Freescale Semiconductor
Application Note

AN3458
Rev. 0, 05/2007

Debugging XGATE Code

Debug Features of S12X(E) MCUs

by: Dirk Heisswolf
MCD Design
Munich, Germany

The XGATE peripheral coprocessor is designed to boost the performance of an S12X(E) controller by unloading the main CPU. The XGATE can perform many tasks performed by the single CPU of previous S12 families.

When it comes to debugging XGATE code, you do not have the option of using an in-circuit debugger as you might have for the CPU12X. You rely on the debug capabilities that are built into each S12X(E) microcontroller. This application note has an overview of the debug features built into the hardware. It is intended for readers who would like to use these features directly or readers who would like to understand how high-level debug environments interact with an S12X(E) MCU.

Section 1 introduces the XGATE and its built-in debug support.

Section 2 describes the S12XDBG module and how to use it to generate intelligent breakpoints and traces.

Section 3 provides examples of debug scenarios. These can be used as a starting point to find a setup for your specific debug need.

Table of Contents

freescale semiconductor

1 Introduction to the XGATE

The XGATE is a coprocessor for the S12X(E) CPU that can serve multiple purposes. It can be used as a DMA controller, it can run driver code for the MCU's peripherals, it can generate low-latency system responses, and it can be used for many other applications. Here is a short overview of the main characteristics of the XGATE and its debug features.

1.1 RISC Core

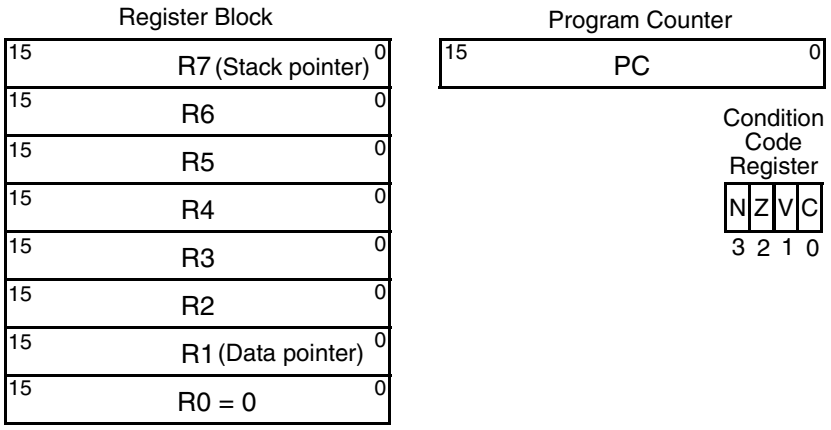

Figure 1. Programmer's Model

The XGATE consists of a RISC core that is triggered through interrupts and is powered down when not in use. On S12X devices, interrupts that are handled by the XGATE cannot be nested. A new interrupt can be serviced only when the previous interrupt activity has finished. On S12XE devices, one level of interrupt nesting is possible.

The RISC core has a set of seven general-purpose registers in its register block, a program counter, and a condition code register (see Figure 1). These registers are mapped to the XGATE's register space. They can be read and modified when the XGATE is stopped for debug purposes.

1.2 Stages of Operation

The interaction with the XGATE module can be categorized into three stages:

- Initial configuration
- Actual application
- Debug

1.2.1 Initial Configuration

After a system reset, the XGATE is not ready to execute application code. It remains in a disabled state, waiting to be configured by the CPU12X. This is the time when the vector base register (XGVBR) and the initial stack pointer registers[1] (XGISP74 and XGISP31) must be set. Also, program code and vector tables,

1. S12XE devices only.

which are supposed to reside in RAM, must now be initialized. Enabling the XGATE (setting the XGE bit in the XGMCTL register) causes the XGATE to proceed to the application stage.

1.2.2 Application

The XGATE performs its work in the application stage. It operates autonomously, except when it detects a software problem. Then it requests help from the CPU12X.

- Idle/running—When the XGATE becomes enabled after the initial configuration, it starts out in a low-power idle state, waiting for a service request from a peripheral module or from the CPU12X. When a service request comes in, it executes its associated thread of code and reenters the idle state upon completion.
- Software error handling—During code execution a number of conditions are checked, which could indicate faulty application code. These error conditions are checked on an S12X(E) device:
  - Execution of an illegal instruction
  - Code execution from register space (address range 0x0000 to 0x0800)
  - Opcode/vector fetch from an odd address
  - 16-bit load/store accesses to an odd address
  - Write accesses to flash memory
  - S12X_MPU access violations[1]

When one of these error conditions occurs, the XGATE stops (even in the middle of an instruction). It enters a software error state that allows the CPU12X to analyze the failure and to reinitialize the XGATE module.

1.2.3 Debugging

The XGATE provides two ways to leave the application stage for debug purposes: debug mode and freeze mode.

Debug mode stops the program execution and provides access to the internal resources of the XGATE's RISC core. Section 1.5, "Debug Features," describes the debug features enabled in this mode. There are three ways to enter debug mode:

- Manually set the XGDBG through a write access to the XGMCTL register
- Execute a BRK instruction
- Generate a breakpoint through the S12XDBG module

To resume normal operation, the XGDBG bit must be cleared through a write access to the XGMCTL register.

In freeze mode (BDM active), the XGATE can also be configured (XGFRZ bit set) to seize program execution whenever the CPU12X enters BDM active mode. This can be helpful for debugging tasks that involve interaction between the CPU12X and the XGATE.

1. S12XE devices only.

1.3 Memory Map

The XGATE is capable of accessing a subset of the MCU’s memory. It has its own memory map, which contains the full register space, a portion of the chip’s RAM, and a portion of the flash memory. The XGATE memory map is linear and static. There are no mapping or page registers.

Figure 2 shows the memory map of an S12X(E) device.

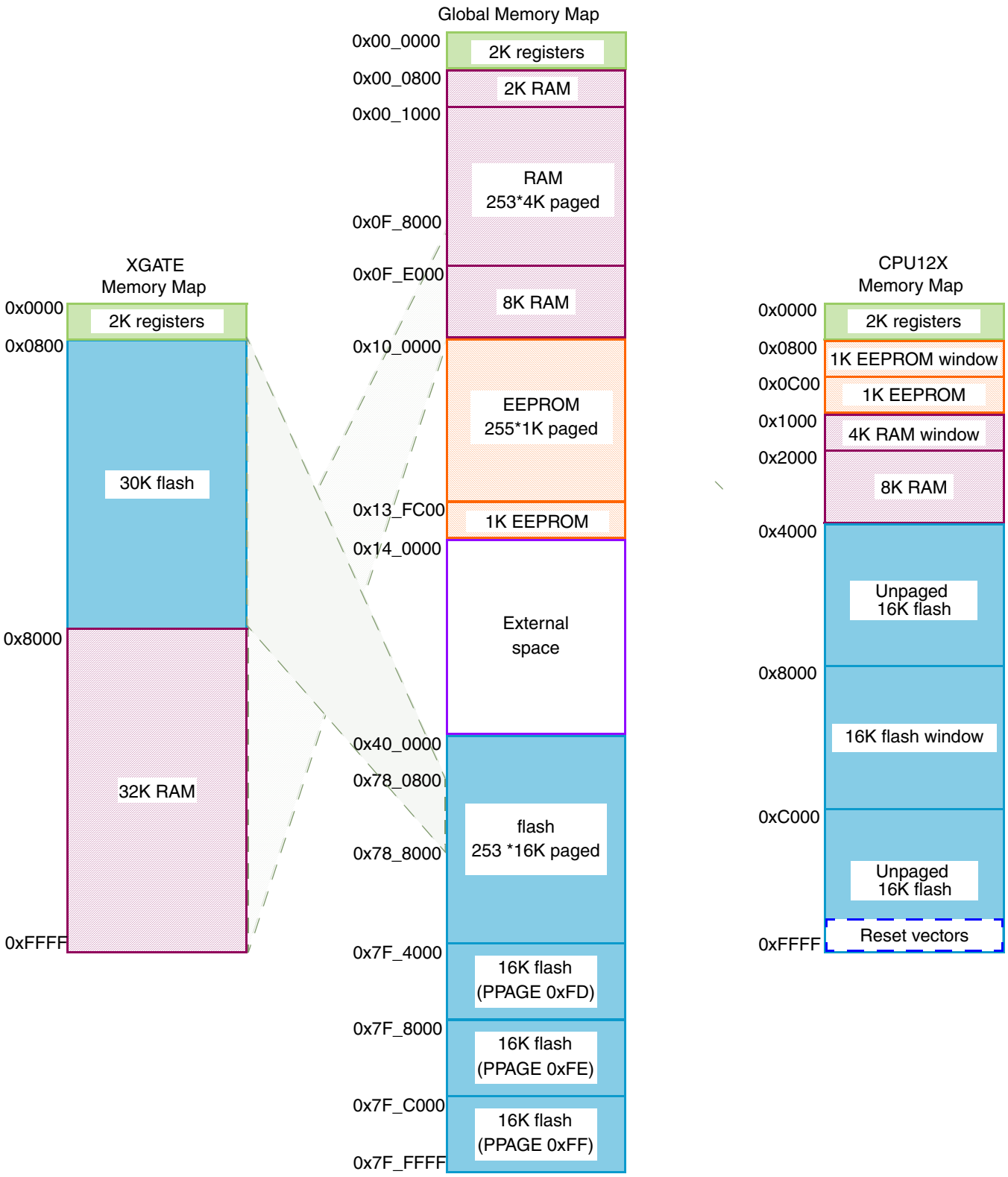

Figure 2. S12X(E) Memory Map

1.4 Status and Control Registers

Similar to other peripherals of S12X(E) devices, a register interface controls the XGATE. This set of registers is mapped to the address range 0x0380 to 0x03AF (in any memory map). To debug XGATE application code, these registers must be accessed via BDM hardware commands or via monitor code running on the main CPU.

Figure 3 summarizes the XGATE's registers and explains their purpose in the three situations: configuration, running application code, and debugging.

Register	Usage		
	Initial Configuration	**Application**	**Debugging**
XGMCTL Module control register			
XGE	• Enable write access to XGISP74, XGISP31, and XGVBR	• Disable incoming requests	—
XGFRZ	—	—	• Suspend XGATE activities while the CPU12X is in BDM active mode • Synchronize concurrent XGATE/CPU12X code
XGDBG	—	—	• Manually enter and leave debug mode
XGSS	—	—	• Execute a single instruction out of debug mode
XGFACT	—	• Keep clocks of peripheral modules running in STOP mode	—
XGSWEF	—	• Resume operation after a software error has occurred (to be cleared by error handler)	—
XGIE	—	• Disable maskable XGATE interrupts	—
XGCHID Channel ID register	—	• Check the state of the XGATE (idle or busy)	• Initiate and terminate threads
XGCHPL Channel priority level	—	• Check the priority level of the current thread	• Initiate a thread with a certain priority level
XGISPSEL XGISPxx select register	• Map either XGISP74, XGISP31, or XGVBR to address 0x0386	—	—
XGISP74 **XGISP31** Initial stack pointer registers	• Select the stack segment for each priority level	—	—

Figure 3. XGATE Register Usage

Register	Usage		
	Initial Configuration	Application	Debugging
XGVBR Vector base register	• Select the vector table	—	—
XGIF Channel interrupt flags	—	• Poll and clear channel interrupt flags	—
XGSWT Software triggers	—	• Trigger XGATE requests or CPU12X interrupts by software	—
XGSEM Semaphores	—	• Synchronize concurrent XGATE/CPU12X code	—
XGCCR Condition code register	—	—	• Read and modify condition code bits
XGPC Program counter	—	—	• Determine the current program counter • Jump to a different location in the program
XGR1 **XGR2** **XGR3** **XGR4** **XGR5** **XGR6** **XGR7** General-purpose registers	—	—	• Read and modify register content

Figure 3. XGATE Register Usage (continued)

1.5 Debug Features

The XGATE module has a number of built-in debug features that are enabled when debug mode is entered (XGDBG bit set).

1.5.1 Manually Starting and Terminating Threads

In debug mode, threads can be started by writing a non-zero value to the channel ID register (XGCHID). This has the same effect as if the equivalent request of the peripheral module or CPU12X would have been received by a running application. The execution of the thread begins when debug mode is left.

On S12XE devices, the priority of a thread can be set by writing to the XGCHID and the XGCHPL register simultaneously.

To terminate a thread in debug mode, 0x00 must be written to the XGCHID register. This has the same effect as a RTS instruction being executed by the XGATE's RISC core.

1.5.2 Single Stepping

If a thread is active (XGCHID ≠ 0x00) in debug mode, a single instruction can be executed by writing a 1 to the XGSS bit. Debug mode is temporarily left while the execution takes place.

1.5.3 Manipulating RISC Core Registers

When the XGATE is in debug mode, the program counter (PC), the condition code register (CCR), and all general-purpose registers (R1 to R7) are mapped into the module's register space. These registers can then be read or written by the CPU12X or by BDM hardware commands.

2 Introduction to the S12XDBG Module

The S12XDBG module provides two important features for debugging XGATE code: intelligent breakpoints and a trace buffer to record bus transactions. The next sections explain how to operate this module.

2.1 Comparators

The S12XDBG module has four comparators (A, B, C, and D) to monitor bus transactions of the XGATE and the CPU12X. Each comparator can be assigned to either one of the cores. The comparators check for different properties of a bus transaction. The data bus can be monitored by comparators A and C only. The data size can be monitored by comparators B and D only (Figure 4).

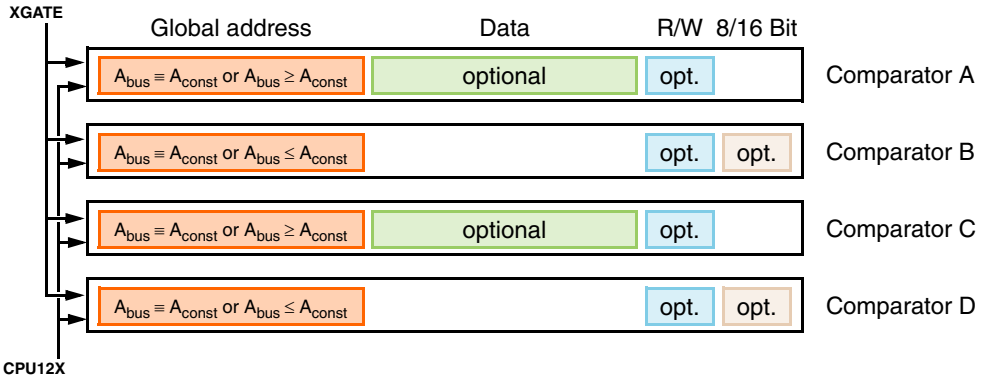

Figure 4. Comparators of the S12XDBG module

2.1.1 Tagged and Forced Comparator Outputs

The output of each comparator can be processed on two ways: it can be used directly (forced triggers) or passed to the XGATE or CPU12X as instruction tag (tagged triggers). These tags are fed into the instruction queue of the core (Figure 5). When the instruction is about to be executed, the tag is passed back.

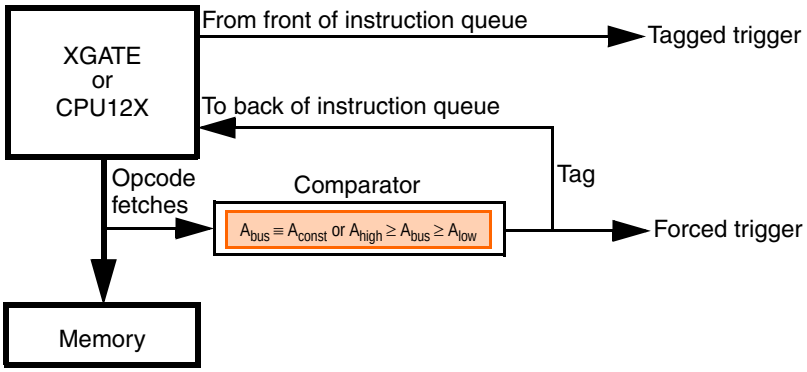

Figure 5. Tagged vs. Forced Comparator Outputs

This detour through the XGATE or CPU12X core makes it possible to generate a comparator hit immediately before a selected instruction becomes executed. The use of direct output generates comparator hits when the opcode of a selected instruction is fetched.

This example illustrates the difference between the two types of comparator outputs (Figure 6). The program code contains a loop that performs eight iterations. The comparator is set to the first instruction after the loop. Running this code generates eight forced comparator hits (due to opcode prefetching of the BNE instruction), but only one tagged comparator is hit.

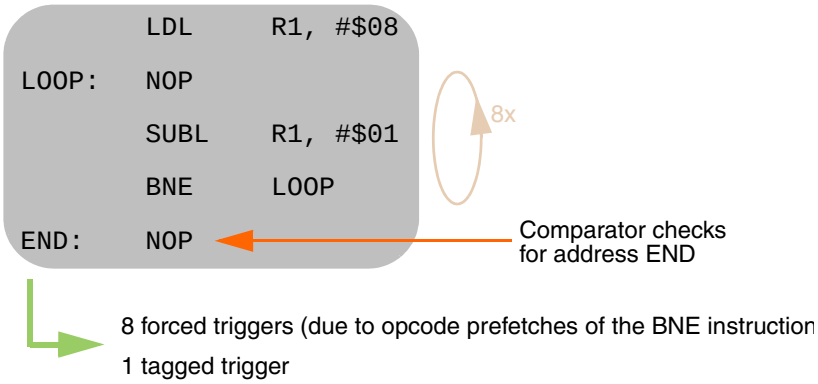

Figure 6. Behavior of Tagged and Direct Comparator Outputs

Tagged triggers are used for setting breakpoints before an instruction boundary. Forced triggers are used for setting breakpoints on data accesses.

2.2 Matches

The comparator outputs of the S12XDBG module (direct and tagged) are fed into a match logic (Figure 7). This match logic provides the option to combine two comparator outputs to perform address range checking. In most of the examples in Section 3, “Examples,” the comparator outputs (A to D) map to the match outputs (1 to 4).

Figure 7. Match Events

2.3 State Sequencer

The outputs of the match logic control a finite state machine called the state sequencer (Figure 8). The state sequencer has five states: an initial disarmed state (state0), three intermediate states with configurable transitions (state1 to state 3), and a final state that can trigger a breakpoint or invoke the trace buffer.

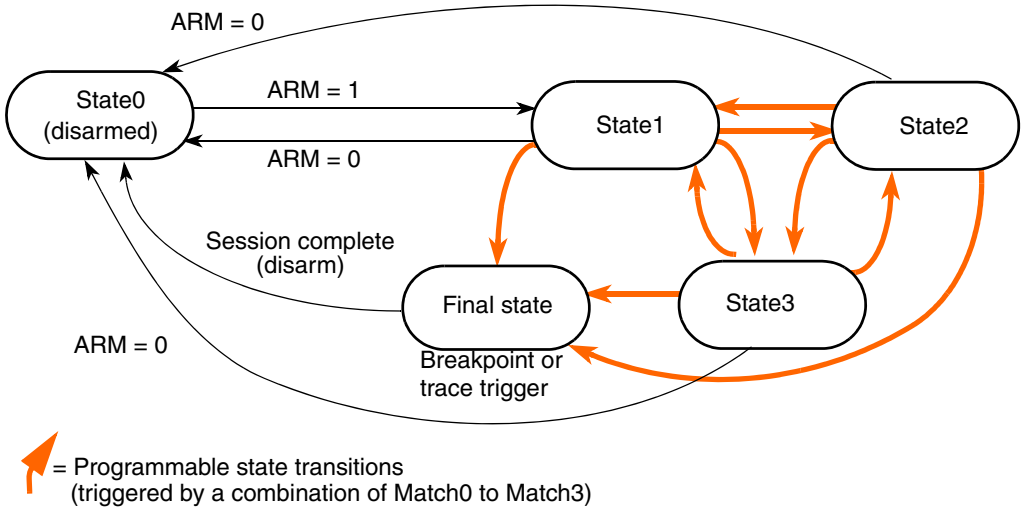

Figure 8. State Sequencer Diagram

A breakpoint can be set up by configuring a transition from state1 to the final state. Complex breakpoint conditions can be achieved by multiple transitions between state1, state2, and state3 followed by a transition to final state.

2.4 Tracing

The S12XDBG module contains an internal trace buffer that can record program flow and data transfers of the XGATE and the CPU12X.

2.4.1 Trace Modes

The trace buffer of the S12XDBG module can record up to 128 entries. Four trace modes allow efficient use of the buffer capacity (Figure 9).

Trace Mode	Type of Bus Transaction			
	Any executed instruction	Any change of flow instruction	First change of flow instruction of a loop	Any data transfer
Normal Mode	No	Yes	Yes	No
Loop1 Mode	No	No	Yes	No
Detail Mode	No	No	No	Yes
Pure PC Mode	Yes	No	No	No

Figure 9. Trace Modes

- Normal mode—Normal mode produces a trace of the program flow. To save trace buffer entries, only changes of the linear flow (conditional branches, indexed jumps, and interrupts) are recorded. By matching this trace against the program memory, the complete program flow can be reconstructed.
- Loop1 mode—Loop1 mode works exactly like normal mode with one exception, it ignores recurring entries resulting from conditional branches. This mode reduces trace buffer entries when executing loops.

  Tracing a loop that does not contain conditional branch instructions inside its body, results in a single trace buffer entry. However, tracing the same loop in normal mode generates one entry per loop iteration.
- Detail mode—Detailed mode generates a trace of data transfers (no opcode fetches).
- Pure PC mode— Pure PC also traces the program flow. Unlike normal or loop1 mode, a trace buffer entry is generated for every instruction.

2.4.2 Trace Alignment

The alignment of the trace can be adjusted relative to an event generated by the state sequencer. S12XDBG module offers three options (Figure 10).

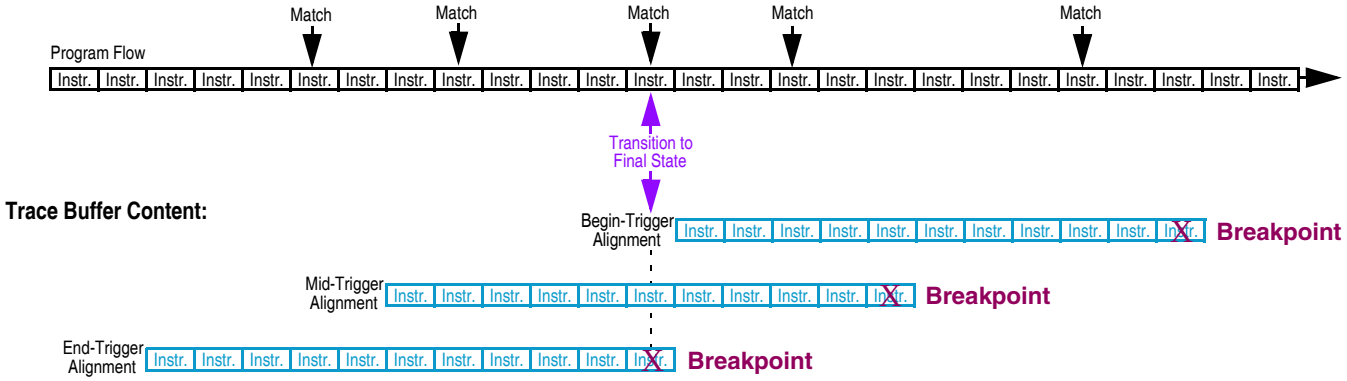

Figure 10. Trace Alignment

- Begin trigger alignment—The trace begins when the state sequencer enters the final state. The S12XDBG module keeps the cores running until the trace buffer is full, then it executes a breakpoint.
- End trigger alignment—The trace buffer behaves as a FIFO in this case. Tracing begins immediately when the S12XDBG module is armed. When the trace buffer is full, the most recent entry replaces the oldest one. When final state is reached a breakpoint is executed immediately. The trigger event appears at the end of the trace.
- Mid-trigger alignment—Mid-trigger alignment is a combination of the other two alignment methods. The trace buffer behaves like a FIFO, but, when final state is reached, all cores run until half the buffer is filled with new entries. Then a breakpoint is executed. The trigger event appears in the middle of the trace.

3 Examples

The following sections show a number of debug scenarios that can be performed with the S12XDBG module. Each example comes with a detailed setup of the debug module that may be used as template for further debug challenges.

3.1 Software Breakpoints

Because XGATE code is usually executed from RAM, software, breakpoints are the simplest debug method. All that needs to be done is to write a BRK instruction (0x0000) to the desired address location. As soon as this BRK becomes executed, the XGATE enters debug mode and the S12XDBG module transitions to final state. In debug mode, the program counter of the XGATE (XGPC) shows the address of the breakpoint. Before continuing program execution, the BRK instruction must be replaced by the original opcode. Software breakpoints work even if the S12XDBG module is disabled. Their main advantage is that they can be set in nearly unlimited number.

3.2 Simple Hardware Breakpoints

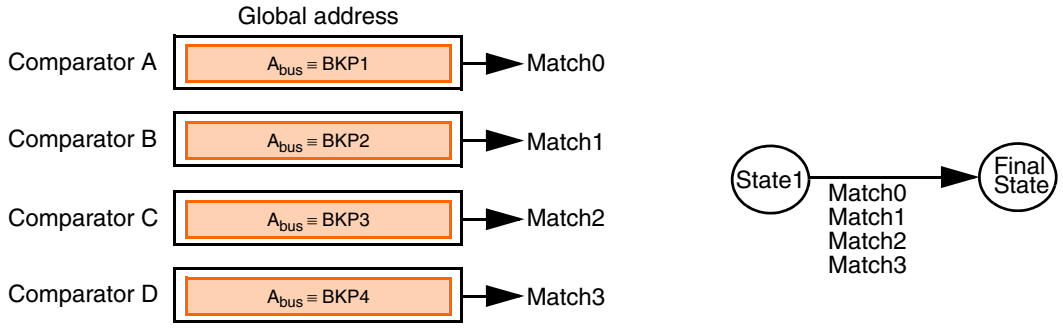

Figure 11. S12XDBG Configuration

When executing code from flash memory, hardware breakpoints are the method of choice. Figure 11 shows how to set four hardware breakpoints (BRK1 to BRK3). The four comparators of the S12XDBG module are configured to check the breakpoint addresses against the address bus (A_{bus}) of the XGATE. The tagged comparator outputs are directly mapped to the four outputs of the match logic (match0 to match3). The setup of the state sequencer causes a transition to final state as soon as any of the four match events occurs.

Figure 12 shows the register setup for this configuration. Relevant register bits are highlighted.

Name	Bit 7	6	5	4	3	2	1	Bit 0	
DBGC1	ARM	TRIG	XGSBPE	BDM	DBGBRK		COMRV		0x04
	0	0	0	0	01		—		
DBGTCR	TSOURCE		TRANGE		TRCMOD		TALIGN		0x00
	00		00		00		00		
DBGC2					**CDCM**		**ABCM**		**0x00**
					00		**00**		
DBGSCR1					**SC3**	**SC2**	**SC1**	**SC0**	**0x02**
					0	**0**	**1**	**0**	
DBGSCR2					SC3	SC2	SC1	SC0	0x00
					0	0	0	0	
DBGSCR3					SC3	SC2	SC1	SC0	0x00
					0	0	0	0	

Figure 12. S12XDBG Register Setup

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
Comparator A (COMRV = 0)	DBGACTL		NDB	**TAG**	BRK	RW	RWE	**SRC**	**COMPE**	**0x23**
			0	**1**	0	0	0	**1**	**1**	
	DBGAAH		0	0	0	0	0	0	0	0x00
	DBGAAM DBGAAL	**BKP1**								**BKP1**
	DBGADH	0	0	0	0	0	0	0	0	0x00
	DBGADL	0	0	0	0	0	0	0	0	0x00
	DBGADHM	0	0	0	0	0	0	0	0	0x00
	DBGADLM	0	0	0	0	0	0	0	0	0x00
Comparator B (COMRV = 1)	DBGBCTL	SZE	SZ	**TAG**	BRK	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	0	0	0	**1**	**1**	
	DBGBAH		0	0	0	0	0	0	0	0x00
	DBGBAM DBGBAL	**BKP2**								**BKP2**
Comparator C (COMRV = 2)	DBGCCTL		NDB	**TAG**	BRK	RW	RWE	**SRC**	**COMPE**	**0x23**
			0	**1**	0	0	0	**1**	**1**	
	DBGCAH		0	0	0	0	0	0	0	0x00
	DBGCAM DBGCAL	**BKP3**								**BKP3**
	DBGCDH	0	0	0	0	0	0	0	0	0x00
	DBGCDL	0	0	0	0	0	0	0	0	0x00
	DBGCDHM	0	0	0	0	0	0	0	0	0x00
	DBGCDLM	0	0	0	0	0	0	0	0	0x00
Comparator D (COMRV = 3)	DBGDCTL	SZE	SZ	**TAG**	BRK	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	0	0	0	**1**	**1**	
	DBGDAH		0	0	0	0	0	0	0	0x00
	DBGDAM DBGDAL	**BKP4**								BKP4

Figure 12. S12XDBG Register Setup (continued)

3.3 Breakpoint if a Specific Data Byte Is Written

In the next example, two breakpoints are set. Each one triggers when a certain data byte is written (Figure 13). A byte in the memory map can be written by accessing the byte directly or by performing a word access to the preceeding address location. Therefore, comparators A and C are needed to check the two byte addresses (A_{byte1} and A_{byte2}) and comparators B and D are required to check for word accesses to the preceeding addresses (A_{byte1}–1 and A_{byte2}–1). The forced trigger of each comparator is mapped to the associated match output (match0 to 3). The state sequencer again is configured to transition to final state when any of the four match events occurs.

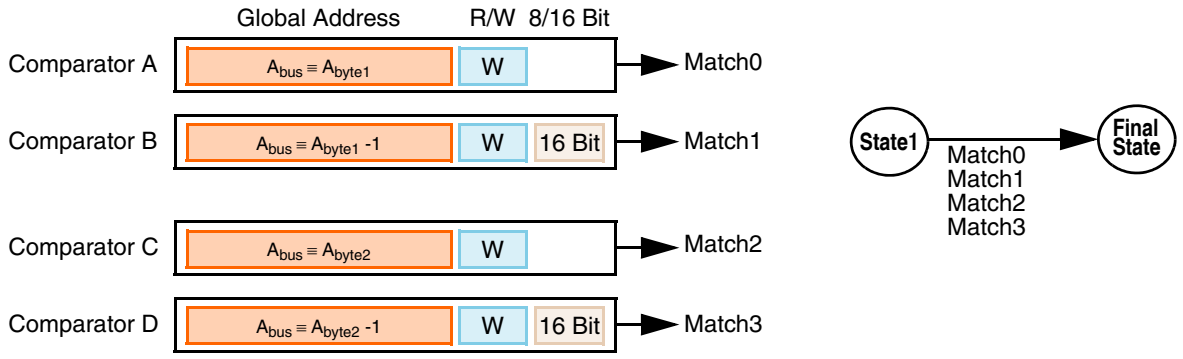

Figure 13. S12XDBG Configuration

The generated breakpoint may not stop the XGATE at the instruction that performed the write access. The XGATE may stop up to two instructions later.

Figure 14 shows the register setup for this configuration. Relevant register bits are highlighted.

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
	DBGC1	ARM	TRIG	XGSBPE	BDM	DBGBRK		COMRV		0x04
		0	0	0	0	01		—		
	DBGTCR	TSOURCE		TRANGE		TRCMOD		TALIGN		0x00
		00		00		00		00		
	DBGC2					**CDCM**		**ABCM**		**0x00**
						00		**00**		
	DBGSCR1					**SC3**	**SC2**	**SC1**	**SC0**	**0x02**
						0	**0**	**1**	**0**	
	DBGSCR2					SC3	SC2	SC1	SC0	0x00
						0	0	0	0	
	DBGSCR3					SC3	SC2	SC1	SC0	0x00
						0	0	0	0	
Comparator A (COMRV = 0)	DBGACTL		**NDB**	**TAG**	BRK	**RW**	**RWE**	**SRC**	**COMPE**	**0x07**
			0	**0**	0	**0**	**1**	**1**	**1**	
	DBGAAH		0	0	0	0	0	0	0	0x00
	DBGAAM DBGAAL	**A_{byte1}**								**A_{byte1}**
	DBGADH	0	0	0	0	0	0	0	0	0x00
	DBGADL	0	0	0	0	0	0	0	0	0x00
	DBGADHM	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0x00**
	DBGADLM	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0x00**

Figure 14. S12XDBG Register Configuration

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
Comparator B (COMRV = 1)	DBGBCTL	SZE	SZ	TAG	BRK	RW	RWE	SRC	COMPE	0x87
		1	0	0	0	0	1	1	1	
	DBGBAH		0	0	0	0	0	0	0	0x00
	DBGBAM DBGBAL	A_{byte1}-1								A_{byte1}-1
Comparator C (COMRV = 2)	DBGCCTL		NDB	TAG	BRK	RW	RWE	SRC	COMPE	0x07
			0	0	0	0	1	1	1	
	DBGCAH		0	0	0	0	0	0	0	0x00
	DBGCAM DBGCAL	A_{byte2}								A_{byte2}
	DBGCDH	0	0	0	0	0	0	0	0	0x00
	DBGCDL	0	0	0	0	0	0	0	0	0x00
	DBGCDHM	0	0	0	0	0	0	0	0	0x00
	DBGCDLM	0	0	0	0	0	0	0	0	0x00
Comparator D (COMRV = 3)	DBGDCTL	SZE	SZ	TAG	BRK	RW	RWE	SRC	COMPE	0x87
		1	0	0	0	0	1	1	1	
	DBGDAH		0	0	0	0	0	0	0	0x00
	DBGDAM DBGDAL	A_{byte2}-1								A_{byte2}-1

Figure 14. S12XDBG Register Configuration (continued)

3.4 Breakpoint if a Thread Sequence Is Executed

In the next example, a breakpoint must be generated immediately after three XGATE threads (X, Y, and Z) are executed in a sequence (Figure 15). For this purpose, comparators B, C, and D are set to the RTS instructions of the three threads. Comparator C monitors thread X, comparator D checks for the execution of thread Y, and comparator B is associated with thread Z (Figure 16). The tagged comparator outputs trigger the corresponding match events. The state sequencer is configured to expect a sequence of match2, match3, and match1 to enter final state.

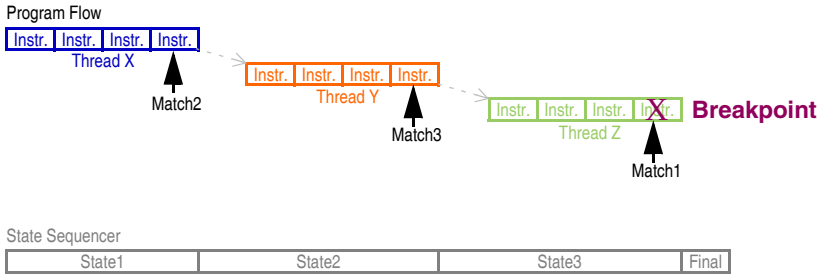

Figure 15. Breakpoint Condition

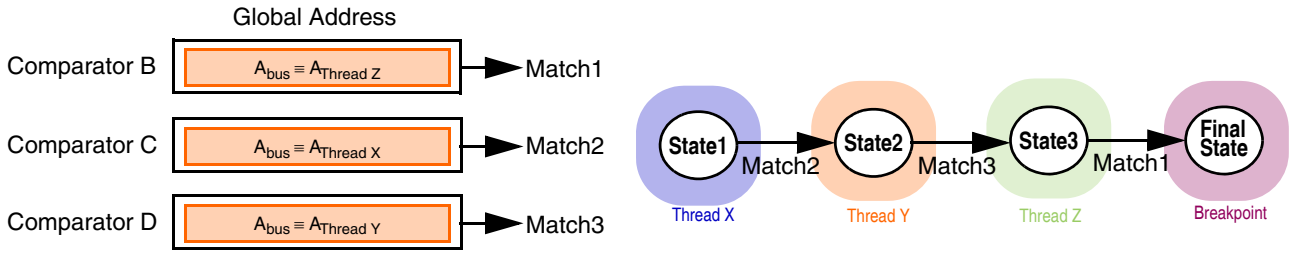

Figure 16. S12XDBG Configuration

Figure 17 shows the register setup for this configuration. Relevant register bits are highlighted.

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
	DBGC1	ARM	TRIG	XGSBPE	BDM	DBGBRK		COMRV		0x04
		0	0	0	0	01		—		
	DBGTCR	TSOURCE		TRANGE		TRCMOD		TALIGN		0x00
		00		00		00		00		
	DBGC2					**CDCM**		**ABCM**		**0x00**
						00		**00**		
	DBGSCR1					**SC3**	**SC2**	**SC1**	**SC0**	**0x03**
						0	**0**	**1**	**1**	
	DBGSCR2					**SC3**	**SC2**	**SC1**	**SC0**	**0x04**
						0	**1**	**0**	**0**	
	DBGSCR3					**SC3**	**SC2**	**SC1**	**SC0**	**0x08**
						1	**0**	**0**	**0**	
Comparator A (COMRV = 0)	DBGACTL		NDB	TAG	BRK	RW	RWE	SRC	**COMPE**	**0x00**
			0	0	0	0	0	0	**0**	
	DBGAAH		0	0	0	0	0	0	0	0x00
	DBGAAM	0	0	0	0	0	0	0	0	0x00
	DBGAAL	0	0	0	0	0	0	0	0	0x00
	DBGADH	0	0	0	0	0	0	0	0	0x00
	DBGADL	0	0	0	0	0	0	0	0	0x00
	DBGADHM	0	0	0	0	0	0	0	0	0x00
	DBGADLM	0	0	0	0	0	0	0	0	0x00
Comparator B (COMRV = 1)	DBGBCTL	SZE	SZ	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	**0**	0	0	**1**	**1**	
	DBGBAH		0	0	0	0	0	0	0	0x00
	DBGBAM DBGBAL	$A_{Thread\ Z}$								$A_{Thread\ Z}$

Figure 17. S12XDBG Register Setup

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
Comparator C (COMRV = 2)	DBGCCTL		NDB	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
			0	**1**	**0**	0	0	**1**	**1**	
	DBGCAH		0	0	0	0	0	0	0	0x00
	DBGCAM DBGCAL	$A_{Thread\ X}$								$A_{Thread\ X}$
	DBGCDH	0	0	0	0	0	0	0	0	0x00
	DBGCDL	0	0	0	0	0	0	0	0	0x00
	DBGCDHM	0	0	0	0	0	0	0	0	0x00
	DBGCDLM	0	0	0	0	0	0	0	0	0x00
Comparator D (COMRV = 3)	DBGDCTL	SZE	SZ	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	**0**	0	0	**1**	**1**	
	DBGDAH		0	0	0	0	0	0	0	0x00
	DBGDAM DBGDAL	$A_{Thread\ Y}$								$A_{Thread\ Y}$

Figure 17. S12XDBG Register Setup (continued)

3.5 Breakpoint If Threads Are Executed Out of Order

In the next example, two XGATE threads (thread X and thread Y) are expected to execute in an alternating order (Figure 18). A breakpoint is generated as soon as the order of execution is violated. For this setup comparators A and D are configured to both look at the beginning of thread X. Comparator B checks for the execution of thread Y. The tagged comparator outputs are mapped to the corresponding match events. The state sequencer is configured to reach final state whenever two consecutive match events of the same kind (match0/3 or match1) occur.

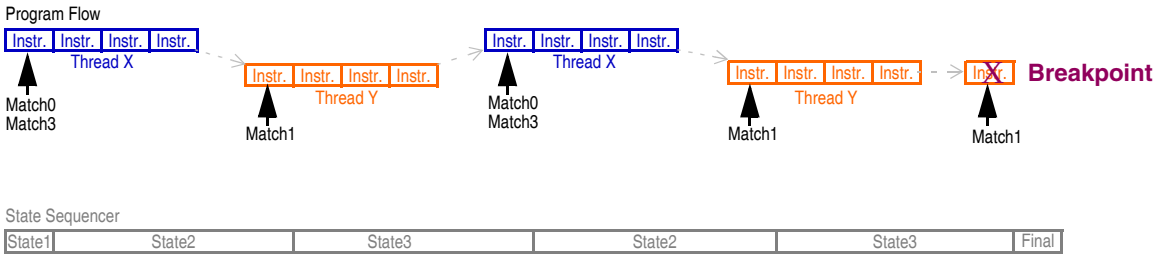

Figure 18. Breakpoint Condition

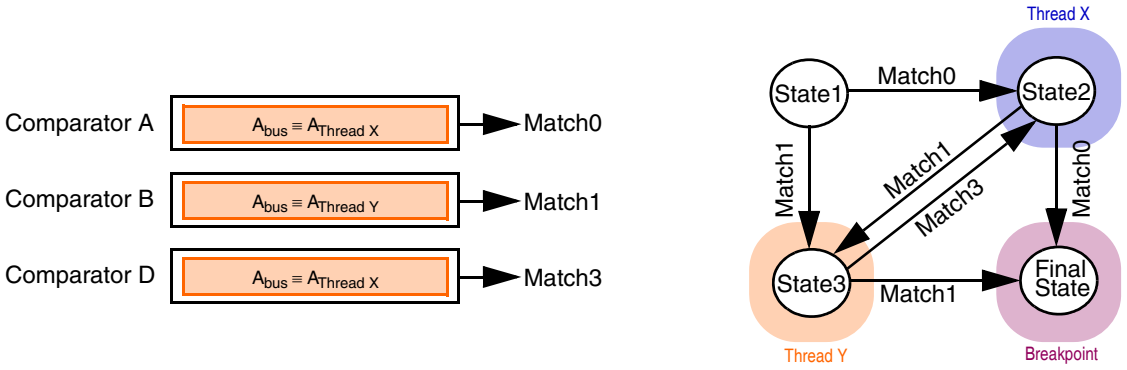

Figure 19. S12XDBG Configuration

Figure 20 shows the register setup for this configuration. Relevant register bits are highlighted.

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
	DBGC1	ARM	TRIG	XGSBPE	BDM	DBGBRK		COMRV		0x04
		0	0	0	0	01		—		
	DBGTCR	TSOURCE		TRANGE		TRCMOD		TALIGN		0x00
		00		00		00		00		
	DBGC2					**CDCM**		**ABCM**		**0x00**
						00		**00**		
	DBGSCR1					**SC3**	**SC2**	**SC1**	**SC0**	**0x06**
						0	**1**	**1**	**0**	
	DBGSCR2					**SC3**	**SC2**	**SC1**	**SC0**	**0x07**
						0	**1**	**1**	**1**	
	DBGSCR3					**SC3**	**SC2**	**SC1**	**SC0**	**0x0B**
						1	**0**	**1**	**1**	
Comparator A (COMRV = 0)	DBGACTL		NDB	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
			0	**1**	**0**	0	0	**1**	**1**	
	DBGAAH		0	0	0	0	0	0	0	0x00
	DBGAAM DBGAAL	**$A_{Thread\ X}$**								**$A_{Thread\ X}$**
	DBGADH	0	0	0	0	0	0	0	0	0x00
	DBGADL	0	0	0	0	0	0	0	0	0x00
	DBGADHM	0	0	0	0	0	0	0	0	0x00
	DBGADLM	0	0	0	0	0	0	0	0	0x00
Comparator B (COMRV = 1)	DBGBCTL	SZE	SZ	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	**0**	0	0	**1**	**1**	
	DBGBAH		0	0	0	0	0	0	0	0x00
	DBGBAM DBGBAL	**$A_{Thread\ Y}$**								**$A_{Thread\ Y}$**

Figure 20. S12XDBG Register Setup

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
Comparator C (COMRV = 2)	DBGCCTL		NDB	TAG	BRK	RW	RWE	SRC	**COMPE**	**0x00**
			0	0	0	0	0	0	**0**	
	DBGCAH		0	0	0	0	0	0	0	0x00
	DBGCAM	0	0	0	0	0	0	0	0	0x00
	DBGCAL	0	0	0	0	0	0	0	0	0x00
	DBGCDH	0	0	0	0	0	0	0	0	0x00
	DBGCDL	0	0	0	0	0	0	0	0	0x00
	DBGCDHM	0	0	0	0	0	0	0	0	0x00
	DBGCDLM	0	0	0	0	0	0	0	0	0x00
Comparator D (COMRV = 3)	DBGDCTL	SZE	SZ	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	**0**	0	0	**1**	**1**	
	DBGDAH		0	0	0	0	0	0	0	0x00
	DBGDAM	$A_{Thread\ X}$								$A_{Thread\ X}$
	DBGDAL									

Figure 20. S12XDBG Register Setup (continued)

3.6 Breakpoint on Violations of Mutual Exclusive Code

In this scenario, two concurrent threads run on the XGATE and the CPU12X (Figure 21). Both threads share a system resource. Each has a critical code sequence in which it expects to have exclusive access to this system resource. To debug a concurrency problem, a breakpoint must be generated as soon as both cores execute their critical code sequence simultaneously.

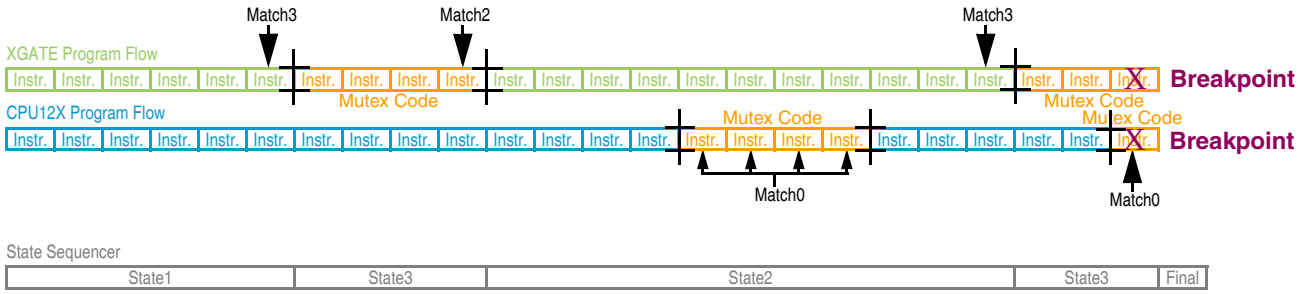

Figure 21. Breakpoint Condition

To setup this type of breakpoint, comparators A and B are configured to perform a range check on the critical code sequence of the CPU12X (Figure 22). A match0 event occurs for every instruction that the CPU12X executes within this range. The entry and exit of the XGATE's critical code sequence are detected by comparator D (points to the instruction before the sequence) and C (points to the end of the sequence). The state sequencer tracks the state of the XGATE. Every time the XGATE enters its critical code sequence, the FSM transitions to state3. Every time the critical sequence is left, the FSM leaves state3 as well. While the state sequencer remains in state3, it needs to pay attention to match0 events. Because this is the indicator that both cores execute their critical code simultaneously, a transition to final state must occur in this case.

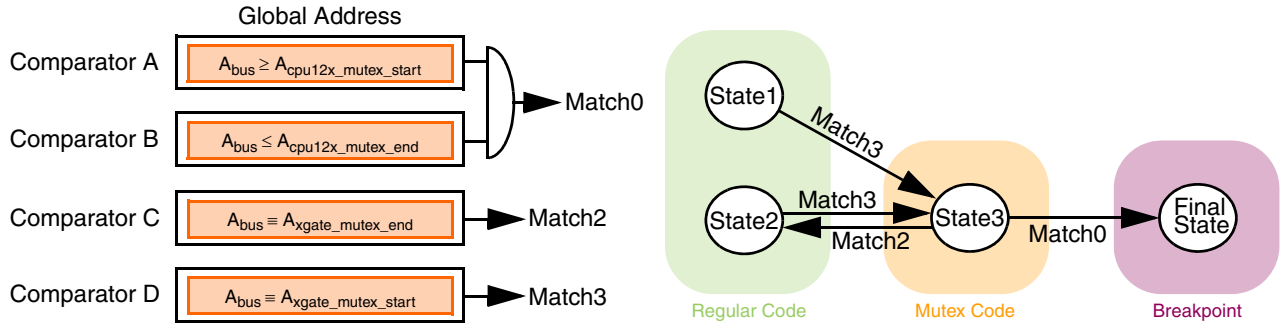

Figure 22. S12XDBG Configuration

Figure 23 shows the register setup for this configuration. Relevant register bits are highlighted.

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
	DBGC1	ARM	TRIG	XGSBPE	BDM	DBGBRK		COMRV		0x0C
		0	0	0	0	11		—		
	DBGTCR	TSOURCE		TRANGE		TRCMOD		TALIGN		0x00
		00		00		00		00		
	DBGC2					**CDCM**		**ABCM**		**0x01**
						00		**01**		
	DBGSCR1					**SC3**	**SC2**	**SC1**	**SC0**	**0x0B**
						1	**0**	**1**	**1**	
	DBGSCR2					**SC3**	**SC2**	**SC1**	**SC0**	**0x04**
						0	**1**	**0**	**0**	
	DBGSCR3					**SC3**	**SC2**	**SC1**	**SC0**	**0x09**
						1	**0**	**0**	**1**	
Comparator A (COMRV = 0)	DBGACTL		NDB	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x21**
			0	**1**	**0**	0	0	**0**	**1**	
	DBGAAH DBGAAM DBGAAL		**Acpu12x_mutex_start**							**Acpu12x_mutex_start**
	DBGADH	0	0	0	0	0	0	0	0	0x00
	DBGADL	0	0	0	0	0	0	0	0	0x00
	DBGADHM	0	0	0	0	0	0	0	0	0x00
	DBGADLM	0	0	0	0	0	0	0	0	0x00
Comparator B (COMRV = 1)	DBGBCTL	SZE	SZ	TAG	BRK	RW	RWE	SRC	**COMPE**	**0x01**
		0	0	0	0	0	0	0	**1**	
	DBGBAH DBGBAM DBGBAL		**Acpu12x_mutex_end**							**Acpu12x_mutex_end**

Figure 23. S12XDBG Register Setup

	Name	Bit 7	6	5	4	3	2	1	Bit 0	
Comparator C (COMRV = 2)	DBGCCTL		NDB	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
			0	**1**	**0**	0	0	**1**	**1**	
	DBGCAH		0	0	0	0	0	0	0	0x00
	DBGCAM DBGCAL		$A_{xgate\_mutex\_end}$							$A_{xgate\_mutex\_end}$
	DBGCDH	0	0	0	0	0	0	0	0	0x00
	DBGCDL	0	0	0	0	0	0	0	0	0x00
	DBGCDHM	0	0	0	0	0	0	0	0	0x00
	DBGCDLM	0	0	0	0	0	0	0	0	0x00
Comparator D (COMRV = 3)	DBGDCTL	SZE	SZ	**TAG**	**BRK**	RW	RWE	**SRC**	**COMPE**	**0x23**
		0	0	**1**	**0**	0	0	**1**	**1**	
	DBGDAH		0	0	0	0	0	0	0	0x00
	DBGDAM DBGDAL		$A_{xgate\_mutex\_start}$-2							$A_{xgate\_mutex\_start}$-2

Figure 23. S12XDBG Register Setup (continued)

4 References

1. MC9S12XEP100 Data sheet, Freescale Semiconductor Inc., 2005.
2. MC9S12XDP512 Data sheet, Freescale Semiconductor Inc., 2005.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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

AN3458
Rev. 0, 05/2007

freescale™ semiconductor