Freescale Semiconductor
Application Note

AN3555
Rev. 0, 11/2007

Investigating XGATE Software Errors

XGATEV2 (S12X) Software Error Debugging Aid

by: Dirk Heisswolf
MCD Design
Munich, Germany

1 Introduction

The XGATE peripheral coprocessor includes a safety feature called Software Error Detection. The Software Error Detection feature lets the XGATE detect conditions which should not occur during program execution. The detection of these conditions indicate that there are problems in the application code.

Whenever XGATE detects a Software Error Condition, it ceases program execution immediately and flags an interrupt to the CPU12X to perform corrective action. In a typical application this corrective action simply consists of initialization and system restart. However during the development and debug phase of application code it is desirable to know the exact cause of XGATE Software Errors.

This document helps you investigate XGATE Software Errors on S12X devices[1]. The S12XE (and S12XF)

Contents

1. Appendix A.1 provides a list of relevant S12X MCUs

product family is discussed in part II of this document series.

CAUTION

This application note describes the behavior of the XGATE as it is shipped on S12X devices as of today. Some descriptions show behavior which is not specified in the reference manual and may be invalid on future implementations of the XGATE. The intent of this document is to provide helpful information for debugging application code. It is not to be understood as an addition to the XGATE specification.

2 The Software Error Detection Feature

Software Error Detection is a safety feature of the XGATE. It provides a mechanism to stop program execution and call for help, whenever the XGATE is instructed to do something illegal.
Illegal actions are defined by a set of Software Error Conditions (Section 2.1). These Software Error Conditions consist of features which are not supported by the MCU's architecture and violations of the MCU's memory protection scheme. Software Error Conditions should never occur in an application. They are considered to be an indicator for a software problem.
Whenever a Software Error Condition is detected, XGATE ceases program execution and performs two distinct actions: It flags an interrupt to the CPU12X (Section 2.2) and enters a special Software Error State (Section 2.3).
To minimize the harm that erroneous XGATE application code can do to the system, the XGATE ceases code execution in the instant that the problem is detected. It doesn't finish the execution of the current instruction.
While stopping program execution in the middle of an instruction execution is desirable from a safety viewpoint, it does add some complexity when it comes to the debugging of Software Errors.
Instructions are stopped before their specified operation has been completed. Which part of the operation has been processed and which not depends on the implementation of the XGATE module.
At present there are two revisions available: XGATEV2 which is integrated in most in S12X devices and XGATEV3 (enhanced XGATE) which is built into S12XE MCUs. These versions have slightly different implementations of the Software Error Detection feature. This document discusses the behavior of Software Errors, as they are implemented on XGATEV2.

2.1 Software Error Conditions

Not every instruction given to the XGATE can or should be executed. Instructions which invoke an unimplemented hardware feature cannot be processed with a meaningful outcome. Instructions that violate the memory protection scheme must be prevented to preserve data consistency. Section 2.1.1 and Section 2.1.2 list the Software Error Conditions for these two categories. A compact summary of all conditions is given in Figure 2.

2.1.1 Invocation of Unsupported Features

There are four Software Error Conditions that indicate the usage of an unsupported hardware feature:

- Undefined Opcodes - The XGATE's instruction set leaves a number of opcodes undefined. Their execution results in Software Errors. Figure 1 lists all illegal XGATE instructions. This list may decrease on later releases of the XGATE, as new features are added.

Functionality	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Illegal Instructions																
Undefined opcode	0	0	0	0	0	don't care			0	0	0	0	0	0	0	1
	0	0	0	0	0	don't care			0	0	0	0	0	0	1	dc
	0	0	0	0	0	don't care			0	0	0	0	0	1	don't care	
	0	0	0	0	0	don't care			0	0	0	0	1	don't care		
	0	0	0	0	0	don't care			0	0	0	1	don't care			
	0	0	0	0	0	don't care			0	0	1	don't care				
	0	0	0	0	0	don't care			0	1	don't care					
	0	0	0	0	0	don't care			1	0	don't care					
	0	0	0	0	0	don't care			1	1	0	don't care				
	0	0	0	0	0	don't care			1	1	1	0	don't care			
	0	0	0	0	0	don't care			1	1	1	1	1	0	1	1
	0	0	0	0	0	don't care			1	1	1	1	1	1	don't care	
	0	0	0	0	0	1	don't care		0	0	0	0	0	0	0	0
	0	0	0	0	1	don't care						0	0	don't care		
	0	0	0	0	1	don't care							1	0	0	0
	0	0	0	1	0	don't care									0	1

Figure 1. Illegal Instructions

- Misaligned word accesses (16-bit accesses to an odd address)

  The S12X architecture does not allow the XGATE to performmisaligned word accesses to its memories. So any of the following three accesses results in a Software Error:

  — Opcode fetches from an odd address

  — 16 Bit data read accesses from an odd address

  — 16 Bit data write accesses to an odd address

- Unimplemented address locations

  On S12X devices it is possible to hide the Flash memory if the MCU is secured and in expanded mode. Accesses to these unimplemented address locations trigger a Software Error.

- Write accesses to Flash space

  The Flash memory on S12X devices can not be programmed by the XGATE through write access to the address space of the nonvolatile memory. As general application code would never try to write to these memory locations; a Software Error is issued if the write attempt takes place.

2.1.2 Violations of the Memory Protection Scheme

The second category of Software Error Conditions contains artificial restrictions for memory access. Their purpose is to minimize the damage that faulty application code can cause. Two types of memory protection are implemented on S12X products.

- Code execution from register space -
  On every S12X product family the XGATE is not allowed to execute code from the register space. This serves two purposes. It is a safety feature. As code execution from register space is generally not a good practice; it can be used as an indicator for runaway code. It is also a security feature used to prevent intruders from accessing system resources of a secured MCU.
  Vector fetches of the XGATEV2 are treated as opcode fetches. Fetching a vector from register space results in a Software Error.

- User defined memory protection -
  S12X MCUs offer a simple RAM write protection scheme. The RAM is divided into three sections: A CPU12X section, an XGATE section, and a shared section. The borders between these sections are configurable (Figure 4). Whenever the XGATE tries to write to the S12X section, a Software Error is triggered.

Software Error Condition		
1	Execution of an unimplemented opcode	
2	Vector fetch (VF)	from an unimplemented address
3		from register space
4	Opcode fetch (OF)	from an odd address
5		from an unimplemented address
6		from register space
7	Byte read access (BR)	from an unimplemented address
8	Word read access (WR)	to an odd address
9		to an unimplemented address
10	Byte write access (BW)	from an unimplemented address
11		to Flash memory
12		causing a RAM protection violation
14	Word write access (WW)	to an odd address
15		from an unimplemented address
16		to Flash memory
17		causing a RAM protection violation

Figure 2. Summary of Software Error Conditions

2.2 Software Error Interrupts

As soon as the XGATE detects a Software Error Condition it sets the Software Error Interrupt Flag (XGSWEIF) that triggers a maskable[1] interrupt to the CPU12X. This interrupt request persists until the XGSWEIF flag is cleared in an interrupt service routine.

1. Maskable via the I-bit in the CPU12X Condition Code Register.

2.3 Software Error State

If the XGATE ceases code execution due to a Software Error, it enters a Software Error State. This state gives the application code a chance to log and fix the problem (typically by reinitializing the application) before resuming normal operation. The XGATE stays in the Software Error State until the XGSWEIF flag is cleared. Leaving the Software Error State always terminates the current thread.

The investigation of XGATE Software Errors as it is described in this document (see Section 4) must be performed while the XGATE remains in the Software Error State.

3 Debugging Goal

To understand the cause of a Software Error the following information is pertinent:

- Which channel was active when the problem occurred?
- Which instruction was executed or which vector was fetched?
- In which cycle of the instruction or vector fetch did the error occur?
- Which Software Error Condition was triggered?
- What was the state of the RISC core's registers, when the problem occurred?

Unfortunately the XGATE doesn't have any means to provide this information directly. It does however provide enough information to help track down the cause of the problem.

There are only a limited number of scenarios that lead to a Software Error. Each one of these scenarios leaves a characteristic signature inside the XGATE's registers after the Software Error has occurred.

To investigate the cause of a Software Error these signatures are broken down into a set of observable conditions. These conditions must be checked one after the other. Software Error scenarios can be excluded from further investigation if an associated observable condition is not fulfilled. The elimination process is complete if only one of the potential Software Error scenarios remains. The desired debug information can be extracted as soon as the Software Error scenario is determined.

The XGATE typically provides enough information to conclude the prehistory of the error. However there are a few cases that leave two possible scenarios that could have led to the problem. In these cases further investigation based on the context of the application code is required.

4 Investigating Software Errors

The following sections explain the process of finding the cause of an XGATE Software Error in detail.

4.1 Sources of Information

Information required for debugging XGATE Software Errors on S12X devices can be derived from the 18 registers shown in Figure 3.

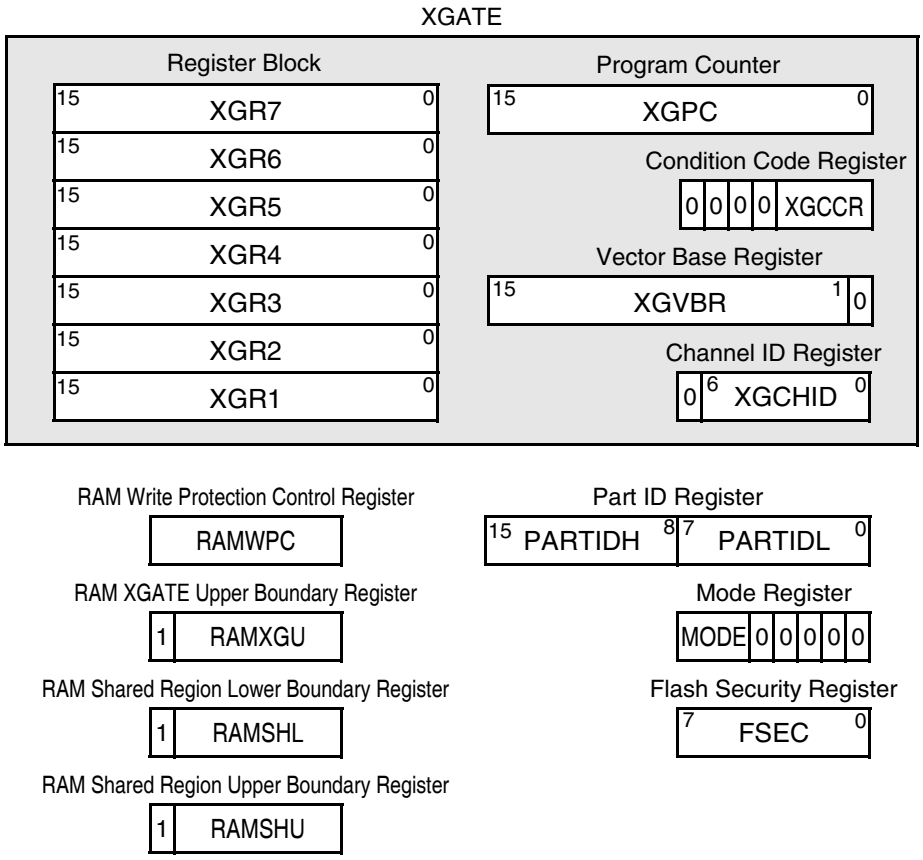

Figure 3. Sources of Information for Debugging S12X Devices

These registers are:

- The XGATE's status and debug registers (XGCHID, XGVBR, CGPC, XGCCR, XGR7..XGR1)
  They provide information about the state of the XGATE's RISC core and the current instruction.
- The MCU's Part ID Register (PARTIDH, PARTIDL)
  This register is used to determine which implementation of the XGATE is on the device
- The S12XMMC's RAM protection registers (RAMWPC, RAMXGU, RAMSHL, RAMSHU)
  These registers determine the protected memory ranges
- The Mode register and the Flash Security register (MODE, FSEC)
  These registers are necessary to find out the unmapped areas within the XGATE's memory map

As some of the registers can be used to obtain addresses of relevant memory locations, it is important that the memory content remains unchanged for the investigation of Software Errors.

4.2 Memory Protection

As mentioned in Section 2.1, there are two categories of Software Error Condition. The first one, “Invocation of Unsupported Features” (see Section 2.1.1), includes access to unmapped addresses in the memory map. On S12X devices there is only one scenario in which an address range inside the XGATE’s memory map is unmapped. If the MCU is in expanded mode while it is secured; the Flash is disabled for security reasons. In this case, access to the Flash range would result in Software Error Conditions 2, 5, 7, 9, 10, or 15 (see Figure 2).

The other category of Software Error Conditions are “Violations of the Memory Protection Scheme“ (see Section 2.1.2). In addition to the general restriction, that opcode and vector fetches from register space are forbidden, S12X devices offer a configurable RAM write protection scheme. There are three registers (RAMXGU, RAMSHL, and RAMSHU) to set the boundaries of two write protected address ranges. Write accesses to these areas trigger Software Error Conditions 12 or 17 (see Figure 2).

Figure 4 illustrates the full memory protection scheme of an S12X device.

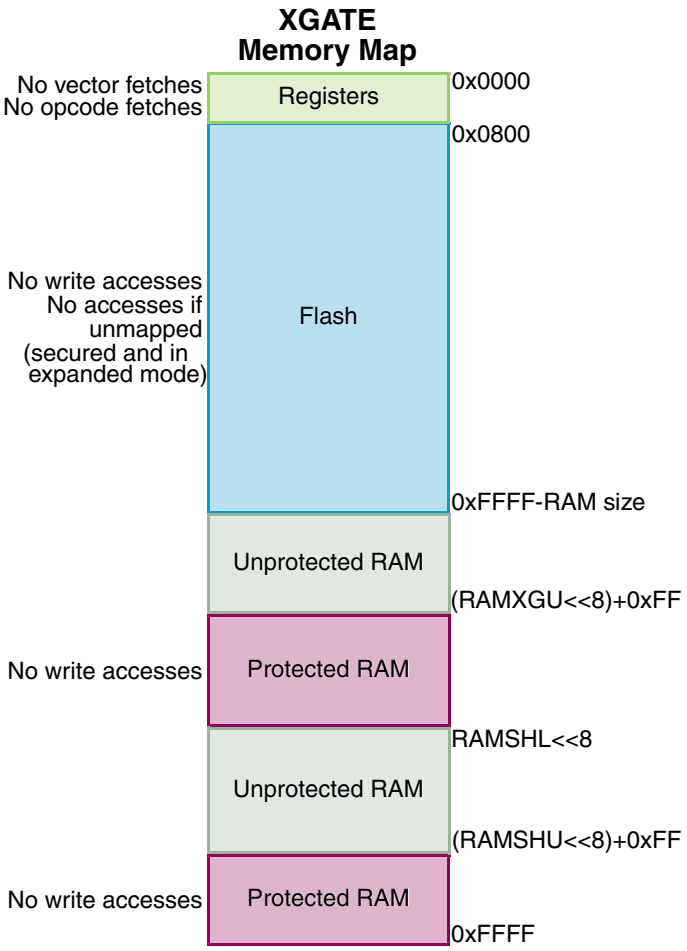

Figure 4. XGATE Memory Protection on S12X Devices

4.3 Observable Conditions

When investigating XGATE Software Errors, there is a set of significant conditions that can be directly checked on the source registers (see Section 4.1) and the current content of the MCU's memories. For every cause of an Software Error, a certain subset of these observable conditions must be true. If one condition in this subset can be proven false, then the associated cause can be excluded from further examination.

On S12X devices there are 31 observable conditions that are relevant for the investigation of Software Errors. These conditions are listed in Figure 5.

	Mnemonic	Description	Details
1	PC = 0xFFFE	The program counter is set to the value 0xFFFE	XGPC = 0xFFFE
2	PC & 1 = 1	The program counter contains an odd address	XGPC[0] = 1
3	PC & 1 = 0	The program counter contains an even address	XGPC[0] = 0
4	PC+2 ∈ register space	The incremented program counter is located in the register space	(XGPC = 0xFFFE) || (XGPC < 0x07FE)
5	PC ∉ register space	The program counter is not located in the register space	(XGPC ≥ 0x0800)
6	PC+2 ∈ flash space	The incremented program counter is located in the flash space	(XGPC ≥ 0x07FE) && (XGPC ≤ 0xFFFD-RAM size)
7	PC ∉ secured flash space	The program counter is not located in the secured flash space	(MODE&0xC0 = 0x00) || (FSEC&0x03 = 0x02) || (XGPC > 0xFFFF-RAM size)
8	PC+2 = [VBR+4*CHID]	The 1st word of the channel vector contains the value of the incremented program counter	XGPC = [XGVBR+4*XGCHID] - 2
9	CCR = 0x0	The condition code register is cleared	XGCCR = 0x00
10	R1 = 0x0000	General purpose register R1 is cleared	XGR1 = 0x0000
11	R1 = [VBR+4*CHID+2]	The 2nd word of the channel vector contains the content of general purpose register R1	XGR1 = [XGVBR+4*XGCHID+2]
12	R2..R7 = 0x0000	General Purpose Registers R2 to R7 are cleared	(XGR2 = 0x0000) && (XGR3 = 0x0000) && (XGR4 = 0x0000) && (XGR5 = 0x0000) && (XGR6 = 0x0000) && (XGR7 = 0x0000)
13	VBR+4*CHID = 0xFFFE	The location of the channel vector is 0xFFFE	XGVBR+4*XGCHID = 0xFFFE
14	VBR+4*CHID ≠ 0xFFFE	The location of the channel vector is not 0xFFFE	XGVBR+4*XGCHID ≠ 0xFFFE

Figure 5. Observable Conditions for Investigating Software Errors on S12X Devices (Sheet 1 of 3)

	Mnemonic	Description	Details
15	VBR+4*CHID ∈ register space	The location of the channel vector is inside the register space	`XGVBR+4*XGCHID < 0x0800`
16	VBR+4*CHID ∉ register space	The location of the channel vector is not inside the register space	`XGVBR+4*XGCHID ≥ 0x0800`
17	VBR+4*CHID ∈ flash space	The location of the channel vector is inside the flash space	`(XGVBR+4*XGCHID ≥ 0x0800) &&` `(XGVBR+4*XGCHID ≤ 0xFFFF-RAM size)`

18	VBR+4*CHID ∉ secured flash space	The location of the channel vector is not inside the secured flash space	`(MODE&0xC0 = 0x00) ‖`
			`(FSEC&0x03 = 0x02) ‖`
			`(XGVBR+4*XGCHID < 0x0800) ‖`
			`(XGVBR+4*XGCHID Š> 0xFFFF-RAM size)`

19	[PC] ∈ legal opcodes	The program counter points to a legal opcode	`(([XGPC]&0xF880 ≠ 0x0000) ‖ ([XGPC]&0x007F = 0x0000)) &&`
			`(([XGPC]&0xF880 ≠ 0x0080) ‖ ([XGPC]&0x0070 = 0x0070)) &&`
			`([XGPC]&0xF8FF ≠ 0x00FB) &&`
			`([XGPC]&0xF8FC ≠ 0x00FC) &&`
			`([XGPC]&0xFCFF ≠ 0x0400) &&`
			`([XGPC]&0xF818 ≠ 0x0800) &&`
			`([XGPC]&0xF80F ≠ 0x0808) &&`
			`([XGPC]&0xF803 ≠ 0x1001)`

20	[PC] ∉ legal opcodes	The program counter points to an illegal opcode	`(([XGPC]&0xF880 = 0x0000) && ([XGPC]&0x007F ≠ 0x0000)) ‖`
			`(([XGPC]&0xF880 = 0x0080) && ([XGPC]&0x0070 ≠ 0x0070)) ‖`
			`([XGPC]&0xF8FF = 0x00FB) ‖`
			`([XGPC]&0xF8FC = 0x00FC) ‖`
			`([XGPC]&0xFCFF = 0x0400) ‖`
			`([XGPC]&0xF818 = 0x0800) ‖`
			`([XGPC]&0xF80F = 0x0808) ‖`
			`([XGPC]&0xF803 = 0x1001)`

21	[PC] ∈ STB	The program counter points to a STB instruction	`([XGPC]&0xF800 = 0x5000) ‖` `(([XGPC]&0xF800 = 0x7000) && ([XGPC]&0x0003 ≠ 0x0003))`
22	[PC] ∈ STW	The program counter points to a STW instruction	`([XGPC]&0xF800 = 0x5800) ‖` `(([XGPC]&0xF800 = 0x7800) && ([XGPC]&0x0003 ≠ 0x0003))`
23	[PC] ∈ LDB	The program counter points to LDB instruction	`([XGPC]&0xF800 = 0x4000) ‖` `(([XGPC]&0xF800 = 0x6000) && ([XGPC]&0x0003 ≠ 0x0003))`
24	[PC] ∈ LDW	The program counter points to a LDW[1] instruction	`([XGPC]&0xF800 = 0x4800) ‖` `(([XGPC]&0xF800 = 0x6800) && ([XGPC]&0x0003 ≠ 0x0003))`

25	[PC+2] ∉ legal opcodes	The incremented program counter points to an illegal opcode	`(([XGPC+2]&0xF880 = 0x0000) && ([XGPC+2]&0x007F ≠ 0x0000)) ‖`
			`(([XGPC+2]&0xF880 = 0x0080) && ([XGPC+2]&0x0070 ≠ 0x0070)) ‖`
			`([XGPC+2]&0xF8FF = 0x00FB) ‖`
			`([XGPC+2]&0xF8FC = 0x00FC) ‖`
			`([XGPC+2]&0xFCFF = 0x0400) ‖`
			`([XGPC+2]&0xF818 = 0x0800) ‖`
			`([XGPC+2]&0xF80F = 0x0808) ‖`
			`([XGPC+2]&0xF803 = 0x1001)`

26	data address & 1 = 1	The address of the data access is odd	`b ⇐([XGPC]>>5) & 0x0007`
			`i ⇐([XGPC]>>2) & 0x0007`
			`offs5 ⇐([XGPC]&0x000F) |`
			`([XGPC]&0x0010)*0x07FF`
			
			`data_address[0] = 1`

27	data address ∈ protected RAM	The address of the data access is within a protected RAM area	`if ([XGPC]&0x2000 = 0x0000)`
			`data_address ⇐ XGRb + offs5`
			
			`if ([XGPC]&0x2002 = 0x2000)`
			`data_address ⇐ XGRb + XGRi`
			
			`(RAMWPC&0x01 = 0x01) &&`
			`(((data_address > (RAMXGU<<8)+0xFF) &&`
			`(data_address < RAMSHL<<8)) ‖`
			`(data_address > (RAMSHU<<8)+0xFF))`

28	data address ∈ flash space	The address of the data access is within the flash space	`if ([XGPC]&0x2802 = 0x2002)`
			`data_address ⇐ XGRb + XGRi + 1`
			
			`if ([XGPC]&0x2802 = 0x2802)`
			`data_address ⇐ XGRb + XGRi + 2`
			
			`data_address ≥ 0x0800) &&`
			`data_address Š≤0xFFFF-RAM size)`

Figure 5. Observable Conditions for Investigating Software Errors on S12X Devices (Sheet 2 of 3)

Mnemonic		Description	Details
29	Secured flash	The flash has been removed from the XGATE's memory map (due to security)	(MODE&0xC0 ≠ 0x00) && (FSEC&0x03 ≠ 0x02)
30	L15Y	Device mask set ends with "L15Y"	(PARTIDH = 0xC4) && (PARTIDL&0xF0 = 0x10)
31	$\overline{\text{L15Y}}$	Device i mask set does not end with "L15Y"	(PARTIDH ≠ 0xC4) || (PARTIDL&0xF0 ≠ 0x10)

Figure 5. Observable Conditions for Investigating Software Errors on S12X Devices (Sheet 3 of 3)

1 Excluding LDW RD, #IMM16

4.4 Software Error Scenarios

The number of scenarios leading to a XGATE Software Error is limited. They can be broken down into 22 cases. The following sections describe all of these scenarios and show how they can be identified. These descriptions refer to the cycle notation of the XGATE's RISC core (see Figure 6).

`V` — Vector fetch: always an aligned word read, lasts for at least one RISC core cycle

`P` — Program word fetch: always an aligned word read, lasts for at least one RISC core cycle

`r` — 8 bit data read: lasts for at least one RISC core cycle

`R` — 16 bit data read: lasts for at least one RISC core cycle

`w` — 8 bit data write: lasts for at least one RISC core cycle

`W` — 16 bit data write: lasts for at least one RISC core cycle

`A` — Alignment cycle: no read or write, lasts for zero or one RISC core cycle

`f` — Free cycle: no read or write, lasts for one RISC core cycle

Special Cases

`PP/P` — Branch: `PP` if branch taken, `P` if not

Figure 6. Access Detail Notation

4.4.1 Vector Fetches

The vector fetch sequence of the XGATE consists of three memory accesses: A fetch of the starting address of the thread, a fetch of the data segment pointer, and a prefetch of the first instruction. There are four Software Error scenarios that could occur during the vector fetch sequence. They are listed in Figure 7. Observable conditions in this table printed in grey are redundant.

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information
V	1	Vector fetch for the initial PC	2	Vector fetch from an unimplemented address	1	PC = 0xFFFE	XGATE Channel: XGCHID Vector Address: XGVBR+4*XGCHID
					3	PC & 1 = 0	
					4	PC+2 ∈ register space	
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					9	CCR = 0x0	
					10	R1 = 0x0000	
					12	R2..R7 = 0x0000	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					17	VBR+4*CHID ∈ flash space	
					29	Secured flash	
			3	Vector fetch from register space	1	PC = 0xFFFE	
					3	PC & 1 = 0	
					4	PC+2 ∈ register space	
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					9	CCR = 0x0	
					10	R1 = 0x0000	
					12	R2..R7 = 0x0000	
					14	VBR+4*CHID ≠ 0xFFFE	
					15	VBR+4*CHID ∈ register space	
					18	VBR+4*CHID ∉ secured flash space	
V	2	Vector fetch for the initial R1 content	3	Vector fetch from register space	8	PC+2 = [VBR+4*CHID]	XGATE Channel: XGCHID Vector Address: 0xFFFE
					9	CCR = 0x0	
					10	R1 = 0x0000	
					12	R2..R7 = 0x0000	
					13	VBR+4*CHID = 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	

Figure 7. Potential Software Errors During the Vector Fetch Sequence (Sheet 1 of 2)

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information
P	3	1st opcode prefetch	4	Opcode fetch from an odd address	2	PC & 1 = 1	XGATE Channel: XGCHID Vector Address: XGVBR+4*XGCHID Initial Program Counter: XGPC+2 = [XGVBR+4*XGCHID]
					8	PC+2 = [VBR+4*CHID]	
					9	CCR = 0x0	
					11	R1 = [VBR+4*CHID+2]	
					12	R2..R7 = 0x0000	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
			5	Opcode fetch from an unimplemented address .	6	PC+2 ∈ flash space	
					8	PC+2 = [VBR+4*CHID]	
					9	CCR = 0x0	
					11	R1 = [VBR+4*CHID+2]	
					12	R2..R7 = 0x0000	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					29	Secured flash	
			6	Opcode fetch from register space	4	PC+2 ∈ register space	
					7	PC ∉ secured flash space	
					8	PC+2 = [VBR+4*CHID]	
					9	CCR = 0x0	
					11	R1 = [VBR+4*CHID+2]	
					12	R2..R7 = 0x0000	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	

Figure 7. Potential Software Errors During the Vector Fetch Sequence (Sheet 2 of 2)

4.4.2 Illegal Instructions

On S12X devices, the XGATEV2 has actually been shipped in two revisions. Both revisions differ slightly in the way they execute illegal instructions. Figure 8 shows the behavior for both these versions. The mask set number and the PARTID register value can be used to distinguish devices. MCUs with mask sets ending with “L15Y” (PARTIDH register reads 0xC4) contain one revision. The other XGATE revision can be found on all remaining S12X MCUs.

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information
f	4	Illegal Instruction is to be executed	1	Execution of an unimplemented opcode (Device mask set ends with “L15Y“)	3	PC & 1 = 0	XGATE Channel: XGCHID Program Counter: XGPC+2 Instruction: [XGPC+2]
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					25	[PC+2] ∉ legal opcodes	
					30	L15Y	
			1	Execution of an unimplemented opcode (Device mask set does not end with “L15Y“)	3	PC & 1 = 0	XGATE Channel: XGCHID Program Counter: XGPC Instruction: [XGPC]
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					20	[PC] ∉ legal opcodes	
					31	$\overline{\text{L15Y}}$	

Figure 8. Software Errors Caused by Illegal Instructions

4.4.3 Opcode Prefetches in a Linear Flow

The term Single Cycle Instructions includes all instructions that execute in one XGATE cycle, like logic or arithmetic operations. All these instructions execute one opcode prefetch, which could occur from register space if the instruction is located at address 0xFFFE. Figure 9 gives a description of this scenario.

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information
P	5	Opcode prefetch in linear flow	7	Opcode fetch from register space	1	PC = 0xFFFE	XGATE Channel: XGCHID Program Counter: 0xFFFE Instruction: [0xFFFE] CCR: unchanged R1..R7: unchanged
					3	PC & 1 = 0	
					4	PC+2 ∈ register space	
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					19	[PC] ∈ legal opcodes	

Figure 9. Software Errors Caused by Opcode Prefetches in a Linear Program Flow

There are three special instructions that require an alignment cycle (optional cycle) after their opcode prefetch: SIF, SSEM, and CSEM. As alignment cycles cannot trigger Software Errors, these operations may be treated as single cycle instructions.

Software Errors caused by single cycle instructions can not be clearly distinguished from Software Errors caused by jumps or branches to address 0x0000 (Section 4.4.4). An exact determination of the error cause can only be made based on the context of the application.

4.4.4 Branch and Jump Instructions

Branches and jumps require two XGATE bus cycles. They execute two opcode fetches to fill the RISC core's instruction queue after they have set the program counter to the new address location.
Conditional branches behave as single cycle instructions if the branch condition is false. In this case only the first opcode fetch cycle is executed, the second one is skipped.
The Software Error Conditions that might occur during these cycles are listed in Figure 10. Observable conditions in this table printed in grey are redundant.

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information	
P	see Section 4.4.3							
P[1]	6	Opcode prefetch at destination address	5	Opcode fetch from an odd address[2]	2	PC & 1 = 1	XGATE Channel: Program Counter: Instruction: Destination address: Rd:	XGCHID see Section 4.4.4 JAL R? (see Section 4.4.4) XGPC+2 return address
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
			6	Opcode fetch from an unimplemented address .	6	PC+2 ∈ flash space	XGATE Channel: Program Counter: Instruction: Destination address: Rd:[2]	XGCHID see Section 4.4.4 see Section 4.4.4 XGPC+2 return address
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					29	Secured flash		
			7	Opcode fetch from register space	4	PC+2 ∈ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		

Figure 10. Potential Software Errors Caused by Jump or Branch Instructions

1 Omitted for conditional branches if the branch condition is false
2 JAL instructions only

Branches to odd addresses are not possible, as the branch offset is always given in word entities. Only the JAL instruction is able to trigger this Software Error Condition.

Debugging of branch and jump instructions can be a tricky if the Software Error is detected in the second program fetch cycle.

Right before execution of the second P-cycle, the program counter is updated to the instruction preceeding the destination address, which causes two problems for the investigation of Software Errors:

- The address of the current instruction is lost. The XGATE doesn't provide a direct way to determine which branch or jump instruction has caused the Software Error.
- Branches or jumps to address 0x0000 leave exactly the same register signature as instructions executed from address 0xFFFE. These two scenarios cannot be clearly distinguished.

This change of the program counter in the middle of the instruction actually makes it easier to determine the Software Error Condition as the critical address of the opcode fetch can be checked directly (XGPC+2). Finding the exact instruction that caused the problem, however requires some deeper investigation. In general actions can be taken:

- All branch instructions that could have caused the Problem can be found through code linting. Any branch instruction that directs the program flow to the address XGPC+2 is a candidate for causing the problem. To ease this step all branch instructions that point into the register space must be eliminated. However, this is not always possible as these opcodes might by chance also appear in data structures.
- Assuming that the Software Error was caused by a JAL instruction, then the return address (the address following the JAL instruction) is stored in one of the General Purpose Registers (XGR1..XGR7).
  A list of jump instructions that potentially triggered the Software Error can be obtained by iterating through the General Purpose Registers. If all of the following conditions are fulfilled...
  - XGR*n* & 1 ≡ 0
  - XGR*n* > 0x0800
  - [XGR*n*-2] ≡ 0x00F6 | (256**n*) [1]

  ... then the JAL R*n* instruction located at address XGR*n*-2 is a candidate for causing the Software Error.

Any further investigation depends on the context of the application code.

4.4.5 Load and Store Instructions

All load and store instructions perform two bus accesses: an opcode prefetech and a data access. The Software Error Conditions which can be triggered by these instructions are shown in Figure 11.

1. Opcode of the JAL R*n* instruction

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information	
P	see Section 4.4.3							
r	7	Data read or write access	7	Byte read from an unimplemented address	3	PC & 1 = 0	XGATE Channel:	XGCHID
					5	PC ∉ register space	Program Counter:	XGPC
					7	PC ∉ secured flash space	Instruction:	[XGPC]
					14	VBR+4*CHID ≠ 0xFFFE	Data address:	see Section 4.4.5
					16	VBR+4*CHID ∉ register space	R1..R7:	unchanged
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					23	[PC] ∈ LDB		
					28	data address ∈ flash space		
					29	Secured flash		
R			8	Word read from an odd address	3	PC & 1 = 0		
					5	PC ∉ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					24	[PC] ∈ LDW		
					26	data address & 1 = 1		
			9	Word read from an unimplemented address	3	PC & 1 = 0		
					5	PC ∉ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					24	[PC] ∈ LDW		
					28	data address ∈ flash space		
					29	Secured flash		

Figure 11. Potential Software Errors Caused by Load or Store Instructions (Sheet 1 of 4)

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information	
w	7	Data read or write access	10	Byte write to an unimplemented address	3	PC & 1 = 0	XGATE Channel:	XGCHID
					5	PC ∉ register space	Program Counter:	XGPC
					7	PC ∉ secured flash space	Instruction:	[XGPC]
					14	VBR+4*CHID ≠ 0xFFFE	Data address:	see Section 4.4.5
					16	VBR+4*CHID ∉ register space	R1..R7:	unchanged
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					21	[PC] ∈ STB		
					28	data address ∈ flash space		
					29	Secured flash		
			11	Byte write to flash space	3	PC & 1 = 0		
					5	PC ∉ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					21	[PC] ∈ STB		
					28	data address ∈ flash space		
			12	Byte write to protected RAM	3	PC & 1 = 0		
					5	PC ∉ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					21	[PC] ∈ STB		
					27	data address ∈ protected RAM		

Figure 11. Potential Software Errors Caused by Load or Store Instructions (Sheet 2 of 4)

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information
W	7	Data read or write access	13	Word write to an odd address	3	PC & 1 = 0	XGATE Channel: XGCHID Program Counter: XGPC Instruction: [XGPC] Data address: see Section 4.4.5 R1..R7: unchanged
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					19	[PC] ∈ legal opcodes	
					22	[PC] ∈ STW	
					26	data address & 1 = 1	
			14	Word write to an unimplemented address	3	PC & 1 = 0	
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					19	[PC] ∈ legal opcodes	
					21	[PC] ∈ STB	
					28	data address ∈ flash space	
					29	Secured flash	
			15	Word write to flash space	3	PC & 1 = 0	
					5	PC ∉ register space	
					7	PC ∉ secured flash space	
					14	VBR+4*CHID ≠ 0xFFFE	
					16	VBR+4*CHID ∉ register space	
					18	VBR+4*CHID ∉ secured flash space	
					19	[PC] ∈ legal opcodes	
					21	[PC] ∈ STB	
					28	data address ∈ flash space	

Figure 11. Potential Software Errors Caused by Load or Store Instructions (Sheet 3 of 4)

Cycle	Description		Software Error Condition		Obserservable Condition		Relevant Debug Information	
W	7	Data read or write access	16	Word write to protected RAM	3	PC & 1 = 0	XGATE Channel: Program Counter: Instruction: Data address: R1..R7:	XGCHID XGPC [XGPC] see Section 4.4.5 unchanged
					5	PC ∉ register space		
					7	PC ∉ secured flash space		
					14	VBR+4*CHID ≠ 0xFFFE		
					16	VBR+4*CHID ∉ register space		
					18	VBR+4*CHID ∉ secured flash space		
					19	[PC] ∈ legal opcodes		
					21	[PC] ∈ STB		
					27	data address ∈ protected RAM		

Figure 11. Potential Software Errors Caused by Load or Store Instructions (Sheet 4 of 4)

To investigate the cause of a failed data access, the data address must be reconstructed out of the opcode and the referred general purpose registers. These registers (destination register and auto-inc/decremented index register) are not changed if a Software Error occurs.

Because destination registers and auto-in/decremented index registers do not change if the instruction does not execute properly; the data address and therefore the reason for Software Error can be clearly determined.

4.5 Finding the Cause of a Software Error

There are 16 Software Error conditions and 31 observable conditions on an S12X device. To maintain an overview of their relations Figure 12 summarizes the relevant information of Section 4.4. The rows in this table list all observable conditions, the columns itemize the Software Error scenarios. All conditions which must hold true for a scenario are marked with an "**x**".

Whenever an observable condition can be proven false, all Software Error conditions which are marked in this row can be eliminated. The goal of the investigation is now to eliminate all but one scenario (column) by iteration though the conditions (rows). When iterating through the conditions, only those which can eliminate additional scenarios need to be checked. All others are redundant and can be skipped.

There are cases where this elimination process leaves two possible Software Error scenarios (see Section 4.6.3). In these cases further investigation depending in the context of the program code is required.

Observable Condition		Cycle	1		2		3			4		5		6			7									
			Vector fetch for the initial PC		Vec. fetch for the init. R1		1st opcode prefetch			Illegal Instruction is to be executed		Opcode prefetch in linear flow		Opcode prefetch at destination address			Data read or write access									
		SWE Condition	VF fr. unimpl. addr.	VF fr. reg. space	VF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	Unimpl. opc.	Unimpl. opc.	OF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	BR fr. unimpl. addr.	WR fr. odd addr.	WR fr. unimpl. addr.	BW to unimpl. addr.	BW to Flash space	BW to prot. RAM	WW to odd addr.	WW to unimpl. addr.	WW to Flash space	WW to prot. RAM		
			2	3	3	4	5	6	1	1	6	4	5	6	7	8	9	10	11	12	13	14	15	16		
1	PC = 0xFFFE		X	X							X															
2	PC & 1 = 1					X						X														
3	PC & 1 = 0		X	X					X	X	X				X	X	X	X	X	X	X	X	X	X		
4	PC+2 ∈ register space		X	X				X			X			X												
5	PC ∉ register space		X	X						X	X				X	X	X	X	X	X	X	X	X	X		
6	PC+2 ∈ flash space						X						X													
7	PC ∉ secured flash space		X	X				X		X	X			X	X	X	X	X	X	X	X	X	X	X		
8	PC+2 = [VBR+4*CHID]				X	X	X	X																		
9	CCR = 0x0		X	X	X	X	X	X																		
10	R1 = 0x0000		X	X	X																					
11	R1 = [VBR+4*CHID+2]					X	X	X																		
12	R2..R7 = 0x0000		X	X	X	X	X	X																		
13	VBR+4*CHID = 0xFFFE				X																					
14	VBR+4*CHID ≠ 0xFFFE		X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
15	VBR+4*CHID ∈ register space			X																						
16	VBR+4*CHID ∉ register space		X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
17	VBR+4*CHID ∈ flash space		X																							
18	VBR+4*CHID ∉ secured flash space			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
19	[PC] ∈ legal opcodes										X				X	X	X	X	X	X	X	X	X	X		
20	[PC] ∉ legal opcodes									X																
21	[PC] ∈ STB																	X	X	X						
22	[PC] ∈ STW																				X	X	X	X		
23	[PC] ∈ LDB														X											
24	[PC] ∈ LDW															X	X									

Figure 12. Mapping Observable Conditions to Software Errors Conditions (Sheet 1 of 2)

Observable Condition		Cycle	Vector fetch for the initial PC		Vec. fetch for the init. R1	1st opcode prefetch			Illegal Instruction is to be executed		Opcode prefetch in linear flow	Opcode prefetch at destination address			Data read or write access									
			1		2	3			4		5	6			7									
		SWE Condition	VF fr. unimpl. addr.	VF fr. reg. space	VF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	Unimpl. opc.	Unimpl. opc.	OF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	BR fr. unimpl. addr.	WR fr. odd addr.	WR fr. unimpl. addr.	BW to unimpl. addr.	BW to Flash space	BW to prot. RAM	WW to odd addr.	WW to unimpl. addr.	WW to Flash space	WW to prot. RAM
			2	3	3	4	5	6	1	1	6	4	5	6	7	8	9	10	11	12	13	14	15	16
25	[PC+2] ∉ legal opcodes								X															
26	data address & 1 = 1															X					X			
27	data address ∈ protected RAM																			X				X
28	data address ∈ flash space														X		X	X	X			X	X	
29	Secured flash		X				X						X		X		X	X				X		
30	L15Y								X															
31	$\overline{\text{L15Y}}$									X														

Figure 12. Mapping Observable Conditions to Software Errors Conditions (Sheet 2 of 2)

4.6 Examples

The following examples demonstrate the investigation process on S12X devices.

4.6.1 Unsuccessful Vector Fetch

In the first example the following register values (see Figure 13) can be read if the XGATE is in the Software Error State.

Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0380 XGMCTL	0x00								1	0	1	0	0	0	1	1
									XGE	XGFRZ	XGDBG	XGSS	XG FACT		XG SWEIF	XGIE
0x0382 XGMCHID									0x38							
0x0386 XGVBR	0x0200															

Figure 13. XGATE Register Values

These register values reveal that condition 16 (see Figure 5) is false. Proving this condition false is enough to conclude the cause of the software error (see Figure 14).

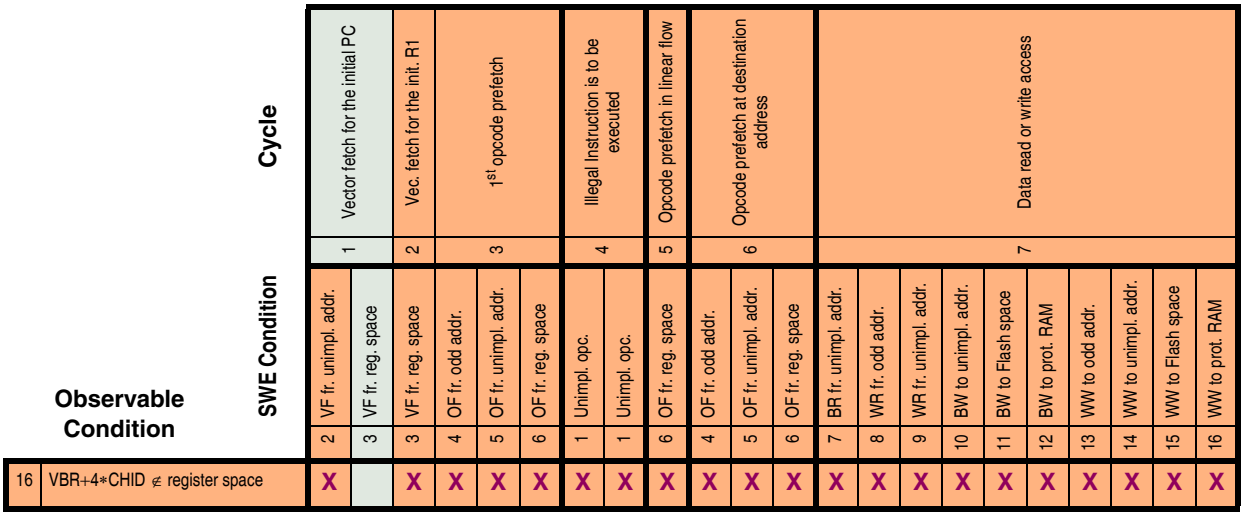

Figure 14. Detecting a vector fetch from register space

Channel 0x38 has been triggered and the XGATE tried to fetch the first vector from register space. The Vector Base Register (XGVBR) doesn't point to the correct vector table.

4.6.2 Illegal Opcode

In the second example the following register values (see Figure 15) can be observed when the XGATE is in the Software Error State. The program code shown in Figure 16 can be read from the RAM and it is assumed that the error handler of the CPU12X didn't modify any of these RAM locations.

Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0380 XGMCTL	0x00								1 XGE	0 XGFRZ	1 XGDBG	0 XGSS	0 XG FACT	0	1 XG SWEIF	1 XGIE
0x0382 XGMCHID									0x24							
0x0386 XGVBR	0x8000															
0x039E XGPC	0x9238															

Figure 15. XGATE Register Values

```
9234    01 00                    NOP
9236    01 00                    NOP
9238    0D                       DB      $0D ;CR
9239    44 6F 6E 27 74 20 65 78  FCC     "Don't execute me!"
        65 63 75 74 65 20 6D 65
        21
```

Figure 16. XGATE Code Sequence

Only six observable conditions need to be checked to determine the cause of the problem (see Figure 17).

The illegal instruction 0x0D has been executed at address 0x9238. And without checking the PARTID register one can be certain that the mask set number of the device does not end with “L15Y”.

Observable Condition		Cycle	1		2	3			4		5	6			7										
			Vector fetch for the initial PC		Vec. fetch for the init. R1	1st opcode prefetch			Illegal Instruction is to be executed		Opcode prefetch in linear flow	Opcode prefetch at destination address			Data read or write access										
		SWE Condition	VF fr. unimpl. addr.	VF fr. reg. space	VF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	Unimpl. opc.	Unimpl. opc.	OF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	BR fr. unimpl. addr.	WR fr. odd addr.	WR fr. unimpl. addr.	BW to unimpl. addr.	BW to Flash space	BW to prot. RAM	WW to odd addr.	WW to unimpl. addr.	WW to Flash space	WW to prot. RAM
			2	3	3	4	5	6	1	1	6	4	5	6	7	8	9	10	11	12	13	14	15	16
2	PC & 1 = 1					X						X												
4	PC+2 ∈ register space		X	X				X			X			X										
6	PC+2 ∈ flash space						X						X											
13	VBR+4*CHID = 0xFFFE				X																			
19	[PC] ∈ legal opcodes										X				X	X	X	X	X	X	X	X	X	X
25	[PC+2] ∉ legal opcodes								X															

Figure 17. Detecting the execution of an illegal instruction.

4.6.3 Failed Opcode Prefetch

The cause of the next Software Error is a little bit more difficult to identify. If the XGATE remains in Software Error State the register values shown in Figure 18 and the memory content shown in Figure 19 can be read. It is assumed that the CPU12X doesn’t modify any of the XGATE’s program code during the investigation of the Software Error.

Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x001A PARTIDH									0xC1							
0x001B PARTIDL									0x01							
0x0380 XGMCTL	0x00								1	0	1	0	0	0	1	1
									XGE	XGFRZ	XGDBG	XGSS	XG FACT		XG SWEIF	XGIE
0x039E XGPC	0xFFFE															
0x03A2 XGR1	0x1060															
0x03A4 XGR2	0xD050															
0x03A6 XGR3	0xD04E															
0x03A8 XGR4	0x6787															
0x03AA XGR5	0x3689															
0x03AC XGR6	0x0126															
0x03AE XGR7	0x003F															

Figure 18. XGATE Register Values

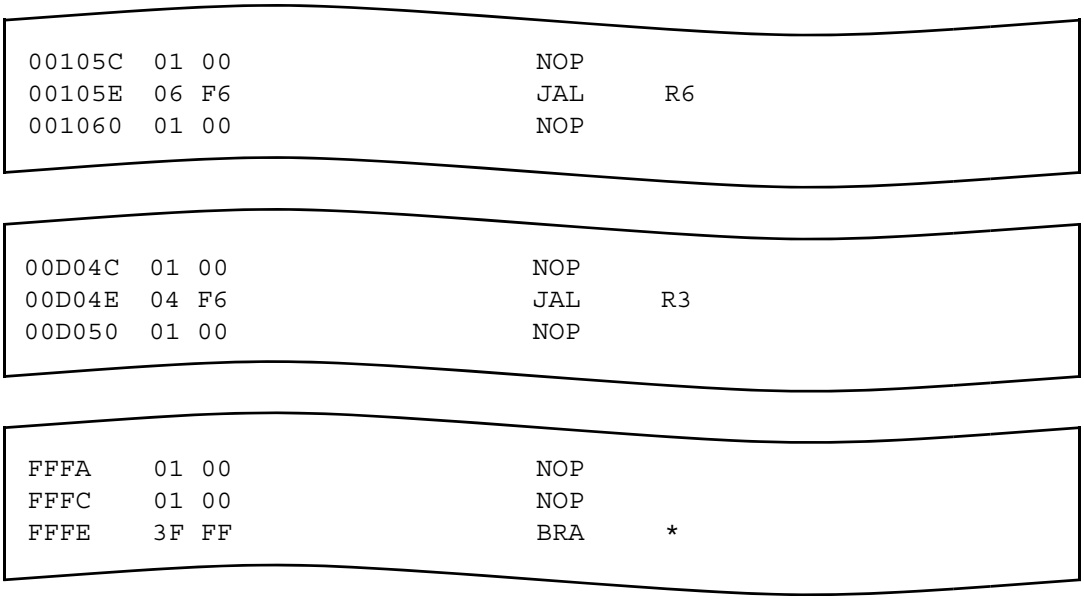

Figure 19. XGATE Code Sequence

Iterating through the observable conditions (see Figure 20) shows that it takes nine false conditions to narrow down the possible causes of the Software Error to two. The two possibilities are:

- An instruction has been executed from address 0xFFFE and an opcode prefetch to address 0x000 has occurred
- A Jump or Branch to address 0x0000 has been executed

None of the 31 identified conditions (see Figure 5) is able to distinguish between these possible causes, so further investigation must be done.

Scanning through the entire 64KB address space of the XGATE shows that there is no Branch instruction which directs the program flow to address 0x0000.

Jump (JAL) instructions leaves a return address in of the General Purpose Registers (XGR1-XGR7). Scanning through those registers shows that

- XGR7 and XGR6 don’t contain a valid return address, because they point into the register space
- XGR5 and XGR4 contain odd values which are no valid return addresses
- XGR3, XGR2, and XGR1 point into valid program space, but there are no matching JAL instructions preceding each return address

By excluding Jump and Branch instructions it is certain that the problem was caused by executing an instruction from address 0xFFFE.

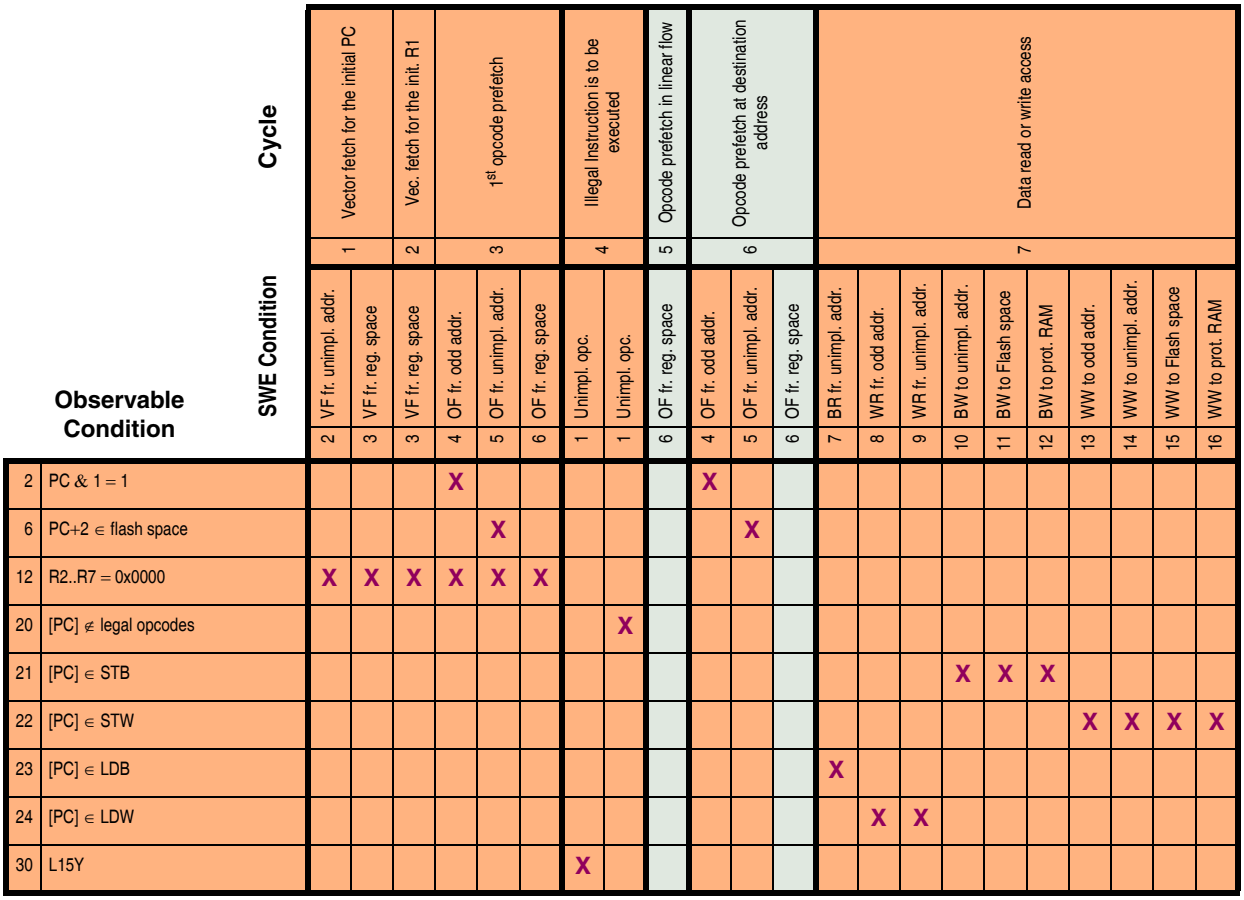

Figure 20. Detecting Two Possible Causes

4.6.4 Failed Write Access

In the last example, the Software Error is triggered resulting in the register values shown in Figure 21.

Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0101 FSEC									0xFE							
0x0380 XGMCTL	0x00								1 XGE	0 XGFRZ	1 XGDBG	0 XGSS	0 XG FACT	0	1 XG SWEIF	1 XGIE
0x0382 XGMCHID									0x44							
0x0386 XGVBR	0xD000															
0x039E XGPC	0x1236															
0x03AC XGR6	0x1000															
0x03AE XGR7	0x3000															

Figure 21. XGATE Register Values

The program counter of the XGATE points to the following content of the RAM (see Figure 22).

```
001234  01 00                    NOP
001236  75 DE                    STB     R5, (R6,-R7)
001238  01 00                    NOP
```

Figure 22. XGATE Code Sequence

By looking at nine observable conditions (see Figure 23) it is evident, that the Software Error was caused by an illegal write access. Looking at the instruction at the program counter and the associated general purpose register shows, that a write to address 0x4000 was attempted. This address is located in the Flash space and Software Error Condition 11 is fulfilled.

Observable Condition		Cycle	1		2	3			4		5	6			7											
			Vector fetch for the initial PC		Vec. fetch for the init. R1	1st opcode prefetch			Illegal Instruction is to be executed		Opcode prefetch in linear flow	Opcode prefetch at destination address			Data read or write access											
		SWE Condition	VF fr. unimpl. addr.	VF fr. reg. space	VF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	Unimpl. opc.	Unimpl. opc.	OF fr. reg. space	OF fr. odd addr.	OF fr. unimpl. addr.	OF fr. reg. space	BR fr. unimpl. addr.	WR fr. odd addr.	WR fr. unimpl. addr.	BW to unimpl. addr.	BW to Flash space	BW to prot. RAM	WW to odd addr.	WW to unimpl. addr.	WW to Flash space	WW to prot. RAM
			2	3	3	4	5	6	1	1	6	4	5	6	7	8	9	10	11	12	13	14	15	16
2	PC & 1 = 1					X						X												
4	PC+2 ∈ register space		X	X				X			X			X										
12	R2..R7 = 0x0000		X	X	X	X	X	X																
20	[PC] ∉ legal opcodes									X														
22	[PC] ∈ STW																				X	X	X	X
25	[PC+2] ∉ legal opcodes								X															
26	data address & 1 = 1															X					X			
27	data address ∈ protected RAM																			X				X
29	Secured flash		X				X						X		X		X	X				X		

Figure 23. Detecting an Illegal Write Access

5 References

1. Datasheet to MC9S12XDP512, Freescale Semiconductor Inc., 2005.

Appendix A

A.1 List of S12X MCUs

The following tables list the S12X MCUs which are available as of the release of this application note. All of these devices contain an XGATEV2 as it is described in this document.

S12XA Family

Part Number	Mask Set	Part ID
MC9S12XA256	1M84E	0xC001

Figure 5-24. MCUs of the S12XA Family

A.1.1 S12XB FamilyS12XD Family

Part Number	Mask Set	Part ID
MC9S12XB256	1M84E	0xC001
MC9S12XB128	2M42E	0xC102

Figure 5-25. MCUs of the S12XB Family

Part Number	Mask Set	Part ID
MC9S12XDP512[1]	0L15Y	0xC410
	1L15Y	0xC411
MC9S12XDT512[1]	1L15Y	0xC411
MC9S12XDT384[1]	1L15Y	0xC411
MC9S12XDQ256	0M84E	0xC000
	1M84E	0xC001
MC9S12XDT256	1M84E	0xC001
MC9S12XD256	1M84E	0xC001
MC9S12XDG128	0M42E	0xC100
	1M42E	0xC101
	2M42E	0xC102
MC9S12XDG128	2M42E	0xC102
MC9S12XD128	2M42E	0xC102
MC9S12XD64	2M42E	0xC102

Figure 5-26. MCUs of the S12XD Family

[1] Please pay attention to Section 4.4.2

A.1.2 S12XH Family

Part Number	Mask Set	Part ID
MC9S12XHZ512	0M80F	0xE460
	1M80F	0xE461
MC9S12XHZ384	1M80F	0xE461
MC9S12XHZ256	1M80F	0xE461

Figure 5-27. MCUs of the S12XH Family

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3555
Rev. 0
11/2007

freescale™
semiconductor