUE = loopback enabled FALSE = loopback disabled

13.4.18.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_IR_ENABLED	Infrared interface is enabled

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

13.4.18.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device handle */
pUART_A_t brtptr = (pUART_A_t) __PWS_UART0;

/* Loopback for buffered UART version A */
retval = BRT_A_Loopback(
                        brtptr,     /* Buffered UART Device Handle */
                        TRUE        /* Enable UART Loopback         */
                        );

/* Perform an optional check on the BRT_A_Loopback() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.19 BRT_A_IrLoopback

The following paragraphs describe the BRT_A_IrLoopback function.

13.4.19.1 Description

BRT_A_IrLoopback enables or disables the feedback path on the infrared interface. This function may be called any time after BRT_A_Init.

13.4.19.2 Prototype

```
ddErr_t BRT_A_IrLoopback    (
                            pBRT_A_t        BRTPtr,
                            bool            Enable
                            );
```

13.4.19.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Enable	bool	Indicates whether looping from transmitter to receiver in infrared interface is enabled or disabled	TRUE = loopback enabled FALSE = loopback disabled

13.4.19.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
UART_A_ERR_IR_DISABLED	Infrared interface is disabled

Freescale Semiconductor, Inc.

13.4.19.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Infrared loopback for buffered UART version A */
retval = BRT_A_IrLoopback(
                          brtptr,    /* Buffered UART Device Handle */
                          TRUE       /* UART Infrared Loopback */
                          );

/* Perform an optional check on the BRT_A_IrLoopback() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.20 BRT_A_GetStatus

The following paragraphs describe the BRT_A_GetStatus function.

13.4.20.1 Description

BRT_A_GetStatus returns the contents of the UART status register (USR) if receiver = false. If receiver = true, it returns the upper eight bits of the UART receive data register (URX). This function may be called any time after BRT_A_Init.

13.4.20.2 Prototype

```
ddErr_t BRT_A_GetStatus     (
                            pBRT_A_t        BRTPtr,
                            u2              *Statusp,
                            bool            Receiver
                            );
```

13.4.20.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
Statusp	u2 *	Address of location to store status word	Any non-zero core processor data address
Receiver	bool	Determines whether to return general UART status or receiver status	TRUE = return receiver status FALSE = return general status

13.4.20.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data pointer is NULL

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.20.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
u2 status;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Get status for buffered UART version A */
retval = BRT_A_GetStatus(
                        brtptr,     /* Buffered UART Device Handle */
                        &status,    /* Status Pointer              */
                        TRUE        /* Return Receiver Status      */
);

/* Perform an optional check on the BRT_A_GetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.21 BRT_A_RX_ISF

The following paragraphs describe the BRT_A_RX_ISF function.

13.4.21.1 Description

BRT_A_RX_ISF is the UART_A level 2 receive interrupt service function. On a receive interrupt, it moves the receive data register contents to the rear of the receive buffer and increments the receive buffer count. If the receive buffer count goes over the receive buffer threshold, the function returns a threshold status. If the receive buffer is full, it will return a buffer full status.

This function is called when the UART device interrupts. It is called by an interrupt dispatcher which is invoked through the actual device interrupt vector. BRT_A_BufInit must be called before this function is entered in order to initialize buffer parameters.

13.4.21.2 Prototype

```
ddErr_t BRT_A_RX_ISF        (
                            pBRT_A_t            BRTPtr
                            );
```

13.4.21.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address

13.4.21.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
UART_A_ERR_THRESHOLD	Buffer threshold exceeded
UART_A_ERR_BUFFER_FULL	Receive buffer full

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.21.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;
...

/* *INTERRUPT* From within ISR, perform receive interrupt processing */
ddErr_t retval;

retval = BRT_A_RX_ISF(
                        brtptr      /* Buffered UART Device Handle */
                        );

/* Perform an optional check on the BRT_A_RX_ISF() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
...

/* Return from interrupt */
```

Freescale Semiconductor, Inc.

13.4.22 BRT_A_TX_ISF

The following paragraphs describe the BRT_A_TX_ISF function.

13.4.22.1 Description

BRT_A_TX_ISF is the UART level 2 transmit interrupt service function. On a transmit interrupt, it checks the transmit buffer for data. If available, it will move the value at the front of the transmit buffer to the transmit data register and decrement the transmit buffer count. If the transmit buffer count falls below the transmit buffer threshold, the function returns a threshold status. If the transmit buffer is empty, it will return a buffer empty status.

This function is called when the UART device interrupts. It is called by an interrupt dispatcher which is invoked through the actual device interrupt vector. BRT_A_BufInit must be called before this function is entered in order to initialize buffer parameters.

13.4.22.2 Prototype

```
ddErr_t BRT_A_TX_ISF        (
                            pBRT_A_t            BRTPtr
                            );
```

13.4.22.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address

13.4.22.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
UART_A_ERR_THRESHOLD	Buffer threshold exceeded
UART_A_ERR_BUFFER_EMPTY	Transmit buffer empty

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.22.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;
...

/* *INTERRUPT* From within ISR, perform transmit interrupt processing */
ddErr_t retval;

retval = BRT_A_TX_ISF(
                        brtptr      /* Buffered UART Device Handle */
                        );

/* Perform an optional check on the BRT_A_TX_ISF() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
...

/* Return from interrupt */
```

Freescale Semiconductor, Inc.

13.4.23 BRT_A_GetRegister

The following paragraphs describe the BRT_A_GetRegister function.

13.4.23.1 Description

BRT_A_GetRegister gets a UART register. This function may be called at any time.

13.4.23.2 Prototype

```
ddErr_t BRT_A_GetRegister    (
                             pBRT_A_t                    BRTPtr,
                             UART_A_RegisterSwitch_t     UARTRegisterSwitch,
                             u2                          *GetRegisterPtr
                             );
```

13.4.23.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
UARTRegisterSwitch	UART_A_RegisterSwitch_t	Selects among UART registers	UART_A_URX_SWITCH UART_A_UCR1_SWITCH UART_A_UCR2_SWITCH UART_A_UBRGR_SWITCH UART_A_USR_SWITCH UART_A_UTS_SWITCH UART_A_UPCR_SWITCH UART_A_UDDR_SWITCH UART_A_UPDR_SWITCH
GetRegisterPtr	u2 *	Address of copy of selected UART register	Any non-zero core processor data address

13.4.23.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_ADDRESS	Data Pointer is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.23.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
u2 received;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Get receive register value for buffered UART version A */
retval = BRT_A_GetRegister(
                  brtptr,           /* Buffered UART Device Handle */
                  UART_A_URX_SWITCH,/* Register Selector */
                  &received         /* Returned Value */
                  );

/* Perform an optional check on the BRT_A_GetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

13.4.24 BRT_A_SetRegister

The following paragraphs describe the BRT_A_SetRegister function.

13.4.24.1 Description

BRT_A_SetRegister sets a UART register. This function may be called at any time.

13.4.24.2 Prototype

```
ddErr_t BRT_A_SetRegister    (
                             pBRT_A_t                    BRTPtr,
                             UART_A_RegisterSwitch_t     UARTRegisterSwitch,
                             u2                          RegisterValue
                             );
```

13.4.24.3 Arguments

Formal Name	Type	Description	Range/Valid
BRTPtr	pBRT_A_t	UART device descriptor handle	Any non-zero core processor data address
UARTRegisterSwitch	UART_A_RegisterSwitch_t	Selects among UART registers	UART_A_UTX_SWITCH UART_A_UCR1_SWITCH UART_A_UCR2_SWITCH UART_A_UBRGR_SWITCH UART_A_UTS_SWITCH UART_A_UPCR_SWITCH UART_A_UDDR_SWITCH UART_A_UPDR_SWITCH
RegisterValue	u2	Data to be copied to the selected UART register	Any 16-bit value

13.4.24.4 Return Values

Return Value	Description
DD_ERR_NONE	No error (force long)
DD_ERR_INVALID_HANDLE	Device handle is NULL
DD_ERR_INVALID_REGISTER	Invalid register selection

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

13.4.24.5 Code Usage Example

```
/* Include the buffered UART version A header file */
#include “brt_a.h”
...
ddErr_t retval;
...

/* Create a buffered UART version A device descriptor and handle */
BRT_A_t brt     = { (pUART_A_t)__PWS_UART0 };
pBRT_A_t brtptr = &brt;

/* Set loopback enable in test register for buffered UART version A */
retval = BRT_A_SetRegister(
                  brtptr,           /* Buffered UART Device Handle */
                  UART_A_UTS_SWITCH,      /* Register Selector */
                  brtptr->UART->UTS | UTS_LOOP_MASK   /* Value to Set */
                  );

/* Perform an optional check on the BRT_A_SetRegister() return code */
if ( retval != DD_ERR_NONE )
{
    /* do application-specific error status reporting */
}
```

Freescale Semiconductor, Inc.

Appendix A Return Codes

A.1 Return Codes List

Table A-1 provides a complete listing of the return codes used by functions in the M•CORE MMC2001 device driver library.

Table A-1 Return Codes

Return Code	Functions Returning	Description
Generic		
DD_ERR_BAD_RESULT_ADDR	EdgePort_A_GetRegister EIM_A_GetRegister ISPI_A_GetRegister ISPI_A_Receive ISPI_A_GetInterrupt Status KPP_A_GetRegister KPP_A_GetStatus KPP_A_KeyColumn Scan LCD_A_GetBusyStatus LCD_A_Read LCD_A_GetRegister TRM_A_GetTODAlarm Status	Result Address is NULL
DD_ERR_DIVIDE_BY_ZERO	BRT_A_Init BRT_A_SetBaudRate	Divide-by-zero error
DD_ERR_INVALID_ADDRESS	BRT_A_BufStatus BRT_A_Receive BRT_A_Transmit BRT_A_ReadPin BRT_A_GetStatus BRT_A_GetRegister INTC_A_Init INTC_A_GetRegister LCD_A_CGWrite PWM_A_GetRegister UART_A_GetStatus UART_A_GetRegister UART_A_Receive UART_A_ReadPin	Data pointer is NULL
DD_ERR_INVALID_BAUD_ RATE	ISPI_A_Init	Baud Rate is out of range
DD_ERR_INVALID_BUFFER_ LENGTH	BRT_A_Init	Length of buffer is not a power of two

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table A-1 Return Codes (Continued)

Return Code	Functions Returning	Description
DD_ERR_INVALID_CLOCK_ DIVIDER	BRT_A_Init BRT_A_SetBaudRate UART_A_Init UART_A_Set_Divider	Clock divider parameter is out of range
DD_ERR_INVALID_HANDLE	All	Device handle is NULL
DD_ERR_INVALID_REGISTER	BRT_A_GetRegister BRT_A_SetRegister EdgePort_A_GetRegister EdgePort_A_SetRegister EIM_A_GetRegister EIM_A_SetRegister INTC_A_GetRegister INTC_A_SetRegister ISPI_A_GetRegister ISPI_A_SetRegister KPP_A_GetRegister KPP_A_SetRegister LCD_A_GetRegister LCD_A_SetRegister PWM_A_GetRegister PWM_A_SetRegister TRM_A_GetRegister TRM_A_SetRegister UART_A_GetRegister UART_A_SetRegister	Invalid register selection
DD_ERR_INVALID_SIZE	BRT_A_Init UART_A_Init	Size parameter is out of range
DD_ERR_NONE	All	No error (force long)
DD_ERR_NO_INTERRUPT	ISPI_A_Receive	Interrupt Service Request (SPSR bit 14) is not set
DD_ERR_PIN_SENSE	ISPI_A_Init	Pin sense is not valid
INTC_A		
INTC_A_ERR_INVALID_ INTERRUPT_SOURCE	INTC_A_SetISF INTC_A_SetSSF	Source designator is invalid
ISPI_A		
ISPI_A_ERR_INVALID_CLOCK_ COUNT	ISPI_A_IntervalEnable ISPI_A_ManualEnable ISPI_A_SlaveEnable	Clock Count invalid
ISPI_A_INVALID_DRIVE_TYPE	ISPI_A_Init	Drive Type invalid
ISPI_A_ERR_INVALID_ INTERVAL_COUNT	ISPI_A_IntervalEnable	Interval Count invalid
ISPI_A_ERR_OVERRUN	ISPI_A_Receive	Overrun Condition (SPSR bit 15) is set
ISPI_A_ERR_OVERRUN_AND_ NO_INTERRUPT	ISPI_A_Receive	No Interrupt Request and ISPI Overrun detected

Freescale Semiconductor, Inc.

Table A-1 Return Codes (Continued)

Return Code	Functions Returning	Description
KPP_A		
KPP_A_ERR_INVALID_ COLUMN	KPP_A_KeyColumn Scan	Column parameter not one column
KPP_A_ERR_ZERO_ COLUMNS	KPP_A_Init KPP_A_KeyControl KPP_A_KeyColumn Scan	Columns parameter is zero
KPP_A_ERR_ZERO_ROWS	KPP_A_Init KPP_A_KeyColumn Scan	Rows parameter is zero
LCD_A		
LCD_A_ERR_INVALID_BITMAP	LCD_A_CGWrite	Invalid character in bitmap. Valid bytes in character bitmap are less than 0 x 20 because upper three bits are truncated (run-time error check).
LCD_A_ERR_INVALID_CHAR_ FONT	LCD_A_Init LCD_A_CGWrite	Selected invalid character font
LCD_A_ERR_INVALID_CHAR_ PAT	LCD_A_CGWrite	Selected an invalid CGRAM character pattern for character font
LCD_A_ERR_INVALID_ COMPONENT	LCD_A_Shift	Components available are display or cursor
LCD_A_ERR_INVALID_DATA_ LENGTH	LCD_A_Init	Selected an invalid data length for interface
LCD_A_ERR_INVALID_ DDRAM_ADDR	LCD_A_SetDDRAMAddr	Address provided is out of range
LCD_A_ERR_INVALID_INC_ MODE	LCD_A_Init LCD_A_SetEntryMode	Selected an invalid increment mode
LCD_A_ERR_INVALID_LINE_ VALUE	LCD_A_Init	Selected an invalid number of lines for display
LCD_A_ERR_INVALID_SETUP	LCD_A_Init	Selected the two_line display with the 5 x 10 character font
LCD_A_ERR_TIME_OUT_ STATUS	LCD_A_WaitNotBusy	Busy flag did not clear
PWM_A		
PWM_A_ERR_CLOCKSEL	PWM_A_Init	ClockSel out of range
PWM_A_ERR_PERIOD	PWM_A_UpdateOutput	Period is out of range
PWM_A_ERR_WIDTH	PWM_A_UpdateOutput	Width is out of range
PWM_A_ERR_STATUS	PWM_A_GetStatus	Status param is NULL
PWM_A_ERR_IRQHIPTR	PWM_A_GetIRQ	IRQHiPtr is NULL

Freescale Semiconductor, Inc.

Table A-1 Return Codes (Continued)

Return Code	Functions Returning	Description
TRM_A		
TRM_A_BAD_RESULT_ADDR	TRM_A_GetPITStatus TRM_A_GetRegister TRM_A_GetTODAlarm Status	Result Pointer is NULL
TRM_A_BAD_TIMEOUT_VAL	TRM_A_InitWD	Watchdog time-out is out of range
UART_A		
UART_A_ERR_BREAK_ DETECT	BRT_A_Receive UART_A_Receive	Break sequence detected
UART_A_ERR_BUFFER_ EMPTY	BRT_A_TX_ISF	Transmit buffer empty
UART_A_ERR_BUFFER_FULL	BRT_A_RX_ISF	Receiver buffer full
UART_A_ERR_DATA_ PENDING	BRT_A_Receive BRT_A_Transmit UART_A_Receive, UART_A_Transmit	Data not available
UART_A_ERR_FRAMING_ ERROR	BRT_A_Receive UART_A_Receive	Framing error detected
UART_A_ERR_INVALID_DATA_ VALUE	BRT_A_Transmit	Data value outside frame size
UART_A_ERR_INVALID_ PARITY	BRT_A_Init BRT_A_ParityError UART_A_Init, UART_A_ParityError	Bad parity selection parameter
UART_A_ERR_INVALID_PIN	BRT_A_Init BRT_A_ReadPin BRT_A_WritePin UART_A_Init UART_A_ReadPin UART_A_WritePin	GPIO operation on non-existent pin
UART_A_ERR_INVALID_ STOP_BITS	BRT_A_Init UART_A_Init	Stop bits not one or two
UART_A_ERR_INVALID_ TRIGGER	BRT_A_Init UART_A_Init	FIFO trigger value not within range
UART_A_ERR_INVALID_TXRX	BRT_A_BufReset BRT_A_Enable BRT_A_Disable BRT_A_IntEnable BRT_A_IntDisable BRT_A_SetThresholdUAUA RT_A_Enable, UART_A_Disable, UART_A_IntEnable, UART_A_IntDisable	Transmit/receive selection is out of range
UART_A_ERR_IR_DISABLED	BRT_A_IrLoopback UART_A_IrLoopback	Infrared interface is disabled

Freescale Semiconductor, Inc.

Table A-1 Return Codes (Continued)

Return Code	Functions Returning	Description
UART_A_ERR_IR_ENABLED	BRT_A_Loopback UART_A_Loopback	Infrared interface is enabled
UART_A_ERR_LOOPBACK_ ENABLED	UART_A_Infrared, UART_A_IRLoopback	Serial loopback is enabled
UART_A_ERR_OVERRUN_ ERROR	BRT_A_Receive UART_A_Receive	Previously received datum has not been read
UART_A_ERR_PARITY_ ERROR	BRT_A_Receive UART_A_Receive	Parity error detected
UART_A_ERR_RECEIVE_ ERROR	UART_A_Receive	General receive error
UART_A_ERR_THRESHOLD	BRT_A_RX_ISF BRT_A_TX_ISF	Buffer threshold exceeded

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

A

ABI 25
ANSI C 25
Application programming interface (API) 15
Applications
    binary interface (ABI) 25
    Binary Interface Standards Manual (*MCOREABI/D*) 25
Autovectored interrupts 18

B

BRT_A (*See* UART_A level 2) 239

C

cfg 25
Character generator
    random access memory (CGRAM) 111
    read-only memory (CGROM) 111
Clear-to-send (CTS) 197
Counting rate 145
CTS 197

D

Data
    C compiler 29
    random access memory (DDRAM) 111
    type definitions 33
DD_ERR_NONE 27
Device
    descriptor 17
    driver code 15
    handle 16
Direct register access 28
Dispatch routine 19
Distribution tree 25

E

Edge Port module (EdgePort) 35
Edge-detecting interrupt 35
EdgePort_A 35
    API definitions
        EdgePort_A_GetRegister 37
        EdgePort_A_SetRegister 39
EIM_A 41
    API definitions
        EIM_A_GetRegister 43
        EIM_A_SetRegister 45
Error
    checking 27
    code definitions 25
errors.h 27
Evaluation board (EVB) 29
External interface module (EIM) 41

F

First in, first out (FIFO) 197
Free-running counter 145

H

Hitachi HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver) 112

I

inc 25
incplib.h 25
INT/FINT vectors 18
INTC_A 47
    API definitions
        INTC_A_GetRegister 60
        INTC_A_Init 50
        INTC_A_IntDisable 54
        INTC_A_IntEnable 52
        INTC_A_SetISF 56
        INTC_A_SetRegister 62
        INTC_A_SetSSF 58
    data type definitions 48
Interrupt
    controller module (INTC) 47
    dispatch routine 19
    integration 18
    service
        function (ISF) 19
        routine (ISR) 18
Interval mode serial peripheral interface (ISPI) 65
ISF 19, 47
ISPI_A 65
    API definitions
        ISPI_A_ClearOverrun 87
        ISPI_A_Disable 77
        ISPI_A_GetInterruptStatus 89
        ISPI_A_GetRegister 91
        ISPI_A_GPControl 83
        ISPI_A_Init 68
        ISPI_A_InterruptReceive 81
        ISPI_A_IntervalEnable 73
        ISPI_A_Loopback 85
        ISPI_A_ManualEnable 71
        ISPI_A_SetRegister 93
        ISPI_A_SlaveEnable 75
        ISPI_A_Transmit 79
    data type definitions 66
    operating modes 65
ISR 18

K

Keypad port (KPP) 95
KPP_A 95
    API definitions

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

KPP_A_GetRegister 106
KPP_A_GetStatus 101
KPP_A_Init 97
KPP_A_KeyColumnScan 103
KPP_A_KeyControl 99
KPP_A_SetRegister 108
data type definitions 95, 113
KS0066U LCD controller 111

L

LCD_A 111
API definitions
LCD_A_CGWrite 137
LCD_A_ClrDisplay 119
LCD_A_DisplayControl 125
LCD_A_GetRegister 140
LCD_A_Init 116
LCD_A_Read 135
LCD_A_ReturnHome 121
LCD_A_SetDDRAMAddr 131
LCD_A_SetEntryMode 123
LCD_A_SetRegister 142
LCD_A_Shift 127
LCD_A_WaitNotBusy 129
LCD_A_Write 133
libplib.a 25
Library
and interface files 25
macros 26

M

M•CORE
Applications Binary Interface Standards Manual (*MCOREABI/D*) 20, 25
device driver library system relationships 15
Evaluation System User's Manual (*MCOREEVSUM/D*) 20
Reference Manual (*MCORERM/AD*) 20
MMC2001 Reference Manual (*MMC2001RM/D*) 20

P

PARAM_CHECKING flag 26
PIT 165
Platform board (PFB) 111
plibdefs.h 26
plibtut.elf 32
Prescaler taps 145
Programmable interval timer (PIT) 165
Pulse-width modulator (PWM) 145
PWM_A 145
abstract data type definitions 35, 42, 146
API definitions
PWM_A_GetIRQ 158
PWM_A_GetRegister 160
PWM_A_GetStatus 156
PWM_A_Init 148
PWM_A_SetRegister 162
PWM_A_Start 150
PWM_A_Stop 152
PWM_A_UpdateOutput 154
data type definitions 35, 41, 146
free-running counter 145
period register 145
width register 145

R

Real-time operation system (RTOS) 15
References 20
Request-to-send (RTS) 197
Reset source/chip configuration register (RSCR) 165
Return codes 27, 291
RSCR 165
RTOS 15
RTS 197

S

Service 16
relationships
level 1 16
level 2 18
signalling function (SSF) 19
Single-step debugger 29
Specified interrupt signaling function (SSF) 47

T

Time-of-day counter (TOD) 165
Timer/reset module (TRM) 165
TOD 165
TRM_A 165
API definitions
TRM_A_ClearPITInterrupt 185
TRM_A_ClearTODAlarmInterrupt 171
TRM_A_ControlPITEnable 187
TRM_A_ControlPITInterrupt 189
TRM_A_ControlTODAlarmEnable 173
TRM_A_ControlTODAlarmInterrupt 175
TRM_A_GetPITStatus 191
TRM_A_GetRegister 193
TRM_A_GetTODAlarmStatus 169
TRM_A_InitPIT 181
TRM_A_InitRSCR 167
TRM_A_InitWD 177
TRM_A_ServiceWD 179
TRM_A_SetPITModulus 183
TRM_A_SetRegister 195
data type definitions 166
Tutorial 29

Freescale Semiconductor, Inc.

U

UART_A
data type definitions 198, 239
level 1 197
API definitions
UART_A_Disable 206
UART_A_Enable 204
UART_A_GetRegister 234
UART_A_GetStatus 232
UART_A_Infrared 222
UART_A_Init 201
UART_A_IntDisable 210
UART_A_IntEnable 208
UART_A_IrLoopback 230
UART_A_Loopback 228
UART_A_ParityError 226
UART_A_ReadPin 218
UART_A_Receive 214
UART_A_SendBreak 224
UART_A_SetDivider 212
UART_A_SetRegister 236
UART_A_Transmit 216
UART_A_WritePin 220
level 2
API definitions
BRT_A_BufInit 245
BRT_A_BufReset 247
BRT_A_BufStatus 249
BRT_A_Disable 253
BRT_A_Enable 251
BRT_A_GetRegister 287
BRT_A_GetStatus 281
BRT_A_Infrared 271
BRT_A_Init 242
BRT_A_IntDisable 257
BRT_A_IntEnable 255
BRT_A_IrLoopback 279
BRT_A_Loopback 277
BRT_A_ParityError 275
BRT_A_ReadPin 267
BRT_A_Receive 263
BRT_A_RX_ISF 283
BRT_A_SendBreak 273
BRT_A_SetBaudRate 259
BRT_A_SetRegister 289
BRT_A_SetThreshold 261
BRT_A_Transmit 265
BRT_A_TX_ISF 285
BRT_A_WritePin 269
Universal asynchronous receiver/transmitter (UART)
level 1 197
level 2 239

V

Vector base register (VBR) 47
Vectored interrupts 18

W

Watchdog timer (WD) 165
Wrapper 19

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Record of Changes

Revision	Date	Description
1.0	17 FEB 99	Original

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Record of Changes

Freescale Semiconductor, Inc.