SuperH

SuperH™ (SH) 64-bit RISC Series

SH-5 System Architecture, Volume 3: Debug

Last updated 19 March 2002

Issued by the SuperH Documentation Group on behalf of SuperH, Inc.

SuperH, Inc.
San Jose, U.S.A. - Bristol, United Kingdom - Tokyo, Japan

www.superh.com

SuperH

Contents

Preface

This document is part of the SuperH SH-5 CPU system documentation suite detailed below. Comments on this or other books in the documentation suite should be made by contacting your local sales office or distributor.

SuperH SH-5 document identification and control

Each book in the documentation suite carries a unique identifier in the form:

05-SA-nnnnn Vx.x

Where, *n* is the document number and *x.x* is the revision.

Whenever making comments on a SuperH SH-5 document the complete identification 05-SA-1000n Vx.x should be quoted.

SuperH SH-5 system architecture documentation suite

The SuperH SH-5 system architecture documentation suite comprises the following volumes:

- SH-5 System Architecture, Volume 1: System (05-SA-10001)
- SH-5 System Architecture, Volume 2: Peripherals (05-SA-10002)
- SH-5 System Architecture, Volume 3: Debug (05-SA-10003)

SuperH

1 Debug/trace architecture

1.1 Overview of debug features

The SH-5 debug system provides both traditional CPU debug, and advanced CPU and system debug features.

A number of different features are available, the number of instances is scalable on a per-implementation basis. A brief overview of these debug features is given below.

In the following description of debug features, the term 'tool' refers to any form of software development system, typically consisting of a computer plus a debug adaptor or emulator.

1.1.1 Communication with a tool

SH-5 provides two interfaces through which it can communicate with a software development tool; a dedicated high-speed interface (called the SHdebug link) and a JTAG interface. A tool can use only one of these interfaces at a time.

The SHdebug link is the preferred debug interface as communications between a tool and SH-5 is much faster using the SHdebug link rather than JTAG. However, some future SH-5 based ASICs may be pin-limited and not have enough pins available for a SHdebug link interface. The JTAG interface allows a tool to communicate with SH-5 and have access to all the on-chip debug features described in this document but at a substantially reduced performance compared with that offered by the SHdebug link. In addition to being used for system debug, the JTAG interface can also be used for its normal functions of boundary scan and internal scan.

Either debug interface provides a logical connection between a tool and the SH-5 SuperHyway bus. This logical connection gives the tool full access to the physical address map, and all the nodes connected to the bus.

A 16 Mbyte portion of the address map is mapped to memory physically located within the tool. Accesses to this area result in read or write messages over the selected debug interface. These can then be handled by the tool to provide “remote-memory” systems. By using this feature in conjunction with facilities to stop, start and run the CPU from a specified address, ROM-less target systems are possible during product development phases.

In addition to these download and control operations, the selected debug interface may also be used to either spill or read-out trace information.

1.1.2 Trigger pins

SH-5 provides a trigger in (DM_TRIG_N) and a trigger out pin (DM_TROUT_N). These allow external analysis hardware (such as a logic analyzer) to be connected.

The trigger out pin can also be configured to provide external visibility of timing events (such as interrupt latency), and to detect internal states (such as FIFO overflow).

1.1.3 Watchpoint detection

The CPU includes facilities to watchpoint on several events which occur in normal code execution:

- Instruction address - for breakpoints in ROM, or ranged breakpoints.
- Operand address - to detect range-based memory writes.
- Instruction value - to perform flexible profiling and register watchpoints.
- PC branch - to perform branch tracing, call graph profiling and sample based profiling.

The watchpoints can be triggered in complex manners using generic pre-conditions to combine them in sequence, and also to combine them with event counters. The pre-conditions also allow them be made process (ASID) and instruction mode (SHmedia vs. SHcompact) specific.

1.1.4 Watchpoint actions

Watchpoints can perform a number of actions:

- Raise a CPU debug exception (to invoke the debug exception handler).
- Capture all parameters associated with the debug event, and generate a trace message. The trace message can optionally include timestamp information, and can also include data values.
- Set or clear chain latches, which allow debug events to be chained together in complex sequences.
- Decrement event counters, which allow events to be disabled until they have occurred a specified number of times.
- Increment performance counters.
- Reset all performance counters.
- Control the state of the trigger-out pin.

1.1.5 Fast printf

A memory-mapped register is available, which when written to results in a specified message being sent to the tool. These messages can be read by the tool and used to implement arbitrary communication functions, such as:

- Dump of specific trace/data or timing information.

  Can be used to provide minimally intrusive code instrumentation facilities.
- Virtual I/O - for target/tool communications (such as file/tty access to the tool).

  These facilities are used to implement “software backplanes” (scalable host/target debug systems).

1.1.6 Bus analyzer

A bus analyzer is provided on SH-5’s SuperHyway bus.

This provides SuperHyway request or SuperHyway response packet watchpoint facilities, and can be used to generate trace information, and provide performance information. The bus analyzer can be combined with CPU watchpoints in order to provide sophisticated conditions for filtering debug events.

1.1.7 Performance counters

A number of CPU based events can be setup to increment a number of performance counters. These can be used to count arbitrary debug events from the CPU and the bus analyzers.

CPU performance monitor channels are also available which allow a number of distinct CPU states (cache hits/misses, interrupts taken) to be observed.

1.2 Key concepts

This section defines some key concepts and mechanisms associated with the debug system.

The following “shorthand” terms are used throughout this document:

- WPC - watchpoint controller.
- DM - debug module.
- BA - bus analyzer.
- WP - watchpoint. Used as in “WP channel”.

Note: Some WP channels are implemented in the WPC, others are implemented in the debug module or the bus analyzers.

1.2.1 SHdebug link

The SHdebug link is one of the two debug interfaces which can be selected for target/tool communications. It provides a full-duplex interface, with a 1-bit wide input path, and a 4-bit wide output path. It is implemented as part of the debug module (*Section 1.9: Debug protocols and interfaces on page 118*). The design of this module allows the width of the output data path to be increased to meet the debugging bandwidth needs of different applications.

The SHdebug link provides:

- Full access to the physical address map (RAM, ROM, on-chip devices, external-devices). This allows access to the debug registers (see *Section 1.2.4: Debug registers on page 16*).
- SH-5-originated access to a 16 Mbyte address space mapped over the SHdebug link.

  Allows a target debug agent (or any other code) to execute on the CPU without requiring any external RAM or ROM, and thus enables use of SH-5 without a traditional monitor ROM.

  The debug/development tool must not access this 16 Mbyte region via the SHdebug link (this would require the SH-5 reflect the request back to the tool). The SH-5 behavior is undefined if this is attempted. The tool is expected to service such memory accesses locally, without involving the SH-5.
- Control of the CPU via memory-mapped register.

  Allows the CPU to be suspended, resumed, forced to execute from a specified address, forced to generate a fast printf message, or forced to take a debug interrupt.
- Streaming operations for CPU and bus trace information.

  Allows trace information gathered from the CPU and the on-chip busses to be copied to a specified area in the physical memory map (such as RAM or the SHdebug link). This area acts as an external FIFO. The trace information can also be sent directly to the SHdebug link using a special mode which compresses the trace message contents. This gives better throughput on the link.

The SHdebug link is suitable for connection to a debug adaptor board as part of a development tool (to provide code download and debug facilities). It can also be connected to specialized hardware debug systems (such as logic analyzers) to provide more complex facilities.

1.2.2 JTAG debug interface

The JTAG port of SH-5 is the other interface which can be selected for target/tool communications. The JTAG debug interface provides the same communication functions between target and tool as described in *Section 1.2.1* above, except that it uses the standard JTAG access method and has a much lower bandwidth.

The other key difference is that JTAG does not support target-initiated communications. In order for the tool to recognize that SH-5 has an unsolicited message pending, the tool must poll the target at regular intervals.

1.2.3 Watchpoint controller (WPC)

The WPC is part of the CPU. It provides “CPU-centric” debugging operations. It is based on two main features:

- Instruction architecture debug support. The provision of a BRK instruction, single stepping, instruction address/operand address/instruction value watchpoints, branch detection facilities, and a dedicated exception vector.
- The ability to cause a context switch from application being debugged to debugger externally from the CPU (via the tool or via another CPU). This can be achieved without the co-operation of the application being debugged or its operating system.

1.2.4 Debug registers

Locality of registers

The debug mechanisms are implemented in the watchpoint controller, the bus analyzer channels and the debug module.

In order to provide the necessary control information, the registers associated with the debug system are located in one of these modules. This locality does not affect the high-level semantics of the operations, but it is naturally exposed as part of the register’s address. Therefore a naming convention is used to denote this:

DM.* debug module (including SuperHyway bus analyzer registers). All DM. register addresses are offsets from DM_BASE_ADDR.

WPC.* watchpoint controller. All WPC register addresses are offsets from WPC_BASE_ADDR.

Access to registers

The debug registers implemented both inside and outside of the WPC (that is, outside of the CPU) are memory mapped. They are accessed using the physical address map and thus can be accessed:

- Via the CPU instruction stream. Following any store, a SYNCO instruction followed by a SYNCI instruction can be used to ensure that the updated WPC and DM states take effect before the instruction after the SYNCI.
- Externally using a debug tool connected to either of the selectable debug interfaces (that is, without involving the CPU).

  Accesses to the WPC and DM registers should only be performed with **Load8/ Store8** transactions with a mask value of 0xFF. All other accesses are undefined. The instructions listed in *Table 1* may be used to correctly access these registers from the instruction stream.

Mode	Direction	Instruction	Notes
SHmedia	**store8**	**ST.Q**	
		STHI.Q	Effective address = 8N+7
		STLO.Q	Effective address = 8N
		FST.D	
		FST.P	
	load8	**LD.Q**	
		LDHI.Q	Effective address = 8N+7
		LDLO.Q	Effective address = 8N
		FLD.D	
		FLD.P	

Table 1: Instructions for accessing WPC and DM registers

Mode	Direction	Instruction	Notes
SHcompact	**store8**	**FMOV DRm, @Rn**	
		FMOV DRm, @-Rn	
		FMOV DRm, @(R0,Rn)	
		FMOV XDm, @Rn	
		FMOV XDm, @-Rn	
		FMOV XDm, @(R0,Rn)	
	load8	**FMOV @Rm, DRn**	
		FMOV @Rm+, DRn	
		FMOV @(R0,Rm), DRn	
		FMOV @Rm, XDn	
		FMOV @Rm+, XDn	
		FMOV @(R0,Rm), XDn	

Table 1: Instructions for accessing WPC and DM registers

Access to undefined areas of the WPC/DM address map

Accesses to memory addresses within the WPC/DM address map which do not correspond to architected registers are undefined.

Whilst being undefined, these accesses have the following properties:

- They will not lock the SuperHyway (that is, SuperHyway success or error responses will be generated).
- Reads will return undefined data.
- Writes will potentially affect other architected registers (that is, the WPC/DM architected registers do not necessarily have their addresses fully decoded).

1.2.5 Debug module

The debug module manages:

- The SHdebug link and the debug interface to the JTAG TAP controller.

  Provides a connection to the SuperHyway bus, and also a route to extract trace information.

- The DM_TRIN_N and the DM_TROUT_N pins.
- An on-chip FIFO (known as the DM FIFO).

  This acts as a temporary buffer for trace messages. Trace messages from the DM FIFO can be dealt with as follows:

  - Sent to the selected debug interface, JTAG or SHdebug link. If trace messages are generated faster than they can be transferred to the selected debug interface, either the CPU can be stalled or new trace messages can be discarded.
  - Accumulated in the DM FIFO until it fills. Once the DM FIFO fills, either the CPU can be stalled or new trace messages are discarded. In this mode, memory-mapped registers allow the DM FIFO contents to be read.
  - Old messages in the DM FIFO overwritten by new ones so that the DM FIFO contains the most recent trace messages generated. In this mode, memory-mapped registers allow the DM FIFO contents to be read.
  - Written to an area of the target system's RAM (known as a trace buffer).

- Trace buffer

  One of the available destinations for trace messages is to write these into an area of target system memory allocated as a trace buffer. Debug module register fields set the size and the base address of this trace buffer area. The size can set between 64 Kbytes and 64 Mbytes.

  The trace buffer can operate in two different ways:

  - As circular buffer, with old entries being overwritten by new entries once the buffer fills. The buffer always contains the most recent trace messages.
  - As a fixed length buffer which does not wrap around. Once the buffer fills, trace messages are discarded which means that the buffer contains the earliest trace messages.

1.2.6 Bus analyzers

As part of SH-5's on-chip debug capability, the SuperHyway bus arbiter contains bus analyzer channels to provide:

- SuperHyway request packet or response packet watchpoints.
  These allow either requests or responses to be monitored, and a normal watchpoint action (see *Section 1.1.4: Watchpoint actions on page 13*) to be generated.
- A bus capture buffer for capturing complete bus transactions whenever a bus watchpoint hit occurs. These captured SuperHyway packets are sent to the debug module and used to create trace messages which are written to the debug module FIFO. These trace messages can be sent either to the tool or written to a FIFO area in target system memory.
- Capture selected performance parameters of the on-chip bus to allow system software to "tune" the parameters of individual application-specific modules or bus arbiters.

1.2.7 Debug monitor

Whenever a watchpoint matches, a debug monitor can optionally be invoked. The debug monitor consists of a debug exception handler whose code and data can exist in any location in the physical memory map. For example:

- Partly in an area of the target system flash memory and partly in target system RAM memory.
- Totally in an area of the target system RAM memory or flash ROM memory.
- In the debug adapter portion of a tool where the debug adapter contains its own processor and local memory.
- In the development host portion of a tool in which the debug adapter is simply a signal converter.

In these last two alternatives, the debug exception vector is setup to force the CPU's MMU and cache to be disabled, and the vector points to an address in the debug module's address space. Instruction fetches (SuperHyway **Load8/16/32** requests) are passed to the tool, either via the SHdebug link or via JTAG depending on which interface is configured as the debug interface.

Similarly, data accesses within the debug module's address space result in SuperHyway requests (for example, **Load8/16/32**, **Store8/16/32**, **Swap8**) being passed to the tool through the SHdebug link or JTAG port.

1.2.8 Chain latches

SH-5 provides a series of chain-latches. Chain-latches allow watchpoint hits, in the WPC or bus analyzers, to enable or disable any other watchpoints.

Chain latches consist of an item of state, which is either set or clear. No assumptions are made about the type of sequential device which will be used in the implementation. *Figure 1* shows a functional block diagram of the chain-latches.

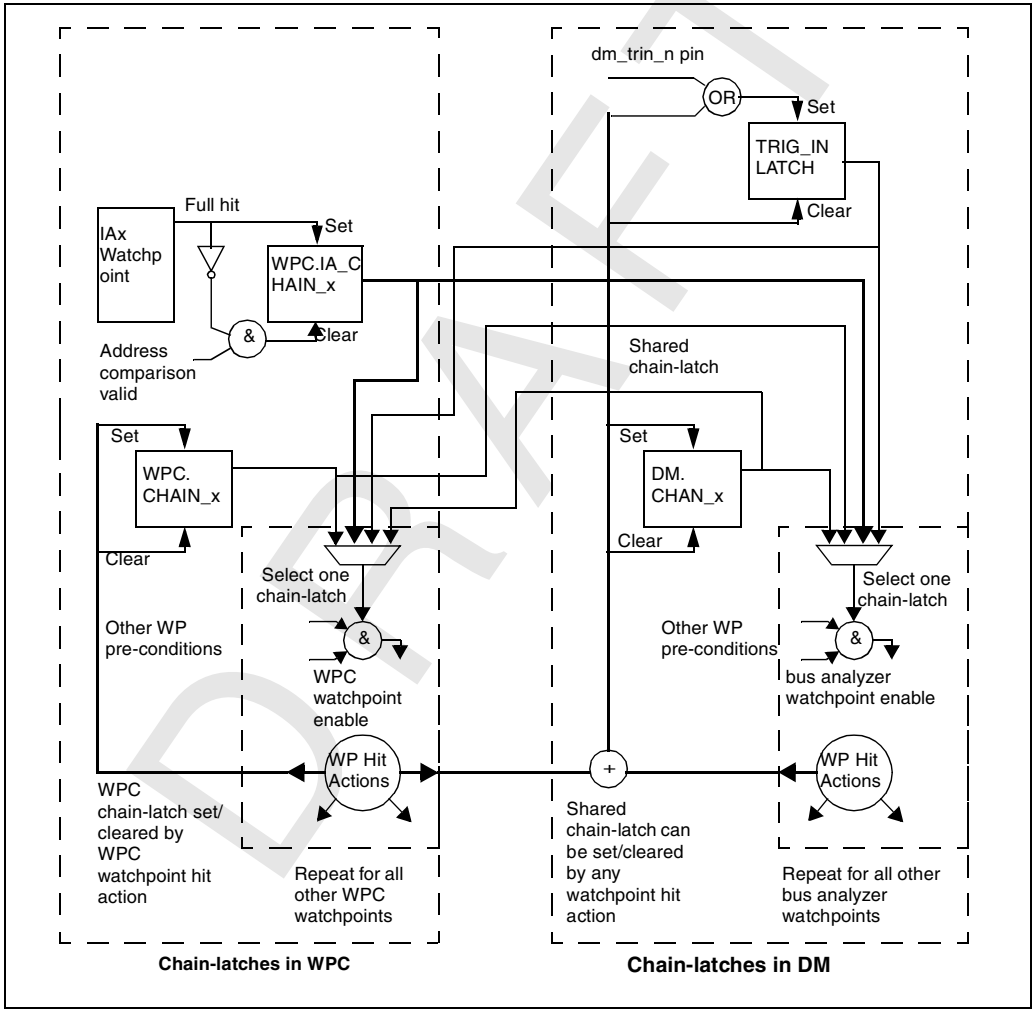

Figure 1: Chain-latch concept

All of the watchpoints (including the branch trace and the fast printf function) can use a chain-latch to enable or disable the function.

An implementation may provide a maximum of 16 chain-latches, thus a 4-bit field is used for the chain-latch ID. *Section 4.1.4: Chain latches on page 243* defines the chain-latch IDs.

Note *The trigger-in signal shown in Figure 1 has some of the characteristics of a chain latch and is described in Trigger-in chain-latch on page 24.*

Chain-latch capabilities

There are three groups of chain-latches with slightly different capabilities.

1 Each of the IA watchpoints has an associated chain-latch. There is no ACTION register field for controlling these chain-latches. Instead, the state of each chain-latch is determined solely by whether a full hit occurred for the corresponding IA channel on the immediately preceding instruction. The chain-latch outputs are available as pre-conditions for all WPC and debug module/bus analyzer watchpoints.

2 Generic chain-latches in the WPC which can be set or cleared by any WPC watchpoint hits. The chain-latch outputs are available as pre-conditions for all WPC and bus analyzer watchpoints.

3 Generic chain-latches in the DM which can be set or cleared by WPC or bus analyzer watchpoint hits. The chain-latch outputs are available as pre-conditions for all WPC and bus analyzer watchpoints.

Table 2 summarizes the control and use of all chain-latches.

Latch name	Pre-condition for WPC watchpoint	WPC watchpoint can alter	Pre-condition for DM and bus analyzer watchpoint	Bus analyzer watchpoint can alter
WPC.IAX_CHAIN	OA, IV, WPC_PERF	no	FPF, BR, PL	no
WPC.CHAIN_X	IA, OA, IV, WPC_PERF	yes (IA, OA, IV only)	FPF, BR, PL	no

Table 2: Chain-latch use

Latch name	Pre-condition for WPC watchpoint	WPC watchpoint can alter	Pre-condition for DM and bus analyzer watchpoint	Bus analyzer watchpoint can alter
DM.CHAIN_X	IA, OA, IV, WPC_PERF	yes (IA, OA, IV only)	FPF, BR, PL	yes

Table 2: Chain-latch use

Chain state

Each latch has 1 bit of state. This denotes if the latch is set (1) or clear (0). The state is affected by the WP channels.

If the state of a chain latch is being changed by conflicting conditions, for example, being set by a watchpoint hit and being cleared by a different watchpoint hit on the same clock cycle, then the resulting state of the chain-latch is undefined.

Chain-latches can be included in the pre-trigger and action-condition operations of each of the watchpoint channels.

The pre-conditions of each WP channel allow the WP match to succeed only if the specified chain latch is set.

Table 3 shows how the state of the generic chain-latches in the WPC and the DM can be changed. *Table 4* gives similar information for the IA watchpoints in the WPC. Refer to *Section 1.6: WP channel matching on page 66* for a definition of *Full-Hit*.

Chain alter (see *Table 18 on page 49*)	FULL_WP_HIT (see *Section 1.6: on page 66*)	New state
0bxx	0	Unchanged
0b00 or 0b01	1	Unchanged
0b10	1	Clear
0b11	1	Set

Table 3: Generic chain-latch action

Instruction comparison valid	Full-hit	New state
1	1	Set
0	x	Unchanged
1	0	Clear

Table 4: IA chain-latch action

Trigger-in chain-latch

The DM_TRIN_N pin is manifested as a chain-latch (see *Section 1.8.7: Trigger-in chain-latch on page 94*). This chain-latch has some special properties which allow its state to be directly affected by the level on the DM_TRIN_N pin, or to be edge triggered.

The resynchronization circuitry used to manifest the pin state as a chain-latch state results in an implementation-defined delay (see *Chain-latch latency on page 244*) delay between the pin changing state and the chain-latch's value altering.

The normal chain-latch operations are available on the WP channels to clear or set the trigger chain-latch's state as required.

{WPC/DM}_CHAIN_*x* control register description

These registers allow debug software to read the state of the chain latches and to directly set or clear these latches.

{WPC/DM}.CHAIN_*x*				where *x* = chain ID OR = TRIG_IN	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CHAIN_ STATE	0	1	✓	Chain-latch state	RW
	Operation		Contains a chain-latch's value. The chain-latch can be set or cleared by watchpoint being hit, according to the programming of the watchpoint's ACTION registers. Software can read the state of this latch at any time, and can also directly set or clear the latch.		
	When read		Returns 0 when the latch is clear and 1 when the latch is set.		
	When written		Sets or clears the chain-latch. Value - Description 0: Clear the chain-latch 1: Ignored		
	HARD reset		Undefined		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 5: {WPC/DM}_CHAIN_*x* definition

WPC.IA_CHAIN_*x*				where x = chain ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CHAIN_ STATE	0	1	✓	Chain-latch state	RO
	Operation		Contains an IA watchpoint chain-latch's value. The chain-latch is set whenever an IA watchpoint full-hit occurs. It is cleared whenever an IA watchpoint full-hit does not occur. The new value is visible to precondition checks from the next instruction (inclusive) onwards. Software can read the state of this latch at any time.		
	When read		Returns 0 when the latch is clear and 1 when the latch is set.		
	When written		Ignored		
	HARD reset		Undefined		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 6: WPC.IA_CHAIN_*x* definition

1.2.9 Event counters

SH-5 provides a series of event counters, these are used in conjunction with WP channels to provide count-based matching of debug events.

An implementation may provide a maximum of 16 event counters, thus a 4-bit field is used for the event counter ID. The counter has a maximum size of 64 bits.

Any particular implementation may provide fewer than 16 counters, and those provided may have fewer than 64 bits. See *Section 4.1.2: Event counters on page 242* for implementation-specific details.

A number of event counters are available, some are implemented in the WPC and are accessible only by CPU core watchpoint channels, whilst others are implemented in the DM and are accessible only by bus analyzer watchpoint channels.

If multiple WP channels are setup to affect the same event counter, only a single event counter decrement will be performed for each simultaneous channel match (rather than the alternative of performing one decrement per simultaneous WP channel match).

Register description

{WPC/DM}.ECOUNT_VALUE_*x*				where x = event counter ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[63:0]	64	✓	Counter value	RW
	Operation		Contains the counter's value. The implementation defines the significant size of the counter (known as ECOUNT.SIZE, see *Section 4.1.2: Event counters on page 242*). Bits [0:(ECOUNT.SIZE-1)] count down when a WP channel is set to decrement this counter. When the counter value reaches zero, no further decrementing occurs. Even if a watchpoint PRE register has an event counter enabled, debug software can disable the counter by setting the value to zero. Bits [ECOUNT.SIZE, 63] are undefined.	
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 7: {WPC/DM}.ECOUNT_VALUE_*x* register definition

1.2.10 Performance counters

SH-5 provides a series of performance counters, these are used in conjunction with the WP channels and WP facilities to provide observation of internal CPU and bus events. Some of the counters are physically located within the WPC and others are physically located within the debug module.

- Performance counters within the WPC may be incremented either when a WPC watchpoint hit occurs or when a WPC_PERF channel match occurs. Refer to *Section 1.18: WP channel type WPC_PERF on page 165*.
- Performance counters within the DM may be incremented when a PL watchpoint hit occurs.

An implementation may provide a maximum of 16 performance counters, thus a 4 bit field is used for the performance counter ID. The counter has a maximum size of 64 bits.

Any particular implementation may provide fewer than 16 counters, and those provided may have fewer than 64 bits. See *Section 4.1.3: Performance counters on page 243* for implementation-specific details.

The counters may be written to at any time. They are modulo-N counters, and thus will wrap around.

Register description

{WPC/DM}.PCOUNT_VALUE_*x*				where *x* = performance counter ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[63:0]	64	✓	Counter value	RW
	Operation		Contains the counter's value. The implementation defines the significant size of the counter (known as PCOUNT.SIZE, see *Section 4.1.3: Performance counters on page 243*). Bits [0, (PCOUNT.SIZE - 1)] count up when a WP channel is set to increment this counter. Bits [PCOUNT.SIZE, 63] are undefined.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 8: {WPC/DM}.PCOUNT_VALUE_*x* register definition

1.3 CPU control

The CPU has a memory-mapped register (WPC.CPU_CTRL_ACTION) which can be used to control it (both from the instruction stream, and from the tool directly).

The DM has a memory-mapped register (DM.FORCE_DEBUGINT) which can be used to force a debug interrupt on the CPU.

1.3.1 Suspending/resuming the CPU

The CPU can be made to cease fetching and issuing instructions and enter the suspended state by writing CPU_CTRL_OP_SUSPEND to WPC.CPU_CTRL_ACTION.

The suspended state may be exited by writing CPU_CTRL_OP_RESUME to WPC.CPU_CTRL_ACTION.

Entering the suspended state causes a CPU to drain its execution pipelines. This takes an implementation defined period of time. When a CPU is suspended its execution context may be changed in any of the following ways:

- The selection of either RESVEC or DBRVEC vectoring through DBRMODE, and the DBRVEC value may be changed (see *Section 1.7: Reset, panic and debug events on page 73*);
- The CPU may be manually reset;
- The state of peripherals may be examined or safely changed.
- The state of memory may be examined or safely changed.

These operations can be performed when the CPU is running, but by suspending it beforehand it is possible to determine that no CPU state is changing apart from that being affected by operations being applied from the tool.

When the CPU is suspended, the stall_state bit of the DM.TRCTL register is set (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*).

When suspending the CPU from the instruction stream[1], the store instruction which writes to WPC.CPU_CTRL_ACTION should be followed by a sequence of NOP instructions, sufficient to allow the CPU pipeline to drain before the suspend takes effect.

1. In a single CPU system this is not generally a useful thing to do.

1.3.2 Control operations

The control operation is defined by a 2-bit value:

Operation name	value	Explanation
CPU_CTRL_OP_SUSPEND	0b00	Suspends execution of the receiving CPU. See *Section 1.3.1: Suspending/resuming the CPU on page 30*.
CPU_CTRL_OP_RESUME	0b01	Resumes execution from suspended state of the receiving CPU
CPU_CTRL_OP_CPURESET	0b10	Generate a CPURESET event on the receiving CPU. See *Section 1.7: Reset, panic and debug events on page 73*.
CPU_CTRL_OP_DEBUGRESET	0b11	Generate a DEBUGRESET event for the whole device. See *Section 1.7: Reset, panic and debug events on page 73*.

Table 9: CPU control operation values

WPC.CPU_CTRL_ACTION register definition

When written to, this register performs a CPU control operation:

WPC.CPU_CTRL_ACTION				0x104000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OPCODE	[1:0]	2	—	Control operation code	RW
	Operation		A CPU control operation as defined in *Table 9 on page 31*.		
	When read		Returns current value		
	When written		Performs the operation defined in *Table 9 on page 31*.		
	HARD reset		Undefined		

Table 10: WPC.CPU_CTRL_ACTION register definition

WPC.CPU_CTRL_ACTION				0x104000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:2]	62	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 10: WPC.CPU_CTRL_ACTION register definition

If the value CPU_CTRL_OP_DEBUGRESET is written to WPC.CPU_CTRL_ACTION twice or more in succession, the second and subsequent writes will be ignored. To perform a second DEBUGRESET event, the WPC.CPU_CTRL_ACTION register must have a different value (such as CPU_CTRL_OP_RESUME) written to it before writing CPU_CTRL_OP_DEBUGRESET for the second time. In particular, it is assumed that the bootstrap code entered after a DEBUGRESET event will write CPU_CTRL_OP_RESUME to WPC.CPU_CTRL_ACTION as one of its steps.

In contrast, each write of the value CPU_CTRL_OP_CPURESET to the WPC.CPU_CTRL_ACTION register will cause a CPURESET event, regardless of the previous value written to the register.

1.3.3 Changing DBRMODE and/or DBRVEC whilst the CPU is suspended

If the values of DBRMODE and DBRVEC are modified whilst the CPU is suspended, the SH-5 will ignore the new values unless a CPURESET occurs before the CPU resumes.

The correct sequence of actions for setting a new PC to be used when the CPU resumes is:

- write CPU_CTRL_OP_SUSPEND to WPC.CPU_CTRL_ACTION (or bring the SH-5 up in a suspended state by a hardware reset).
- set DBRMODE to 1 and DBRVEC to the required address
- write CPU_CTRL_OP_CPURESET to WPC.CPU_CTRL_ACTION
- write CPU_CTRL_OP_RESUME to WPC.CPU_CTRL_ACTION

If the CPURESET action is not used, the effect will be to ignore the new settings of DBRMODE and DBRVEC. In particular:

- If the SH-5 was brought up in a suspended state by a hardware reset (see the discussion of DM_ISYNC and SUSPEND in *Section 3.4.2: Reset functions available from debug tools on page 219*), the PC after the CPU resumes will be 0x0 (the power-on reset value of RESVEC).
- If the SH-5 was suspended by writing CPU_CTRL_OP_SUSPEND to WPC.CPU_CTRL_ACTION, the PC will not be modified by the suspend and resume operation.

1.3.4 Debug interrupt

A non-maskable, but blockable debug interrupt is available (see *Section 1.7.3: Event specific information on page 80*). DEBUGINT has a priority level of 16, thus it can be taken regardless of the value of SR.IMASK.

Occurrence of DEBUGINT forces execution of the event handler (see *Section 1.7: Reset, panic and debug events on page 73*). If the CPU was in sleep mode, a wake-up transition will occur prior to the DEBUGINT launch.

This interrupt is forced using the DM.FORCE_DEBUGINT register (see*Table 11 on page 34*). Full details of the interrupt mechanism are given in *Section : DEBUGINT - debug interrupts on page 81*. (That section also describes how to clear DEBUGINT conditions.)

DM.FORCE_DEBUGINT register definition

DM.FORCE_DEBUGINT				0x100088	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FORCE	0	1	✓	DEBUGINT force	RW
	Operation		Forces a DEBUGINT event on the CPU.		
	When read		Returns an undefined value.		
	When written		Writing ‘1’ sets the FORCED_DEBUG_INTERRUPT bit of DM.EXP_CAUSE and will force a DEBUGINT event (see *Section : on page 81*). Subsequent writes of ‘1’ will have no effect until the DEBUGINT has been cleared. Writing ‘0’ has no effect.		
	HARD reset		0		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 11: DM.FORCE_DEBUGINT register definition

1.4 Watchpoint channels

SH-5 supports a number of WP channels, these provide differing features, but have a common overall structure.

The WP channels are implemented in different parts of the SH-5, some are in the WPC controller itself (that is, within the SH-5 CPU), others are within the debug module, whilst others are in the bus analyzer.

1.4.1 WP channel type

The SH-5 debug system supports up to 16 distinct channel types. Each WP channel has a single fixed, unchangeable type.

Channel name	WP channel type (4 Bits)	Type of watchpoint hit	Explanation
BRK	0b0000	Breakpoint or single step	Execution of an embedded BRK instruction, a single step, or a forced debug interrupt. This is not a true WP channel - it has no precondition or action registers. See *Section 1.10: WP channel type BRK on page 136*.
IA	0b0001	Instruction address watchpoint	CPU is about to execute an instruction from a PC address within a IA watchpoint range.
OA	0b0010	Operand address watchpoint	CPU is about to execute an instruction which will write to memory within a memory range covered by an OA watchpoint.
IV	0b0011	Instruction value watchpoint	CPU is about to execute an instruction which has a bit pattern matching an IV watchpoint.
BR	0b0100	Non-sequential PC branch	CPU has branched to a non-sequential PC value (either conditional branch, unconditional branch, subroutine call or exception/interrupt).

Table 12: WP Channel Types

Channel name	WP channel type (4 Bits)	Type of watchpoint hit	Explanation
FPF	0b0101	Fast printf	Fast printf forced (see *Section 1.1.5: Fast printf on page 13*, and *Section 1.15*)
PL	0b0110	SuperHyway bus analyzer watchpoint	SuperHyway bus analyzer watchpoint has occurred. See *Section 1.16: WP channel type PL on page 164*.
DM	0b1000	FIFO activity	Debug module's FIFO activity (as selected by *Section 1.8.10: on page 97*).
WPC_PERF	0b1001	Performance events	Performance information updated (see *Section 1.18*).
*	All other values	N/A	

Table 12: WP Channel Types

1.4.2 WP Channel - generic register structure

Each WP channel is controlled by a set of registers. Most WP channels follow a generic register form (some channels have implicit features and so have a reduced set of registers). All the WP channel registers appear in the physical memory map.

The generic form of WP channels consists of:

- PRE condition registers, which must all match in order to trigger the WP channel.
- Optional channel-specific MATCH condition registers, which must match in order to trigger the WP channel.
- ACTION registers which define the action to perform when the WP channel triggers.

The following WP channels do not follow the generic form:

- The BR channel has only a single register, DM.WP_BR_FILTER, defining all of its PRE conditions, MATCH conditions and ACTIONS.
- The FPF and PERF channels have implicit actions, and so does not have ACTION registers.
- The BRK and DM channels have implicit preconditions and actions, and so do not have these registers.

The WP channel match sequence described in *Section 1.6: WP channel matching on page 66*.

WP-channel	Register name	Abbreviation
IA, OA, IV, WPC_PERF	WPC.WP_NX_PRE DM.WP_NX_PRE	Defines a set of pre conditions to apply when performing channel matching. Described in *Section 1.6: WP channel matching on page 66*.
PL, FPF	DM.WP_NX_PRE	
IA, OA, IV	WPC.WP_NX_ACTION DM.WP_NX_ACTION	Defines a set of actions to apply when the debug event matches. Described in *Section 1.5: Debug event actions on page 48*.
DM, PL	DM.WP_NX_ACTION	
IA, OA, IV, WPC_PERF, PL, DM	WPC.WP_NX_MATCH DM.WP_NX_MATCH	Defines a set of match criteria which are specific to the WP Channel's type (that is, IA watchpoints contain an address range, IV watchpoints contain an instruction value and instruction mask). Described in *Section 1.9: Debug protocols and interfaces on page 118* *Section : Implicit action: on page 138* *Section 1.12: WP channel type OA on page 143* *Section 1.13: WP channel type IV on page 152* *Section 1.15: WP channel type FPF on page 163* *Section 1.16: WP channel type PL on page 164* *Section 1.17: WP channel type DM on page 164* *Section 1.18: WP channel type WPC_PERF on page 165*
	n = name of the channel (for example IA for IA channel) *x* = a 4 bit value to specify the channel ID (relative to *n*), for example x= 2 for IA2.	

Table 13: WP channel generic registers

Each WPC channel has two PRE registers, one which is implemented in the WPC and the second which is implemented in the debug module.

WPC.WP_n*x*_PRE

Each channel has a PRE register implemented in the WPC.

WPC.WP_*nx*_PRE				where *n* = *{IA/OA/IV/WPC_PERF}*, *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BASIC_ENABLE	0	1	—	Enable	RW
	Operation		Enables or disables the WP channel. Value - Description 0: basic match disabled 1: basic match enabled		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		0		
ASID_ENABLE	1	1	—	ASID match enable	RW
	Operation		Enables or disables the inclusion of the current ASID value in the debug event match. Value - Description 0: ASID match disabled 1: ASID match enabled. Will only trigger when the current ASID matches the ASID_VALUE field.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 14: WPC.WP_*{IA/OA/IV/WPC_PERF}x*_PRE register definition

WPC.WP_*nx*_PRE				where *n* = *{IA/OA/IV/WPC_PERF}*, *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CHAIN_ENABLE	2	1	—	Chain-latch enable	RW
	Operation		Enables or disables the inclusion of a specified chain-latch in the debug event match. Value - Description 0: Chain-latch match disabled 1：Chain-latch match enabled. Will only trigger when the chain-latch specified by CHAIN_ID is set.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CHAIN_ID	[6:3]	4	—	Chain-latch ID	RW
	Operation		Defines the chain-latch used in the debug event match. Chain latches located in the WPC and in the DM are all available for selection. See *Section 4.1.4: Chain latches on page 243*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 14: WPC.WP_*{IA/OA/IV/WPC_PERF}x*_PRE register definition

WPC.WP_*nx*_PRE				where *n = {IA/OA/IV/WPC_PERF}*, *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ECOUNT_ENABLE	7	1	—	Event counter enable	RW
	Operation		Enables or disables the inclusion of a specified event counter in the debug event match. Value - Description 0: Event count match disabled 1：Event count match enabled. Will only trigger when the event counted defined by ECOUNT_ID contains 0.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ECOUNT_ID	[11:8]	4	—	Event counter ID	RW
	Operation		Defines the event counter used in the debug event match. See *Section 4.1.2: Event counters on page 242*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ASID_VALUE	[19:12]	8	—	ASID match value	RW
	Operation		Defines the ASID value in the debug event match.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 14: WPC.WP_*{IA/OA/IV/WPC_PERF}x*_PRE register definition

WPC.WP_*nx*_PRE				where *n = {IA/OA/IV/WPC_PERF}*, *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ISAMODE_ ENABLE	[21:20]	2	—	CPU ISA mode selection	RW
	Operation		Allows the CPU ISA mode to be included in the debug event match. For the IV WP channels, this field is still present, but its value is ignored in the pre-condition checking. IV channels never match in SHcompact mode. Value - Description 0b00, 0b11: match irrespective of the current CPU ISA mode. 0b01: only match if CPU is executing SHmedia instructions. 0b10: only match if CPU is executing SHcompact instructions.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
SR_MD_ ENABLE	[23:22]	2	—	CPU user/privileged mode selection	RW
	Operation		Allows the CPU user/privileged mode to be included in the debug event match. Value - Description 0b00, 0b11: match irrespective of the current CPU user/ privileged mode. 0b01: only match if CPU is in user mode. 0b10 : only match if CPU is in privileged mode.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 14: WPC.WP_*{IA/OA/IV/WPC_PERF}x*_PRE register definition

WPC.WP_*nx*_PRE				where *n* = *{IA/OA/IV/WPC_PERF}*, *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:24]	40	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 14: WPC.WP_*{IA/OA/IV/WPC_PERF}x*_PRE register definition

DM.WP_{*IA/OA/IV*}*x*_PRE:

Each channel has a PRE register implemented in the DM.

DM.WP_*nx*_PRE				where *n*= *{IA/OA/IV}*, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	0	1	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 15: DM.WP_{*IA/OA/IV*}*x*_PRE register definition

DM.WP_*nx*_PRE				where *n= {IA/OA/IV}*, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ASID_ENABLE	1	1	—	ASID match enable	RW
	Operation		The WPC.WP_NX_ PRE register determines the inclusion of the current ASID value in the debug event match. The field defined here is not involved in the debug event match, it determines whether the ASID value is placed into trace messages. Value - Description 0: include the ASID value (at the point of the trigger) in the trace message. 1: do not include ASID value in trace message.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:2]	62	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 15: DM.WP_{*IA/OA/IV*}*x*_PRE register definition

DM.WP_PL*x*_PRE:

Each bus analyzer WP channel has one PRE register implemented in the debug module:

DM.WP_PL*x*_PRE				where *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BASIC_ENABLE	0	1	—	Enable	RW
	See BASIC_ENABLE field of *Table 14 on page 38*.				
—	1	1	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
CHAIN_ENABLE	2	1	—	Chain-latch enable	RW
	Operation		See the CHAIN_ENABLE field of *Table 14 on page 38*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CHAIN_ID	[6:3]	4	—	Chain-latch ID	RW
	Operation		See the CHAIN_ID field of *Table 15 on page 42*		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 16: DM.WP_PLx_PRE register definition

DM.WP_PL*x*_PRE				where *x* = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ECOUNT_ENABLE	7	1	—	Event counter enable	RW
	Operation		See the ECOUNT_ENABLE field of *Table 14 on page 38*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ECOUNT_ID	[11:8]	4	—	Event counter ID	RW
	Operation		See the ECOUNT_ID field of *Table 14 on page 38*		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:12]	52	—	Reserved	RES
	Operation		Reserved		
	When read		Return 0		
	When written		Ignored		
	HARD reset		0		

Table 16: DM.WP_PLx_PRE register definition

DM.WP_FPF_PRE:

The fast printf function has an associated PRE register.

DM.WP_FPF_PRE				0x800280	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BASIC_ENABLE	0	1	—	Enable	RW
	Operation		See BASIC_ENABLE field of *Table 14 on page 38*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ASID_ENABLE	1	1	—	ASID match enable	RW
	Operation		Enables or disables the inclusion of the current ASID value in the debug event match. Irrespective of this setting, the ASID value is always included in the FPF message. Value - Description 0: ASID match disabled. 1: ASID match enabled. Will only trigger when the current ASID matches the ASID_VALUE field.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CHAIN_ENABLE	2	1	—	Chain-latch enable	RW
	Operation		See the CHAIN_ENABLE field of *Table 14 on page 38*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 17: DM.WP_FPF_PRE register definition

DM.WP_FPF_PRE				0x800280	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CHAIN_ID	[6:3]	4	—	Chain-latch ID	RW
	Operation		See the CHAIN_ID field of *Table 15 on page 42*		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[11:7]	5	—	Reserved	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ASID_VALUE	[19:12]	8	—	ASID match value	RW
	Operation		See the ASID_VALUE field of *Table 14 on page 38*		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[20,63]	44	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 17: DM.WP_FPF_PRE register definition

1.5 Debug event actions

Multiple actions can be raised when a debug event is detected:

- Raise a CPU debug exception event (to involve the event handler).

  The event handler vector is separate from the normal exception vectors. This allows a target-debug agent to be loosely integrated, or even totally decoupled from the target software being debugged.

  The event handler mechanism is described in *Section 1.7: Reset, panic and debug events on page 73*.

  For all WPC watchpoint channels, a CPU debug exception causes all other potential watchpoint hit actions for this channel (except decrement of WPC event counters) to be ignored. This ensures that a consistent set of state is made available to the exception handler at launch[1]. It also causes suppression of actions from other watchpoint channels that hit on the same instruction; see *Section 1.6.4: Behavior when more than one WPC channel matches an instruction on page 70*.
- Capture all parameters associated with the debug event, and generate a trace message of a defined flavor.
- Set or clear chain latches, which allow debug events to be connected together.
- Decrement event counters.
- Increment performance counters.
- Reset all performance counters.
- Control the state of a trigger-out pin (which is used to interface to external debug equipment).
- Perform an action specific to the WP channel type.

Some WP channels support a subset of the above actions, these subsets are described in the following channels.

Each WPC channel has two ACTION registers to define its event actions, one (WPC.WP_{IA/OA/IV}X_ACTION) which is implemented in the WPC and the second

1. Different actions are potentially carried out on different clock cycles to that on which a debug exception is raised. Thus enabling the multiple actions with exception would present inconsistent state to the exception handler at launch time.

(DM.WP_{IA/OA/IV}X_ACTION) which is implemented in the debug module. This is because WPC channel actions can affect architectural state in both the WPC and in the DM. Actions are always specified in a control register that is in the module where the affected state is. In contrast, bus analyzer watchpoint channels can only affect architectural state in the debug module, so they each have just a single ACTION register (DM.WP_PLX_ACTION) which is in the debug module.

WPC.WP_{*IA/OA/IV*}x_ACTION:

WPC.WP_*nx*_ACTION				**where *n* = *{IA/OA/IV}* *x* = channel ID (relative to *N*)**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ EXCEPTION	0	1	—	Exception enable	RW
	Operation		Enables or disables a debug exception for the WPC watchpoint channel. Value - Description 0: debug exception disabled 1: debug exception enabled If debug exception is enabled, all the other action fields specified for this channel in its WPC and DM action registers (except ACTION_ECOUNT) are ignored when the watchpoint hit occurs.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 18: WPC.WP_*{IA/OA/IV}x*_ACTION register definition

WPC.WP_*nx*_ACTION				where *n* = *{IA/OA/IV}* *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ECOUNT	1	1	—	Event count decrement enable	RW
	Operation		Enables or disables decrement of an event counter for the WP channel. The event counter is specified by the ECOUNT_ID field of this register. *Section 1.6: WP channel matching on page 66* defines the terms used below. Value - Description 0: event count decrement disabled 1 : if PARTIAL_WP_HIT, decrement enabled. No other action will occur unless the specified event counter contains 0.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ECOUNT_ID	[5:2]	4	—	Event counter ID	RW
	Operation		Defines the event counter used in the debug event match. See *Section 4.1.2: Event counters on page 242*. Only those event counters located in the WPC can be used. If the value in this field refers to a DM event counter, no counter decrement action occurs.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 18: WPC.WP_*{IA/OA/IV}x*_ACTION register definition

WPC.WP_*nx*_ACTION				where *n* = *{IA/OA/IV}* *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_CHAIN_ALTER	[7:6]	2	—	Enable chain-latch alteration	RW
	Operation		Specifies if and how a chain latch is modified according to the match state. The chain latch is specified by the CHAIN_ID field. *Section 1.6: WP channel matching on page 66* defines the terms used below. Value - Description 0b00: do not alter the chain latch. 0b01: do not alter the chain latch. 0b10: if EXTRA_HIT, clear the chain latch. 0b11: if EXTRA_HIT, set the chain latch.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CHAIN_ID	[11:8]	4	—	Chain-latch ID	RW
	Operation		Defines the chain-latch used in conjunction with ACTION_CHAIN_ALTER. Only certain chain-latches can be controlled by each watchpoint. See *Section 4.1.4: Chain latches on page 243*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 18: WPC.WP_*{IA/OA/IV}x*_ACTION register definition

WPC.WP_*nx*_ACTION				where *n* = *{IA/OA/IV}* *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[12,17]	6	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ACTION_PCOUNT	18	1	—	Performance counter increment enable	RW
	Operation		Enables or disables increment of a performance counter (specified by PCOUNT_ID) for the WP channel. Value - Description 0: performance count increment disabled 1 : performance count increment enabled		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ACTION_RESET_ALL_PCOUNT	19	1	—	Reset all performance counters	RW
	Operation		Allows all the WPC performance counters to be reset when the WP channel triggers. The performance counters in the DM are not affected. Value - Description 0: do not reset 1: reset all performance counters		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 18: WPC.WP_*{IA/OA/IV}x*_ACTION register definition

WPC.WP_*nx*_ACTION				where *n* = *{IA/OA/IV}* *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PCOUNT_ID	[23:20]	4	—	Performance counter ID	RW
	Operation		Defines the WPC performance counter used in the counter increment. See *Section 1.2.10: Performance counters on page 28*. Only those performance counters located in the WPC can be used. If the value in this field refers to a DM performance counter undefined effects will occur.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:24]	40	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 18: WPC.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*{IA/OA/IV}x*_ACTION:

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ INTERRUPT	0	1	—	OA data match interrupt	RW
	Operation		This field applies only to OA watchpoints. When DM.WP_OAX_ACTION.OA_MATCH == 1, it determines whether an interrupt is generated when an OA watchpoint hit is detected and a successful OA data match occurs. If DM.WP_OAX_ACTION.OA_MATCH == 0, the data match is not considered and a debug interrupt will be generated regardless of the data value. Normally, the WPC.WP_OAX_ACTION.ACTION_EXCEPTION field would be more useful in this case, because it will raise a pre-execution exception on the instruction that hit the watchpoint. A debug interrupt is always asynchronous and could be delivered long after the instruction has completed. Undefined effects may occur if this bit is set to '1' and the action_exception bit of the corresponding WPC.WP_X_ACTION register is also set to '1'. Value - Description 0: debug interrupt disabled. 1: debug interrupt enabled.	
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[5:1]	5	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ACTION_CHAIN_ALTER	[7:6]	2	—	Enable chain-latch alteration	RW
	See the ACTION_CHAIN_ALTER field of *Table 18 on page 49*				
CHAIN_ID	[11:8]	4	—	Chain-latch ID	RW
	This field should be set to the same value as corresponding CHAIN_ID field of *Table 18 on page 49*				

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_TRACE	[13:12]	2	—	Trace enable	RW
	Operation		Enables or disables generation of a trace message for the WP channel (see *Section 1.8.5: IA watchpoint trace modes on page 92*). The trace information generated varies according to the WP channel. *Section 1.6: WP channel matching on page 66* defines the terms used below. Trace for the OA WP channels can be further controlled using the DM.WP_OAX_ACTION:OA_MATCH field and the DM.OA_MATCH_* registers (see *Section 1.12.3: Data match registers on page 147*). Trace is generated as described in *Section 1.8.5: on page 92*, according to the mode specified: Value - Description 0b00: trace generation disabled 0b01: trace generation disabled 0b10: trace generation enabled - single trace mode. This only applies for IA watchpoints, undefined effects occur if this is programmed for non-IA channels. 0b11: trace generation enabled - multi trace mode		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TRACE_TYPE	14	1	—	Trace message type	RW
	Operation		Specifies the type of the trace message generated (see *Section : Trace message types on page 119*).: Value - Description 0: trigger trace message 1: background trace message		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ENABLE_TRACE_TIMESTAMP	15	1	—	Enable trace timestamp	RW
	Operation		Enable timestamp in trace message: Value - Description 0: no timestamp 1: trace message includes timestamp value		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_TRIG_OUT	16	1	—	Trigger out enable	RW
	Operation		Enables or disables generation of a trigger out message for the WP channel. The effect of this on the DM_TROUT_N pin is determined by the programming of the DM.TRCTL register. See *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*. Value - Description 0: The DM_TROUT_N pin is unaffected by hits on this WP channel. 1: If DM.TRCTL.DM_TRIG_OUT_MODE == 0b001, then an active-low pulse is produced on the DM_TROUT_N pin each time a hit occurs on this WP channel (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*). If DM.TRCTL.DM_TRIG_OUT_MODE has any other value, hits on this WP channel have no effect on the DM_TROUT_N pin.	
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
OA_MATCH	17	1	—	OA data match enable	RW
	Operation		This field applies only to OA watchpoints. It determines whether DM chain latch, trace generation and interrupt actions are dependent on a successful OA data match. Value - Description 0: data match function is not enabled. 1: data match function is enabled.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_*nx*_ACTION				where *n*= {*IA/OA/IV*}, *x* = channel ID (relative to *N*)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:18]	46	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 19: DM.WP_*{IA/OA/IV}x*_ACTION register definition

DM.WP_PL*x*_ACTION

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ INTERRUPT	0	1	—	Interrupt enable	RW
	Operation		Enables or disables a debug interrupt for a bus analyzer WP channel. Value - Description 0: debug interrupt disabled 1: debug interrupt enabled		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 20: DM.WP_PL*x*_ACTION register definition

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ ECOUNT	1	1	—	Event count decrement enable	RW
	Operation		Enables or disables decrement of an event counter for a bus analyzer WP channel. The event counter is specified by the ECOUNT_ID field of this register. *Section 1.6: WP channel matching on page 66* defines the terms used below. Value - Description 0: event count decrement disabled 1: if PARTIAL_WP_HIT, decrement enabled. No other action will occur unless the specified event counter contains 0.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ECOUNT_ID	[5:2]	4	—	Event counter ID	RW
	Operation		Defines the event counter used in the debug event match. See *Section 4.1.2: Event counters on page 242*. Only those event counters located in the DM can be used.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ACTION_CHAIN_ ALTER	[7:6]	2	—	Enable chain-latch alteration	RW
	See the action_chain_alter field of *Table 18 on page 49*				

Table 20: DM.WP_PL*x*_ACTION register definition

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CHAIN_ID	[11:8]	4	—	Chain-latch ID	RW
	Operation		Defines the chain-latch used in conjunction with ACTION_CHAIN_ALTER. Only certain chain-latches can be controlled by each watchpoint. See *Section 4.1.4: Chain latches on page 243*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ACTION_TRACE	[13:12]	2	—	Trace enable	RW
	See the ACTION_TRACE field of *Table 19 on page 54*				
TRACE_TYPE	14	1	—	Trace message type	RW
	See the TRACE_TYPE field of *Table 19 on page 54*				
ENABLE_TRACE_TIMESTAMP	15	1	—	Enable trace timestamp	RW
	See the ENABLE_TRACE_TIMESTAMP field of *Table 19 on page 54*				
ACTION_TRIG_OUT	16	1	—	Trigger out enable	RW
	Operation		Enables or disables generation of a trigger out message for the WP channel. The effect of this on the DM_TROUT_N pin is determined by the programming of the DM.TRCTL register. See *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*. Value - Description 0: The DM_TROUT_N pin is unaffected by hits on this WP channel. 1: If DM.TRCTL.DM_TRIG_OUT_MODE == 0b010, then a low-going pulse is produced on the DM_TROUT_N pin each time a hit occurs on this WP channel (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*). If DM.TRCTL.DM_TRIG_OUT_MODE has any other value, hits on this WP channel have no effect on the DM_TROUT_N pin.		

Table 20: DM.WP_PL*x*_ACTION register definition

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ACTION_ PCOUNT	17	1	—	Performance counter increment enable	RW
	See the action_pcount field of *Table 18 on page 49*				
ACTION_RESET_ ALL_PCOUNT	18	1	—	Reset all performance counters	RW
	Operation		Allows all the DM performance counters to be reset when the WP channel triggers. The performance counters in the WPC are not affected. Value - Description 0: do not reset 1: reset all performance counters		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ACTION_ PCOUNT_ID	[22:19]	4	—	Performance counter ID	RW
	Operation		Defines the DM performance counter used in the counter increment. See *Section 1.2.10: Performance counters on page 28*. Only those performance counters located in the DM can be used. If the value in this field refers to a WPC performance counter, no counter increment action occurs		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 20: DM.WP_PL*x*_ACTION register definition

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PL_MODULE	[30:23]	8	—	SuperHyway bus physical module number	RW
	Operation		This field only applies to SuperHyway bus analyzer watchpoints. Defines the identity of a physical SuperHyway bus master module (one of 256 possible masters) associated with the DM.WP_PLX_CTRL.SRC field for the purpose of freezing the bus master when a watchpoint hit occurs. The relationship between physical module number and SuperHyway protocol source ID is specific to the chip implementation and known to the debug programmer. The implementation specific information is held in *Table 91: DM.PLx_ACTION.pl_module/ DM.PLX_FRZ.freeze_x/SuperHyway module mapping on page 251*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 20: DM.WP_PL*x*_ACTION register definition

DM.WP_PL*x*_ACTION				where *x* = channel ID (relative to PL)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FREEZ_EN	31	1	—	Freeze enable	RW
	Operation		This field only applies to SuperHyway bus analyzer watchpoints. It has no effect on WPC watchpoints. Specifies whether the SuperHyway bus master specified by the PL_MODULE field will be inhibited from generating further SuperHyway transactions. This “freeze” function should be enabled only when a specific source is defined. Value - Description 0: No freeze action 1: Freeze is enabled. Note: For SuperHyway bus, the freeze action logic has no knowledge of whether the DM.WP_PLX_CTRL register specifies a specific source or any source. If “any source” is selected and FREEZ_EN is set, the results of the freeze action are undefined.	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 20: DM.WP_PL*x*_ACTION register definition

1.5.1 WPC.ADDR_IN_TRACE register definition

This register defines what information is passed from the WPC to the DM whenever an OA or IV watchpoint hit occurs. This information determines the contents of the DATA_ADDRESS field of OA and IV trace messages, and also determines the data seen by the OA channel's data match comparator (see *Section 1.12.3: Data match registers on page 147*).

WPC.ADDR_IN_TRACE				0x104018	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MUX_ADDR	0	1	—	Control of WPC/DM interconnect	RW
	Operation		Controls the information generated for OA/IV watchpoint hits, and thus the information placed in the DATA_ADDRESS fields of OA/IV trace messages, and the data seen by the OA data match comparator. Value - Description 0: information generated does not contain the operand address but instead contains the 64-bit data word written to memory by the instruction. 1: information generated contains both the 32-bit operand address and least significant 32-bits (bits [0, 31]) of the data written to memory by the instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 21: WPC.ADDR_IN_TRACE register definition

1.6 WP channel matching

The following subsections and diagram explain how watchpoint channel matching is achieved. Where:

PRE = precondition register (WPC or DM as appropriate),

ACTION = action register (WPC or DM as appropriate).

1.6.1 SR.WATCH bit

The SR.WATCH bit is used to enable or disable all actions related to WPC-based WP channels (IA, OA, IV, BR, WPC_PERF) in both the WPC and the DM. It has no effect on the other WP channels (BRK, FPF, PL, DM).

If SR.WATCH is '0', all the action conditions (*Section 1.5: Debug event actions on page 48*) associated with all the WPC-based WP channels defined above are disabled. Thus these channel's actions which would normally launch the event handler, affect event/performance counters, chain latches, trigger out pins or generate trace for example, will be voided.

SR.WATCH is automatically cleared to '0' by the hardware when launching a reset, panic or debug event handler. This allows all instructions executed within the event handler to be invisible to the debug system itself in order to prevent generation of unwanted trace, and to prevent the CPU from trying to launch a debug exception during the execution of the event handler itself.

This may appear to be a nontypical situation (that is, breakpoints would not be set in the debug event handler code), but unexpected exceptions might occur due to "wide" programming of IA/OA and particularly IV watchpoint channels.

Event handler software should re-enable action conditions by restoring SR.WATCH to '1' when leaving its critical region. This is achieved using the RTE instruction to write the contents of the saved status register (SSR) to the status register (SR). This will not result in the BR channel triggering for the branch caused by the RTE instruction, as the state of SR.WATCH is considered at the start of an RTE instruction's execution (see *WP channel type BR on page 156*).

1.6.2 Precondition terms

INITIAL_HIT includes the state of SR.WATCH for some WPC channels, and is always 1 for other WPC channels.

```
If channel is of type {IA, OA, IV, BR, WPC_PERF }
  INITIAL_HIT = SR.WATCH
else INITIAL_HIT = 1
```

PARTIAL_WP_HIT includes INITIAL_HIT, basic enable, optional ASID value, optional CPU mode checks, optional chain-latch, match conditions specific to the channel.

```
PARTIAL_WP_HIT =
 ( (INITIAL_HIT && pre.basic_enable) &&
   ((!pre.asid_enabled) || (pre.asid_enabled && pre.asid_value ==
<ASID>)) &&
    (CPUMODE_OK (pre.isamode_enable, pre.sr_md_enable)) &&
      ((!pre.chain_enable) ||
      (pre.chain_enable && (CHAIN_LATCH_VALUE[pre.chain_id] == 1)))
&&
         (ChannelSpecificMatches) )
```

FULL_WP_HIT includes event counter.

PARTIAL_WP_HIT and FULL_WP_HIT are not evaluated sequentially - the event counter state used when evaluating FULL_WP_HIT is the same state used when evaluating the corresponding PARTIAL_WP_HIT.

Thus a PARTIAL_WP_HIT which decrements an event counter such that it reaches '0' will not result in FULL_WP_HIT occurring for this hit. This will happen for the subsequent PARTIAL_WP_HIT.

```
FULL_WP_HIT = ( PARTIAL_WP_HIT &&
 ((!pre.ecount_enable) || (pre.ecount_enable &&
(ECOUNT_VALUE[pre.ecount_id] == 0))) )
```

1.6.3 Actions

The following actions are carried out, for each attempt at matching a given channel.

Potentially decrement the event counter, irrespective of FULL_WP_HIT.

```
if ( (PARTIAL_WP_HIT) && (action.action_ecount) ) {
  DecrementEventCounter (action.event_id);
}
```

If FULL_WP_HIT generate an exception:

```
if (FULL_WP_HIT && action.action_exception) {
  GenerateException();
}
```

Generation of trigger-out, interrupts, trace and DM chain-latch modification is more complex, as for OA channels it can involve the use of the DM's data/value mask comparator.

For OA channels, EXTRA_HIT may include a match dependant on the DM's data value/mask comparator matching the data written by the instruction which triggered the OA channel. For all other channels EXTRA_HIT is always '1'.

```
If channels is of type {OA} {
  EXTRA_HIT = (!DM.WP_OAx_ACTION.oa_match) ||
           (DM.WP_OAx_ACTION.oa_match &&
    ( (oa_data_written & DM.OA_MATCH_DATAMASK &
mask_out_invalid_bits() ==
       (DM.OA_MATCH_DATAVALUE & DM.OA_MATCH_DATAMASK &
mask_out_invalid_bits() )))
} else EXTRA_HIT == 1
```

Potentially either clear or set a WPC chain latch:

```
if ( FULL_WP_HIT && (!action.action_exception) ) {
  maybeUpdateWPCChainLatch = 1;
} else {
  maybeUpdateWPCChainLatch = 0;
}
if (maybeUpdateWPCChainLatch && action.action_chain_alter == 0b10) {
   CHAIN_LATCH[action.chain_id] = 0;
}
if (maybeUpdateWPCChainLatch && action.action_chain_alter == 0b11) {
  CHAIN_LATCH[action.chain_id] = 1;
}
```

Potentially either clear or set a DM chain latch:

```
if ( (FULL_WP_HIT && EXTRA_HIT) && (!action.action_exception) ) {
  maybeUpdateDMChainLatch = 1;
} else {
  maybeUpdateDMChainLatch = 0;
}
if (maybeUpdateDMChainLatch && action.action_chain_alter == 0b10) {
   CHAIN_LATCH[action.chain_id] = 0;
}
if (maybeUpdateDMChainLatch && action.action_chain_alter == 0b11) {
  CHAIN_LATCH[action.chain_id] = 1;
}
```

Potentially generate trace (see *Section 1.8.5: IA watchpoint trace modes on page 92* for full details):

```
if ((FULL_WP_HIT && EXTRA_HIT) && action.action_trace &&
     !action.action_exception) {
  GenerateTrace();
}
```

Potentially generate an interrupt:

```
if ((FULL_WP_HIT && EXTRA_HIT) && !action.action_exception &&
     action.action_interrupt) {
  GenerateInterrupt();
}
```

Potentially modify performance counters:

```
if (FULL_WP_HIT && !action.action_exception && action.action_pcount)
{
  IncrementPerformanceCounter(action.pcount_id);
}
if (FULL_WP_HIT && !action.action_exception &&
action.reset_all_pcount) {
  ResetAllPerformanceCounters();
}
```

Potentially signal the trigger out pin:

```
if ((FULL_WP_HIT && EXTRA_HIT) &&
     !action.action_exception && action.action_trig_out) {
  PulseTriggerOutPin();
}
```

Note: The ACTION_TRACE, ACTION_TRIG_OUT *and* ACTION_CHAIN_ALTER *actions can occur regardless of whether action_interrupt is enabled.*

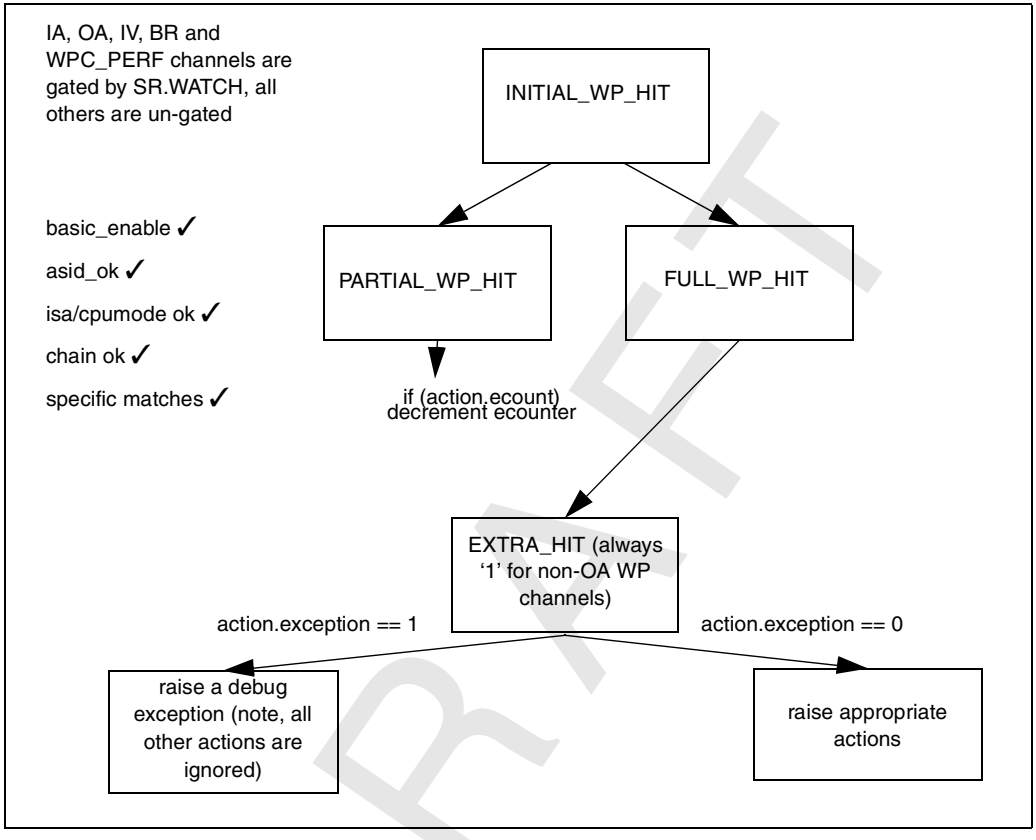

Figure 2: Channel matching algorithm

1.6.4 Behavior when more than one WPC channel matches an instruction

This section describes the defined behavior when more than one WPC channel (IA, IV or OA) matches on the same instruction. Most of the text describes the behavior when in SHmedia mode. The SHcompact behavior is similar, but with some divergence; it is described at the end.

When none of the matching channels have ACTION_EXCEPTION programmed in their action registers, the effect is to logically 'OR' together the actions across all the matching channels. Thus, if 2 channels both specify the same performance counter decrement, that performance counter will still only be decremented by 1 per matching instruction.

When one or more of the matching channels have ACTION_EXCEPTION programmed in their action registers, the behavior is as follows. The following WPC and DM actions are suppressed, regardless of which matching channel (excepting or non-excepting) requests them:

- set or clear WPC or DM generic chain latch,
- increment WPC performance counter,
- reset WPC performance counters,
- trace generation,
- raise debug interrupt,
- trigger out message generation.

If some matching channels specify ACTION_ECOUNT in their action registers, whether or not a particular WPC event counter is decremented is determined by whether the following conditions hold:

- there must be at least one matching channel with ACTION_EXCEPTION and ACTION_ECOUNT set, with ECOUNT_ID referring to the particular event counter

AND

- at least one such channel must be of the type corresponding to the debug exception that is actually launched.

For example,

- if DEBUGIA is launched, only matching IA channels with both ACTION_EXCEPTION and ACTION_ECOUNT set can cause a WPC event counter to be decremented. Matching IV and OA channels have their ACTION_ECOUNT settings ignored.

When a SHcompact mode instruction hits multiple WPC channels, there may be some divergence from the above description, depending on the particular SHcompact instruction involved. In particular, non-exception actions programmed on a matching IA channel may occur even when a DEBUGOA exception is launched from a matching OA channel.

The SHcompact instructions that are affected are:

```
AND.B #imm, @(R0, GBR)
OR.B #imm, @(R0, GBR)
STS.L FPSCR, @-Rn
STS.L MACH, @-Rn
TAS.B @Rn
XOR.B #imm, @(R0, GBR)
```

1.6.5 Handling of non-debug exceptions

This section describes the behavior when a WPC channel (IA, IV, OA or BR) matches on an instruction that causes any non-debug pre-execution exception to be launched. A non-debug exception is one other than DEBUGIA, DEBUGIV, or DEBUGOA.

When executing in SHmedia mode, the actions associated with all IA, IV, OA and BR channels are suppressed. Thus the debug handler sees all WPC event counters, performance counters and chain latches in the state they were prior to the instruction.

When executing in SHcompact mode, the behavior is similar. However non-exception actions programmed on an IA channel may still occur, even if an non-debug exception occurs, for certain instructions in some situations. The affected instructions and exception types are listed in *Table 22*.

SHcompact instruction	Exception type(s) that do not suppress IA channel actions
AND.B #imm, @(R0, GBR)	WRITEPROT
LDS.L @Rm+, FPSCR	FPUDIS
MAC.L @Rm+, @Rn+	RADDERR, RTLBMISS, READPROT (for @Rm+)
MAC.W @Rm+, @Rn+	RADDERR, RTLBMISS, READPROT (for @Rm+)
MOV.L @(disp, PC), Rn	RADDERR, RTLBMISS, READPROT
MOV.W @(disp, PC), Rn	RADDERR, RTLBMISS, READPROT
OR.B #imm, @(R0, GBR)	WRITEPROT
STS.L FPSCR, @-Rn	WADDERR, WTLBMISS, WRITEPROT
STS.L MACH, @-Rn	WADDERR, WTLBMISS, WRITEPROT

Table 22: SHcompact exceptions which do not suppress IA channel actions

SHcompact instruction	Exception type(s) that do not suppress IA channel actions
TAS.B @Rn	WRITEPROT
TRAPA #imm	TRAP
XOR.B #imm, @(R0, GBR)	WRITEPROT
any	ILLSLOT, SLOTFPUDIS (if preceding branch instruction had IA channel hit)

Table 22: SHcompact exceptions which do not suppress IA channel actions

1.7 Reset, panic and debug events

The SH-5 instruction set architecture defines a control register (RESVEC) which defines a vector used to handle reset, panic and debug events (refer to the *Core Architecture* manuals for full details). It is assumed that the reader is familiar with this document.

As described in the instruction set architecture, SH-5 provides the ability to divert these events from the normal RESVEC vector, to a separate vector known as DBRVEC.

This allows a debugger to be loosely coupled to the application it is debugging, such that it can "intercept" all the reset, panic and debug events by overlaying its own event handler in the place of RESVEC, and thus can handle these events within the debug agent, or vector them to the application's normal event handler as required.

Irrespective of whether RESVEC or DBRVEC is selected, each cause of reset, panic, debug event uses the same:

- vector offsets,
- EXPEVT codes,
- DM.EXP_CAUSE values (see *DEBUGINT - debug interrupts on page 81*).

These registers and values are defined in *Section 1.7.3: Event specific information on page 80*.

1.7.1 RESVEC/DBRVEC selection

By default, any occurrence of reset, panic or debug events will vector through RESVEC as defined in the instruction set architecture. The WPC provides two memory mapped registers which allow this behavior to be modified:

- WPC.CPU_DBRVEC

  Defines a separate vector known as DBRVEC. The least significant bit of DBRVEC provides the same facilities as RESVEC in that it allows the MMU to be disabled when launching the DBRVEC handler in response to debug events (DEBUGIA, DEBUGIV, DEBUGOA, DEBUGSS, BREAK, PANIC and CPURESET).

- WPC.CPU_DBRMODE

  Selects whether debug events vector through RESVEC or through DBRVEC.

Note: From the CPU core's viewpoint, DEBUGRESET *is indistinguishable from* POWERON *reset. Thus* DEBUGRESET *will always vector through* RESVEC *irrespective of the setting of* WPC.CPU_DBRMODE*.*.

WPC.CPU_DBRMODE

WPC.CPU_DBRMODE				0x104008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ENABLE	0	1	—	DBRVEC enable	RW
	Operation		Selects whether RESVEC or DBRVEC is used for reset, panic and debug Events. Value - Description 0: Use RESVEC 1: Use DBRVEC		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		0		

Table 23: WPC.CPU_DBRMODE register definition

WPC.CPU_DBRMODE				0x104008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 23: WPC.CPU_DBRMODE register definition

WPC.CPU_DBRVEC

WPC.CPU_DBRVEC				0x104010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MMUOFF	0	1	—	MMU (and hence cache) disable	RW
	Operation		Specifies whether the MMU is disabled when raising debug events through DBRVEC. Value - Description 0: Do not alter the state of SR.MMU. 1: Upon launch of the DBRVEC event handler for debug events, the MMU will be forced to be disabled (that is, the MMU bit of SR will be forced to 0).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	1	1	—	Reserved	RES
	Operation		Software should always write 0 to these bits. Software should always ignore the value read from these bits.		
	When read		Reads as 0 (behavior of other implementations may vary).		
	When written		Writes ignored (behavior of other implementations may vary).		
	HARD reset		0 (behavior of other implementations may vary).		
ADDRESS	[2,31]	30	—	DBRVEC address	RW
	Operation		Defines the address used for Reset, Panic and debug Events when DBRVEC is selected. Note that DEBUGRESET is *always* vectored through RESVEC.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 24: WPC.CPU_DBRVEC register definition

WPC.CPU_DBRVEC				0x104010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
EXP	[63:32]	32	—	EXPANSION	EXP
	Operation		These bits may be used on future implementations to expand the address space using a sign-extended convention. Software should always write a sign-extension of bit 31 into these bits. This approach is necessary if software on this implementation is to be executed on a future implementation with more implemented address space.		
	When read		Reads as a sign-extension of bit 31 (behavior of other implementations may vary).		
	When written		Writes ignored (behavior of other implementations may vary).		
	HARD reset		Sign-extension of bit 31 (behavior of other implementations may vary).		

Table 24: WPC.CPU_DBRVEC register definition

1.7.2 Event handling sequence

The reset, panic and debug event handler sequence is as per the normal event handling sequence as defined in event handling in *SH-5 Core Architecture Volume 1*.

Some important additional behavior, specific to debug events is provided in the following sections.

Multiple simultaneous debug events

Debug events are a subclass of reset, panic and debug events. They correspond to the cases where:

- A synchronous CPU watchpoint match occurs on the BRK, instruction address, operand address or instruction value WP channels.
- A debug interrupt, bus analyzer or debug module watchpoint match occurs on the BRK channel (due to a debug interrupt), or due to bus analyzer or debug module WP channels.

The debug event sequence is identical for both of these circumstances, but there are differences in the data available to the exception handler according to the WP channel type. This state is described in the WP channel type sections of this document (*Section 1.10* through *Section 1.18*).

In a given processor cycle, multiple debug events can match, each of which can potentially have its action set to raise a debug exception. In these circumstances, a single debug event (the highest priority debug exception which matches) will be raised. These priorities are defined in *Section 1.7.3: Event specific information on page 80*. There are two general approaches that can be used to cope with instructions that hit more than one excepting condition:

1 The debug event handler could work out which other conditions would have hit and carry out all the actions.

2 The debug handler can temporarily disable the exception action from the channel(s) which hit and caused the exception. When the excepting instruction is restarted by the RTE at the end of the handler, lower priority exceptions will get a chance to launch. The handler should enable single stepping, so that a DEBUGSS exception will be taken on completion of the instruction. The DEBUGSS handler can re-instate any channels' ACTION_EXCEPTION actions that were temporarily disabled.

MMU disable

Either RESVEC or DBRVEC is used to vector debug events (see *Section 1.7.1*).

Both RESVEC and DBRVEC provide a MMUOFF field to allow the MMU to be disabled when launching the event handler due to debug events (when launching for non-debug events such as reset or panic, the MMU is always disabled).

Disabling the MMU allows the debug event handler to execute without having to reserve TLB entries in the application being debugged. Thus it is possible to totally decouple the debug event handler from the debugger. It also allows execution of code with the caches disabled, thus it is possible to perform debug event handling without perturbing the caches.

If the MMUOFF bit of RESVEC/DBRVEC is set, the MMU will be forced to be disabled whenever the debug event handler is launched (that is, the MMU bit of SR will be forced to 0).

The MMU bit of the status register determines whether the MMU is enabled. The standard exception handler launch sequence involves copying the status register (SR) to the saved status register (SSR). Therefore the previous enable/disable status of the MMU is available such that it can be restored by the exception handler's exit sequence (that is, the RTE instruction).

SR.BL bit

When the BL bit of SR is '1':

- Attempts to raise a debug exception of type DEBUGIA, DEBUGIV, DEBUGOA, DEBUGSS or BREAK will result in a panic event being raised instead. EXPEVT will be set to indicate the normal event type, even though a panic event is generated. This allows the panic handler to distinguish between the synchronous debug event types.

  Panic events can be recovered from (as the previous EXPEVT contents are recorded in PEXPEVT), thus a target debug agent will need to handle both panic events and other types of debug event.
- Attempts to raise a debug interrupt (DEBUGINT) will block until the BL bit is cleared.

Note: If the cause of the debug interrupt is de-asserted whilst BL is set, the debug interrupt will be lost.

SR.WATCH bit

See *Section 1.6.1: SR.WATCH bit on page 66*.

SR.STEP bit

When SR.STEP is '1', an event will be raised whenever an instruction execution completes. The type of event generated depends on the value of SR.BL:

- If SR.BL is '0', a debug exception of type DEBUGSS will be raised.
- If SR.BL is '1', a panic event will be raised.

In both these cases EXPEVT is set as per *Table 26: Synchronous debug exceptions on page 81*

1.7.3 Event specific information

The reset, panic and debug event vector and offset values are shown below. The following subsections define the codes used to identify the individual reasons for the debug events.

Event	Base + Offset	Used when	
POWERON	0x0	Power-on/debug reset (EXPEVT for DEBUGRESET is the same as that for POWERON reset).	The instruction set architecture defines EXPEVT event codes for these events.
DEBUGRESET	0x0		
CPURESET	RESVEC + 0x0	RESVEC enabled	
	DBRVEC + 0x0	DBRVEC enabled	
PANIC	RESVEC + 0x0	RESVEC enabled	
	DBRVEC + 0x0	DBRVEC enabled	
DEBUGIA, DEBUGIV, DEBUGOA, DEBUGSS, BREAK	RESVEC + 0x100	RESVEC enabled	EXPEVT defines the reason (see *Section : Synchronous debug exceptions on page 81*
	DBRVEC + 0x100	DBRVEC enabled	
DEBUGINT	RESVEC + 0x200	RESVEC enabled	DM.EXP_CAUSE defines the reason (see *Section : DEBUGINT - debug interrupts on page 81*). It is also used to clear the sources of the debug Interrupt.
	DBRVEC + 0x200	DBRVEC enabled	

Table 25: Reset, panic, debug event vectoring

Synchronous debug exceptions

The synchronous debug exceptions are shown in *Table 26*.They are listed in decreasing order of priority.

DEBUGIA/IV/OA/SS, BREAK: RESVEC/DBRVEC offset 0x100			
WP Channel details		**Exception type**	**EXPEVT**
IA	Instruction Address	DEBUGIA	0x900
IV	Instruction Value	DEBUGIV	0x920
BRK	BRK executed	BREAK	0x940
OA	Operand Address	DEBUGOA	0x960
BRK	Single step	DEBUGSS	0x980

Table 26: Synchronous debug exceptions

DEBUGINT - debug interrupts

DEBUGINT: RESVEC/DBRVEC offset = 0x200		
WP Channel details		**Bits of DM.EXP_CAUSE set for this channel**
DM	FIFO activity	.DM_FIFO_INTERRUPT == 1
PL	SuperHyway bus analyzer	.PL_INTERRUPT == 1
BRK	debug Interrupt	.FORCED_DEBUG_INTERRUPT == 1
OA	OA interrupt with data comparison (see *Section 1.12.3: Data match registers on page 147*)	.OA_MATCH_INTERRUPT == 1

Table 27: Debug interrupt reasons

The CPU will launch a DEBUGINT when these 3 conditions are met:

- SR.BL = 0
- At least one bit of DM.EXP_CAUSE is set. If SR.BL=1 when a cause bit is first asserted, at least one cause bit must still be asserted when SR.BL changes to 0.
- In at least one clock cycle since the last DEBUGINT launch or reset (of any kind), all bits of DM.EXP_CAUSE have been clear.

If a bit in DM.EXP_CAUSE is set then subsequently cleared, with SR.BL=1 all the time, no DEBUGINT will occur.

If the handler for DEBUGINT deals with some conditions signalled in DM.EXP_CAUSE but leaves others asserted, there will not be another DEBUGINT after the handler returns. A handler for DEBUGINT must be written to handle all asserted conditions, clear the handled conditions in DM.EXP_CAUSE (see below) and ensure that DM.EXP_CAUSE has been read back with all bits clear before returning. This ensures a new DEBUGINT can occur when a cause is next asserted.

The write semantics of DM.EXP_CAUSE are such that a handler can clear the cause bits selectively.

Each bit of DM.EXP_CAUSE corresponds to a source of DEBUGINT, *Table 28* shows the state indicated when reading these bits, and the effect of writing them. *Table 29* shows which bits in the register correspond to specific interrupt sources.

State of DM.EXP_CAUSE bits			
Value read	**Meaning**	**Value written**	**Effect**
0	Interrupt source was not asserted	0	The interrupt source will remain not asserted. If the interrupt source has been asserted since the register was read, this will actually clear the interrupt without its condition being properly handled. For this reason, writing 0 to a bit which was read as 0 is not advised.
		1	Writing 1 has no effect on the bit's value.

Table 28: State of reading/writing DM.EXP_CAUSE bits

State of DM.EXP_CAUSE bits			
Value read	**Meaning**	**Value written**	**Effect**
1	Interrupt source was asserted	0	The interrupt source will be de-asserted. Note that some interrupt sources require additional actions to fully de-assert them (see *Table 29 on page 83*).
		1	Writing 1 has no effect on the bit's value.

Table 28: State of reading/writing DM.EXP_CAUSE bits

The DEBUGINT handler should write '**0**' to the appropriate bit(s) in DM.EXP_CAUSE as soon as it can. This will minimize the risk of losing new interrupts that arrive during the execution of the DEBUGINT handler.

DM.EXP_CAUSE				0x100010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DM_FIFO_INTERRUPT	0	1	✓	DM FIFO interrupt	RW
	Operation		This field is set whenever an interrupt is generated due to DM FIFO activity as selected by the FF_THRESH field of DM.TRCTL (see *Table 31 on page 97*).		
	When read		Returns current value		
	When written		Updates value. Writing 0 will partially clear the cause of the DM FIFO interrupt. In order to fully clear the interrupt, the DM FIFO must be setup to remove the cause (either reprogram DM.TRCTL.FF_THRESH such that it will not generate debug Interrupts or empty the DM FIFO). Writing 1 has no effect - any pending DM FIFO interrupts will not be lost, and subsequent reads will return 1 (unless the source of the interrupt is unexpectedly removed).	
	HARD reset		0		

Table 29: DM.EXP_CAUSE register definition

DM.EXP_CAUSE				0x100010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PL_INTERRUPT	1	1	✓	SuperHyway watchpoint interrupt	RW
	Operation		This field is set due to a transitory state on the SuperHyway triggering a bus analyzer watchpoint which has .ACTION_EXCEPTION set as 1.		
	When read		Returns current value		
	When written		Updates value. Writing 0 will clear the PL_INTERRUPT. Writing 1 has no effect - any pending SuperHyway interrupts will not be lost, and subsequent reads will return 1 (unless the source of the interrupt is unexpectedly removed).		
	HARD reset		0		
FORCED_DEBUG_ INTERRUPT	2	1	✓	Forced debug interrupt	RW
	Operation		Contains 1 if the debug interrupt was raised by writing '1' to the DM.FORCE_DEBUGINT.FORCE register field (see *Section 1.3.4: on page 33*).		
	When read		Returns current value		
	When written		Updates value. Writing 0 will clear the forced debug interrupt. Writing 1 has no effect - any pending forced debug interrupts will not be lost, and subsequent reads will return 1 (unless the source of the interrupt is unexpectedly removed).		
	HARD reset		0		

Table 29: DM.EXP_CAUSE register definition

DM.EXP_CAUSE				0x100010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OA_MATCH_ INTERRUPT	3	1	✓	OA watchpoint data match interrupt	RW
	Operation		This field is set due to a transitory state in the instruction stream triggering an OA watchpoint channel which resulted in a successful data match and has .ACTION_INTERRUPT set as 1, and subsequent reads will return 1 (unless the source of the interrupt is unexpectedly removed).		
	When read		Returns current value		
	When written		Updates value. Writing 0 will clear the OA watchpoint data match interrupt. Writing 1 has no effect. Any pending OA watchpoint data match interrupts will not be lost.		
	HARD reset		0		
—	[63:4]	60	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 29: DM.EXP_CAUSE register definition

1.8 Debug module

The debug module, as shown in *Figure 3*, connects to the:

- SHdebug link interface,
- JTAG TAP controller,
- Watchpoint controller (WPC) logic in the CPU core,
- SuperHyway bus (as both a master and as a slave),

- Bus capture buffers in the SuperHyway bus analyzers.

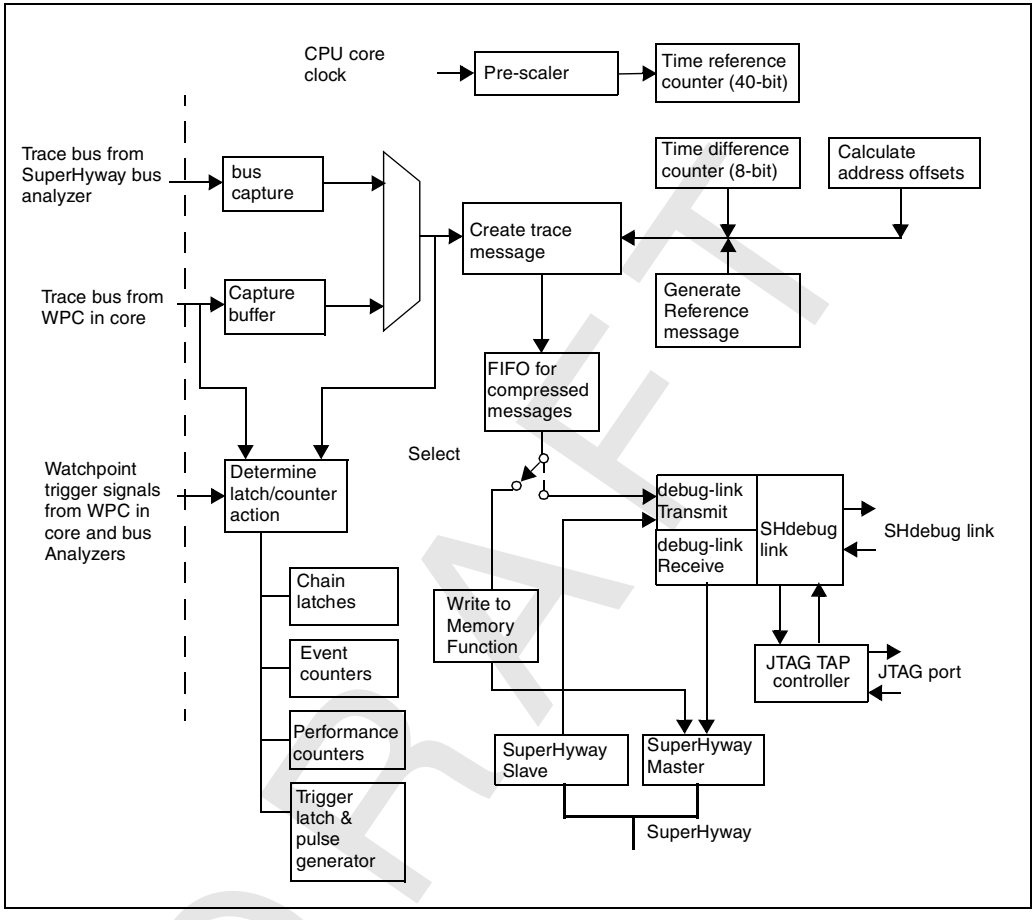

Figure 3: Debug module functions

The debug module has the following main functions:

- Determine the destination of trace messages received from the watchpoint controller (WPC) in the CPU core, or from either of the bus analyzers.
- Provide FIFO buffering for trace messages awaiting transmission to a tool or writing to target memory trace buffer.
- Send trace messages to a tool using either the SHdebug link or the JTAG debug interface.

- Write trace messages into a trace buffer in the target system's memory.
- Set and clear any of the chain-latches in the debug module.
- Control the trigger-out signal (DM_TROUT_N).
- Route SuperHyway bus transactions to or from the tool using either the SHdebug link or the JTAG debug interface.

1.8.1 Address spaces

The debug module is assigned 32 Mbytes of SuperHyway target address space. 16 Mbytes of this consists of DM registers, the other 16 Mbytes is mapped to the tool via the SHdebug link/JTAG port.

1.8.2 Fast printf

Fast printf is an extremely simple and powerful technique in which the processor emits a piece of data and the current PC value whenever a program writes to a special CPU register. External hardware and software print this data as requested by the programmer, usually in the "console" window of a graphical debugger running on a tool. Support of fast printf requires enhancements to the programming tools, as well as appropriate debugging facilities. The fast printf function allows a programmer to embed fast printf code into programs and probe any register or memory contents.

The fast printf function is part of the debug module logic and uses the DM.FPF register. Whenever the DM.FPF register is written, the debug module creates a trace message containing the program counter, ASID and the data value. This trace message is sent directly to the tool, bypassing the DM FIFO, irrespective of how the trace destination is configured. The FPF trace message is sent to the tool at the first opportunity, that is, immediately following any other trace message currently being sent.

Writes to the DM.FPF register are treated differently from writes to other SuperHyway bus registers. The debug module delays the SuperHyway bus response to such writes until the FPF trace message has been sent to the tool.

The fast printf function is available as a WP channel (see *Section 1.4: Watchpoint channels on page 35*). This allows its operation to be selectively enabled and disabled by one of the chain latches.

This facility will be used to implement efficient tool/target data communications. This operation will typically be performed in an environment where tracing may also be performed at the same time. Thus, it is sensible if the fast printf data is always written directly to the debug port to avoid interfering with the FIFO operation.

1.8.3 DM FIFO/trace buffer in target system memory

The debug module contains a FIFO (see *Section 4.1.5: DM FIFO on page 246*) which holds trace messages awaiting transmission to the tool, or waiting to be written into a trace buffer in the target system's memory.

The DM.TRCTL.DESTN (*Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*) determines one of the following actions for all trace data loaded into the DM FIFO:

- Trace messages in the DM FIFO are sent to the tool using the currently-selected debug interface (known as "trace link" mode).
- Trace messages in the DM FIFO are written into an area of target system memory assigned as a trace buffer (known as "trace buffer" mode). The DM.TRBUF register (*Table 32 on page 107*) allows this to be further selected as either "circular trace buffer" mode or "trace buffer hold" mode. The format is which the trace messages are written is described in *Section 1.8.12: DM.TRPTR (trace pointer register) on page 110*.

*Note: In "trace buffer" mode the DM writes the trace messages to target system memory using SuperHyway **Store8/16/32** transactions. These transactions are visible to the SuperHyway bus analyzers (see Chapter 2: SuperHyway bus analyzer on page 179), and thus if a bus analyzer is programmed such that it will match on these transactions, an infinite number of bus analyzer hits (and thus an infinite number of trace message) will be generated.*

- Trace messages remain in the DM FIFO until read by the CPU or by the tool (known as "DM FIFO trace hold" mode).
- old messages in the DM FIFO are overwritten by new ones so that the DM FIFO contains the most recent trace messages generated (known as "circular DM FIFO" mode).

When in “trace link”, “DM FIFO trace hold” or “trace buffer hold” mode the trace system can be programmed to either stall the CPU, or discard trace messages (see *Stall/discard overview on page 91*). In the FIFO modes, trace messages are packed into the DM FIFO in an implementation defined manner, according to the size of the trace message:

- Some DM FIFO implementations may be byte-based so that a variant number of trace messages can be held in the DM FIFO, this provides “best fit” and thus makes most efficient use of the available FIFO space.
- Some DM FIFO implementations may pack trace entries in a fixed manner (that is, 3*64-bit) intervals.

In normal operation, fields of trace messages corresponding to PC values or bus analyzer address values are encoded relative to the last such address in order to make the trace messages as compact as possible. If the trace message which “seeded” these values is lost, the subsequent trace messages using relative values cannot be interpreted until a further absolute value is issued in a subsequent trace message.

For this reason, when in “DM FIFO trace hold”/“trace buffer hold” mode or “circular DM FIFO”/“circular trace buffer” mode, all trace messages use absolute rather than relative values to ensure that the trace messages can be utilized even when a preceding message which “seeded” these values has been lost. See *Section : Encoding of address offsets on page 120* for full details.

Trace messages can be read from the DM FIFO an entry at a time using the DM.FIFO_{0/1/2} registers (see *Section 1.8.13: DM.FIFO_0/DM.FIFO_1/DM.FIFO_2 (FIFO port register) on page 112*). When the DM FIFO is in “circular DM FIFO” mode, care must be taken to ensure trace is not being generated whilst it is being extracted or trace messages may be lost. See *Section 1.8.13: DM.FIFO_0/DM.FIFO_1/DM.FIFO_2 (FIFO port register) on page 112* for full details.

1.8.4 Watchpoint hit buffering and trace message generation

Figure 4 is a functional block diagram showing the buffering within the DM for watchpoint hit information used to create trace messages.

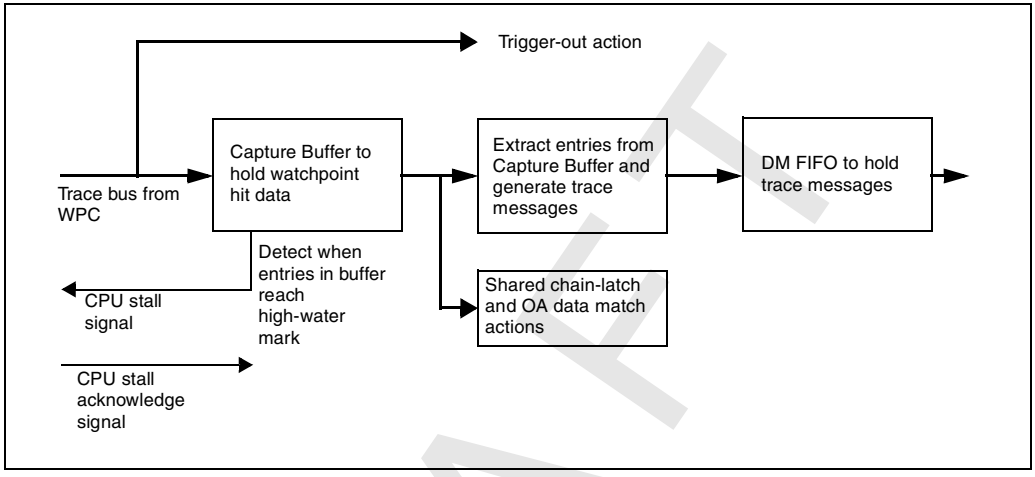

Figure 4: Buffering within the DM

The trace bus which connects the WPC to the DM is able to transfer information about watchpoint hits every CPU clock cycle. The bus contains individual tag bits for each WPC watchpoint, permitting multiple watchpoint hits detected in the same clock cycle to be signalled. This raw watchpoint hit information is loaded into a capture buffer, together with the current value of the shadow program counter (see *Section 1.8.14: DM.PC (shadow program counter register) on page 117*). The trace message generation logic of the DM extracts watchpoint hit details from the capture buffer and loads corresponding trace messages into the DM FIFO.

WPC watchpoint hit actions performed within the DM include:

- trace generation,
- setting/clearing a shared chain-latch,
- controlling the trigger-out signal,
- generating an OA data match interrupt.

Stall/discard overview

The DM.TRCTL register (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*) has a field (DM.TRCTL.STALL_MODE) which determines the action if there is no space available in the capture buffer at the time a WPC watchpoint hit or BR watchpoint hit occurs. The two possible actions are:

- stall the CPU until space becomes available in the capture buffer or
- discard watchpoint hits which are unable to be written to the capture buffer.

All actions (other than controlling the non-BR hit trigger-out signal) are performed at the output of the capture buffer. This means that in stall mode (see *Table 31: DM.TRCTL definition on page 97*), actions will be delayed when the capture buffer fills and the processor stalls. In discard mode, when the capture buffer fills, the following occur until space becomes available:

- subsequent watchpoint hits will be discarded,

  When space does become available in the capture buffer, the next trace message generated due to a WP hit will have its overstall bit set to 1 to denote that WP hits have been lost.
- DM ACTION_CHAIN_ALTER actions for IA/IV/OA/BR hits are voided,
- ACTION_TRIG_OUT actions for BR hits are voided,
- ACTION_TRACE actions for IA/IV/OA/BR hits are voided.

(WPC ACTION_CHAIN_ALTER actions for IA/IV/OA hits are performed normally in this situation and are not voided.)

Generation of trigger-out pulses for IA/IV/OA hits is performed at the input to the capture buffer, and thus is performed even in discard mode. However, the length of the trigger-out pulse can cover several distinct WP hits occurring.

Stall mode

The CPU pipeline cannot be instantly stalled. When a stall signal from the DM is asserted, the pipeline stops fetching new instructions which causes the pipeline to empty an implementation defined number of clock cycles later (see *DM FIFO high-water mark on page 246*). Only at this time is the CPU stalled and unable to generate any more watchpoint hits.

The determination of the size of the capture buffer must allow for this CPU stall latency. The capture buffer includes “high-water” detection logic which asserts the

CPU stall signal when the number of entries in the buffer is within an implementation defined number of entries of the maximum size of the buffer. This ensures that the capture buffer has sufficient space for possible further watchpoint hit entries produced during the time that the CPU pipeline is draining.

Stall/discard status

Regardless of whether stall mode or discard mode is selected, the hit which is written into the capture buffer (either overwriting the previous hit or after the CPU stall has ended) has a status bit set to indicate that the stall/discard action occurred. This status bit is included in the trace message sent to the tool or stored in target system memory.

1.8.5 IA watchpoint trace modes

When an IA watchpoint hit invokes a CPU debug exception, the resident monitor has the option of changing the watchpoint parameters to avoid another immediate hit as soon as the monitor returns to the original running thread.

This behavior is not possible when the watchpoint action is trace, but involves no exception handling. Trace is entirely passive, and thus unless a exception handler is invoked there is no mechanism to prevent further trace messages from being emitted.

This can generate large amounts of trace, which are not necessarily useful.

For example:

Consider an instruction address watchpoint placed on address range A(n) to A(m).

1 The first instruction execution between A(n) to A(m) will generate a instruction address trace message.

2 All subsequent executions within A(n) to A(m) will also generate trace.

In many circumstances, only the first trace message is of interest, subsequent accesses within A(n) to A(m) are not of interest until the execution has proceeded outside of A(n) to A(m) that is, until the WP channel match has failed.

SH-5's IA WP channels support two trace modes to allow such filtering to occur without exception handler intervention. These modes are known as multi-trace and single-trace mode.

Multi-trace mode

Each and every watchpoint hit for the WP channel generates a trace message.

This mode of operation is likely to generate a large volume of trace messages and thus is mainly suitable for use when a large FIFO is available, or either when CPU stall mode is enabled, or when the chain-latches are being used to filter the trace data to manageable quantities.

Single trace mode

A single trace message is produced for a given IA WP channel, until the channel fails to match.

The other actions associated with the channel (such as incrementing performance counters, affecting chain-latches for example) are handled regardless of the trace mode.

Trace generation algorithm

Each IA channel has a 1 bit state value (TRACE_GENERATED).

Whenever the channel fails to match, TRACE_GENERATED is set to 0.

The following pseudo-C details the trace generation process.

```
IF (WPchannel.matched) { // match obeys WP channel pre conditions
  IF ( (WPchannel.multi_mode) || (!WPchannel.trace_match_state) ) {
    WPchannel.trace_match_state = 1;
    GenerateTraceMessage();
  } ELSE {
    // trace already generated for this match, so don't generate one
    increment performance counters etc. as defined by action
} ELSE {
  // channel match failed
  WPchannel.trace_match_state = 0;
}
```

1.8.6 Timestamping and reference messages

The debug module includes a single 40-bit timestamp counter. This can be used to add timing information to both CPU trace and bus analyzer trace information. Using a single timestamp counter ensures that both trace types are coherent in a

single time domain. The timestamp function in the debug module includes a programmable pre-scaler driven from the DM clock (see *Table 31: DM.TRCTL definition on page 97*). This allows a debug user to set the timestamp increment size to suit the application being debugged.

Reference messages (*Table 47 on page 133*) are occasionally generated to “reseed” the absolute values used in the address and timestamp value compression scheme.

The circumstances under which reference messages are generated are:

- Immediately prior to the generation of the first non-FPF trace message.

  Thus SH-5 will not generate any reference messages until the point at which other trace would be generated. This will only occur if the debug tool has programmed sources of trace generation (that is, watchpoint channels), and has suitably configured the trace destination. From this point onwards, reference messages may be generated at regular intervals.

- Immediately prior to the generation of a non-FPF trace message which was generated at least 256 time intervals after the previous trace message.

  This provides regular “reseeding” of the absolute address and timestamp values.

Each DM.WP_{IA/OA/IV/PL}X_ACTION register has a field which determines whether timestamps are added to trace messages. If added, the single byte timestamp field specifies a time difference from the last reference message.

1.8.7 Trigger-in chain-latch

The DM_TRIN_N pin is manifested as a chain-latch, in order to allow for its inclusion in WP channel pre-conditions and actions. It has no associated action, and thus must be used in conjunction with a WP channel in order to cause an action.

The DM.TRCTL register (*Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*) determines how the trigger chain-latch’s state is determined. It can be set in “manual mode” where the DM_TRIN_N pin does not affect it, or connected to the DM_TRIN_N as either edge or level triggered, with both rising and falling triggers supported.

The normal chain-latch operations are available on the WP channels to clear or set the trigger chain-latches state as required. However, these do not apply when it is in manual mode.

1.8.8 Trigger-out

The trigger-out (DM_TROUT_N) pin can be configured by the DM.TRCTL register to operate in one of the following modes:

1 Each time a WPC watchpoint (a CPU core watchpoint) trigger-out event is detected, an active-low pulse is output on the DM_TROUT_N pin.

   This allows the trigger-out pin to be used to trigger external equipment to assist with system-level debugging. For example, triggering a logic analyzer or qualifying a logic analyzer trigger so that the analyzer is able to capture system-level states/events only following a specific watchpoint hit. Alternatively, the trigger-out pin can be used to accurately measure certain real-time performance characteristics of the SH-5 system-on-a-chip. A logic analyzer can capture all activity from the pin and be used to show the occurrences of a specific watchpoint hit, for example, indicating how often a particular routine is being invoked.

   It is possible for CPU watchpoint hits to occur every CPU clock period but it is impractical for the trigger-out signal to respond to watchpoint hits occurring at this rate since this would require output pulses of width equal to half the CPU clock period (that is, a pulse width of 1.25 ns at 400 MHz). Instead, trigger-out watchpoint hits are sent to pulse stretching logic within the debug module which asserts the DM_TROUT_N pin for a longer period. Trigger-out watchpoint hits which occur during the time the DM_TROUT_N signal is asserted or during the equal time the signal is recovering are ignored. The pulse width and recovery times for SH-5 are specified in *Section 4.1.9: Trigger out pulse width on page 249*.

2 Each time a bus analyzer watchpoint trigger-out event is detected, an active-low pulse is output on the DM_TROUT_N pin.

   This mode of operation of trigger-out can be used for the same functions described above but relating to bus analyzer watchpoint hits.

   It is possible for bus analyzer watchpoint hits to occur every bus clock period. Even though the bus clock frequency is lower than that of the CPU core, the frequency is still too high for a trigger-out pulse to occur every bus clock period. bus analyzer trigger-out watchpoint hits are sent to pulse stretching logic similar to that described above.

3 The trigger-out pin is connected to the output of the WPC.CHAIN_0 chain-latch.

Since the chain-latch can be directly set by one WPC watchpoint hit and directly cleared by a different WPC watchpoint hit, the trigger-out pin can be used to accurately measure the time between different WPC watchpoint events.

4 The trigger-out pin is connected to the output of the DM.CHAIN_0 chain-latch.

Since the chain-latch can be directly set by one WPC or bus analyzer watchpoint hit and directly cleared by a different watchpoint hit, the trigger-out pin can be used to measure the time between different watchpoint events. There are two main differences between this TRIGGER-OUT function and the one described above for the WPC.CHAIN_0 chain-latch:

- The DM.CHAIN_0 chain latch can be set or cleared by any bus analyzer or WPC watchpoint.
- The action logic for DM.CHAIN_0 chain-latch setting/clearing is located at the output of the capture buffer. If stall-mode is selected, the chain-latch setting/clearing action will be delayed by an unpredictable amount of time as the capture buffer fills. If discard-mode is selected, watchpoint hit events will be discarded when the capture buffer fills and expected chain-latch setting/clearing actions may not occur.

5 If stall mode is selected, the trigger-out pin goes low during the time that the CPU is stalled. This allows external equipment connected to the trigger-out pin to observe CPU stall behavior.

1.8.9 DM.FPF register definition

When written to, this register generates a fast printf message which is sent to the tool (see *Section 1.15: WP channel type FPF on page 163*).

DM.FPF				0x100018	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
VALUE	[63:0]	64	—	Fast Printf data value	RW
	Operation		Specifies the data value to be placed in the FPF_DATA field of the fast printf message.		
	When read		Returns last written value		
	When written		Generates a fast printf message as defined in *Table 45 on page 130*. Unless a SYNCO instruction follows the store instruction which writes to this register, a bogus PC value may be placed in the FPF message generated. See *Access to registers on page 17*.		
	HARD reset		Undefined		

Table 30: DM.FPF register definition

1.8.10 DM.TRCTL (trace/trigger register)

This register determines the destination of trace messages, the stall/discard mode and controls the function assigned to the trigger-out (DM_TROUT_N) pin.

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	0	1	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DM_TRIG_IN_MODE	[2:1]	2	—	Trigger-in mode	RW
	Operation		Determines the state of the trigger chain-latch relative to the DM_TRIN_N pin as follows: Value - Description 0b00: edge triggered, a rising edge on DM_TRIN_N sets the trigger chain-latch. The latch is only automatically cleared by a WP channel's action explicitly clearing it. 0b01: edge triggered, a falling edge on DM_TRIN_N sets the trigger chain-latch. The latch is only automatically cleared by a WP channel's action explicitly clearing it. 0b10: continuous sampling, level ==1 on DM_TRIN_N sets the trigger chain-latch, level == 0 on DM_TRIN_N clears the trigger chain-latch 0b11: manual mode. The trigger chain latch will only be set or cleared by explicit writes to the DM.CHAIN_TRIG_IN register. Alterations attempted by WP channel's action_chain_alter setting have no effect. The trigger chain-latch can be manually set or cleared by writing to DM.CHAIN_TRIG_IN, but this is not recommended.	
	When read		Returns current value		
	When written		Updates value		
	HARD reset		0b11		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_CLEAR	3	1	—	DM FIFO clear	RW
	Operation		Used by software to change the DM FIFO status to empty when the trace destination is “DM FIFO trace hold”. Value - Description 0: No action 1: Clear FIFO. This operation is not instantaneous, the value of the FF_CLEAR field can be used to determine that the clear operation has completed.		
	When read		Returns current value. 0 if the FIFO is empty 1 otherwise		
	When written		If trace destination is “DM FIFO trace hold” mode, writing 1 will clear the FIFO. Writing 1 in other modes will produce undefined effects. Writing 0 in any mode produces no effect.		
	HARD reset		0		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_THRESH	[5:4]	2	—	DM FIFO interrupt threshold	RW
	Operation		Controls the generation of a debug interrupt when the trace destination is “DM FIFO trace hold” mode (destn = 0b00). With other trace destinations no interrupt can ever be generated due to trace messages. Value - Description 0b00: No debug interrupt generated in response to DM FIFO writes. 0b01: Generate debug interrupt whenever an entry is written to the DM FIFO. 0b10: Generate debug interrupt when DM FIFO reaches its high-water mark (see *DM FIFO high-water mark on page 246*). 0b11: Undefined		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0b00		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_STATUS	[7:6]	2	✓	DM FIFO status	RO
	Operation		Indicates the status of the DM FIFO. When the trace destination is programmed as “DM FIFO trace hold” mode (destn == 0b00) or “Circular DM FIFO” mode (destn == 0b11), this indicates whether there are trace messages waiting to be read. In “Trace link mode” or “Trace buffer mode”, this indicates whether there are trace messages still waiting to be written to the selected trace destination. *Note: These status bits do not show whether there are any WPC watchpoint hits waiting in the capture buffer. Depending on the programming of the DM action registers, these may cause new trace messages to be generated later.* Value - Description 0b00: DM FIFO contains some trace data. 0b01: DM FIFO is empty. 0b10: DM FIFO is full. 0b11: Undefined		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0b01		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
STALL_MODE	8	1	—	CPU stall mode	RW
	Operation		Defines the current CPU stall mode. This determines what happens when the DM capture buffer fills. Value - description 0: (Discard mode). Do not stall the CPU when the capture buffer reaches its almost-full threshold. Instead, keep filling the capture buffer with watchpoint hit entries from the WPC and when no space is available, discard watchpoint hits which are unable to be written into the capture buffer. 1: (Stall mode). Stall the CPU when the DM capture buffer reaches its almost-full threshold. The CPU does not instantly react to a stall request from the DM and by using an almost-full threshold, enough space is available in the capture buffer for any additional watchpoint hit entries which may occur.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		0		
STALL_STATE	9	1	✓	Current CPU stall/suspend state	RO
	Operation		Defines if the CPU is currently stalled due to the DM's capture buffer being almost-full or if the CPU is suspended. Stall behavior is controlled by the stall_mode field. Suspend behavior is controlled by the WPC.CPU_CTRL_ACTION register (*Section 1.3.1: Suspending/resuming the CPU on page 30*).		
	When read		Returns current value. Value - Description 0: not stalled. 1: stalled		
	When written		Ignored		
	HARD reset		0		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DESTN	[11:10]	2	—	Trace destination	RW
	Operation		Determines the destination of trace messages generated by the CPU core watchpoint controller or either of the two bus Analyzers. Value - Description 0b00: (DM FIFO trace hold mode). All trace messages are held in the DM FIFO until read. Data can be read from the DM FIFO at any time. When the DM FIFO fills, DM.TRCTL.STALL_MODE determines whether the CPU is stalled or whether new trace messages are discarded. 0b01: (Trace link mode). Trace messages in the DM FIFO are sent to the currently-selected debug interface (SHdebug link or JTAG). Data in the DM FIFO cannot be accessed in this mode and undefined data will be returned if the DM FIFO register is read. 0b10: (Trace buffer mode). Trace messages in the DM FIFO are written to a trace buffer area in target system memory. Data in the DM FIFO cannot be accessed in this mode and undefined data will be returned if the DM FIFO register is read. The trace buffer can be configured into “trace buffer hold” or “circular trace buffer” mode using the DM.TRBUF register (see *Table 32: DM.TRBUF definition on page 107*). 0b11: (Circular DM FIFO Mode). Trace messages are held in the DM FIFO until read. As new trace messages are generated they overwrite the oldest messages in the DM FIFO. Debug software can read these messages as described in *Section 1.8.13: DM.FIFO_0/ DM.FIFO_1/DM.FIFO_2 (FIFO port register) on page 112*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRESCALER	[19:12]	8	—	Timestamp counter pre-scaler	RW
	Operation		The timestamp counter is driven by a pre-scaler which uses DM clock as its source. The value of the pre-scaler determines the time increment of the timestamp counter. Value - Description 0x00: No division, the time increment is the same as DM clock 0x01 to 0xFF - DM clock divided by (value + 1)		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0xFF		
DL_N_JTAG	20	1	—	Debug interface selected	RO
	Operation		This field shows the debug interface which is currently selected. Refer to *Section 3.3.3: Debug interface selection on page 213*. Value - Description 0- JTAG interface selected 1- SHdebug link interface selected		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		1		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DM_TRIG_OUT_MODE	[23:21]	3	—	Trigger-out mode	RW
	Operation		Defines the mode of operation of the trigger-out function. *Section 4.1.3: Performance counters on page 243* defines the length of trigger out pin pulses. Value - Description 0b000: The trigger-out function is disabled and the DM_TROUT_N pin is held high. 0b001: An active-low pulse is output each time a WPC (a CPU core watchpoint) channel triggers, and the triggering channel has .ACTION_TRIG_OUT set as one of its actions. 0b010: An active-low pulse is output each time a bus analyzer channel triggers, and the triggering channel has .ACTION_TRIG_OUT set as one of its actions. 0b100: The trigger-out pin is connected to the output of the WPC chain-latch, WPC.CHAIN_0. 0b101: The trigger-out pin (DM_TROUT_N) is connected to the output of the DM shared chain-latch, DM.CHAIN_0. 0b110: The trigger-out pin is connected to the stall-acknowledge signal from the CPU core. If stall mode is selected, the DM_TROUT_N signal goes low during the time that the CPU is stalled when the capture buffer is full. 0b111: Undefined	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 31: DM.TRCTL definition

DM.TRCTL				0x100040	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:24]	40	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 31: DM.TRCTL definition

1.8.11 DM.TRBUF (trace buffer register)

This register is specified in *Table 32*. Fields in this register are used only when debug software allocates a portion of target system memory space as a trace buffer.

DM.TRBUF				0x100048	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TR_MODE	[1:0]	2	—	Trace Buffer mode selection	RW
	Operation		Defines the mode of operation of the trace buffer in the target system's memory. The value in this field is only significant if DM.TRCTL.DESTN = 0b10 (that is, the trace destination is programmed as trace buffer mode). Value - Description 0b00: Trace buffer disabled. This allows debug software to disable tracing whilst the other fields (such as TR_SIZE and TR_BASE) are programmed. 0b01: "Circular trace buffer" mode. The trace buffer acts as a wrap-around buffer with the newest trace entry overwriting the oldest. 0b10: "Trace buffer hold" mode. The trace buffer acts as a fixed length buffer. DM.TRCTL.STALL_MODE (see *Section 1.8.10: on page 97* effects the behavior (see *Section : Stall/discard overview on page 91*). 0b11: Undefined	
	When read		Returns current value		
	When written		Updates current value. Undefined behavior may result if this field is changed whilst a trace message is being written to memory. Software should disable all watchpoint channel actions that could produce trace messages and wait long enough to allow any pending messages to drain to memory. The time to wait is implementation-dependent.		
	HARD reset		0		

Table 32: DM.TRBUF definition

DM.TRBUF				0x100048	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TR_SIZE	[5:2]	4	—	Trace buffer size	RW
	Operation		Defines the size of the trace buffer in the target system's memory. The trace buffer should be disabled (by writing 0b00 to .TR_MODE) before writing this field. Value - Description 0x0: 64 Kb 0x1: 128 Kb 0x2: 256 Kb 0x3: 512 Kb 0x4: 1 Mb 0x5: 2 Mb 0x6: 4 Mb 0x7: 8 Mb 0x8: 16 Mb 0x9: 32 Mb 0xA: 64 Mb 0x[B, F]: Undefined	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[15:6]	10	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 32: DM.TRBUF definition

DM.TRBUF				0x100048	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TR_BASE	[31:16]	16	—	Trace Buffer base	RW
	Operation		When an area of target system memory is to be configured as a trace buffer, bits [31:16] of DM.TRBUF represent bits [31:16] of the physical base address of the trace buffer. The trace buffer must be aligned on a 64 Kbyte boundary. The trace buffer should be disabled (by writing 0b00 to .TR_MODE) before writing this field. Setting TR_BASE to the value of the DM (that is, to spill trace to the debug link) will lead to undefined effects. If this mode of operation is desired, it should be achieved by programming DM.TRCTL.DESTN to trace link mode.	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 32: DM.TRBUF definition

1.8.12 DM.TRPTR (trace pointer register)

This register is specified in *Table 33*. The TR_PTR field of this register allows debug software to determine the location in the trace buffer where the next trace message will be written

Trace messages vary in size, the largest sized message fits within 3 64-bit words. In order that debug software can read the trace messages backwards, the DM writes trace messages into the trace buffer at fixed 3 * 64-bit intervals.

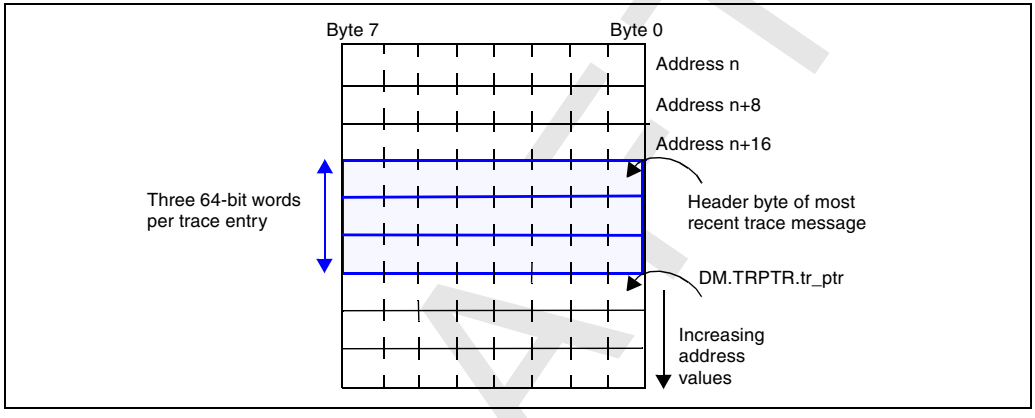

Figure 5: DTRC messages in trace buffer

When extracting trace messages, the debug software must be aware of whether the trace buffer is in “circular trace buffer” mode and if so, use modulo arithmetic on the addresses as appropriate. When the trace buffer is about to wrap around, trace messages will not be “split” between the top of the buffer and the bottom, the complete message will be written from offset 0.

DM.TRPTR				0x100050	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[2:0]	3	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
TR_PTR	[31:3]	29	3	Next message pointer	RO
	Operation		This field is used to determine the start address at which the next trace entry will be written into the trace buffer, and thus is used to determine how many entries have been written. It is automatically set to the value of DM.TRBUF.TR_BASE whenever DM.TRBUF.TR_BASE is written. When the DM generates a trace entry, it is updated to indicate the address at which the next entry will be written. The implementation is free to update this field during the process of writing the trace entry to the trace buffer. Thus the field may temporarily contain a value which is not an exact multiple of the trace entry size above DM.TRBUF.TR_BASE (the trace entry size is implementation specific - see *Section 4.1.5: DM FIFO on page 246*). The number of complete entries present in the trace buffer is determined by subtracting the value of DM.TRBUF.TR_BASE from the value in this field, dividing by the implementation-specific trace entry size, and rounding down. *Figure 5: DTRC messages in trace buffer on page 110* shows an example of a trace buffer and the pointer value.	
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		

Table 33: DM.TRPTR definition

DM.TRPTR				0x100050	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 33: DM.TRPTR definition

1.8.13 DM.FIFO_0/DM.FIFO_1/DM.FIFO_2 (FIFO port register)

The DM.FIFO_X registers allow debug software to read trace data from the debug module FIFO when the trace destination is in either “DM FIFO trace hold” mode or “circular DM FIFO” mode.

Care should be taken to ensure that trace is not being generated whilst the trace is being extracted, as this can result in the loss of trace messages.

Trace data is extracted one trace message at a time, 3 registers are used due to the maximum size of a trace entry (<=3*64 bits).

The 3 FIFO port registers are indirectly coupled to the FIFO. The mechanism for transferring the oldest trace message from the FIFO into the FIFO port registers consists of the following sequence:

- Software writes a ‘1’ into the FF_READ_REQ field of the DM.FIFO_REQ register. This initiates the transfer of the oldest trace message from the FIFO to the FIFO port registers.
- Software reads the FF_READ_ACK field of the DM.FIFO_ACK register until its value is ‘1’. *Table 34* shows all possible values of FF_READ_REQ and FF_READ_ACK.
- Trace data can now be read from the FIFO port registers. Normally, DM.FIFO_0 will be read first to determine the size of the trace message but the registers can be read in any order and can be read any number of times.

- Before requesting another trace message transfer, software should read the FF_STATUS field of the DM.TRCTL register to determine if more trace messages exist in the FIFO.

FF_READ_REQ	FF_READ_ACK	Meaning
0	0	No FIFO transfer has ever been requested.
1	0	FIFO transfer in progress.
0	1	Transfer complete. This state exists until the next transfer request.

Table 34: FIFO transfer request/acknowledge states

A suitable pseudo-code sequence to transfer and read a FIFO entry is given below:

```
if (DM.TRCTL.ff_status}
  // FIFO contains at least 1 entry so transfer it
  // request that the FIFO transfers 1 entry
  write (DM.FIFO_REQ.ff_read_req = 1)

  // wait until the FIFO has signalled the transfer operation is
complete
  while (DM.FIFO_ACK.ff_read_ack == 0) {
    donothing
  }

  // the FIFO has now transferred an entry
  read DM.FIFO_0, DM.FIFO_1, DM.FIFO_2
}
```

Note: *If a FIFO transfer is requested when the FIFO is empty, the contents of* DM.FIFO_0, DM.FIFO_1 *and* DM.FIFO_2 *are undefined.*

DM.FIFO_x				where x = {0/1/2}	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_PORT	[63:0]	64	✓	DM FIFO data port	RO
	Operation		Trace data in the debug module FIFO can be read via this port when the FIFO is in “DM FIFO trace hold” mode or “circular DM FIFO” mode.		
	When read		Returns a DM FIFO data word when the FIFO is in “DM FIFO trace hold” mode or “circular DM FIFO” mode (DM.TRCTL.DESTN = 0b00 or 0b11). When the trace message destination is the development tool or target system trace buffer, reads have no effect on the DM FIFO and return undefined data. *Section 1.9.3: DTRC messages on page 119* defines the format of trace messages.		
	When written		Ignored		
	HARD reset		Undefined		

Table 35: DM.FIFO_{*0/1/2*} definition

DM.FIFO_REQ				0x100070	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_READ_REQ	0	1	✓	DM FIFO transfer request	RW
	Operation		In “DM FIFO trace hold” mode or “circular DM FIFO” mode, this is one of the fields used to transfer data from the DM FIFO to the DM FIFO port registers.		
	When read		Undefined		
	When written		Value - Description 0: No action 1: Initiates a transfer of the oldest trace message in the DM FIFO to the DM FIFO port registers.		
	HARD reset		0		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 36: DM.FIFO_REQ definition

DM.FIFO_ACK				0x100078	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FF_READ_ACK	0	1	✓	DM FIFO transfer acknowledge	RO
	Operation		In “DM FIFO trace Hold” mode or “circular DM FIFO” mode, this is one of the fields used to transfer data from the DM FIFO to the DM FIFO port registers.		
	When read		Value - Description 0: Transfer of trace message data between the DM FIFO and the DM FIFO port registers is in progress. Software should not attempt to read the DM FIFO port registers. 1: One trace message has been transferred from the DM FIFO to the DM FIFO port registers. The DM FIFO port registers can now be read. This field will remain set until another transfer request command has been written to the DM.FIFO_REQ.FF_READ_REQ field.		
	When written		Ignored		
	HARD reset		0		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 37: DM.FIFO_ACK definition

1.8.14 DM.PC (shadow program counter register)

This register allows debug software to read the current value of the shadow PC and the shadow ASID registers both located in the debug module.

DM.PC				0x100020	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SHADOW_PC	[31:0]	32	✓	Program counter	RO
	Operation		The debug module maintains a shadow program counter which is kept in step with the program counter of the CPU core. The LSB of this field indicates the current ISA Mode (1 == SHmedia, 0 == SHcompact).		
	When read		Returns the current value of the shadow program counter. Reads do not affect the value of this counter.		
	When written		Ignored		
	HARD reset		Undefined		
SHADOW_ ASID	[39:32]	8	✓	Application space ID	RO
	Operation		The debug module maintains a shadow ASID register which is kept in step with the ASID field of the SR register in the CPU core.		
	When read		Returns the current value of the shadow ASID register. Reads do not affect the value of this register.		
	When written		Ignored		
	HARD reset		Undefined		
—	[63:40]	24	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 38: DM.PC definition

1.9 Debug protocols and interfaces

1.9.1 Endianness

The debug system is entirely little endian in respect of:

- all FIFO contents,
- all trace messages generated,
- all values encoded in DBUS transactions.

However, within DBUS transactions the correlation between mask values and addresses is dependant on the endianness of the SH-5 system. This is defined in *Section 3.6: DBUS protocol on page 226*.

1.9.2 Overall message structure

The SH-5 debug module can initiate two types of transaction over the currently-configured debug interface (SHdebug link or JTAG):

- those associated with SuperHyway bus transactions (called DBUS messages),
- trace messages from on-chip debug logic (called DTRC).

In the reverse direction, the tool can generate only DBUS messages.

The message structure is the same, regardless of the width of the SHdebug link data path or which debug interface is used. A 3-bit message type field in the first word of the message defines the message contents.

First word of message

The 3-bit message type field in the first word of each debug message can be used by the tool to determine the action required for the message:

- Process the DBUS request or response.
- Write the background trace message into FIFO memory within the tool.
- Forward the trigger trace message to tool software for immediate action.
- Update the reference time register in the tool.

When the SHdebug link is the currently-selected debug interface and SH-5 has no data to send, the debug module maintains a line idle condition on the SHdebug link.

Message type name	Message type field	Meaning
MHDR_IDLE	0b000	Line idle condition
MHDR_DBUS	0b001	DBUS message.
MHDR_DTRC_BACK	0b010	DTRC background trace message.
MHDR_DTRC_TRIG	0b011	DTRC trigger trace message.
MHDR_REF	0b100	Reference message
	0b101, 0b110 or 0b111	Not currently defined

Table 39: Message type values

1.9.3 DTRC messages

DTRC messages contain information captured by both the WPC, DM and bus analyzers.

These are stored in the debug FIFO, or sent to the tool without any CPU involvement. Such trace messages, sent by the on-chip debug module, include:

- information from each CPU watchpoint hit,
- branch trace information,
- fast printf output,
- information captured in bus analyzer bus capture buffers.

Trace message types

Watchpoints and bus analyzer bus capture buffers can generate two types of trace message:

- a trigger trace message,
- a background trace message.

The watchpoint hit information contained in these two types of message is the same; the only difference is the use of a different message header code. The use of different message header codes permits the tool to take different actions for the two types of trace message.

Each WP channel's action register allows the type of trace message generated to be defined. See *Section 1.5: Debug event actions on page 48*.

Encoding of address offsets

To minimize the sizes of trace messages being sent over the SHdebug link, program counter addresses, and bus analyzer addresses are compressed wherever possible.

However, when the trace destination is configured as "DM FIFO trace hold", "circular DM FIFO", "circular trace buffer" or "trace buffer hold" mode, addresses are always encoded as absolute values. This allows trace to be reconstructed even if some trace packets are lost, as each message can be interpreted without reference to other messages.

An encoding method is used whereby either one or two bytes are used to represent signed address offsets as either a 7-bit or a 15-bit 2's complement value. These offsets are relative to the previous address of the same type. If the address cannot be expressed as an offset, an absolute 32-bit value is encoded instead. Therefore:

- Program counter addresses are stored as effective addresses, and are encoded either as an 32-bit absolute address, or as a 7-bit or 15-bit 2's complement value relative to the previous PC address.
- Bus analyzer addresses are encoded either as absolute 32-bit addresses, or as a 7-bit or 15-bit 2's complement value relative to the previous bus analyzer address.

Address offsets are calculated as [NEW_ADDRESS - PREVIOUSLY_SENT_ADDRESS]. As shown in *Figure 5*, bit-7 of the first byte is used to indicate whether a second byte follows.

Absolute or relative encoding of an address is indicated by one of the following header fields:

- PC_ABSOLUTE
- SRC_ABSOLUTE (in branch trace messages only)
- DEST_ABSOLUTE (in branch trace messages only)
- ADDR_ABSOLUTE (in bus analyzer trace messages only)

When debug software is analyzing trace message information, it uses an absolute address as the reference for reconstructing the addresses in subsequent trace messages. The reference message (*Section 1.8.6: Timestamping and reference messages on page 93* and *Table 47 on page 133*) also contains the absolute PC and

bus analyzer addresses and provides a regular source of reference addresses to assist in address reconstruction.

Note that FPF messages contain an absolute PC value, but that they do **not** reseed the PC reference value.

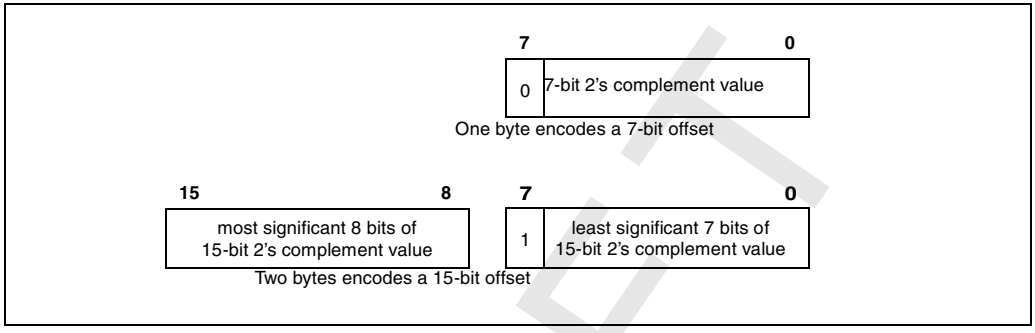

Figure 6: Encoding of address offsets

A "C" implementation of a compression decode routine is given below.

```
/* _DecodeCompressedOffset
    pre: byteStream points to signed compressed value
    post: returns decoded value
*/
int _DecodeCompressedOffset (char *byteStream)
{
  int result;

  if ((byteStream[0] & 128) == 0) {
    // its a 1 byte value
    //

    result = (byteStream[0] & 63); // extract the least sig 6 bits

    // check if its negative (include bit 7)
    if (byteStream[0] & 64) {
      result = (-64 + result);
    }

  } else {
    // its a 2 byte value
    //
```

```
    // extract the least sig 7 bits
    result = (byteStream[0] & 127);

    // additionally, extract the most sig 7 bits
    result = result | ((byteStream[1] & 127) << 7);

    // check if its negative (include bit 15)
    if (byteStream[1] & 128) {
      result = (-16384 + result);
    }
  }
  return result;
}
```

DTRC message definitions

Each trace message consists of a 16-bit header followed by a variable number of bytes depending on the message type, options enabled and the amount of compression achieved.

Common trace message fields

Common trace message fields			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	Defines the basic contents of the debug message as described in *Section : First word of message on page 118*. Only field values of 0b010 (DTRC background trace message) and 0b011 (DTRC trigger trace message) relate to trace messages.
SOURCE_MODULE	3-bits	[5:3]	Defines the on-chip source module which provides the information in the trace message[a]. Value - Description 0: WPC (CPU watchpoint controller) 1: SuperHyway bus analyzer 2-7 - Reserved for watchpoint logic in additional CPU cores or future accelerator modules.
EVENT_TYPE	5-bits	[10:6]	Defines the watchpoint channel in the source module which generated the trace message. Refer to *Table 4.1.6: Trace message header fields on page 247* for watchpoint channel numbers in the WPC and each of the bus analyzers.
OVER_STALL	1-bit	[11]	This bit has two meanings depending on whether the stall-mode field of DM.TRCTL selects CPU stall mode or discard mode. See *Table 31: DM.TRCTL definition on page 97*. In stall mode, this bit is set when the CPU was stalled for some indeterminate time prior to this trace message being generated because there was no space available in the debug module FIFO. In discard mode, this bit is set to indicate that one or more watchpoint hits before this hit were discarded because there was no space available in the capture buffer.

Table 40: Common trace message fields

Common trace message fields			
Field	**Size**	**Header bit positions**	**Description**
PC_ABSOLUTE	1-bits	[12]	Defines whether the PC field contains a 4-byte absolute address or a 1- or 2-byte relative address. A relative address is the signed offset from the most recent PC value sent in a previous trace message (of any type). Value - Description 0: Relative address offset 1: Absolute 4-byte address
OTHER	3-bits	[15:13]	Specific for each watchpoint channel type.
TIMESTAMP	1-byte	N/A	This optional field occurs in the trace message when the WP channel's action includes enable_trace_timestamp == 1 (see *Section 1.5: Debug event actions on page 48*). This one-byte value specifies a number of timer increments since the last Reference trace message (see *Table 47*).
ASID	1-byte	N/A	This optional field occurs whenever the watchpoint channel if setup to match any ASID. When the WP channel's pre condition includes asid_enable ==1 (see *Section 1.6: WP channel matching on page 66*), then the ASID field does not appear in the trace messages.
PC	1, 2 or 4 bytes	N/A	If PC_Absolute is '0', this field is a 1-byte or 2-byte compressed address as a signed offset from the most recent PC value sent in a previous trace message (of any type). If PC_Absolute is '1', this field consists of the 4-byte absolute value of the PC at the point of the WP trigger.

Table 40: Common trace message fields

a. Although the FPF function is implemented in the DM, its header denotes as appearing from the WPC (as there is no specified DM header).

Specific trace messages

IA watchpoint trace message (3-bytes minimum, 8-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b010 or 0b011
SOURCE_MODULE	3-bits	[5:3]	0 (WPC)
EVENT_TYPE	5-bits	[10:6]	0x00 through 0x03 (see *Table 89: SH-5 evaluation device trace message codes on page 247*).
OVER_STALL	1-bit	[11]	
PC_ABSOLUTE	1-bits	[12]	
Reserved	3-bits	[15:13]	Not used
TIMESTAMP	0 or 1 byte	N/A	
ASID	0 or 1 byte	N/A	
PC	1, 2 or 4 bytes	N/A	

Table 41: IA watchpoint trace message

OA watchpoint trace message (7-bytes minimum, 16-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b010 or 0b011
SOURCE_MODULE	3-bits	[5:3]	0 (WPC)
EVENT_TYPE	5-bits	[10:6]	0x04 through 0x05 (see *Table 83: SH-5 evaluation device WP channels on page 241*).
OVER_STALL	1-bit	[11]	
PC_ABSOLUTE	1-bits	[12]	

Table 42: OA watchpoint trace message

OA watchpoint trace message (7-bytes minimum, 16-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
DATA_FIELD_SIZE	3-bits	[15:13]	Defines how much data was stored to memory by the triggering instruction. Value - Description 0b000: The instruction which hit the watchpoint did not write to a memory location. 0b001: Undefined 0b010: Undefined. 0b011: Undefined. 0b100: 1 byte. The instruction which hit the watchpoint did a 1 byte write to a memory location. 0b101: 2 byte write (as above) 0b110: 4 byte write (as above) 0b111: 8 byte write (as above)
TIMESTAMP	0 or 1 byte	N/A	
ASID	0 or 1 byte	N/A	
PC	1, 2 or 4 bytes	N/A	

Table 42: OA watchpoint trace message

OA watchpoint trace message (7-bytes minimum, 16-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
DATA_ADDRESS	4, 5, 6 or 8 bytes	N/A	If .DATA_FIELD_SIZE indicates a 0 byte store, this field is 4 bytes in length. If .DATA_FIELD_SIZE indicates a 1 byte store, this field is 5 bytes in length. Bits [39:32] correspond to the 8-bits of data stored to memory. If .DATA_FIELD_SIZE indicates a 2 byte store, this field is 6 bytes in length. Bits [47:32] correspond to the 16-bits of data stored to memory. If .DATA_FIELD_SIZE indicates a 4 byte store, this field is 8 bytes in length. Bits [63:32] correspond to the 32-bits of data stored to memory. If .DATA_FIELD_SIZE indicates an 8 byte store, this field is 8 bytes in length. Bits [63:32] correspond to bits [31:0] of the 64 bits of data stored to memory. The meaning of bits [31:0] is dependant on the setting of WPC.ADDR_IN_TRACE: If .MUX_ADDR == 1, bits[31:0] correspond to the operand address at which the triggering instruction stored data to memory. If .MUX_ADDR == 0, bits[31:0] correspond to bits [63:32] of the 64 bits of data stored to memory by the triggering instruction. They only contain valid information if .DATA_FIELD_SIZE indicates an 8 byte store. For the store instructions that support misaligned addresses, the contents of this field are described in *Handling of misaligned store instructions on page 134*

Table 42: OA watchpoint trace message

IV watchpoint trace message (7-bytes minimum, 16-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b010 or 0b011
SOURCE_MODULE	3-bits	[5:3]	0 (WPC)
EVENT_TYPE	5-bits	[10:6]	0x06 through 0x07 (see *Table 89: SH-5 evaluation device trace message codes on page 247*).
OVER_STALL	1-bit	[11]	
PC_ABSOLUTE	1-bit	[12]	
DATA_FIELD_SIZE	3-bits	[15:13]	Defines how much data was stored to memory by the triggering instruction. Value - Description 0b000: The instruction which hit the watchpoint did not write to a memory location 0b001: Undefined 0b010: Undefined 0b011: Undefined 0b100: 1 byte. The instruction which hit the watchpoint did a 1 byte write to a memory location 0b101: 2 byte write (as above) 0b110: 4 byte write (as above) 0b111: 8 byte write (as above)
TIMESTAMP	0 or 1 byte	N/A	
ASID	0 or 1 byte	N/A	
PC	1, 2 or 4 bytes	N/A	

Table 43: IV watchpoint trace message

IV watchpoint trace message (7-bytes minimum, 16-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
DATA_ADDRESS	4, 5, 6 or 8 bytes	N/A	Identical to the DATA_ADDRESS field of OA trace messages (see *Table 42: OA watchpoint trace message on page 125*). However, as IV channels can trigger for instructions which do not write to memory a further clarification is required - If the triggering instruction did not write to memory, bits[31:0] contain undefined data.

Table 43: IV watchpoint trace message

BR watchpoint trace message (4-bytes minimum, 12-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b010 (Always defined as a background trace message)
SOURCE_MODULE	3-bits	[5:3]	0 (WPC)
EVENT_TYPE	5-bits	[10:6]	0x08 (see *Table 89: SH-5 evaluation device trace message codes on page 247*).
OVER_STALL	1-bit	[11]	
DEST_ABSOLUTE	1-bits	[12]	
SRC_ABSOLUTE	1-bits	[13]	
RESERVED	2-bits	[15:14]	
TIMESTAMP	0 or 1 byte	N/A	
ASID	0 or 1 byte	N/A	

Table 44: BR watchpoint trace message

BR watchpoint trace message (4-bytes minimum, 12-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
DESTN_ADDRESS	1, 2 or 4 bytes	N/A	If DEST_ABSOLUTE is '0', this field is a 1-byte or 2-byte compressed address as a signed offset from the most recent PC value sent in a previous trace message (of any type). If DEST_ABSOLUTE is '1', this field consists of the 4-byte absolute value of the destination address for the branch.
SOURCE_ADDRESS	1, 2 or 4 bytes	N/A	If SRC_ABSOLUTE is '0', this field is a 1-byte or 2-byte compressed address as a signed offset from the most recent PC value sent in a previous trace message (of any type). If SRC_ABSOLUTE is '1', this field consists of the 4-byte absolute value of the source address of the branch. *Note: For SHcompact delayed branches, the source address is that of the branch delay slot, not of the branching instruction.*

Table 44: BR watchpoint trace message

FPF watchpoint trace message (15-bytes)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b011 (Always defined as a trigger trace message)
SOURCE_MODULE	3-bits	[5:3]	0 (WPC). The actual source module is DM, but there is no value reserved for this, so the WPC value is used.
EVENT_TYPE	5-bits	[10:6]	0x09 (see *Table 89: SH-5 evaluation device trace message codes on page 247*).

Table 45: FPF Watchpoint Trace Message

FPF watchpoint trace message (15-bytes)			
Field	**Size**	**Header bit positions**	**Description**
OVER_STALL	1-bit	[11]	
RESERVED	4-bits	[15:12]	
ASID	1 byte	N/A	The ASID is always included in the trace message.
PC	4 bytes	N/A	A full PC address is always sent. Note that this value does not reseed the PC reference value.
FPF_DATA	8 bytes	N/A	The data written to the fast printf register is always a 64 bit value.

Table 45: FPF Watchpoint Trace Message

PL Watchpoint trace message (7-bytes minimum, 20-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b010 or 0b011
SOURCE_MODULE	3-bits	[5:3]	1 (SuperHyway bus analyzer)
EVENT_TYPE	5-bits	[10:6]	0x00 through 0x01 (see *Table 89: SH-5 evaluation device trace message codes on page 247*).
OVER_STALL	1-bit	[11]	Set to indicate that one or more trace messages before this one were discarded because there was no space available in the debug module FIFO.

Table 46: PL Watchpoint trace message

PL Watchpoint trace message (7-bytes minimum, 20-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
MATCH_LOSS	1-bits	[12]	Set to indicate that some SuperHyway cells/tokens which should have been captured were lost because the hit occurred when either: - The previous, or current) captured cell/token was being transferred to the DM. - The bus capture buffer was frozen (by the PL channel raising a debug interrupt). See *Debug interrupt actions on page 184*). No watchpoint hit is registered for these additional bus transactions.
ADDR_ABSOLUTE	1-bits	[13]	
RESERVED	2-bits	[15:14]	
TIMESTAMP	0 or 1 byte	N/A	
SOURCE	1-byte	N/A	
DESTINATION	1-byte	N/A	
OPCODE	1-byte	N/A	
TRANSACTION ID	1-byte	N/A	
DATA_MASK	0 or 1 byte	N/A	When the opcode corresponds to a **Flush** (0x18), **Purge** (0x8), **Success** (0x80) or **Failure** (0x81) this field is 0 bytes in length (that is, it is not encoded) For other opcodes it is encoded as a 1 byte field and contains the mask value from the captured bus packet.

Table 46: PL Watchpoint trace message

PL Watchpoint trace message (7-bytes minimum, 20-bytes maximum)			
Field	**Size**	**Header bit positions**	**Description**
ADDRESS	0, 1, 2 or 4 bytes	N/A	If Opcode is **Success** (0x80) or **Failure** (0x81) this field is 0 bytes in length (that is, an address is not encoded). Otherwise: If ADDR_ABSOLUTE is '0', this field is a 1-byte or 2-byte compressed address as a signed offset from the bus transaction address calculated for the previous trace message for this watchpoint. If ADDR_ABSOLUTE is '1', this field consists of the 4-byte absolute value of the transaction address.
TRANSACTION_ DATA	0 or 8 bytes	N/A	When the opcode is **Load8** (0x31), **Load16** (0x41), **Load32** (0x51) or **Failure** (0x81) this field is 0 bytes in length (that is, it is not encoded). For all other opcodes this field is 8 bytes in length, and corresponds either to valid data from the captured bus packet, or undefined data when the captured bus packet contained no data. The opcode field can be used to determine whether the bus packet was a request which contained data as described above. If the opcode shows the bus packet was a **Success** (0x80) response, software will need knowledge of the corresponding request to determine whether valid data is encoded here.

Table 46: PL Watchpoint trace message

Reference message (14-bytes)			
Field	**Size**	**Header bit positions**	**Description**
MESSAGE_TYPE	3-bits	[2:0]	0b100
RESERVED	5-bits	[7:3]	

Table 47: Reference message

Reference message (14-bytes)			
Field	**Size**	**Header bit positions**	**Description**
TIME_VALUE	5-bytes	N/A	The value of the 40-bit timestamp counter in the debug module.
PC_ADDRESS	4-bytes		The absolute 4-byte address of the shadow program counter at the time this message is generated. This address becomes the new reference PC value and the relative address in a trace message which follows will be based on this value.
BA_ADDRESS	4-bytes		The absolute 4-byte reference address associated with the SuperHyway bus analyzer. This value becomes the new bus analyzer reference address and the relative address in a bus analyzer trace message which follows will be based on this value.

Table 47: Reference message

Handling of misaligned store instructions

This section describes the contents of the DATA_FIELD_SIZE and DATA_ADDRESS fields in OA and IV trace messages that result from STHI.Q, STHI.L, STLO.Q or STLO.L instructions. The field contents are shown in *Table 49*. The 8 bytes within the register containing the data for the store are identified by A, B, C, D, E, F, G, H. The association between these labels and the bytes in the register is shown in *Table 48*.

	Data source register for store (referred to as ***Ry*** in the instruction definitions)							
	MSB							LSB
	Bits [63:56]	Bits [55:48]	Bits [47:40]	Bits [39:32]	Bits [31:24]	Bits [23:16]	Bits [15:8]	Bits [7:0]
Byte name in *Table 49*	H	G	F	E	D	C	B	A

Table 48: Byte naming convention used in table *Table 49*

Endian mode	Instruction	EA[2:0]	# bytes stored	ADDR_IN_TRACE	DATA_FIELD_SIZE	Data_address							
						Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
						Bits [63:56]	Bits [55:48]	Bits [47:40]	Bits [39:32]	Bits [31:24]	Bits [23:16]	Bits [15:8]	Bits [7:0]
Little	STHI.Q	0..3	0..3	0	0b110	H	G	F	E	?	?	?	?
				1		H	G	F	E	Operand address			
		4..7	4..7	0	0b111	D	C	B	A	H	G	F	E
				1		D	C	B	A	Operand address			
	STLO.Q	0..3	7..4	0	0b111	D	C	B	A	H	G	F	E
				1		D	C	B	A	Operand address			
		4..7	3..0	0	0b110	D	C	B	A	?	?	?	?
				1		D	C	B	A	Operand address			
Big	STHI.Q	0..3	0..3	0	0b110	D	C	B	A	?	?	?	?
				1		D	C	B	A	Operand address			
		4..7	4..7	0	0b111	D	C	B	A	H	G	F	E
				1		D	C	B	A	Operand address			
	STLO.Q	0..3	7..4	0	0b111	D	C	B	A	H	G	F	E
				1		D	C	B	A	Operand address			
		4..7	3..0	0	0b110	H	G	F	E	?	?	?	?
				1		H	G	F	E	Operand address			
Any	STHI.L	0..3	0..3	0	0b110	D	C	B	A	?	?	?	?
				1		D	C	B	A	Operand address			
Any	STLO.L	0..3	3..0	0	0b110	D	C	B	A	?	?	?	?
				1		D	C	B	A	Operand address			

Table 49: DATA_FIELD_SIZE and DATA_ADDRESS in IV/OA traces for misaligned store instructions

Where an entry in *Table 49* is shown as ?, it means the contents of that byte are unspecified.

The trace data is usually the unshifted register contents, except when 4 or fewer bytes are stored from the upper half of the register, in which case a 32-bit right shift is applied.

1.9.4 DBUS messages

DBUS transactions occur when the CPU or any other bus master module sends a **Load**, **Store** or **Swap** request over the SuperHyway bus to the target address space assigned to the tool. This is the mechanism by which the CPU boots via the currently-selected debug interface and how a debug monitor running on the CPU communicates with the tool.

The tool can also initiate data and control transactions in the reverse direction over the currently-selected debug interface. Such data transactions include:

- Reads and writes to any memory-mapped on-chip resources, without affecting the CPU. The on-chip module's registers appear in the memory map, thus this mechanism allows the resource's state to be inspected, altered and controlled.
- Control of the CPU. As defined in *Section 1.3: CPU control on page 30*.

1.10 WP channel type BRK

Break watchpoints trigger whenever:

- BRK-BRK

  The BRK instruction is executed. The trigger occurs prior to the instruction executing.

- BRK-STEP:

  An instruction is completed in single step mode. The trigger occurs after the instruction executes.

- BRK-INT:

  A debug interrupt was forced by writing to DM.FORCE_DEBUGINT.

All BRK channel triggers (BRK-BRK, BRK-STEP and BRK-DEBUGINT) are unaffected by SR.WATCH.

The BRK channel is not a true WP channel - it has no generic PRE and ACTION registers, and instead has some implicit behavior. BRK is manifested as a WP channel in order to make its exception state available in a uniform manner (see *Section 1.7: Reset, panic and debug events on page 73*).

1.10.1 Match registers

There are no associated PRE or MATCH registers as BRK watchpoints cannot be filtered.

There is no associated ACTION register as the BRK watchpoint implicitly takes a debug exception.

1.10.2 Event specifics

Source CPU

Reason: Execution of a BRK instruction, execution of a single-stepped instruction, or a forced debug interrupt.

Implicit action:

Exception: Standard event handling sequence is followed.

- If due to a BRK instruction execution:

  If SR.BL == '0', the RESVEC/DBRVEC offset is 0x100 (to denote a synchronous debug event of type BREAK).

  If SR.BL == '1', the RESVEC/DBRVEC offset is 0x0 (to denote a PANIC event).

  EXPEVT is set to 0x940 to denote a BREAK exception.

  The SPC register contains the address at which the BRK instruction was placed.

- If due to a single step:

  If SR.BL == '0', the RESVEC/DBRVEC offset is 0x100 (to denote a synchronous debug event of type DEBUGSS).

  If SR.BL == '1', the RESVEC/DBRVEC offset is 0x0 (to denote a PANIC event).

  EXPEVT is set to 0x980 to denote a DEBUGSS exception.

  The SPC register contains the PC value of the next instruction to be executed.

  In SHcompact mode, the delayed branch and delay slot instructions are executed indivisibly - a single step will not be raised between them. Upon occurrence of the step exception, SPC will refer to the delayed branch and the next single step will occur for the first instruction which executes after the delay slot instruction (depending on whether the branch is taken or not taken).

- If due to a forced debug interrupt:

  The RESVEC/DBRVEC offset is 0x200 (to denote a DEBUGINT event).

  DM.EXP_CAUSE.FORCED_DEBUG_INTERRUPT is set to 1.

1.11 WP channel type IA

Instruction address watchpoints trigger whenever an instruction fetch occurs within the defined range.

The trigger occurs only for instruction fetches which will result in an instruction executing (that is, speculative fetches do not trigger).

If SR.WATCH == 0, IA channels will not trigger.

The trigger occurs prior to the instruction executing.

1.11.1 Match registers

Two registers are used to define a start and end address.

WPC.WP_IA*x*_MATCH_START				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDRESS	[31:0]	32	—	Start address	RW
	Operation		Defines the start address for the instruction address range. When the MMU is enabled, this contains an effective address. When the MMU is disabled, this contains a physical address. The least significant bit (bit 0) should always be written as zero. The effect of writing 1 to bit 0 is implementation-defined.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 50: WPC.WP_IA*x*_MATCH_START register definition

WPC.WP_IA*x*_MATCH_END				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDRESS	[31:0]	32	—	End address	RW
	Operation		Defines the end address for the instruction address range. When the MMU is enabled, this contains an effective address. When the MMU is disabled, this contains a physical address. The least significant bit (bit 0) should always be written as zero. The effect of writing 1 to bit 0 is implementation-defined. The address comparison is performed using the start address of a SHmedia or SHcompact instruction (shown as *Iaddr* below). The comparison is inclusive of the match start address, but not of the match end address: *Iaddr* >= start && *Iaddr*< end		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 51: WPC.WP_IA*x*_MATCH_END register definition

1.11.2 Address comparison

The address comparison is performed using the effective address supplied to the instruction (known as Aeffective). When the MMU is enabled, this is an effective address. When the MMU is disabled, this is a physical address.

The comparisons are unsigned. If the start/end address range straddles the unimplemented part of the effective address space (that is, it contains addresses in the range $[2^{(neff - 1)}, 2^{(64 - neff - 1)})$ where neff is the number of implemented effective address bits, the behavior is architecturally undefined. If this type of 'stradding' behavior is required, two IA channels must be programmed, each one limited to addresses in a single region of the implemented address space.

1.11.3 SHcompact behavior

SH-5 may implement SHcompact instruction execution by actual execution of a sequence of SHmedia instructions. In such implementations, IA watchpoint matches are made against the address of the single SHcompact instruction (as the sequence of SHmedia instructions generated have no PC address).

1.11.4 Event specifics

Source CPU

Reason: Fetch of an instruction at an address which is within an enabled IA watchpoint.

Undefined behavior

The following operations of the IA*x* registers are undefined:

- Writing to WPC.WP_IAX_{PRE/MATCH/ACTION}, DM.WP_IAX_{PRE/ACTION} when the WP channel is enabled.

Supported fields in WPC.WP_IA*x*_PRE:

`BASIC_ENABLE, ASID_ENABLE, ASID_VALUE, ISAMODE_ENABLE, SR_MD_ENABLE, ECOUNT_ENABLE, ECOUNT_ID, CHAIN_ENABLE, CHAIN_ID`

Supported fields in DM.WP_IA*x*_PRE:

ASID_ENABLE

Supported fields in WPC.WP_IA*x*_ACTION:

ACTION_EXCEPTION**:** Standard event handling sequence is followed.

If SR.BL == '0', the RESVEC/DBRVEC offset is 0x100 (to denote a synchronous debug event of type DEBUGIA). Otherwise, if SR.BL == '1', the RESVEC/DBRVEC offset is 0x0 (to denote a PANIC event).

EXPEVT is set to 0x900 to denote a DEBUGIA exception.

In addition, the effective instruction address is placed into the SPC register.

ACTION_ECOUNT supported.

ACTION_CHAIN_ALTER/CHAIN_ID supported.

ACTION_PCOUNT/ACTION_RESET_ALL_PCOUNT/PCOUNT_ID supported

Supported fields in DM._WP_IA*x*_ACTION:

ACTION_TRACE/TRACE_TYPE/ENABLE_TRACE_TIMESTAMP**:** See *Table 41: IA watchpoint trace message on page 125*

ACTION_TRIG_OUT**:** Supported

ACTION_CHAIN_ALTER/CHAIN_ID Supported

FREEZE_EN Not supported

1.12 WP channel type OA

Operand watchpoints trigger whenever an instruction performs a memory write within the defined range.

The trigger occurs prior to the instruction executing (that is, prior to the write occurring).

If SR.WATCH == 0, OA channels will not trigger.

Two registers are in the WPC are used to define a start and end address, two registers in the DM are used to define a data value and data mask.

1.12.1 Match registers

WPC.WP_OA*x*_MATCH_START				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[4:0]	5	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ADDRESS	[31:5]	27	—	Start address	RW
	Operation		Defines the start address for the operand address range. See *Section 1.12.2: on page 145*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 52: WPC.WP_OA*x*_MATCH_START register definition

WPC.WP_OA*x*_MATCH_START				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 52: WPC.WP_OA*x*_MATCH_START register definition

WPC.WP_OA*x*_MATCH_END				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[4:0]	5	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ADDRESS	[31:5]	27	—	End address	RW
	Operation		Defines the end address for the operand address range. See *Section 1.12.2: on page 145*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 53: WPC.WP_OA*x*_MATCH_END register definition

WPC.WP_OA*x*_MATCH_END				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 53: WPC.WP_OA*x*_MATCH_END register definition

1.12.2 Address comparison

The address comparison is performed using the effective address supplied to the instruction (known as Aeffective). When the MMU is enabled this is an effective address. When the MMU is disabled, this is a physical address.

The comparisons are unsigned. If the start/end address range straddles the unimplemented part of the effective address space (that is, it contains addresses in the range $[2^{(neff - 1)}, 2^{(64 - neff - 1)})$ where neff is the number of implemented effective address bits, the behavior is architecturally undefined. If this type of 'stradding' behavior is required, two OA channels must be programmed, each one limited to addresses in a single region of the implemented address space. OA channels will never match for store addresses that lie in the unimplemented part of the effective address space. WADDERR exceptions will always be raised for such cases, never DEBUGOA exceptions.

The following instructions can cause OA watchpoint triggers:

SHmedia

- An ST, FST, SWAP, STLO, ALLOCO or OCBI instruction.

  The instruction will access memory from Aeffective upwards (for example, mem[Aeffective +0], mem[Aeffective + 1]).

  Oaddr corresponds to Aeffective with the lower 5 bits masked out.

- A STHI instruction.

  The instruction will access memory from Aeffective downwards (for example, mem[Aeffective - 0], mem[Aeffective - 1]).

  Oaddr corresponds to Aeffective with the lower 5 bits masked out.

SHcompact

- All SHcompact store instructions which use the @ addressing mode.

  The instruction will access memory from *Aeffective* upwards (for example, mem[Aeffective +0], mem[Aeffective + 1]).

  Oaddr corresponds to Aeffective with the lower 5 bits masked out.

The start and end address values (and thus the address comparator) are cache-block size aligned. Thus they ignore the lower 5 bits of the address, and can only be set at 32-byte aligned addresses. This ensures that the OA comparator will not “miss” accesses which would straddle narrower address settings, it also ensures that the relevant caching instructions can correctly trigger the OA channels.

The comparison is inclusive of the match start address and the match end address:

Oaddr = (Aeffective & 0x1F)

addressMatch = ((Oaddr >= start) && (Oaddr <= end))

1.12.3 Data match registers

The DM provides a single data value/data mask comparator (controlled by additional DM.OA_MATCH_* registers) which is used to provide additional filtering of OA channels.

The standard ACTION_EXCEPTION, ACTION_ECOUNT, ACTION_CHAIN_ALTER and PRE_CHAIN_CLEAR (as specified by WPC.WP_OAX_ACTION) apply as normal and are totally unaffected by the programming of these additional DM.OA_MATCH_* registers.

The debug module's data value/data mask comparator can be combined with one, or several of the standard OA channels. This allows an additional set of actions to be performed:

- Trace data can be made conditional on the data value written matching a specified value/mask.

  This is determined by the oa_match field of DM.WP_OAX_ACTION.

- A debug interrupt can be generated, which is triggered whenever a specified OA channel triggers, and the DM data value/mask comparison also triggers.

  This is determined by the action_interrupt field of DM.WP_OAX_ACTION but differs from the standard DEBUGOA exception (specified by the action_exception field of WPC.WP_OAX_ACTION) as follows:

  - It generates an interrupt rather than an exception.

    Interrupts are asynchronous to the instruction stream, and thus do not occur precisely;

  - As it is a DEBUGINT event, it uses offset 0x200 to RESVEC/DBRVEC;

  - The TEA register does not contain the data address.

  - The interrupt is distinguished from other causes of debug interrupt, by DM.EXP_CAUSE.OA_MATCH_INTERRUPT == 1.

Note: The data value / mask comparison is performed on the data passed between the WPC and DM. This is controlled by the WPC.ADDR_IN_TRACE *register (see Section 1.5.1: WPC.ADDR_IN_TRACE register definition on page 65). When programmed to transfer data address, and 32 bits of data,* DM.OA_MATCH_DATAMASK *should be setup to only match on the lower 32 bits of data written by triggering instructions. When an OA channel hit occurs on a misaligned store instruction (*STHI.Q, STLO.Q, STHI.L *or* STLO.L*), the 64 bits of data available for comparison are the same as the data that would appear in a trace message. The composition of the 64 bits of data is described in Section : Handling of misaligned store instructions on page 134.*

DM.OA_MATCH_DATAVAL:

DM.OA_MATCH_DATAVAL				0x100030	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DATA	[63:0]	64	—	Data value	RW
	Operation		Defines a data value which is combined with the data mask specified in DM.OA_MATCH_DATAMASK.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 54: DM.OA_MATCH_DATAVAL register definition

DM.OA_MATCH_DATAMASK:

DM.OA_MATCH_DATAMASK				0x100038	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MASK	[63:0]	64	—	Data mask	RW
	Operation		Defines a data mask. A value of ‘1’ in a bit position causes the data comparator to ignore the corresponding bit of the DM.OA_MATCH_DATAVAL field. A value of ‘0’ in a bit position makes the comparison significant when the bit is valid for the size of data written by the triggering instruction. When the bit value is ‘0’ and the data bit is not valid for the size of data written by the triggering instruction, the data comparator ignores the corresponding bit in the comparison.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 55: DM.OA_MATCH_DATAMASK register definition

1.12.4 SHcompact behavior

The SH-5 may implement SHcompact instruction execution by actual execution of a sequence of SHmedia instructions. In such implementations, OA watchpoint matches are made against this sequence of SHmedia instructions. At most a single OA watchpoint will trigger for this sequence of instructions.

1.12.5 Interrupt action

The OA data match function occurs at the output of the capture buffer. If the OA watchpoint action includes generating an interrupt when a data match occurs, this action may be affected by the stall/discard mode:

- In stall mode, the generation of the interrupt will be delayed when the capture buffer fills and the processor stalls.
- In discard mode, watchpoint hits will be discarded when the capture buffer fills. This means that an expected OA data match interrupt may not occur. The user must use caution when enabling OA data match interrupts with discard mode selected.

1.12.6 Event specifics

Source CPU

Reason: Execution of an instruction which will write data at an address which is within an enabled OA watchpoint.

Undefined behavior

The operation of the OA*x* registers is undefined if a write is made to WPC.WP_OAX_{PRE/MATCH/ACTION}_* or DM.WP_OAX_{PRE/MATCH/ACTION} when the WP channel is enabled.

Supported fields in WPC.WP_OA*x*_PRE:

BASIC_ENABLE, ASID_ENABLE, ASID_VALUE, ISAMODE_ENABLE, SR_MD_ENABLE, ECOUNT_ENABLE, ECOUNT_ID, CHAIN_ENABLE, CHAIN_ID

Supported fields in DM.WP_OA*x*_PRE:

ASID_ENABLE

Fields in WPC.WP_OA*x*_ACTION:

ACTION_EXCEPTION**:** Standard event handling sequence is followed. If SR.BL == '0', the RESVEC/DBRVEC offset is 0x100 (to denote a synchronous debug event of type DEBUGOA). Otherwise, if SR.BL == '1', the RESVEC/DBRVEC offset is 0x0 (to denote a PANIC event).

EXPEVT is set to 0x960 to denote a DEBUGOA exception.

In addition, the effective operand address, is put into the TEA register.

ACTION_ECOUNT Supported.

ACTION_CHAIN_ALTER/CHAIN_ID Supported.

ACTION_PCOUNT/ACTION_RESET_ALL_PCOUNT/PCOUNT_ID**:** Supported

Fields in DM.WP_OA*x*_ACTION:

OA_MATCH**:** Supported

ACTION_INTERRUPT**:** Standard event handling sequence is followed. The RESVEC/DBRVEC offset is 0x200 (to denote a DEBUGINT event).

EXPEVT is not set as this is a debug interrupt, not an exception.

DM.EXP_CAUSE.OA_MATCH_INTERRUPT is set to '1'.

ACTION_TRACE/TRACE_TYPE/ENABLE_TRACE_TIMESTAMP**:**

The data included in trace messages is the same as that used for comparison with the DM.OA_MATCH_* registers. See *Section 1.12.3: Data match registers on page 147* and *Table 42: OA watchpoint trace message on page 125*.

The contents of OA trace messages is controlled by the WPC.ADDR_IN_TRACE register (see *Section 1.5.1: on page 65*).

OA trace messages include a field (DATA_FIELD_SIZE) which indicates the size of the data written (see *Table 42: OA watchpoint trace message on page 125*).

This field specifies the amount of data written as being either 0, 1, 2, 4 or 8 bytes. It also determines which bits of the data are included in the data comparison, valid bits are compared and invalid bits are ignored.

Normally the value encoded corresponds directly to the amount of data written by the triggering instruction.

However, the STHI and STLO instructions are an exception, they can store 3, 5, 6 or 7 bytes. In these cases the value denoted by the DATA_FIELD_SIZE is the actual value rounded up to either 4 or 8. The data included in the trace message for these instructions is defined in *Table 49: DATA_FIELD_SIZE and DATA_ADDRESS in IV/OA traces for misaligned store instructions on page 135*

OCBI and ALLOCO instructions store no data, thus they encode DATA_FIELD_SIZE to show 0 bytes were written, and always trigger irrespective of how the data mask/value comparator is setup.

ACTION_TRIG_OUT**:** Supported

ACTION_CHAIN_ALTER/CHAIN_ID**:** Supported

FREEZE_EN Not supported

1.13 WP channel type IV

Instruction value watchpoints trigger when an instruction is executed whose bit pattern matches an instruction value and mask. The mask field allows isolation of specific fields of the instruction (such as opcode, register fields).

The trigger occurs prior to the instruction executing.

If SR.WATCH == 0, IV channels will not trigger.

One register is used to define an instruction bit pattern and instruction mask.

1.13.1 Match registers

WPC.WP_IV*x*_MATCH_VALUE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IVALUE	[31:0]	32	—	Instruction value	RW
	Operation		Defines an instruction value		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 56: WPC.WP_IV*x*_MATCH_VALUE register definition

WPC.WP_IV*x*_MATCH_MASK				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IMASK	[31:0]	32	—	Instruction mask	RW
	Operation		Defines an instruction mask. A value of ‘1’ in a bit position causes the watchpoint comparator to ignore the corresponding bit of the IVALUE field, whereas a value of ‘0’ in a bit position makes the comparison significant.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 57: WPC.WP_IV*x*_MATCH_MASK register definition

WPC.WP_IV*x*_MATCH_MASK				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:32]	32	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 57: WPC.WP_IV*x*_MATCH_MASK register definition

1.13.2 SHcompact mode

IV watchpoints are not supported for SHcompact instructions and will never trigger when the SH-5 is operating in SHcompact mode.

1.13.3 Event specifics

Source CPU

Reason: Execution of an instruction whose bit pattern matches a defined bit pattern, when used in conjunction with a mask value.

Undefined behavior

The operation of the IV*x* registers is undefined if a write is made to WPC.WP_IVX_{PRE/MATCH/ACTION}_* or to DM.WP_IVX_{PRE/ACTION} when the WP channel is enabled.

Supported fields in WPC.WP_IV*x*_PRE:

BASIC_ENABLE, ASID_ENABLE, ASID_VALUE, SR_MD_ENABLE, ECOUNT_ENABLE, ECOUNT_ID, CHAIN_ENABLE, CHAIN_ID.

ISAMODE_ENABLE Not supported - IV watchpoints only match for SHmedia instructions.

Supported fields in DM.WP_IVx_PRE:

ASID_ENABLE, CHAIN_ENABLE, CHAIN_ID

Fields in WPC.WP_IVx_ACTION:

ACTION_EXCEPTION**:** Standard event handling sequence is followed. If SR.BL == '0', the RESVEC/DBRVEC offset is 0x100 (to denote a synchronous debug event of type DEBUGIV). Otherwise, if SR.BL == '1', the RESVEC/DBRVEC offset is 0x0 (to denote a PANIC event).

EXPEVT is set to 0x920 to denote a DEBUGIV exception.

In addition, the effective instruction address is placed into the TEA register.

ACTION_ECOUNT/ECOUNT_ID Supported.

ACTION_CHAIN_ALTER/CHAIN_ID Supported

ACTION_PCOUNT/ACTION_RESET_ALL_PCOUNT/PCOUNT_ID**:** Supported

Fields in DM.WP_IVx_ACTION:

ACTION_TRACE/TRACE_TYPE/ENABLE_TRACE_TIMESTAMP See *Table 43: IV watchpoint trace message on page 128*.

The contents of IV trace messages is controlled by the WPC.ADDR_IN_TRACE register (see *Section 1.5.1: on page 65*).

IV trace messages include a field (DATA_FIELD_SIZE) which indicates the size of the data written (see *Table 43: IV watchpoint trace message on page 128*).

This field specifies the amount of data written as being either 0, 1, 2, 4 or 8 bytes.

Normally the value encoded corresponds directly to the amount of data written by the triggering instruction.

However, the STHI and STLO instructions are an exception, they can store 3, 5, 6 or 7 bytes. In these cases the value denoted by the DATA_FIELD_SIZE is the actual value rounded up to either 4 or 8. The data included in the trace message for these instructions is defined in *Table 49: DATA_FIELD_SIZE and DATA_ADDRESS in IV/OA traces for misaligned store instructions on page 135*

ALLOCO and OCBI instructions store no data, thus they encode DATA_FIELD_SIZE to show 0 bytes were written.

Note: If the triggering instruction stores 8 bytes of data, and an OA watchpoint channel also triggers for the same instruction, the contents of the DESTINATION_DATA *field of the trace message is as determined by the* WPC.ADDR_IN_TRACE *register (see Section 1.5.1 on page 65).*

ACTION_TRIG_OUT**:** Supported.

ACTION_CHAIN_ALTER/CHAIN_ID**:** Supported

ACTION_TRIG_OUT**:** Supported

FREEZE_EN Not supported

1.14 WP channel type BR

Branch trace watchpoints trigger whenever an non-sequential control flow occurs.

For BR channels to trigger on unconditional or conditional branches or on event launches, SR.WATCH must be 1 at the branch destination. For these types of branch, it means SR.WATCH must be 1 before the branch, and that the BR channel can never trigger on the launch of a debug event.

For BR channels to trigger on an RTE, SR.WATCH must be 1 both before and after the RTE instruction. This conveniently avoids BR channel triggers on return from debug event handlers, mirroring the lack of triggers on their launches.

The trigger occurs immediately after the completion of the branching instruction or action.

One register is used to define the type of branch to trace.

1.14.1 Branch filter register

The branch channel register, DM.WP_BR_FILTER, contains all PRE, MATCH and ACTION fields associated with branch tracing.

There are no PRE or ACTION registers associated with branch tracing. The ACTION is implicitly to generate a trace message. Rather than having an ACTION register, the ENABLE_TRACE_TIMESTAMP and ACTION_TRIG_OUT bits normally present in the ACTION register are held in the DM.WP_BR_FILTER register.

DM.WP_BR_FILTER				0x100028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BASIC_ENABLE	0	1	—	Enable	RW
	See basic_enable field of *Table 14 on page 38*.				
ASID_ENABLE	1	1	—	ASID match enable	RW
	Operation		Enables or disables the inclusion of the current ASID value in the debug event match, and determines whether an ASID field appears in BR trace messages. Value - Description 0: ASID match disabled. Thus the ASID value at the point of trigger will be included in BR trace messages. 1: ASID match enabled. Will only trigger when the current ASID matches the ASID_VALUE field, thus ASID value will not be included in BR trace messages.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CHAIN_ENABLE	2	1	—	Chain-latch enable	RW
	See the CHAIN_ENABLE field of *Table 14 on page 38*.				
CHAIN_ID	[6:3]	4	—	Chain-latch ID	RW
	See the CHAIN_ID field of *Table 15 on page 42*				
ASID_VALUE	[14:7]	8	—	ASID match value	RW
	See the ASID_VALUE field of *Table 14 on page 38*				
SR_MD_ ENABLE	[16:15]	2	—	CPU user/privileged mode selection	RW
	See the SR_MD_ENABLE field of *Table 14 on page 38* See *Section 1.14.3: Precondition checking for events and RTE on page 161* regarding precondition checks for SR_MD_ENABLE when tracing events and RTE.				

Table 58: DM.WP_BR_FILTER register definition

DM.WP_BR_FILTER				0x100028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CONDITIONAL	17	1	—	Conditional branch trace enable	RW
	Operation		Match for all conditional branches. SHmedia instructions: BEQ, BNE, BGT, BGE, BGTU, BGEU, BEQI, BNEI SHcompact instructions: BF, BF/S, BT, BT/S		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
UNCONDITIONAL	18	1	—	Unconditional branch trace enable	RW
	Operation		Match for all unconditional branches. SHmedia instructions: BLINK SHcompact instructions: BRA, BRAF, BSR, BSRF , JMP, JSR, RTS		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 58: DM.WP_BR_FILTER register definition

DM.WP_BR_FILTER				0x100028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RTE	19	1	—	RTE branch trace enable	RW
	Operation		Match for RTE instruction executions (apart from those which restore SR.WATCH to '1', see *Section 1.6.1: SR.WATCH bit on page 66*). See *Section 1.14.3: Precondition checking for events and RTE on page 161* regarding precondition checks for SR_MD_ENABLE when tracing RTE.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
EVENT	20	1	—	Event branch trace enable	RW
	Operation		Match for all interrupts and all exceptions. See *Section 1.14.3: Precondition checking for events and RTE on page 161* regarding precondition checks for SR_MD_ENABLE when tracing events.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ENABLE_TRACE_TIMESTAMP	21	1	—	Enable trace timestamp	RW
	See the ENABLE_TRACE_TIMESTAMP field of *Table 19 on page 54*				
ACTION_TRIG_OUT	22	1	—	Trigger out enable	RW
	See the ACTION_TRIG_OUT field of *Table 19 on page 54*				

Table 58: DM.WP_BR_FILTER register definition

DM.WP_BR_FILTER				0x100028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:23]	41	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 58: DM.WP_BR_FILTER register definition

1.14.2 Event specifics

Source CPU

Reason: Execution of an instruction which performs a branch, or launch of a trap or interrupt handler.

Undefined behavior

The operationsof the BR register are undefined as writing to other fields in DM.WP_BRX_FILTER when the WP channel is enabled.

WPC.WP_BRX_PRE does not exist.

DM.WP_BRX_PRE does not exist.

WPC.WP_BRX_ACTION does not exist.

DM.WP.BRX_ACTION does not exist

Some ACTION bits are available in DM.WP_BRX_FILTER:

ACTION_TRACE/TRACE_TYPE Is implicit - there is no ACTION_TRACE bit in DM.WP_BRX_FILTER. Thus background trace messages are always generated for the BR channel when it triggers.

TRACE_TYPE/ENABLE_TRACE_TIMESTAMP See *Table 44: BR watchpoint trace message on page 129*.

ACTION_TRIG_OUT**:** Supported.

1.14.3 Precondition checking for events and RTE

When an event is traced, the DM.WP_BRX_FILTER.SR_MD_ENABLE field is checked against the CPU mode (SR.MD) of the interrupted or excepting instruction.

This allows the branch channel to filter between interrupts/exceptions occurring in user mode and those occurring in privileged mode.

When an RTE is traced, the DM.WP_BRX_FILTER.SR_MD_ENABLE field is checked against the CPU mode (SR.MD) of the first instruction executed after the RTE.

This allows the branch channel to filter between RTEs which return to user mode and those which return to privileged mode.

An RTE instruction may return to a user-mode instruction which itself excepts and causes a new exception handler to be launched. In this case, it is implementation-defined whether the user-mode instruction is treated as having run in user mode or privileged mode. The DM.WP_BRX_FILTER.SR_MD_MODE bits might not work as expected in this situation.

1.14.4 Source and destination addresses in branch trace messages

Table 59 defines the selection of source and destination addresses for branch trace messages in a number of situations. The branch source address is always the address of the most recent completed instruction occurring before the jump in PC value. In particular, this means an excepting instruction's PC can never be shown as the source of a branch.

Scenario	No.[a]	Source	Destination	Branch type[b]
		(See footnote[c])		
SHmedia branch at PC=X, target PC=Y		X	Y	Branch
SHcompact non-delayed branch at PC=X, target PC=Y		X	Y	Branch
SHcompact delayed branch at PC=X, target PC=Y		X+2	Y	Branch
RTE at PC=X, new PC=Y, Y does not except		X	Y	RTE

Table 59: Selection of branch source and destination addresses

Scenario	No.[a]	Source (See footnote[c])	Destination (See footnote[c])	Branch type[b]
RTE at PC=X, new PC=Y, Y excepts, handler at PC=Z	1	X	Y	RTE
	2	X	Z	Event
SHmedia PC=X excepts, handler at PC=Z (X is not the target of a taken branch)		X-4	Z	Event
SHcompact PC=X excepts, handler at PC=Z (X is not the target of a taken branch)		X-2	Z	Event
SHmedia branch at PC=X, target at PC=Y excepts, handler at PC=Z	1	X	Y	Branch
	2	X	Z	Event
SHcompact delayed branch at PC=X, delay slot at PC=X+2, target at PC=Y excepts, handler at PC=Z	1	X+2	Y	Branch
	2	X+2	Z	Event
SHcompact branch (delayed or non-delayed) at PC=X excepts, handler at PC=Z		X-2	Z	Event
SHcompact delayed branch at PC=X, delay slot at X+2 excepts, handler at PC=Z		X	Z	Event
RTE at PC=X, new PC=Y is a SHcompact branch which excepts, handler at PC=Z	1	X	Y	RTE
	2	X	Z	Event
RTE at PC=X, new PC=Y is a SHcompact delayed branch whose delay slot excepts, handler at PC=Z	1	X	Y	RTE
	2	Y	Z	Event

Table 59: Selection of branch source and destination addresses

a. Some scenarios generate more than one BR channel trigger. When this happens, the triggers are numbered in the order they occur.

b. Where *branch* is shown in this column, it means either unconditional or conditional, depending on the branch type.

c. For addresses of instructions in SHmedia mode, the value in the trace packet has the least significant bit set. For addresses of instructions in SHcompact mode, the value in the trace packet has the least significant bit clear.

1.15 WP channel type FPF

Triggers whenever DM.FPF register is written. The trigger occurs immediately after the execution of the memory write instruction which caused it.

The FPF channel is unaffected by SR.WATCH.

1.15.1 Match registers

There are no associated match or action registers. Certain fields in the generic DM.WP_FPF_PRE register define the pre-conditions.

1.15.2 Event specifics

Source CPU

Reason: The DM.FPF register was written to.

Undefined behavior

Writing to the DM.WP_FPF_PRE register when the WP channel is enabled.

Note: As explained in Access to registers on page 17, instructions which write to WPC/DM memory mapped registers should be followed by a SYNCO instruction. In the case of writing to DM.FPF*, omitting the SYNCO instruction may result in the FPF message having an incorrect PC value.*

Fields in DM.WP_FPF_PRE:

BASIC_ENABLE, ASID_ENABLE, ASID_VALUE, CHAIN_ENABLE, CHAIN_ID - Supported

ECOUNT_ENABLE, ECOUNT_ID Not supported

WPC.WP_FPFX_PRE does not exist

WPC.WP_FPFX_ACTION does not exist

DM.WP_FPFX_ACTION does not exist:

The action is implicit, a trigger trace message is generated (see *Figure 45: FPF Watchpoint Trace Message on page 130*):

1.16 WP channel type PL

Bus analyzer watchpoints trigger whenever the bus analyzer(s) detect a matching transaction on the internal interconnect.

These channels are defined in *Chapter 2: SuperHyway bus analyzer on page 179*.

1.17 WP channel type DM

Debug module watchpoints trigger whenever the DM's FIFO is operating in "trace hold" mode, and entries are written to the FIFO as configured by the DM.TRCTL register (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*).

The DM channel is unaffected by SR.WATCH.

1.17.1 Match registers

There is no associated match register.

1.17.2 Event specifics

The DM channel is not a normal WP channel - it does not use the generic {WPC/DM}.WP_NX_PRE and {WPC/DM}.WP_NX_ACTION register formats as it only has one precondition, and only one action. This precondition and action are implicit, so there are no PRE or ACTION registers for the DM channel.

Source: Debug module

Reason: FIFO activity

The FF_THRESH field of DM.TRCTL (see *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97*) is used to specify that the DM will raise a debug interrupt when the FIFO has either:

- had an entry written to it,
- reached its high-water mark.

Supported actions (determined by ff_thresh field of DM.TRCTL)

INTERRUPT**:** Standard Event Handling sequence is followed. The RESVEC/DBRVEC offset is 0x200 (to denote a DEBUGINT event).

DM.EXP_CAUSE.DM_FIFO_INTERRUPT is set to 1.

The trace information present in the FIFO can be emptied one entry at a time, this is described in *Section 1.8.13: DM.FIFO_0 / DM.FIFO_1 / DM.FIFO_2 (FIFO port register) on page 112*.

1.18 WP channel type WPC_PERF

SH-5 has four performance counters, two located within the WPC and two located in the DM.

A WPC performance counter is incremented when either of the following types of events occur:

- A WPC watchpoint hit occurs and the watchpoint has its action_pcount field == 1 and pcount_id field selecting a specific performance counter.
- Whenever a CPU core event specified by fields of the WPC.WP_PERFX_MATCH_TYPE occurs and the conditions specified by fields of the WPC.WP_PERFX_PRE exist.

One register per WPC_PERF channel is used to define the match conditions. The WPC performance counter channels do not have a WPC.WP_PERFX_ACTION register, they have an implicit action to increment the corresponding WPC performance counter. For example, the WPC.WP_PERF0 channel increments the WPC.PCOUNT_VALUE_0 counter.

The specified match conditions are detected at different blocks of the CPU, and at different stages of the CPU pipeline. Thus the exact timing relationship between different match conditions is implementation dependant.

On each cycle, the match conditions are “OR”ed together and processed, yielding either 0 or 1 increments per cycle. The descriptions of the individual filters (in *Table 60*) specify whether this results in either 0..1 or 0..N increments per instruction.

The counter is incremented a bounded number of cycles after the watchpoint hit or the CPU state occurred. The bounded time is fixed, and thus allows timing relationships between CPU states to be observed.

If SR.WATCH is '0', WPC_PERF channels will not trigger.

1.18.1 Match registers

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ITLB_MISS	0	1	—	Instruction TLB miss	RW
	Operation		Instruction fetch access failed (due to a lack of an instruction TLB translation). Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
ICACHE_ACCESS	1	1	—	Instruction Cache Access	RW
	Operation		An instruction fetch successfully hit the Instruction cache. This includes preload accesses (due to PT instructions), and cache coherency (ICBI) accesses, but excludes prefetches (PREFI). Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ICACHE_MISS	2	1	—	Instruction Cache Miss	RW
	Operation		An instruction fetch missed the Instruction cache. This excludes prefetches (PREFI). Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
OTLB_MISS	3	1	—	OTLB miss	RW
	Operation		Operand access failed (due to a lack of an data TLB translation). Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
OCACHE_ACCESS	4	1	—	Operand Cache Access	RW
	Operation		See *Section 1.18.2: Operand cache access types on page 176*. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OCACHE_MISS	5	1	—	Operand Cache Miss	RW
	Operation		See *Section 1.18.2: Operand cache access types on page 176*. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
OCACHE_ALIAS	6	1	—	Operand Cache Alias	RW
	Operation		See *Section 1.18.2: Operand cache access types on page 176*. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
NONCACHE_ ACCESS	7	1	—	Non-cache Access	RW
	Operation		See *Section 1.18.2: Operand cache access types on page 176*. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
COND_TAKEN_ PREDICTED	8	1	—	Predicted conditional branch taken	RW
	Operation		SHmedia: Conditional branch predicted taken (L-bit of opcode set) and actually taken. Instructions: BEQ, BNE, BGT, BGE, BGTU, BGEU, BEQI, BNEI SHcompact: Conditional branch taken. Instructions: BF, BF/S, BT, BT/S Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
COND_TAKEN_ MISPREDICTED	9	1	—	Mispredicted conditional branch taken	RW
	Operation		SHmedia: Conditional branch predicted not taken (L-bit of opcode clear) but actually taken. Instructions: BEQ, BNE, BGT, BGE, BGTU, BGEU, BEQI, BNEI SHcompact branch instructions do not use prediction and so are not counted by this featured. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
COND_NOT_TAKEN_PREDICTED	10	1	—	Predicted conditional branch not taken	RW
	Operation		SHmedia: Conditional branch predicted not taken (L-bit of opcode clear) and actually not taken. Instructions: BEQ, BNE, BGT, BGE, BGTU, BGEU, BEQI, BNEI SHcompact: Conditional branch not taken. Instructions: BF, BF/S, BT, BT/S Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
COND_NOT_TAKEN_MISPREDICTED	11	1	—	Mispredicted conditional branch not taken	RW
	Operation		SHmedia: Conditional branch predicted not taken (L-bit of opcode clear), but actually taken. Instructions: BEQ, BNE, BGT, BGE, BGTU, BGEU, BEQI, BNEI SHcompact branch instructions do not use prediction and so are not counted by this featured Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
EXCEPTION_TAKEN	12	1	—	Exception taken	RW
	Operation		A exception has been taken. This covers all CPU generated exceptions. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
INTERRUPT_TAKEN	13	1	—	Interrupt taken	RW
	Operation		An interrupt has been taken. This covers all interrupts (and resets) caused from outside of the CPU. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
SHMEDIA_ SHCOMPACT_ SWITCH	14	1	—	Instruction mode switched	RW
	Operation		SHmedia to SHcompact, or SHcompact to SHmedia instruction mode switched. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
USER_ PRIVILEGED_ SWITCH	15	1	—	User/privileged mode switched	RW
	Operation		User mode to privileged mode, or privileged mode to user mode switched. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
SHMEDIA_RETIRED	16	1	—	SHmedia instruction retired	RW
	Operation		A SHmedia instruction was retired without raising an exception (that is, it completed normally). This only applies to pure SHmedia instructions, it does not include SHmedia instructions executed as part of SHcompact. Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
SHCOMPACT_ RETIRED	17	1	—	SHcompact instruction retired	RW
	Operation		A SHcompact instruction was retired without raising an exception (that is, it completed normally). Results in either 0 or 1 increments per instruction.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PIPE_STALL	18	1	—	Pipeline stall	RW
	Operation		A pipeline stall occurred, due to a register or resource hazard in the CPU or FPU. Results in either 0 or N increments per instruction (depending on number of stall cycles for that instruction's execution).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
TARGET_ADDRESS_STALL	19	1	—	Target address stall	RW
	Operation		A branch caused a pipeline stall (or NOP execution) due to its target address not being available (that is, the prepare target instruction (PTA, PTB, PTABS or PTREL) and branch instruction were too close together). Results in either 0 or N increments per instruction (depending on number of stall cycles for that instruction's execution).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INSTR_BRANCH_STALL	20	1	—	Instruction branch stall	RW
	Operation		A branch caused a pipeline stall (or NOP execution) as the instruction following the branch was not available in the instruction queue), however the branch target address was available (so this match type is distinct from TARGET_ADDRESS_STALL). Results in either 0 or N increments per instruction (depending on number of stall cycles for that instruction's execution).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
STALL_CYCLE	21	1	—	Stall cycle	RW
	Operation		A stall cycle occurred - this covers all cases of stall. Results in either 0 or N increments per instruction (depending on number of stall cycles for that instruction's execution).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
CPU_CYCLE	22	1	—	CPU Clock Cycle	RW
	Operation		A CPU clock cycle occurred. When this field is set to '1', the match will occur on every CPU clock cycle regardless of all other performance monitoring causes. This can be used to count CPU clock cycles.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

WPC.WP_WPC_PERF*x*_MATCH_TYPE				where x = channel ID	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[63:23]	41	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 60: WPC.WP_WPC_PERF*x*_MATCH_TYPE register definition

1.18.2 Operand cache access types

Table 60 defines four modes of operand cache access (OCACHE_ACCESS, OCACHE_MISS, OCACHE_ALIAS, NONCACHE_ACCESS) that can be seperately counted. The selection of which mode applies to each case is shown in *Table 61*. The terms ACCESS, ALIAS and MISS mean OCACHE_ACCESS, OCACHE_ALIAS and OCACHE_MISS respectively. The "—" symbol indicates that no performance counter trigger occurs in that situation. For example, the GETCFG and PUTCFG instructions never increment performance counters.

		SR.MMU=1				SR.MMU=0
Effective address in operand cache		**Yes**	**No**	**No**	**No**	**N/A**
Physical address in operand cache		**Yes**	**Yes**	**No**	**No**	**N/A**
Cacheable translation		**Yes**	**Yes**	**Yes**	**No**	**N/A**
Mode	**Instruction**					
SHmedia	LD, ST, FLD, FST	ACCESS	ALIAS	MISS	NONCACHE_ACCESS	
	LD .., R63 (prefetch)	ACCESS	ALIAS	MISS	NONCACHE_ACCESS	
	ALLOCO	ACCESS	ALIAS	MISS	NONCACHE_ACCESS	
	SWAP.Q	ALIAS	ALIAS	MISS	NONCACHE_ACCESS	
	OCBI, OCBP, OCBWB	ALIAS	ALIAS	MISS	NONCACHE_ACCESS	
	GETCFG/PUTCFG	—	—	—	—	
SHcompact	All instructions using the @-addressing mode with the exclusion of JMP, JSR, MOVA, OCBI, OCBP, OCBWB	ACCESS	ALIAS	MISS	NONCACHE_ACCESS	
	OCBI, OCBP, OCBWB	ALIAS	ALIAS	MISS	NONCACHE_ACCESS	

Table 61: Operand cache access modes for performance counting

1.18.3 Event specifics

Source CPU

Reason: CPU state change detected

Undefined behavior

The following operations of the WPC.WP_WPC_PERFX* registers are undefined:

- Writing to WPC.WP_WPC_PERFX_PRE, or to WPC.WP_WPC_PERFX_MATCH when the WP channel is enabled.

Fields in WPC.WP_WPC_PERF*x*_PRE:

BASIC_ENABLE, ASID_ENABLE, ASID_VALUE, ISAMODE_ENABLE, SR_MD_ENABLE, CHAIN_ENABLE, CHAIN_ID - Supported

ECOUNT_ENABLE, ECOUNT_ID Not supported

WPC.WP_WPC_PERFX_ACTION Does not exist

SuperH

2 SuperHyway bus analyzer

2.1 Introduction

As part of SH-5's on-chip debug capability, the SuperHyway router contains a bus analyzer, which provides two SuperHyway watchpoints (WP channel PL).

These provide the same set of pre and action conditions as the other watchpoint channels (*Section 1.6: WP channel matching on page 66* and *Section 1.5: Debug event actions on page 48*). In addition, an extra action feature is available to stop a specific bus initiator from making any further bus requests, that is, to freeze a bus master.

The bus analyzer includes a bus capture buffer for capturing part of a bus request or bus response (called a token) whenever a bus watchpoint hit occurs. A token includes the transaction header plus one 64-bit data word, regardless of the size of the bus transaction. These captured tokens can be sent to the debug module and used to create trace messages which are written to the debug module FIFO, from which they can be sent to a specified destination (see *Section 1.8: Debug module on page 85*). The capture buffer is memory-mapped and when trace mode is not enabled, debug software can directly read a token from the capture buffer.

It is also possible to use the bus analyzers, in conjunction with performance counters (*Section 1.1.7: Performance counters on page 14*) to capture selected performance parameters of the SuperHyway bus to allow system software to "tune" the parameters of individual application-specific modules or bus arbiters.

The SuperHyway bus may be implemented with multiple bus segments to improve performance by permitting transactions to occur on multiple segments concurrently. In such a multi-segment implementation, the SuperHyway bus analyzer will be physically implemented with multiple watchpoint comparators, one set on each segment. Functionally however, watchpoint sets on multiple bus segments are treated as a single set, controlled by the same registers.

SuperHyway bus analyzer watchpoints include the standard WP channel features, and so can make use of chain-latches, event counters, performance counters:

- Chain-latches can be used to combine arbitrary WP channels in sequence (including combining CPU and bus analyzer watchpoints).
- Event counters can be used to provide “trigger after N hits” functionality.

Accesses from the CPU to the WPC memory-mapped registers (resulting from loads and stores in the instruction stream) may be routed internally to the CPU, bypassing the SuperHyway. Whether this happens is implementation-dependent. Hence, whether such loads and stores can be detected by the bus analyzer channels is implementation-dependent.

Accesses to the WPC registers from other SuperHyway modules (for example, the SHdebug link via the debug module) are visible to the bus analyzer channels.

2.2 SuperHyway watchpoint comparators

Control registers associated with each SuperHyway watchpoint allow the following SuperHyway specific parameters to be defined:

- Bus transaction type: Opcode value and opcode mask fields allow any type of request or response to be matched.
- Source device ID (specific ID or any source).
- Destination device ID (top 8-bits of address, maskable).
- Destination address comparison range, for requests only.

Transaction type

The SuperHyway bus may be implemented with multiple segments and separate request and response segments. At a functional level, the SuperHyway bus analyzer has no knowledge of multiple bus segments or separate request and response segments. The debug user sets a single opcode value and an associated opcode mask in DM.WP_PLX_CTRL and this can be a request opcode or a response opcode depending on the type of transaction the debug user wants to monitor. This single opcode is used by the comparators which exist on each request segment and response segment.

Transaction source

Each SuperHyway request and response carries an 8-bit field that identifies the originator of the bus request. Each watchpoint has two fields associated with the source device, one field for the source device value and a second field for a mask. The mask field allows requests and responses from different devices to be matched.

Destination device

The destination device field in the SuperHyway request and response consists of the top bits of the address. In the case of SH-5, the destination field is the top 8-bits of the 32-bit address. Each watchpoint has two fields associated with the destination device, one field for the destination device value and a second field for a mask. The mask field allows requests and responses to different devices to be matched.

It is an implementation specific property as to whether the destination comparator is implemented.

Address

The address field in the request header defines an address in the specified destination device. In the case of SH-5, this address field is 24-bits wide with the least-significant 3-bits forced to zero, that is, addresses are always 8-byte aligned. Watchpoint registers define an address range by means of start address and end address parameters. The address comparison range can be as small as one bus word (8-bytes).

SuperHyway responses do not have an address field, so that the watchpoint address range comparison is ignored for responses.

2.3 Matching on devices with wide address ranges

Some SuperHyway devices (for example, the external memory interface (EMI)) have address ranges that occupy more than one value in the most significant byte. When programming a bus analyzer channel to match accesses to such a device, the programming must be as though multiple devices are being matched, one per value of the highest byte. The following paragraphs give some examples.

- If the desired address range is [0x81001000, 0x81002000], this can be programmed by setting the address start and end registers to 0x1000 and 0x2000 respectively, and the source/destination device value and mask to match precisely the value 0x81.
- If the desired address range is [0x80000000, 0x81FFFFF8], this can be programmed by setting the address start and end registers to 0x0 and 0xFFFFF8 respectively, and the source/destination device value and mask to match the values 0x80 and 0x81 (that is, with a wildcard LSB).
- If the desired address range consists of the 2 subranges [0x80000000, 0x807FFFF8] and [0x810000000, 0x817FFFF8], this can be programmed by setting the address start and end registers to 0x0 and 0x7FFFF8 respectively, and the source/destination device value and mask to match the values 0x80 and 0x81 (that is, with a wildcard LSB).
- It is not possible to match a range like [0x80800000, 0x817FFFF8]. This would require the end address to be set less than the start address, so no address would match for both address comparators.

2.4 Address comparison

For multiple data phase transactions (**Load16/Load32** and **Store16/Store32**) the address on the SuperHyway request bus is the address of the first data word of the transaction. The burst address ordering is linear. Burst requests issued by the MMU/Cache are always critical-word-first and bits [4:3] of the address determine the word order of **Store** requests and of **Load** responses as shown in *Table 62* and *Table 63*.

Address Bits [3,4]	Word Order
00	0, 1, 2, 3
01	1, 2, 3, 0
10	2, 3, 0, 1
11	3, 0, 1, 2

Table 62: SuperHyway word order for Store32, Load32

Address Bit [3]	Word Order
0	0, 1
1	1, 0

Table 63: SuperHyway word order for Store16, Load16

Since the request contains a single address corresponding to the first word of the transaction, the SuperHyway bus analyzer computes the implied address of each subsequent data phase.

2.5 Bus watchpoint hit action

When a bus watchpoint is enabled and a hit occurs, the bus analyzer signals the debug module that the hit has occurred and also optionally captures the bus transaction in a bus capture buffer. Action conditions for the PL watchpoint channels are the same as for other watchpoint channels, but additionally include the inhibition of the originating bus module from generating further bus transactions, that is, freeze the source.

Because of the pipelined nature of bus arbitration, it is not possible to immediately “freeze” a bus master following a bus analyzer watchpoint hit.

This ability to “freeze” the originating bus master applies only if the DM.WP_PLX_CTRL.SRC field identifies the bus master uniquely.

Debug interrupt actions

When bus analyzer debug interrupt is not enabled, but trace is enabled and a watchpoint hit occurs, the contents of the bus capture buffer are immediately sent to the debug module so that a trace message can be generated. The capture buffer is then available to capture another watchpoint hit token.

However, when a bus analyzer has debug interrupt enabled and a watchpoint hit occurs, the contents of the bus capture buffer are not sent to the debug module. Instead, a 1-bit register, DM.WP_PLS_EXCTRL.CBUF_FREEZE, is set by the watchpoint hit and stops the capture buffer from being overwritten by another watchpoint hit. The debug event handler is able to directly read the bus capture buffer to get details of the bus transaction which caused the watchpoint hit and can then re-enable the capture buffer by clearing DM.WP_PLS_EXCTRL.CBUF_FREEZE.

2.6 Freezing bus masters

A watchpoint hit can cause the SuperHyway arbiter to suspend bus transactions from any SuperHyway bus master by controlling the relevant arbiter signal.

When a bus analyzer watchpoint hit occurs, one of the possible actions is to stop the requesting bus master (except the MMU/cache) from initiating further bus transactions.

This “freeze” action is commonly required when the watchpoint hit causes a debug interrupt. In addition to this automatic “freeze” action, control register fields exist to allow software to “freeze” bus masters (except the MMU/Cache) at any time. Debug software can then “unfreeze” the bus master module when appropriate by writing to the DM.PL_FRZ register.

Because of the pipelined nature of bus arbitration, it is not possible to immediately “freeze” a bus master following a bus analyzer watchpoint.

The DM.PL_FRZ register has individual 1-bit fields for each SuperHyway bus master capable of being frozen. In the initial SH-5 evaluation device implementation, the DM.PL_FRZ register has only the least significant 16-bits implemented, giving the capability of freezing up to 16 SuperHyway bus masters. The outputs of this register go to the SuperHyway router as individual signals and the freeze action involves the control of the request signal from a specific physical SuperHyway bus master within the arbiter function of the SuperHyway router. The relationship between SuperHyway source ID and physical SuperHyway module may change in different chips which use the SH-5 core and is specified in the SuperHyway Router Micro-architecture document.

When the DM.WP_PLX_CTRL.SRC field defines a specific bus master, it is the responsibility of debug software to set up the value in the DM.WP_PLX_ACTION.PL_MODULE field to match the chip’s implementation of source ID and physical module.

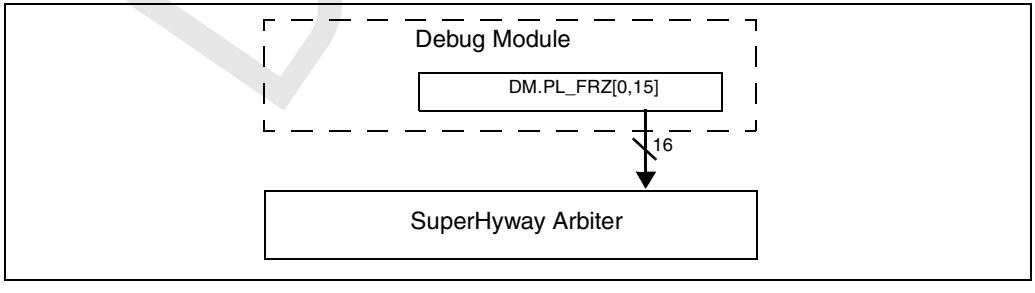

Figure 7: DM to PL arbiter interconnect

As shown in *Figure 7*, the least-significant 16-bits of DM.PL_FRZ connect directly to the SuperHyway arbiter and the individual signals are used by the arbiter to freeze/unfreeze a given master.

2.7 Unfreezing bus masters

The DM.PL_FRZ register allows software to determine which bus masters are currently frozen and to unfreeze one or more bus masters. This register contains 16 identical single bit fields which are associated with up to 16 SuperHyway bus master modules capable of being frozen. Modules not capable of being frozen and which do not have a corresponding FREEZE_X field include the CPU and the DM. The identity of the SuperHyway physical module associated with each FREEZE_X field is chip-specific (see *Section 4.1.12: Bus analyzer module/SuperHyway mapping on page 251*).

DM.PL_FRZ				0x100080	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FREEZE_X	[15:0]	16	✓	Bus master freeze control	RW
	Operation		A value of **1** in this field inhibits the bus request signal for the corresponding SuperHyway bus master, stopping the bus master from initiating any further bus transactions. The freeze state can be set either by a watchpoint hit action or by software writing to this field.		
	When read		Returns 0 when the bus master is able to operate normally and 1 when it is frozen.		
	When written		Sets or clears the freeze state. Value - Description 0: unfrozen 1: frozen		
	HARD reset		0		

Table 64: DM.PL_FRZ register definition

DM.PL_FRZ				0x100080	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[16,63]	48	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 64: DM.PL_FRZ register definition

2.8 WP channel type PL

Separate sets of pre-condition, match, and action-condition registers exist for each of the SuperHyway bus analyzer watchpoints. The fields of the pre-condition registers are described in *Table 16: DM.WP_PLx_PRE register definition on page 44*. The fields of the action-condition registers are described in *Table 20: DM.WP_PLx_ACTION register definition on page 60*.

Match registers

Control registers for each watchpoint channel defines the conditions which enable the watchpoint. Refer to the generic description in *Section 1.6: WP channel matching on page 66*.

In addition to the generic pre registers, special match registers are also used. These are defined in the tables that follow. These registers must only be modified with the watchpoint channel disabled (DM.WP_PLX_PRE.BASIC_ENABLE==0). If they are modified with the channel enabled, the behavior is architecturally undefined.

DM.WP_PL*x*_MATCH_START				where x = channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[2:0]	3	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ADDRESS	[23:3]	21	—	Start address	RW
	Operation		Represents the base (lower) address of the watchpoint comparator address range. Addresses are bus-word aligned (bits [2:0] are forced to zero). For burst transactions, the address on the SuperHyway bus is always critical-word-first and bits [4:3] determine the word order as shown in *Table 62 on page 183*.		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:24]	40	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 65: DM.WP_PL*x*_MATCH_START register definition

DM.WP_PL*x*_MATCH_END				where x= channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[2:0]	3	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
ADDRESS	[23:3]	21	—	End address	RW
	Operation		Represents the limit (upper) address of the watchpoint comparator address range. A watchpoint hit is possible if the transaction address is >= DM.WP_PLX_MATCH_START and <= DM.WP_PLX_MATCH_END. Addresses are bus-word aligned (bits [2:0] are forced to zero).		
	When read		Returns current value		
	When written		Updates value		
	HARD reset		Undefined		
—	[63:24]	40	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 66: DM.WP_PL*x*_MATCH_END register definition

DM.WP_PLx_CTRL				where x= channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DEST_MSK	[7:0]	8	—	Watchpoint destination mask	RW
	Operation		Defines which bits in the DEST_VAL field must match for the watchpoint to trigger. A value of '1' in a bit position causes the watchpoint comparator to ignore the corresponding bit of the DEST_VAL field, a value of '0' in a bit position causes the watchpoint comparator include the bit in the comparison. Thus, a value of 0xFF causes the destination field in each transaction to be ignored and the watchpoint will match all destinations. It is an implementation specific property as to whether the destination comparator is implemented.	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[15:8]	8	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
DEST_VAL	[23:16]	8	—	Watchpoint destination value	RW
	Operation		Combined with the DEST_MSK field and is matched against the destination field of each SuperHyway bus transaction. It is an implementation specific property as to whether the destination comparator is implemented.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 67: DM.WP_PL*x*_CTRL register definition

DM.WP_PLx_CTRL				where x= channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SRC_MSK	[31:24]	8	—	Watchpoint source mask	RW
	Operation		Defines which bits in the SRC_VAL field must match for the watchpoint to trigger. A value of '1' in a bit position causes the watchpoint comparator to ignore the corresponding bit of the SRC_VAL field, a value of '0' in a bit position causes the watchpoint comparator include the bit in the comparison. Thus, a value of 0xFF causes the source field in each transaction to be ignored and the watchpoint will match all sources.	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
SRC_VAL	[39:32]	8	—	Watchpoint transaction source	RW
	Operation		Defines the identity of the transaction source. A value of 0xFF means that the watchpoint ignores the source field and will match the transaction generated by any source.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[47:40]	8	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 67: DM.WP_PL*x*_CTRL register definition

DM.WP_PLx_CTRL				where x= channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
OPCODE_MSK	[55:48]	8	—	Transaction opcode mask	RW
	Operation		Defines which bits in the OPCODE_VAL field must match for the watchpoint to trigger. A value of '1' in a bit position causes the watchpoint comparator to ignore the corresponding bit of the OPCODE_VAL field, a value of '0' in a bit position causes the watchpoint comparator include the bit in the comparison. Thus, a value of 0xFF causes the opcode field in each bus request and response to be ignored and the watchpoint will match all opcodes.	
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
OPCODE_VAL	[63:56]	8	—	Transaction opcode value	RW
	Operation		Combined with the OPCODE_MSK field and is matched against the 8-bit opcode field of each SuperHyway bus request and response.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 67: DM.WP_PL*x*_CTRL register definition

Debug interrupt action registers

In addition to the generic DM.WP_PLX_ACTION registers, the following registers provide additional functions associated with bus analyzer debug interrupts.

DM.WP_PLx_EXCTRL				where x = channel id	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CBUF_FREEZE	0	1	✓	Capture buffer status/freeze control	RW
	Operation		Controls the bus analyzer capture buffer when debug interrupt is enabled. This register is set by hardware following a PL watchpoint hit when debug interrupt is enabled. Debug event handling software should read the contents of the capture buffer, and then finally clear the CBUF_FREEZE register. If PL debug interrupt is not enabled, the contents of this register are undefined.		
	When read		Returns 0 when the capture buffer is empty, 1 when it is frozen following a watchpoint hit resulting in a debug interrupt.		
	When written		Provides a write-1-clear function for clearing the capture buffer frozen state. Value - Description 0: no action. 1: clears the CBUF_FREEZE register if it had previously been set by hardware. No action if debug interrupt is not enabled or if this register is already clear.		
	HARD reset		0		
—	[63:1]	63	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 68: DM.WP_PLx_EXCTRL register definition

Note: If debug interrupt and trace actions are both enabled, the debug interrupt action described above takes priority. Since the contents of the bus capture buffer are not automatically sent to the debug module, no trace message is generated.

DM.WP_PL*x*_CBHDR				where x = channel id Note that when the PLx channel is not enabled, the contents of all fields of this register are undefined.	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PLINK_DEST	[7:0]	8	✓	SuperHyway transaction header	RO
	Operation		This register provides access to the header captured in the bus analyzer capture buffer following a watchpoint hit when debug interrupt is enabled. This field is the destination specified in the bus request or response.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		
PLINK_OPCODE	[15:8]	8	✓	SuperHyway transaction header	RO
	Operation		This field is the opcode specified in the bus request or response.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		
PLINK_TID	[23:16]	8	✓	SuperHyway transaction header	RO
	Operation		This field is the transaction ID specified in the bus request or response.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		

Table 69: DM.WP_PL*x*_CBHDR register definition

DM.WP_PL*x*_CBHDR				where x = channel id Note that when the PLx channel is not enabled, the contents of all fields of this register are undefined.	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PLINK_SOURCE	[31:24]	8	✓	SuperHyway transaction header	RO
	Operation		This field is the source specified in the bus request or response.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		
PLINK_ADDRESS	[55:32]	24	✓	SuperHyway transaction header	RO
	Operation		This field is the address specified in the bus request.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		
PLINK_MASK	[63:56]	8	✓	SuperHyway transaction header	RO
	Operation		This field is the mask specified in the bus request.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		

Table 69: DM.WP_PL*x*_CBHDR register definition

DM.WP_PL*x*_CBDATA				where x = channel id Note that when the PLx channel is not enabled, the contents of all fields of this register are undefined.	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PLINK_DATA	[63:0]	64	✓	SuperHyway transaction data	RO
	Operation		This register provides access to the data word captured in the bus analyzer capture buffer following a watchpoint hit when debug interrupt is enabled.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		

Table 70: DM.WP_PL*x*_CBDATA register definition

Event specifics

Source SuperHyway

Reason Occurrence of a SuperHyway bus transaction which matches according to the PL channel's comparator settings.

Undefined behavior

Writing to DM.WP_PLX_* (except DM.WP_PLX_PRE) when the WP channel is enabled is undefined.

Supported fields in DM.WP_PL*x*_PRE:

```
BASIC_ENABLE, ECOUNT_ENABLE, ECOUNT_ID, CHAIN_ENABLE, CHAIN_ID
```

Supported actions in DM.WP_PL*x*_ACTION:

INTERRUPT: Standard hardware debug handling sequence is followed. DM.EXP_CAUSE.PL_INTERRUPT is set to denote the bus analyzer channel. The RESVEC/DBRVEC offset is 0x200 (to denote an asynchronous debug interrupt).

TRACE: See *Table 46: PL Watchpoint trace message on page 131*.

Note: In "trace buffer" mode (see Section 1.8.3: DM FIFO/trace buffer in target system memory on page 88) the DM writes the trace messages to target system memory using SHwy store8 transactions. These transactions are visible to the SHwy bus analyzers, and thus if a bus analyzer is programmed such that it will match on these transactions, an infinite number of bus analyzer hits (and thus an infinite number of trace messages) will be generated.

TRIG_OUT: supported.

ECOUNT: supported.

PCOUNT: supported.

ALTER: supported.

FREEZE_EN: supported.

SuperH

3 External debug interfaces

3.1 Introduction

A development tool can communicate with SH-5's on-chip debug functions through one of two possible interfaces; a dedicated high-speed interface called the SHdebug link and the JTAG interface.

The JTAG interface is controlled by the on-chip JTAG TAP controller and when JTAG is the currently-selected debug interface, debug messages are transferred between the TAP controller and the debug module. When the SHdebug link is the currently-selected debug interface, the debug module directly controls all SHdebug link functions.

Other pins provide trigger-in and trigger-out signals which allow some of the debug functions to be monitored and controlled by a logic analyzer or other external test equipment.

Refer to *Chapter 1: Debug/trace architecture on page 11* for a description of the debugging functions incorporated into SH-5.

3.2 SHdebug link

3.2.1 Key features

The SHdebug link has completely separate input and output interfaces which provide communication with a debug adapter. The SH-5 evaluation device implementation has an input data path 1-bit wide and an output data path 4-bits wide. These widths may be increased in subsequent SH-5 chips to meet the debugging bandwidth needs of different implementations and applications.

The SHdebug link provides:

- Full access by a tool to the SH-5 physical address map (RAM, ROM, on-chip devices, external-devices).
- SH-5-originated access to a 16 Mbyte address space mapped over the SHdebug link into tool memory.

  This allows a target debug agent (or any other code) to execute on the CPU without requiring any external RAM or ROM, and thus enables use of SH-5 without a traditional monitor ROM.
- Control of the CPU via memory-mapped register.

  See *Section 1.3: CPU control on page 30*.
- Streaming operations for CPU and bus trace information.

  Allows trace information gathered from the CPU and the on-chip busses to be copied to a specified FIFO in the physical memory map (such as RAM or the SHdebug link).

  See *Section 1.8: Debug module on page 85*.

The SHdebug link is normally connected to a tool's debug adaptor board (to provide code download and debug facilities). It can also be connected to specialized hardware debug systems (such as logic analyzers) to provide more complex facilities.

3.2.2 Protocol levels

The SHdebug link has two protocol levels in each direction:

- A low-level protocol which provides start-of-message indication, end-of-message indication and flow control. At this level, message input and message output are completely independent and messages can flow in both directions at the same time.
- A higher-level protocol which identifies the message contents and specifies the response required to each request. Certain message types, that is, trace messages from SH-5 are output-only and require no response from the tool.

  The protocol is encoded little endian.

3.2.3 External pins

The SHdebug link interface pins are described in *Table 71*.

Signal	Lines	Type	Internal pull-up	Description
DM_CLKIN	1	IN_TBD	Yes	Clock from debug adapter, in phase with DM_IN and DM_ISYNC.
DM_CLKOUT	1	OUT_TBD	No	SHdebug link clock source.
DM_OUT[0,3]	4	OUT_TBD	No	Output data, synchronous to the rising edge of DM_CLKOUT.
DM_OSYNC	1	OUT_TBD	No	Output sync, synchronous to the rising edge of DM_CLKOUT.
DM_IN	1	IN_TBD	Yes	Input data, synchronous to the rising edge of DM_CLKIN.
DM_ISYNC	1	IN_TBD	Yes	Input sync, synchronous to the rising edge of DM_CLKIN.
DM_TRIN_N	1	IN_LVTTL	Yes	Trigger input.
DM_TROUT_N	1	OUT_LVTTL	No	Trigger output.

Table 71: SHdebug link external pins

3.2.4 Clocking

The data provided by the SH-5 on the DM_OSYNC and DM_OUT[3:0] pins is stable on the rising edge of DM_CLKOUT.

The SH-5 debug module divides down the bus clock to provides the DM_CLKOUT clock source for the SHdebug link. The divider has a power-on default value of 0xFFFF, giving a SHdebug link clock frequency of approximately 3 kHz with a bus clock speed of 200 MHz. The divider is a field in memory-mapped register DM.CLKOUTDIV, which can be changed by host debug software. Debug software must have knowledge of the bus clock frequency and the maximum operating speed of the debug adapter before changing the divider value.

In the initial SH-5 evaluation device implementation, the divider cannot be set to give a division smaller than 2. With a bus clock frequency of 200 MHz, a value of 2 gives a debug-link clock speed of 100 MHz.

In some applications, the CPU and bus clock frequencies can be dynamically changed by power-management software. By deriving the SHdebug link clock from bus clock, the SHdebug link clock speed automatically follows changes in bus clock speed allowing SHdebug link communication to be maintained over any bus speed range. When SH-5 enters standby state, both the PLL and the master oscillator are disabled which means that the DM_CLKOUT signal assumes a steady DC level. The tool should monitor the STATUS0 and STATUS1 signals to determine when SH-5 has entered standby state.

A tool can issue a command to wake up SH-5 from standby state. However, once the wake-up command has been issued, it can take up to a millisecond for the PLL to stabilize and internal clocks to be enabled. The tool must therefore monitor the STATUS0 and STATUS1 signals and delay any DBUS requests until the chip is operating normally.

The input clock, DM_CLKIN, is independent of the output clock and is used by the debug module to extract serial data from the DM_IN and DM_ISYNC pins.

At high clock speeds, the design of the debug adapter must ensure that the DM_CLKIN, DM_IN and DM_ISYNC signals all have approximately the same I/O pad delays and interconnect delays. The frequency of DM_CLKIN may be higher or lower than the frequency of DM_CLKOUT up to the maximum permitted by the electrical specification of SH-5.

3.2.5 Pin state during reset

Three pins are sampled during a reset sequence initiated externally to the SH-5 through NOTRESETP or NOTRESETM. These pins allow the following to be determined as part of the external reset sequence:

- debug module to be enabled or disabled.

  The signal used is referred to as DM_ENABLE. It is obtained by sampling the DM_CLKIN pin during reset.

- CPU to be brought up in a suspended or running state.

  The signal used is referred to as SUSPEND. It is obtained by sampling the DM_ISYNC pin during reset.

- Reset to be forced to be a DEBUG reset, rather than the normal POWERON or MANUAL reset specified by the RESETP and RESETM pins.

  The signal used is referred to as RESET_MODE. It is obtained by sampling the DM_IN pin during reset.

  The pins are not sampled in this way when a reset is generated by the watchdog timer or from software (see *Section 1.3.2: Control operations on page 31* and *Section 1.7: Reset, panic and debug events on page 73*).

  Section 3.4.2: Reset functions available from debug tools on page 219 defines the reset sequence in detail. Further information is available in the PMU chapter (Refer to the reset controller chapter in Volume 1.).

3.2.6 Start of message indication

The input and output portions of the SHdebug link operate completely independently. SHdebug link messages can flow in both directions simultaneously.

Output from SH-5

The length of output messages is not fixed but is determined by the message contents.

The DM_OSYNC signal is asserted to indicate either a output-idle condition or the start of a new message. Output-idle is represented by a header type field of 0b000. During output-idle, the debug module forces the value of '0' on the DM_OUT[2:0] pins and asserts DM_OSYNC. The first word of a SHdebug link message is indicated by the DM_OSYNC signal being asserted and the output data bus having a data value other than 0b000 on the DM_OUT[2:0] pins.

Messages are output in little-endian order (that is, bit 0 of byte 0 first).

Input to SH-5

Data being sent by an external debug adapter to SH-5 uses a similar method of achieving message-level synchronization. Since all messages in this direction are DBUS requests or responses, there is no need for a header with a message type field. In this direction, the start of each message is indicated by a 1 to 0 transition of the DM_ISYNC signal.

Messages are input in little-endian order (that is, bit 0 of byte 0 first).

3.2.7 Flow control

Flow control of messages from SH-5

SH-5 can send a DBUS request or response message to the tool only when the DBUS receive buffer within the tool (that is, within the debug adapter) can accept another message. During the input-idle state, DM_IN indicates whether the DBUS receive buffer within the tool can accept another message (DM_IN == 0) or is full (DM_IN == 1). The debug module uses this debug adapter receive buffer status to determine when it can send another DBUS request or response message to the tool.

This debug adapter receive buffer status has no effect on SH-5 sending trace messages to the debug adapter. The debug module assumes that the debug adapter can process a continuous stream of trace messages, separated by single idle words, at the selected SHdebug link clock speed.

Flow control of messages from tool

The tool can send a DBUS request or response message to SH-5 only when the DBUS receive buffer within the debug module is empty. Data bit DM_OUT[3], in every output-idle word, indicates whether the SH-5 receive buffer is empty (DM_OUT[3] == 0) or still contains a request or response previously sent by the tool (DM_OUT[3] == 1). The tool uses this buffer status condition to determine when it can send a new message to SH-5.

Undefined behavior will occur if the tool ignores the buffer status indication and sends a message to SH-5 when its DBUS receive buffer is not empty.

3.2.8 SHdebug link output protocol

The SH-5 debug module can initiate two types of transaction over the SHdebug link:

1 Those associated with SuperHyway bus transactions (called DBUS messages). These DBUS transactions occur when a tool reads or writes to SH-5 internal address space and when the SH-5 CPU or other bus master reads or writes address space in the tool.

2 Trace messages from on-chip debug logic (called DTRC).

The protocol is the same, regardless of the width of the SHdebug link output data path. A 3-bit type field in the header word of the message defines the message contents.

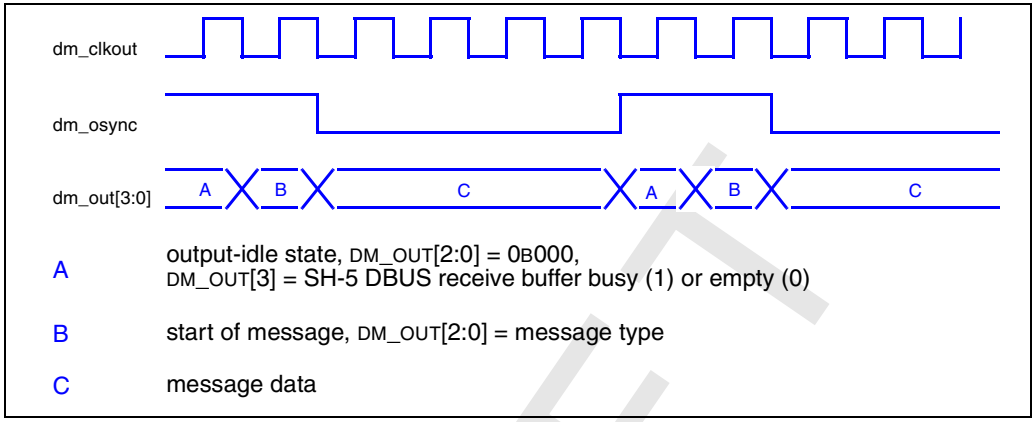

Figure 8: Debug-link output

Start and end of messages

Output-idle words are transmitted over the SHdebug link during times when the debug module has no data to send, for example, when the DM FIFO is empty.

When the FIFO contains several trace messages, they are sent over the SHdebug link with a minimum of one idle (4-bit) word separating the messages.

Output-idle words serve another purpose; as the means of telling the tool the status of the debug module DBUS receive buffer. See *Section : Flow control of messages from tool on page 205*.

Message structure

The values signalled on dm_out[2:0] when DM_OSYNC is asserted are defined in *First word of message on page 118*.

Table 72 and *Table 73* provide some examples of DTRC and DBUS packets as they appear on the SHdebug link, and give details of the meaning assigned to DM_OUT[3] during non-idle transmissions.

DBUS messages

A DBUS message is sent from SH-5 whenever the CPU or another bus master issues requests to the debug module's SuperHyway target address space. For each request message sent from SH-5 there is a corresponding response message sent back from the external debug adapter or development host. The DBUS request message has the same format as the originating SuperHyway message but with the addition of a header to identify it as a DBUS message.

The message type field occupies the least-significant 3-bits of the first byte of the message. The remaining 5-bits of this first byte are unused.

DTRC messages

There are two types of DTRC messages:

- trigger trace messages (message type MHDR_DTRC_TRIG == 0b011)
- background trace messages (message type MHDR_DTRC_BACK == 0b010).

They are distinguished by the message type field, but contain exactly the same data.

The debug adapter can perform different functions with these two types of trace message (using a different message type field simplifies the design of the debug adapter as the message header allows a simple comparison to be made).

Typically, the debug adapter will store background trace messages in a trace buffer memory area for later analysis but will forward trigger trace messages to debug software running on the development host for immediate processing.

DTRC transactions are output-only and require no response.

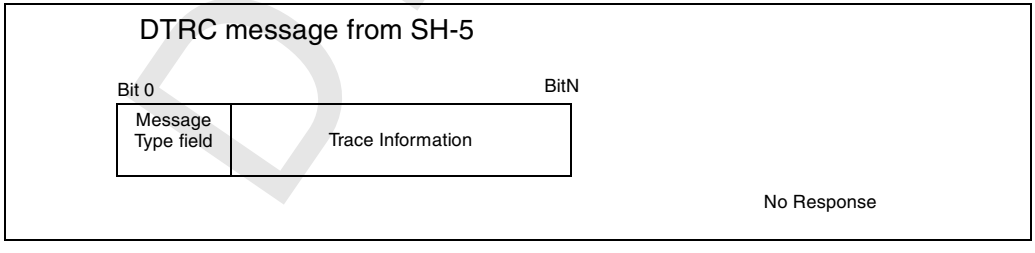

Figure 9: DTRC protocol

With DTRC messages, all bits of the first byte contain useful information. There are not the same unused bit positions that exist in DBUS messages.

Refer to *Section 1.9.3: DTRC messages on page 119* for details of trace message contents.

3.2.9 SHdebug link input protocol

No message header is used on messages sent to SH-5 from the debug adapter since all SHdebug link transactions in this direction are implicitly DBUS requests or responses.

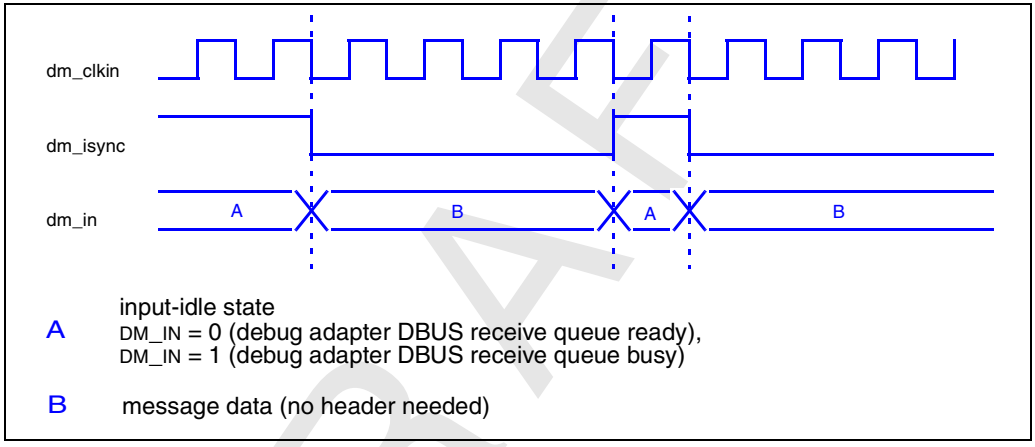

Figure 10: SHdebug link Input

Start and end of input messages

The DM_ISYNC signal is used to distinguish message data on the DM_IN pin from line idle. The transition from DM_ISYNC == 1 to DM_ISYNC == 0 indicates the start of a message and the transition from DM_ISYNC == 0 to DM_ISYNC == 1 indicates the end of a message. Messages are separated by one or more clock periods of idle (DM_ISYNC == 1).

3.2.10 Debug-link message examples

IA watchpoint trace message

Clock cycle	DM_OSYNC state	DM_OUT[0,3] contents
-1	1	Output-Idle [2:0] == 0b000 (MHDR_IDLE) [3] = buffer status
0	1	Header [3:0] [2:0] == 0b010/0b011 (MHDR_DTRC_{BACK/TRIG}) [3] == bit 3 of trace header
1	0	Header [7:4]
2	0	Header [11:8]
3	0	Header [15:12]
4	0	PC Value [3:0]
5	0	PC Value [7:4]
6	0	PC Value [11:8]
7	0	PC Value [15:12]
8	0	PC Value [19:16]
9	0	PC Value [23:20]
10	0	PC Value [27:24]
11	0	PC Value [31:28]
12	1	Output-Idle [2:0] = 0b000 (MHDR_IDLE) [3] = Buffer status

Table 72: Trace message example

DBUS request message

Clock Cycle	DM_OSYNC state	DM_OUT[0,3] contents
-1	1	Output-Idle [2:0] == 0b000 (MHDR_IDLE), [3] == Buffer status
0	1	Header [0,3] [2:0] == 0b001 (MHDR_DBUS) [3] = No useful data
1	0	Dummy nibble (no useful data)
2	0	Opcode [3:0]
3	0	Opcode [7:4]
4	0	Address [3:0]
5	0	Address [7:4]
6	0	Address [11:8]
7	0	Address [15:12]
8	0	Address [19:16]
9	0	Address [23:20]
10	0	Address [27:24]
11	0	Address [31:28]
12	0	Source [3:0]
13	0	Source [7:4]
14	0	TID [3:0]
15	0	TID [7:4]
16	0	Mask [3:0]
17	0	Mask [7:4]
18	1	Output-idle [2:0] == 0b000 (MHDR_IDLE) [3] = Buffer status

Table 73: DBUS Load8 request message example

3.2.1 SHdebug link control registers

3.2.1.1 DM.CLKOUTDIV (debug-link control register)

This register is specified in *Table 74*. The divider field of this register allows debug software to change the clock speed of the SHdebug link.

DM.CLKOUTDIV				0x100090	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DIVIDER	[15:0]	16	—	SHdebug link clock divider	RW
	Operation		This register determines the speed of the SHdebug link clock source (DM_CLKOUT). The input to the divider is bus clock. A value of n corresponds to a clock division of 2(n+1). The maximum SHdebug link output clock speed (divider value = 0) is half the speed of bus clock.		
	When read		Returns current value		
	When written		Updates current value. Debug software must ensure that the debug link and the debug adapter are capable of supporting the selected clock speed. All communication between SH-5 and the debug adapter may be lost if the clock speed is set too high.		
	HARD reset		0xFFFF		
—	[63:16]	48	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 74: DM.CLKOUTDIV definition

3.3 JTAG interface

3.3.1 Introduction

The JTAG interface can be used for connection to a debug tool as an alternative to the SHdebug link, however, the effective bandwidth (messages per second) is much lower. All messages capable of being sent over the SHdebug link can also be sent via the JTAG interface. Because of the lower bandwidth available, configuring SH-5's debug functionality to send trace messages via JTAG may give unacceptable performance but this still may be a useful capability for some applications.

3.3.2 Basic concepts

The implementation of the SH-5 debug interface via a JTAG connection has the following characteristics:

- Standard JTAG functionality is not compromised (for example, the standard JTAG instruction "space" remains unpolluted).
- The port is drivable from standard JTAG state-machine interface devices.
- The port allows "unsolicited" messages to be sent from SH-5 to the tool.
- Realistic real-time performance is achievable.
- Uses identical message structure as the SHdebug link.

The SH-5 JTAG TAP controller implements all the mandatory features of a standard JTAG port, including the "**BYPASS**" and "**IDCODE**" instructions. The JTAG instruction register defaults to the "**IDCODE**" instruction.

Access to debug features is enabled by the loading of a single command ("**DEBUG**") to the JTAG instruction register. The value of this "**DEBUG**" instruction is implementation specific, and is defined in *Section 4.1.10: JTAG IR DEBUG codes on page 249*. An additional command ("**WAKEUP**") is used to wake the SH-5 from a sleep state via the JTAG port. This is also defined in *Section 4.1.10: JTAG IR DEBUG codes on page 249*.

3.3.3 Debug interface selection

A debug tool can communicate with SH-5 using either the SHdebug link or the JTAG interface. At power-on, the debug module defaults to select the SHdebug link as the debug interface.

The debug module changes its selection to the JTAG interface whenever it detects that the "**DEBUG**" command has been written into the JTAG instruction register. This must only be done when the SHdebug-Link is idle, if the JTAG port is selected whilst there is SHdebug-Link traffic outstanding, undefined effects will occur.

Once JTAG has been selected as the debug interface, it remains selected until SH-5 is reset. Software can determine the interface selection state from the read-only field DM.TRCTL.DL_N_JTAG.

3.3.4 JTAG debug message protocol

The input and output debug messages transferred via the JTAG port are identical to those which can be sent via the SHdebug link, including the standard message-type header.

An inherent characteristic of messages between SH-5 and a tool is that messages are of different lengths depending on the message type. The longest message is 41 bytes (a DBUS **Store32** request message from SH-5) and the shortest message is 3 bytes (an IA trace message). The JTAG debug message protocol defined in this section provides a method for both the tool and the SH-5 target to determine the start and end of these variable-length messages.

There are two parts to the JTAG debug message protocol. At the lowest level, data and status bits are transferred between a tool and the debug serial shift register. At a higher level, the protocol provides the mechanism for detecting the start and end of messages consisting of a variable number of bytes.

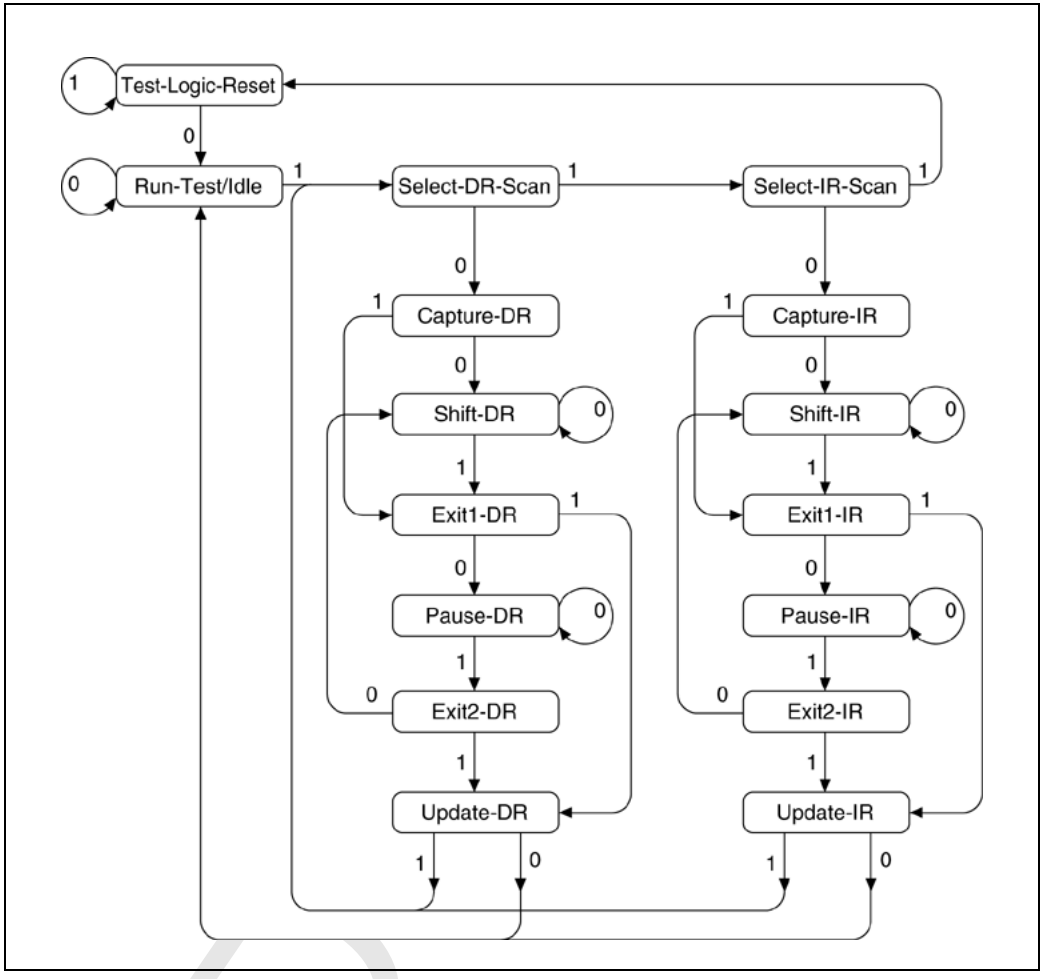

Figure 11: TAP control state transition diagram

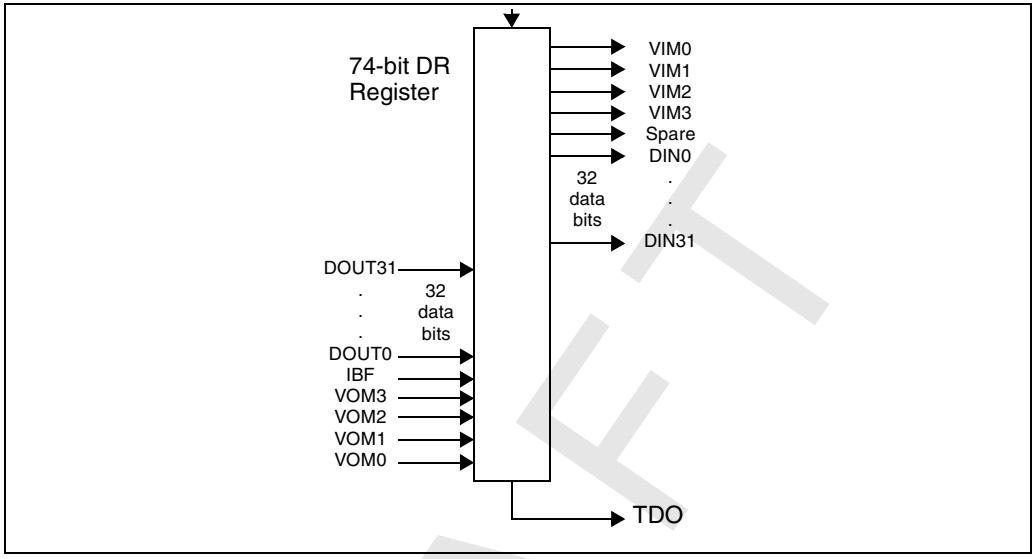

Figure 12: JTAG debug DR register connections

As shown in *Figure 12*, the SH-5 JTAG debug data register is 74 bits long, consisting of:

- four valid input message status bits (VIM3, VIM2, VIM1, VIM0), one associated with each of the four byte positions,
- 32 data input bits,
- 32 data output bits,
- an input buffer full (IBF) status bit,
- four valid output message status bits (VOM3, VOM2, VOM1, VOM0), one associated with each of the four byte positions.

Status Bit	Meaning
VIMx	0: Input byte position has no valid data. 1: The serial word which has just been shifted in contains one byte of an input message in the corresponding byte position.
VOMx	0: Output byte position has no valid data. 1: The serial word which has just been shifted out contains one byte of an output message in the corresponding byte position.
IBF	0: The input message buffer in SH-5's JTAG tap controller is available for a new message. 1: The input message buffer in SH-5's JTAG tap controller is full.

Table 75: Status bit meaning

The use of a single 74-bit DR register allows the following alternative transfers to take place:

1 The tool can poll just the output status from SH-5 (5 bits of DR shifted).

2 The tool can poll the output status plus from one byte to four bytes of an output message from SH-5 (13, 21, 29 or 37 bits of DR shifted).

3 The tool can send status and from one byte to four bytes of an input message to SH-5 (13, 21, 29 or 37 bits of DR shifted).

4 The tool can poll status and from one byte to four bytes of an output message from SH-5 and at the same time send status and from one byte to four bytes of an input message to SH-5 (13, 21, 29 or 37 bits of DR shifted).

As shown in *Figure 12*, when SH-5 is connected to a tool capable of shifting variable length data words, the tool transfers 13, 21, 29 or 37 bits into the DR register giving 8, 16, 24, or 32 message data bits status plus bits to indicate valid input data in each of the four byte positions (plus one unused bit). When polling out an output message, the tool shifts 13, 21, 29 or 37 bits out of the DR register with four of these bits indicating valid output data and a another status bit indicating the state of the SH-5 tap controller debug message input buffer.

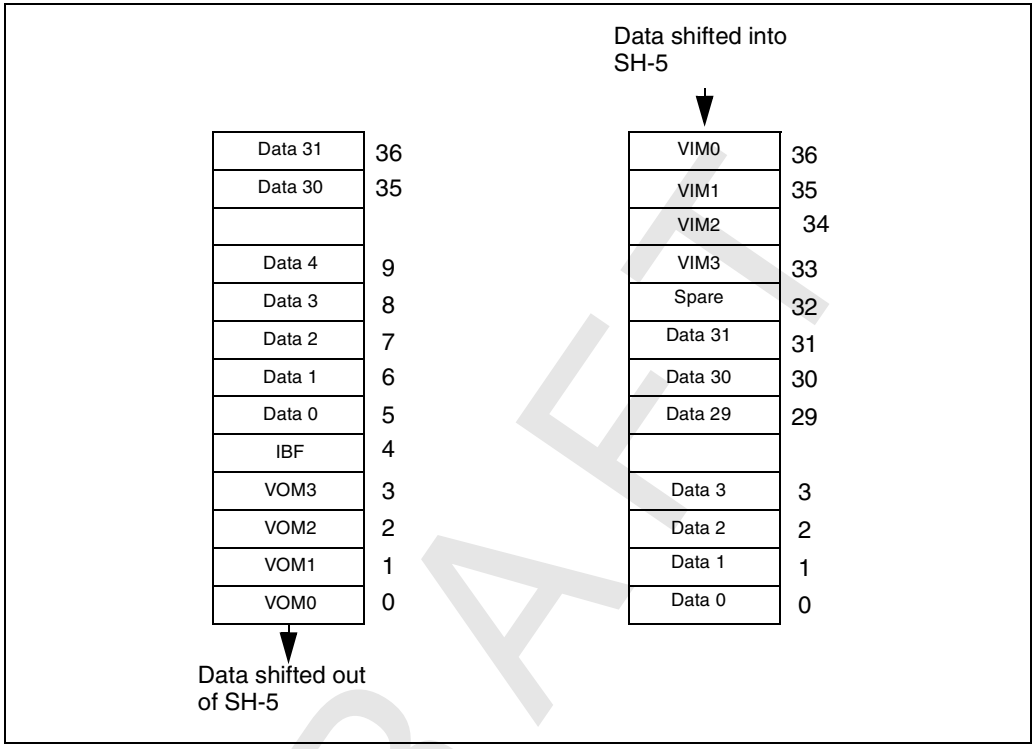

Figure 13: JTAG data transferred using variable shift length

Polling to detect a SH-5 pending output message

In order that the tool can detect when SH-5 has a pending output message and to determine when the input buffer is available for a new input message, the tool must poll the JTAG debug logic at regular intervals to shift out the IBF and VOMx status bits.

The VOMx status bits are located closest to the TDO end of the shift register. The tool can therefore poll out the VOMx and IBF status by shifting just five bits out of the shift register following the capture-DR state. At the same time as these five status bits are being shifted out of the shift register, the VIM status bits are shifted into the input end of the shift register from the TDI pin and is latched during the update-DR state. When the tool has no pending input message to send, and it simply wants to poll the IBF and VOMx status bits, the tool sets all VIMx bits to '0' during the five shift cycles.

Polling out a message

After the five status bits have been shifted out, the tool can determine that an output message exists and then continues shifting a further 8, 16, 24 or 32 times depending on which VOMx bits = ‘1’. The tool now has assembled the first bytes of an output message.

The tool continues this process of shifting out just the IBF and VOMx bits, testing the VOMx bits and then shifting data bits depending on which VOMx bits = ‘1’. The detection of any VOMx bit = ‘0’ indicates the end of the message. Once the end of the message has been reached, the tool does not need to shift any more data bits out.

Tool sending a message to SH-5

Input messages, longer than 4-bytes, are sent as one or more 4-byte segments possibly followed by a segment containing fewer bytes. For each 4-byte segment of a message, the tool shifts 37 bits into TDI with the VIMx status bits in the last positions (the positions closest to TDI). For the last segment of a message containing fewer than 4 bytes, the tool shifts 13, 21 or 29 bits into TDI with appropriate VIMx bits indicating the number of valid bytes.

During message transfers, the first VIMx bit = '0' indicates the end of the input message. When the length of an input message is a multiple of 4 bytes, the end of the message is indicated by a 1-byte segment (13 bits) with all VIMx bits = ‘0’.

Flow control

The SH-5 JTAG tap controller has an input message buffer large enough to hold the largest input message plus an output message buffer large enough to hold the largest output message. The DBUS protocol allows one outstanding response in each direction, so it is possible that the tool may want to send a response message immediately following a new request. Flow control is needed to eliminate the possibility of ever having a message in the input buffer overwritten before its has been moved into the DM. This flow control is done by sending including an input buffer status bit in the DR register, adjacent to the VOM3 bit. The tool can poll out just five bits (if it can handle variable-length shifting), one of which determines whether the input buffer can accept a new message and the other four determine whether there is an output message pending.

There is no need for a tool input buffer status bit in the other direction. The tool will only poll SH-5 when its input buffer has space for a new message from SH-5.

3.4 Debug tool reset/suspend behavior

3.4.1 DEBUG reset

DEBUG reset is similar to POWERON reset except:

1 It does not reset debug registers in the WPC or DM.

2 It does reset the DM's internal state. The following DM features are reset to their power on state:

- DM FIFO,
- DM capture buffers (from WPC and bus analyzers),
- Any pending debug interrupts are cleared,
- Any CPU stall signals applied from the DM to the CPU are cleared.

As described in 2 above, DM internal state is reset. Thus DM registers which reflect this state (for example, the FIFO registers) are updated in the normal manner to correspond to the new internal state of the debug system.

3 It does not reset the active debug link.

Thus the active debug link (either SHdebug link or JTAG) remains the same.

DEBUG reset should not be applied whilst the active debug link (either the SHdebug link or the JTAG port) is transferring information, or undefined behavior will occur.

As explained in *Section 1.7.1: RESVEC/DBRVEC selection on page 74*, DEBUG reset always vectors through RESVEC.

3.4.2 Reset functions available from debug tools

Debug tool connected via SH debug-link

A debug tool such as an ST JEI debug adapter connects to a SH-5 board-level product via a SHdebug link header. The tool is able to directly reset SH-5 using the RESET signal on the header. A jumper on the target board connects this signal to either the RESETP pin or the RESETM pin.

During the reset sequencing initiated by a pulling either RESETP or RESETM low, SH-5 senses the state of the DM_IN. If DM_IN is low, a DEBUG reset is initiated regardless of whether the RESETP pin or the RESETM pin was asserted.

DEBUG reset from an SHdebug link tool can also be performed by writing to the WPC.CPU_CTRL_ACTION register.

Signal	Source	Meaning
DM_CLKOUT	From SH-5	SHdebug link output clock.
DM_OSYNC	From SH-5	SHdebug link output sync.
DM_OUT[3:0]	From SH-5	SHdebug link output data.
DM_CLKIN	From tool	SHdebug link input clock. Also debug module enable/disable control, sampled at the rising edge of RESETP/RESETM. Value - Description 0: The debug module is enabled following reset. 1: The debug module is disabled with its clock source turned off following reset.
DM_ISYNC	From tool	SHdebug link input sync. Also CPU suspend mode (known as the multi-function SUSPEND pin), sampled at the rising edge of RESETP/RESETM. Value - Description 0: CPU remains suspended following reset. 1: CPU operates normally following reset.
DM_IN	From tool	SHdebug link input data. Also provides the RESET_MODE signal. The value is sampled during the entire period when RESETP/RESETM are low. Value - Description 0: Forces a DEBUG reset regardless of whether the RESETP or RESETM pin is asserted. 1: A normal POWERON reset or MANUAL reset is initiated when the corresponding reset pin is asserted.
RESET	Bi-directional	Reset signal driven by Tool (open drain). Tool can also monitor this signal to detect when board-level reset is initiated.

Table 76: SHdebug link header signals

Debug tool connected via JTAG

A debug tool such as a Hitachi E20 connects to a SH-5 board-level product via a JTAG debug header. For debug tools such as a Hitachi E20, the RESET signal in the JTAG interface is an output from the target board which allows the tool to detect when a board-level reset function has occurred, for example, when a user has pressed the reset button.

The E20 does not have the capability to initiate a POWERON, MANUAL or DEBUG reset via signals in the interface. However, the E20 can perform either a CPU reset or a DEBUG reset by writing to the WPC.CPU_CTRL_ACTION register. The E20 has some control over the type of reset performed by a board-level reset button. A RESET_MODE signal is assigned to an E20 pin not currently connected and this allows the tool to force a DEBUG reset when the reset button on the board is pressed.

With an appropriate design of target board, the RESET signal in the JTAG debug interface could be bi-directional allowing the tool to initiate one type of hard reset, either POWERON, MANUAL or DEBUG depending on board-level jumpers.

Signal	Source	Meaning
TCK	From tool	JTAG clock
TDI	From tool	JTAG data in
TDO	From SH-5	JTAG data out
TMS	From tool	JTAG test mode select
TRST	From tool	JTAG reset. The tap controller finite state machine is reset by TRST going low. This pin has no effect on other chip functions.
RESET	Bi-directional	Tool can monitor this signal to detect when board-level reset is initiated.
SUSPEND	From tool	CPU suspend mode following reset, sampled at the rising edge of RESETP/RESETM. Refer to *Section 3.4.3: CPU suspend function on page 223* for more details. Value - Description 0: CPU remains suspended following reset 1: CPU operates normally following reset

Table 77: JTAG debug header signals

Signal	Source	Meaning
RESET_MODE	From tool	Allows the tool to determine the type of reset function performed. The value is sampled during the entire period when NOTRESETP/NOTRESETM are low. Value - Description 0: Forces a DEBUG reset regardless of whether the NOTRESETP or NOTRESETM pin is asserted. 1: A normal POWERON reset or MANUAL reset is initiated when the corresponding reset pin is asserted.
DM_ENABLE	From tool	debug module state following reset, sampled at the rising edge of NOTRESETP/NOTRESETM. Refer to *Section 3.2.5: Pin state during reset on page 203* for more details. Value - Description 0: The debug module is enabled following reset. 1: The debug module is disabled with its clock source turned off following reset.

Table 77: JTAG debug header signals

3.4.3 CPU suspend function

The DM_ISYNC signal has two functions. Its primary function is the synchronization pin for messages sent to SH-5 from a SHdebug link-connected tool. Its secondary function controls CPU suspend state.

At the end of a POWERON, MANUAL or DEBUG reset function, when NOTRESET is pulled high, the CPU can either start executing boot code or can enter a suspended state depending on the state of the DM_ISYNC signal sampled when NOTRESET goes from low to high. If DM_ISYNC is sampled low at the end of the reset phase, the CPU remains suspended on the assumption that various SH-5 registers will be loaded by a connected tool. At some later time, the tool will release the CPU from its suspended state by writing to the WPC.CPU_CTRL_ACTION register. If DM_ISYNC is sampled high at the end of the reset phase, the CPU starts executing boot code.

This DM_ISYNC pin has an internal pull-up resistor to ensure that when no debug tool is connected to a debug connector, the CPU is not suspended at the end of reset.

The CPU suspend function is also available to JTAG-connected tools. The JTAG debug header signal SUSPEND_ is an AC-decoupled version of the DM_ISYNC pin. Since DM_ISYNC is a high-speed signal used by the SHdebug link, board-level products must include a series resistor between SUSPEND_ pin in the JTAG header and the DM_ISYNC pin. This resistor (of value around 1K ohm) must be located close to the DM_ISYNC pin to minimize the effect of the extra trace length on the printed circuit board. A bypass capacitor is also required.

Summary of all functions using RESETP and RESETM

DM_IN state	Action when NOTRESETP asserted	Action when NOTRESETM asserted
High (internal pull-up resistor)	POWERON reset	MANUAL reset
Low	DEBUG reset	DEBUG reset

Table 78: Alternative reset functions

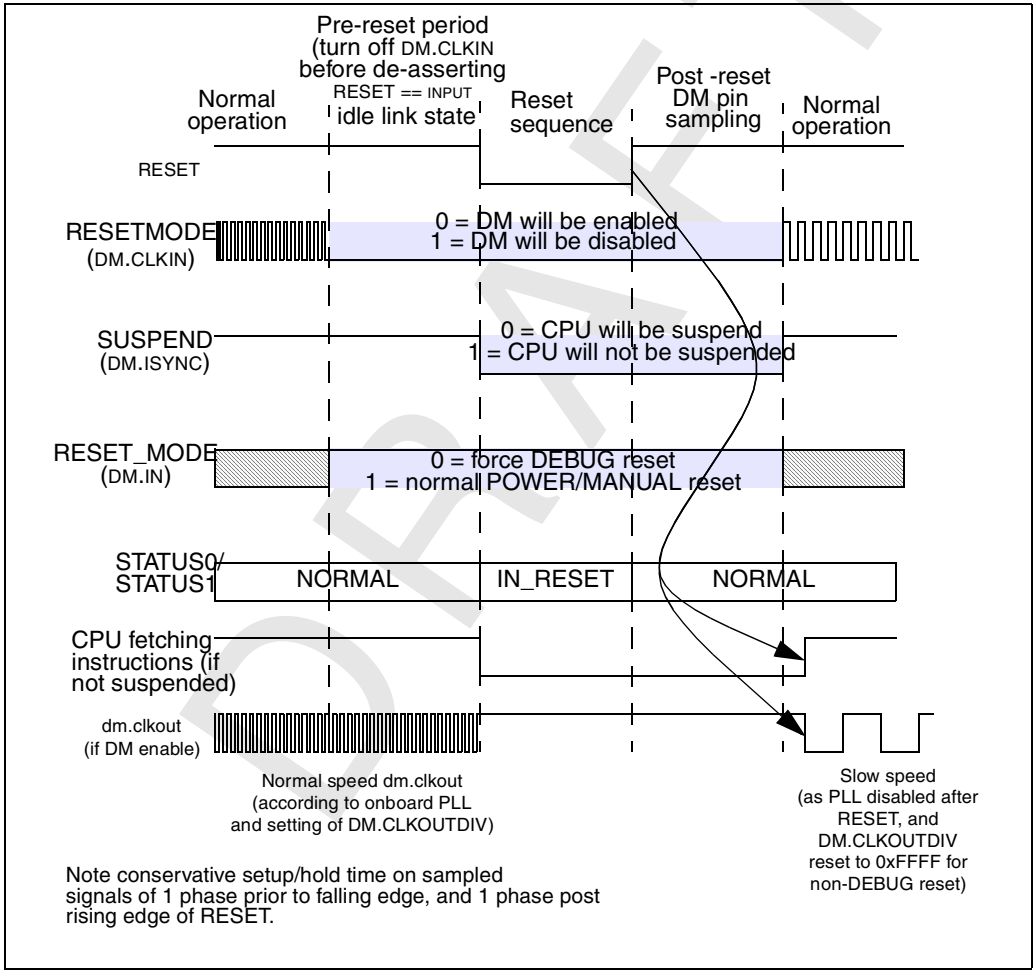

Figure 14: POWERON, MANUAL or DEBUG Resetting (via Reset Signal)

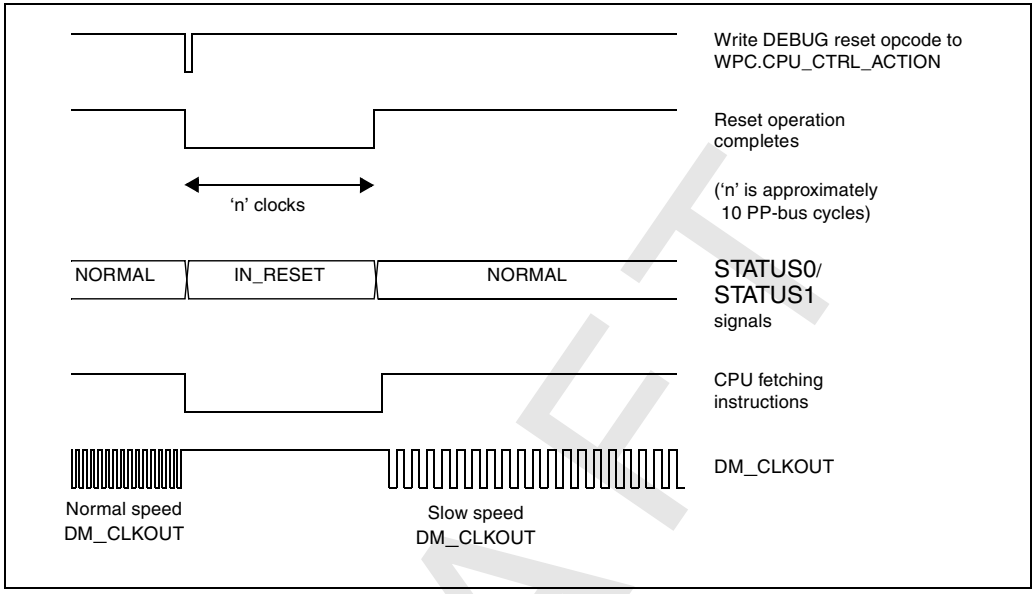

Figure 15: DEBUG reset (via write to WPC.CPU_CTRL_ACTION)

3.5 Trigger functions

SH-5 provides a trigger-in (DM_TRIN_N) and a trigger-out pin (DM_TROUT_N). These allow external analysis hardware (such as a logic analyzer) to control and monitor specific watchpoints.

The trigger out pin can also be configured to provide external visibility of timing events (such as interrupt latency), and to detect internal states (such as trace buffer overflow).

Refer to *Section 1.8.10: DM.TRCTL (trace/trigger register) on page 97* for details of the control register fields that determine the functions of the trigger-in and trigger-out pins.

3.6 DBUS protocol

3.6.1 Overview

DBUS supports the following command types:

- **Load8** - read 8 bytes (1 * 64 bit word)
- **Load16** - read 16 bytes (2 * 64 bit words)
- **Load32** - read 32 bytes (4 * 64 bit words)
- **Store8** - write 8 bytes (1* 64 bit words)
- **Store16** - write 16 bytes (2 * 64 bit words)
- **Store32** - write 32 bytes (4 * 64 bit words)
- **Swap8** - swap 8 bytes (1 * 64 bit word)
- **Flush** (address)
- **Purge** (address)

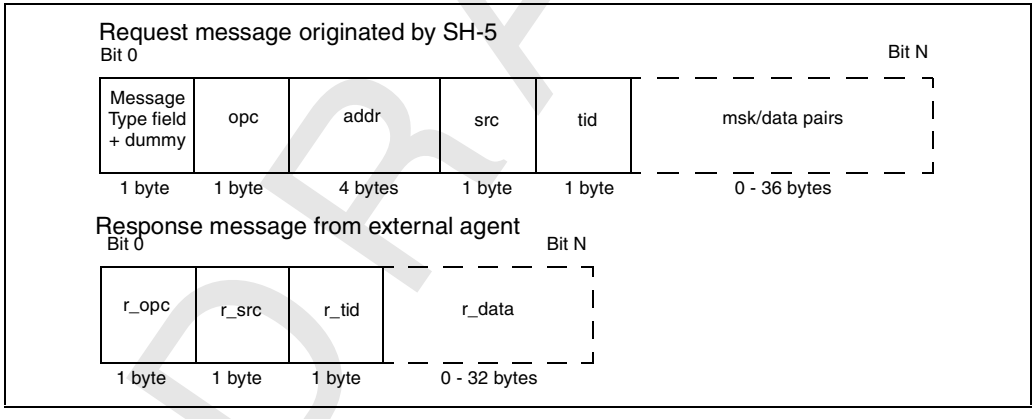

Figure 16: DBUS transaction originated by SH-5

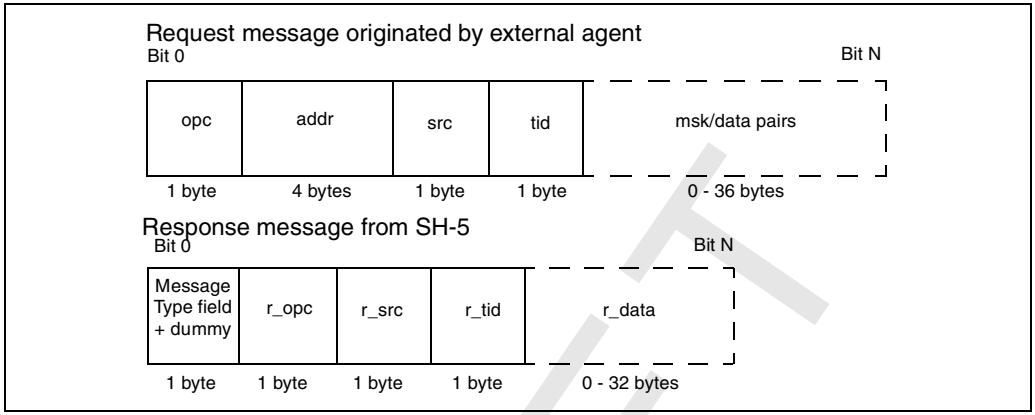

Figure 17: DBUS transaction originated by external agent

Opcode (opc, r_opc)

Opcode is a 1-byte field which defines the contents of the message.

Address (addr)

For request messages originated by the CPU or other SuperHyway module addressed to the SHdebug link portion of the debug module address space (that is, the remote memory area which is hosted on the development host or debug adaptor), the top 8-bits of the address field contain the SuperHyway device code for the debug module.

For request messages originated by the development host or debug adapter, the top 8-bits of the address field identifies the destination SuperHyway device. The top 8 bits of the address must not be the value associated with the remote memory area. The SH-5 behavior is undefined if such an access is attempted.

Source (src, r_src)

This field identifies the source of the request. When generating a response, the tool must ensure that the value in the source field matches the Source field of the original request.

Mask (msk)

This field appears in request transactions, and is used to define which bytes within the data field are significant.

Note: For load requests the DBUS protocol only transfers a single 8-bit msk value, which is used for the series of underlying 8-byte load operations.

The address of individual bytes within the data field is determined by the msk value, and the relationship between these values varies according to the endian mode of the SH-5 system. This is described in *Section 3.6.6: Endian-specific behavior on page 230*.

TID (tid, r_tid)

Transaction identifier. When generating a response, R_TID must match the TID value of the original request.

Dummy

Dummy bits which are included in order to pad the packet to certain alignment and size constraints. Any value can be supplied for the dummy bits.

3.6.2 Nibble order

For SH-5 implementations using a 4-bit-wide output data port (DM_OUT[0,3]), the least-significant nibble of each byte is transmitted first, that is, BYTE-I[0,3] followed by BYTE-I[4,7]. See *Table 73: DBUS Load8 request message example on page 210* for more details.

3.6.3 Pipelining of DBUS requests

The SHdebug link does not support pipelined DBUS requests. The debug module will accept only a single SuperHyway request addressed to the SHdebug link and will forward this over the SHdebug link as a DBUS message. When the DBUS response is received by the debug module from the tool, it is forwarded to SuperHyway. Only then will the debug module accept another SuperHyway request addressed to the SHdebug link.

This means that the transaction ID field in the DBUS message has no useful function since there can never be more than one outstanding transaction. However, checking of the TID in the response is still performed by the SuperHyway initiator and the tool must ensure that the TID in the response message generated by the tool is identical to the TID in request message received from SH-5.

3.6.4 Unsolicited responses

Undefined effects will occur if a tool sends an unsolicited DBUS response to the SH-5.

3.6.5 Critical word ordering

For multiple data phase transactions (that is, **Load16**, **Store16**, **Load32**, **Store32**), the address on the SuperHyway request bus is the address of the first data word of the transaction. The burst address ordering is linear. Burst requests issued by the MMU/cache are always critical-word-first and the low order bits of the address determine the word order of **Store** requests and of **Load** responses as shown in *Table 79* and *Table 80* below.

Address bits[0, 3]	Word order
0b0xxx	0, 1
0b1xxx	1, 0

Table 79: Word order for Store16, Load16

Address bits[0,4]	Word order
0b00xxx	0, 1, 2, 3
0b01xxx	1, 2, 3, 0
0b10xxx	2, 3, 0, 1
0b11xxx	3, 0, 1, 2

Table 80: Word order for Store32, Load32

3.6.6 Endian-specific behavior

The msk field determines which bytes in the data field are significant. The manner in which the bytes within the data field correspond to offsets from the address field is dependant on the endianness of the target system.

This is illustrated in *Table 81*.

Data size	Mask	Byte positions within 8-byte data field								Address offset in endian mode:	
		7	6	5	4	3	2	1	0	Big	Little
8	0xFF	MSB							LSB	0	0
4	0x0F					MSB			LSB	4	0
4	0xF0	MSB			LSB					0	4
2	0x03							MSB	LSB	6	0
2	0x0C					MSB	LSB			4	2
2	0x30			MSB	LSB					2	4
2	0xC0	MSB	LSB							0	6
1	0x1									7	0
1	0x2									6	1
1	0x4									5	2
1	0x8									4	3
1	0x10									3	4
1	0x20									2	5
1	0x40									1	6
1	0x80									0	7

Table 81: DBUS mask/address mapping for big and little endian modes

3.6.7 Opcode definition

Opcode type	Value		Name	Reference
	Binary	Hexadecimal		
Requests	0b00110001	0x31	**Load8**	*Load8 on page 232*.
	0b01000001	0x41	**Load16**	*Load16 on page 233*.
	0b01010001	0x51	**Load32**	*Load32 on page 234*
	0b00110010	0x32	**Store8**	*Store8 on page 235*
	0b01000010	0x42	**Store16**	*Store16 on page 236*
	0b01010010	0x52	**Store32**	*Store32 on page 237*
	0b00110101	0x35	**Swap8**	*Swap8 - swap 8 bytes on page 238*
	0b00011000	0x18	**Flush**	*Flush - flush a physical address on page 239*
	0b00001000	0x8	**Purge**	*Purge - purge a physical address on page 240*
Responses	0b10000000	0x80	**Success**	
	0b10000001	0x81	**Failure**	

Table 82: DBUS opcode values

3.6.8 DBUS transactions

Load8

Request

OPC[7:0]	Opcode **Load8** (0x31).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:0] specifies the offset within the module. ADDR[2:0] is not significant.
SRC[0, 7]	Source identifier, defined by the system not the module.
TID[0, 7]	Transaction identifier, defined by the module not the system.
MSK[7:0]	Byte significance within data ('1' == significant, '0' == not significant). Where the target device is not read sensitive it is acceptable to the target to read all bytes, however the initiator must assume bytes for which MSK == '0' are invalid. See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.
R_DATA[7:0]	Response data, see MSK for significance of each byte.

Load16

Request

OPC[7:0] Opcode **Load16** (0x41).

ADDR[31:0] Address, where ADDR[31:24] identifies the module.

ADDR[23:4] specifies the offset within the module.

ADDR[3] specifies the address of the critical word within a 16 byte quantity (see *Section 3.6.5: Critical word ordering on page 229*).

ADDR[2:0] is not significant.

SRC[7:0] Source identifier, defined by the system not the module.

TID[7:0] Transaction identifier, defined by the module not the system.

MSK[7:0] Byte significance within data ('1' == significant, '0' == not significant).

A single 8 bit mask value is supplied with the request, this value is used for the two 8 byte loads which will be performed by this transaction.

When dealing with **Load16** requests issued from SH-5 to the debug link, the off-chip protocol handling software and hardware can be simplified by ignoring the mask, and thus supplying the R_DATA field of the response with all bytes containing valid data (that is, as if MSK was supplied as a 16-bit value == 0xFFFF).

When a debug tool issues a **Load16** request to SH-5, it must be aware that SH-5 will use the same msk value for both of the 8 byte load operations.

See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.

Response

R_OPC[7:0] Response type, either 0x80 for **Success**, or 0x81 for **Failure**.

R_SRC[7:0] Copy of SRC.

R_TID[7:0] Copy of TID.

R_DATA[127:0] Response data, see MSK for significance of each byte.

Load32

Request

OPC[7:0]	Opcode **Load32** (0x51).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:5] specifies the offset within the module ADDR[4:3] specifies the address of the critical word within a 32 byte quantity (see *Section 3.6.5: Critical word ordering on page 229*). ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
MSK[7:0]	Byte significance within data (‘1’ == significant, ‘0’ == not significant). A single 8 bit mask value is supplied with the request, this value is used for the four 8 byte loads which will be performed by this transaction. When dealing with **Load32** requests issued from SH-5 to the debug link, the off-chip protocol handling software and hardware can be simplified by ignoring the mask, and thus supplying the R_DATA field of the response with all bytes containing valid data (that is, as if MSK was supplied as a 32-bit value of 0xFFFFFFFF). When a debug tool issues a **Load32** request to SH-5, it must be aware that SH-5 will use the same MSK value for the four 8 byte load operations. See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.
R_DATA[255:0]	Response data.

Store8

Request

OPC[7:0]	Opcode **Store8** (0x32).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:3] specifies the offset within the module. ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
MSK[7:0]	Byte significance within word ('1' == significant, '0' == not significant). See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.
DATA[63:0]	Data to write.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.

Store16

Request

OPC[7:0]	Opcode **Store16** (0x42).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:4] specifies the offset within the module. ADDR[3] specifies the address of the critical word within a 16 byte quantity (see *Section 3.6.5: Critical word ordering on page 229*). ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
MSK0[7:0]	Byte significance within word ('1' == significant, '0' == not significant). See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.
DATA0[63:0]	First 8 byte quantity to write.
MSK1[7:0]	As MSK0 but for DATA1 rather than DATA0.
DATA1[63:0]	Second 8 byte quantity to write.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.

Store32

Request

OPC[7:0]	Opcode **Store32** (0x52).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:5] specifies the offset within the module. ADDR[4:3] specifies the address of the critical word within a 32 byte quantity (see *Section 3.6.5: Critical word ordering on page 229*). ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
MSK0[7:0]	Byte significance within word (‘1’ == significant, ‘0’ == not significant). See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.
DATA0[63:0]	First 8 byte quantity to write.
MSK1[7:0]	As MSK0 but for DATA1 rather than DATA0.
DATA1[63:0]	Second 8 byte quantity to write.
MSK2[7:0]	As MSK0 but for DATA2 rather than DATA0.
DATA2[63:0]	Third 8 byte quantity to write.
MSK3[7:0]	As MSK0 but for DATA3 rather than DATA0.
DATA3[63:0]	Fourth 8 byte quantity to write.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.

Swap8 - swap 8 bytes

Perform a 8 byte write and 8 byte read atomically.

Request

OPC[7:0]	Opcode **Swap8** (0x35).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module. ADDR[23:3] specifies the offset within the module. ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
MSK[7:0]	Byte significance within word ('1' == significant, '0' == not significant). See *Table 81: DBUS mask/address mapping for big and little endian modes on page 230*.
DATA[63:0]	Data to write.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.
R_DATA[7:0]	Data read.

Flush - flush a physical address

Forces any data held by a module and associated with a physical address to be made coherent with the actual data held at the physical address. The target device may retain a copy of the data.

Request

OPC[7:0]	Opcode **Flush** (0x18).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module.
	ADDR[23:3] specifies the offset within the module.
	ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
DUMMY[7:0]	Dummy bits.
DATA[63:0]	Address to be flushed.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.

Purge - purge a physical address

Forces any data held by a module and associated with a physical address to be made coherent with the actual data held at the physical address whilst removing any copies of the data held in the target module.

Request

OPC[7:0]	Opcode **Purge** (0x8).
ADDR[31:0]	Address, where ADDR[31:24] identifies the module.
	ADDR[23:3] specifies the offset within the module.
	ADDR[2:0] is not significant.
SRC[7:0]	Source identifier, defined by the system not the module.
TID[7:0]	Transaction identifier, defined by the module not the system.
DUMMY[7:0]	Dummy bits.
DATA[63:0]	Address to be purged.

Response

R_OPC[7:0]	Response type, either 0x80 for **Success**, or 0x81 for **Failure**.
R_SRC[7:0]	Copy of SRC.
R_TID[7:0]	Copy of TID.

4 Implementation specifics

4.1 Scalable parameters

The number of various debug facilities is scalable.

This section defines these specifics for the SH-5 evaluation device.

4.1.1 WP channels

SH-5 evaluation device provides the following channels:

Base channel name	Locality	Function	Number of channels	Actual channel names
BRK	WPC	CPU	1	BRK0
IA	WPC	Instruction address	4	IA0 thru IA3
OA	WPC	Operand address	2	OA0 thru OA1
IV	WPC	Instruction value	2	IV0 thru IV1
BR	DM	Branch trace	1	BR0
FPF	DM	Fast printf	1	FPF0
PL	DM	SuperHyway bus analyzer	2	PL0 thru PL1
DM	DM	Debug module	1	DM0
WPC_PERF	WPC	CPU performance monitor	2	WPC_PERF0 thru WPC_PERF1

Table 83: SH-5 evaluation device WP channels

4.1.2 Event counters

SH-5 evaluation device provides two 16-bit event counters

ECOUNT.SIZE = 16	
Event counter ID (4 bits)	**Name**
0b0000	WPC.ECOUNT_VALUE_0
0b0001	DM.ECOUNT_VALUE_0
All other values	Undefined

Table 84: SH-5 evaluation device event counter Ids

One event counter is implemented in the WPC (for timing critical exception matching). This counter is only accessible to WPC implemented channels.

One event counter is implemented in the DM. This counter is only accessible to the PL watchpoint channels.

Event counter latency

Table 85 shows the latency for the event counters.

Event counter name	Delay from WPC watchpoint hit (CPU cycles)		Delay from Bus Analyzer watchpoint hit (DM cycles)
	For use in precondition check	For use in subsequent counter update	
WPC.ECOUNT_VALUE_X	4	1	Not applicable
DM.ECOUNT_VALUE_X	Not applicable	Not applicable	TBD

Table 85: Update latency for event counters

The latency for WPC.ECOUNT_VALUE_X means that the new value is visible between 1 and 4 instructions after the one that hits a WPC channel and causes the change in value. Put another way, between 0 and 3 following instructions will use the old value of WPC.ECOUNT_VALUE_X in their precondition checks. The actual number of instructions involved depends on whether stalls occur in the pipeline (for example, due to resource or operand dependencies, or delays in instruction fetching).

However, the new value will be used immediately if the following instruction also requires a decrement to the counter. That is, no updates are lost if a succession of instructions each cause a decrement to the counter.

4.1.3 Performance counters

SH-5 evaluation device provides four 48 bit performance counters, two located within the WPC and two located in the DM.

PCOUNT.SIZE = 48		
Performance counter ID (4 bits)	**Name**	**.PCOUNT_ID availability**
0b0000	WPC.PCOUNT_VALUE_0	Only for WPC channels.
0b0001	WPC.PCOUNT_VALUE_1	
0b0010	DM.PCOUNT_VALUE_0	Only for DM channels.
0b0011	DM.PCOUNT_VALUE_1	
All other values	Undefined	

Table 86: SH-5 evaluation device performance counter Ids

4.1.4 Chain latches

SH-5 evaluation device provides 6 chain latches, these latches are not all globally accessible.

The WPC based latches are only available to WPC implemented channels, expect the IA chain latches which are globally readable. The DM based latches are available to all WP channels.

Setting a WP channel to affect unavailable chain-latches will result in undefined behavior.

Chain-latch ID (4-bits)	Latch name	Pre-condition for WPC watchpoints	Pre-condition for DM and bus analyzer watchpoints	Changeable by watchpoint hits
0x0	WPC.IA0_CHAIN	OA, IV, WPC_PERF	FPF, BR, PL	Auto by IA0
0x1	WPC.IA1_CHAIN	OA, IV, WPC_PERF	FPF, BR, PL	Auto by IA1
0x2	WPC.IA2_CHAIN	OA, IV, WPC_PERF	FPF, BR, PL	Auto by IA2
0x3	WPC.IA3_CHAIN	OA, IV, WPC_PERF	FPF, BR, PL	Auto by IA3
0x8	WPC.CHAIN_0	IA, OA, IV, WPC_PERF	FPF, BR, PL	IA, OA, IV
0xC	DM.CHAIN_0	IA, OA, IV, WPC_PERF	FPF, BR, PL	IA, OA, IV, PL
0xF	DM.CHAIN_TRIG_IN	IA, OA, IV, WPC_PERF	FPF, BR, PL,	IA, OA, IV, PL
All other values	undefined			

Table 87: SH-5 evaluation device chain latch IDs

There is not global connectivity for the ACTION_CHAIN_ALTER actions between the WPC and DM based WP channels. Thus, the appropriate registers must be used to achieve this effect - for a WPC channel to alter a DM chain latch, the WPC channel's DM.WP_NX_ACTION.ACTION_CHAIN_ALTER field and the CHAIN_ID fields are used (the channel's WPC.WP_NX_ACTION.ACTION_CHAIN_ALTER field should be set to not alter a chain-latch).

Chain-latch latency

Due to inter-module delays, not all chain latches change state immediately upon the detection of a watchpoint hit. *Table 88* specifies the latency for each of the chain-latch groups.

Latch name	Delay from WPC watchpoint Hit (CPU clocks)	Delay from bus analyzer watchpoint hit (DM clocks)
WPC.IAX_CHAIN	1	Not applicable
WPC.CHAIN_X	4	Not applicable
DM.CHAIN_X	Not specified - see below.	1
Latch name	**Cycles from pin assertion and chain latch state change (DM Clocks)**	
DM.CHAIN_TRIGIN	3 to 4 (depending on resynchronization)	

Table 88: Chain-latch latency

The latency for the WPC.IAX_CHAIN latches means that the new value is always visible to the following instruction.

The latency for the WPC.CHAIN_X latch means that the new value is visible between 1 and 4 instructions later than the instruction that causes the change to the latch value. Put another way, between 0 and 3 following instructions will use the old value of WPC.CHAIN_X in their precondition checks. The actual number of instructions involved depends on whether stalls occur in the pipeline (for example, due to resource or operand dependencies, or delays in instruction fetching).

Latency for DM.CHAIN_x

No figures are specified for the latency of DM.CHAIN_X in association with the watchpoint controller channels. The timing relationship between particular SuperHyway transactions and particular instructions being executed in the CPU is partially asynchronous, due to the presence of caches and instruction buffers inside the CPU core.

There is no general way to specify whether the altering of DM.CHAIN_X by a bus analyzer hit on a particular SuperHyway transaction will or will not influence the precondition checks for a particular CPU instruction on the WPC's IA/IV/OA channels.

Equivalently, if DM.CHAIN_X is altered as a result of an IA/IV/OA channel hit by a particular CPU instruction, there is no general way to specify whether this will or will not influence the precondition checks for the bus analyzer channels on a particular SuperHyway transaction. Moreover, the number of outstanding hits in the debug module's capture buffer also affects the time taken between the watchpoint hit and the update to DM.CHAIN_X.

4.1.5 DM FIFO

SH-5 evaluation device provides a DM FIFO of 504 bytes.

This is arranged a 21 entries each 3*64-bits wide.

Trace entries are packed into the DM FIFO at constant 3*64 bit intervals, thus the DM FIFO can hold a maximum of 21 trace messages at a time.

DM FIFO high-water mark

SH-5 evaluation device's FIFO high-water mark is 8 entries from the FIFO-full condition.

DM capture buffer high-water mark

SH-5 evaluation device's capture buffer high-water mark is 8 entries from the full condition.

When stall mode is selected and a high-water mark detection occurs, the CPU will be stalled in time to avoid overfilling the capture buffer.

Clearing the DM FIFO in DM FIFO trace hold or circular DM FIFO mode.

Even when the source of trace generation are disabled, there is a bounded period when trace can be generated from previously occurring events which are held in the capture buffer.

Thus the recommended sequence for clearing the FIFO is:

```
... disable all sources of trace generation
while (DM.TRCTL.ff_status != empty) {
  -- attempt to clear it
  write DM.TRCTL.ff_clear = 1

  -- loop for the implementation defined limit at which captured
trace may be generated
  tries = 0;
  while ((DM.ff_status != empty) &&
         (tries < MAX_CLEAR_RETRIES)) {
    tries++;
  }

}
```

Note: For SH-5 evaluation device, MAX_CLEAR_RETRIES is 400.

4.1.6 Trace message header fields

SH-5 evaluation device provides the following watchpoint channels which can generate trace messages:

Channel name	Function	Source_module code	Event_type code
IA0	Instruction address	0	0x00
IA1	Instruction address	0	0x01
IA2	Instruction address	0	0x02
IA3	Instruction address	0	0x03
OA0	Operand address	0	0x04
OA1	Operand address	0	0x05
IV0	Instruction value	0	0x06
IV1	Instruction value	0	0x07
BR	Branch trace	0	0x08

Table 89: SH-5 evaluation device trace message codes

Channel name	Function	Source_module code	Event_type code
FPF	Fast printf	0	0x09
PL0	SuperHyway bus analyzer	1	0x00
PL1	SuperHyway bus analyzer	1	0x01

Table 89: SH-5 evaluation device trace message codes

4.1.7 Action and trace generation timing

Within the debug module, the state machine which extracts watchpoint hit entries from the capture buffer and processes actions defined by the associated DM.WP_X_ACTION register requires the following DM clock cycles to perform the actions:

Action	Number of DM clock periods to perform the action
All actions except trace message generation	4
Trace message generation	8 plus one clock cycle for each trace message field[a].

Table 1 SH-5 evaluation device action timing

a. The 16-bit header of each message is considered to be a single field. The size of the field does not affect the number of clock cycles, for example, a 1-byte ASID field and a 4-byte full address field both take one clock cycle to process.

Example

A branch trace message consists of three fields (header, source address, destination address) and can vary in size from 4 bytes to 11 bytes. It takes 11 DM clock periods to extract the entry from the capture buffer and generate a branch trace message, which is the same as 22 CPU clock periods.

4.1.8 Timestamping

Trace messages are generated by emptying information from the capture buffer(s) into the DM FIFO. If the capture buffer(s) or DM are full, the trace message will not be generated until space is available.

Thus the timestamp value generated in the trace message is that at the point of generation (TGEN), not the point of WP triggering (TTRIG).

Normally TGEN == TTRIG. However if the capture buffer(s) or DM FIFO are full and trace message generation is delayed, TGEN = (TTRIG + TDELAY) where TDELAY is the number of timestamp cycles the generation was delayed for. Thus TTRIG > TGEN can also result in reference messages being generated to reseed the timestamp counter.

This is not regarded as a problem, as all trace messages are generated in the same manner, and thus the timestamp values are all in the same time domain, and typically TDELAY will range from 0 to a small number of cycles.

4.1.9 Trigger out pulse width

SH-5 evaluation device's (low-going) trigger-out pulse width is a minimum of 1 DM clock cycle followed by a recovery period also of a minimum of 1 DM clock cycle.

4.1.10 JTAG IR DEBUG codes

DEBUG - the JTAG instruction code used to enable the JTAG port as the debug link is 0b11100.

WAKEUP - the JTAG instruction code used to wake a system from a standby state is 0b10101.

4.1.11 DM.VCR register

The definition of DM.VCR for this implementation is given below..

DM.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PERR_FLAGS	[7:0]	8	✓	P-port error flags	Varies
	Operation		Standard PERR_FLAGS, see System Architecture Volume 1.		
	When read				
	When written				
	HARD reset		0		
MERR_FLAGS	[15:8]	8	✓	P-module error flags (module specific)	Varies
	Operation		Standard MERR_FLAGS, see System Architecture Volume 1.		
	When read				
	When written				
	HARD reset		0		
MOD_VERS	[31:16]	16	—	Module version	RO
	Operation		Used to indicate module version number		
	When read		Returns 0x0		
	When written		Ignored		
	HARD reset		0x0		
MOD_ID	[47:32]	16	—	Module identity	RO
	Operation		Used to identify module		
	When read		Returns 0x1823		
	When written		Ignored		
	HARD reset		0x1823		

Table 90: DM.VCR

DM.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
BOT_MB	[55:48]	8	—	Bottom memory block	RO
	Operation		Used to identify bottom memory block		
	When read		Returns 0x01		
	When written		Ignored		
	HARD reset		0x01		
TOP_MB	[63:56]	8	—	Top memory block	RO
	Operation		Used to identify top memory block		
	When read		Returns 0x01		
	When written		Ignored		
	HARD reset		0x01		

Table 90: DM.VCR

4.1.12 Bus analyzer module/SuperHyway mapping

The mapping from physical module number to SuperHyway module numbers as used by the PL_MODULE field of DM.WP_PLX_ACTION (*Table 20: DM.WP_PLx_ACTION register definition on page 60*) is given below:

DM.PL*x*_ACTION.pl_module	DM.PL*x*_FRZ.freeze_x	SuperHyway module name	SuperHyway source iD
0	bit 0	S5 CPU	0x0D
1	bit 1	DM	0x0C
2	bit 2	DMA	0x0E
3	bit 3	PCI	0x60
4	bit 4	SuperHyway socket	0x70
5	bit 5	F_EMI	0x08

Table 91: DM.PLx_ACTION.pl_module/DM.PLX_FRZ.freeze_x/SuperHyway module mapping

DM.PL*x*_ACTION.pl_ module	DM.PL*x*_FRZ.freeze_x	SuperHyway module name	SuperHyway source iD
All other values	All other bit positions	Unused	Undefined

Table 91: DM.PLx_ACTION.pl_module/DM.PLX_FRZ.freeze_x/SuperHyway module mapping

Bus analyzer destination mask/value implementation

The current SuperHyway implementation does not provide the bus analyzer with SRC information on response segments. Therefore the bus analyzer's destination mask/value fields and comparators are not implemented, and thus behave as if the destination always matches.

4.2 Debug register address map

All the registers associated with the debug architecture are detailed in the following section.

Registers implemented in the WPC have memory mapped addresses within the WPC address range. Registers implemented outside of the WPC have memory mapped addresses within the DM address range.

4.2.1 WPC registers

Each WP channel has eight 64-bit registers available, these consist of:

- a PRE register;
- an ACTION register;
- space for six further registers, used as required by each channel.

Thus, a single instance of a given WPC WP channel type can occupy up to 0x40 bytes.

There are up to 16 instances of each WPC WP channel type, and up to 0x400 bytes per channel type can be occupied.

There are up to 16 different channel types, and thus the WPC CPU channels in total can occur 0x4000 bytes.

This split denotes the address encoding appropriate for mapping all the WPC registers into the WPC address space.

Bank	Address		Offset range
	Bit 23:20	Bit 14	
Chain-latches	0b0001	0b0	0x100000 .. 0x103FFF
CPU control	0b0001	0b1	0x104000 .. 0x104018
Event counters	0b0010	0b0	0x200000 .. 0x203FFF
Performance counters	0b0010	0b1	0x204000 .. 0x207FFF
IA and IV WP channels	0b0100	0b0	0x400000 .. 0x403FFF
OA WP channels	0b0100	0b1	0x404000 .. 0x407FFF

Table 92: WPC debug register layout

4.2.2 DM registers

The basic ordering is DM, WPC. Within each of these there is a regular structure as shown in *Table 92*.

Module bank	In-bank ordering	Offset range	Number of 64-bit registers assigned
Debug module (includes bus analyzer registers)	Module specific setup and chain-latches	0x100000 .. 0x100080	17
	Event and performance counters	0x200000 .. 0x200018	3
	Registers physically located in bus analyzer	0x400000 .. 0x4000F8	16 per channel, 2 channels
	WP channels, including bus analyzer registers physically located in DM	0x800000 .. 0x800248	8 per channel, 10 channels

Table 93: Debug register layout

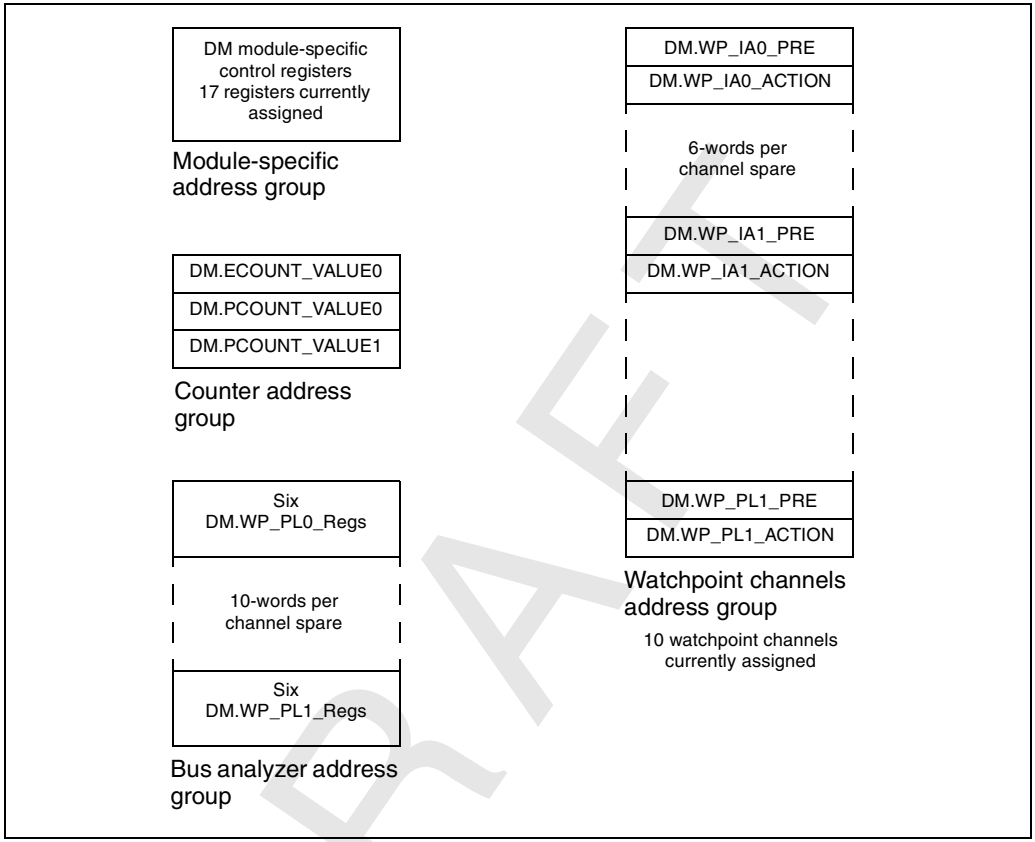

Figure 18: DM control register groups

4.2.3 Complete register list

Note: *The offsets given below are relative to either the CPU base address, or the DM base address, according to whether the register name prefix is "WPC" or "DM". See the SH-5 System Architecture Manual for the address map.*

All registers shown in this table must be accessed using 64-bit operations. When writing to the registers from the instruction stream, suitable instructions are listed in *Table 1: Instructions for accessing WPC and DM registers on page 17*.

	Register name	Offset	References
WPC chain latches			
	WPC.IA_CHAIN_0	0x100000	*Section 1.2.8: Chain latches on page 21* and *Table 6: WPC.IA_CHAIN_x definition on page 26*
	WPC.IA_CHAIN_1	0x100008	
	WPC.IA_CHAIN_2	0x100010	
	WPC.IA_CHAIN_3	0x100018	
	WPC.CHAIN_0	0x100020	
WPC event counters			
	WPC.ECOUNT_VALUE_0	0x200000	*Section 1.2.8: Chain latches on page 21*.
WPC performance counters			
	WPC.PCOUNT_VALUE_0	0x204000	*Section 1.2.10: Performance counters on page 28* and *Table 8 on page 29*.
	WPC.PCOUNT_VALUE_1	0x204008	
WPC CPU control			
	WPC.CPU_CTRL_ACTION	0x104000	*Section 1.3: CPU control on page 30*.
	WPC.CPU_DBRMODE	0x104008	*WPC.CPU_DBRMODE on page 74*.
	WPC.CPU_DBRVEC	0x104010	*WPC.CPU_DBRVEC on page 76*

Table 94: Register map and references

	Register name	Offset	References
WPC/DM interconnect control			
	WPC.ADDR_IN_TRACE	0x104018	*Section 1.5.1: WPC.ADDR_IN_TRACE register definition on page 65.*
WPC channel-specific registers			
OA	WPC.WP_OA0_PRE	0x404000	*Table 14: WPC.WP_{IA/OA/IV/WPC_PERF}x_PRE register definition on page 38*
	WPC.WP_OA0_ACTION	0x404008	*Table 18: WPC.WP_{IA/OA/IV}x_ACTION register definition on page 49*
	WPC.WP_OA0_MATCH_START	0x404010	*Table 52: WPC.WP_OAx_MATCH_START register definition on page 143*
	WPC.WP_OA0_MATCH_END	0x404018	*Table 53: WPC.WP_OAx_MATCH_END register definition on page 144*
	WPC.WP_OA1_PRE	0x404040	See cross references for channel WP_OA0.
	WPC.WP_OA1_ACTION	0x404048	
	WPC.WP_OA1_MATCH_START	0x404050	
	WPC.WP_OA1_MATCH_END	0x404058	
IA	WPC.WP_IA0_PRE	0x400000	*Table 14: WPC.WP_{IA/OA/IV/WPC_PERF}x_PRE register definition on page 38*
	WPC.WP_IA0_ACTION	0x400008	*Table 18: WPC.WP_{IA/OA/IV}x_ACTION register definition on page 49*

Table 94: Register map and references

	Register name	Offset	References
	WPC.WP_IA0_MATCH_START	0x400010	*Table 50: WPC.WP_IAx_MATCH_START register definition on page 139*
	WPC.WP_IA0_MATCH_END	0x400018	*Table 51: WPC.WP_IAx_MATCH_END register definition on page 140*
	WPC.WP_IA1_PRE	0x400040	See cross references for channel WP_IA0.
	WPC.WP_IA1_ACTION	0x400048	
	WPC.WP_IA1_MATCH_START	0x400050	
	WPC.WP_IA1_MATCH_END	0x400058	
	WPC.WP_IA2_PRE	0x400080	
	WPC.WP_IA2_ACTION	0x400088	
	WPC.WP_IA2_MATCH_START	0x400090	
	WPC.WP_IA2_MATCH_END	0x400098	
	WPC.WP_IA3_PRE	0x4000C0	
	WPC.WP_IA3_ACTION	0x4000C8	
	WPC.WP_IA3_MATCH_START	0x4000D0	
	WPC.WP_IA3_MATCH_END	0x4000D8	

Table 94: Register map and references

	Register name	Offset	References
IV	WPC.WP_IV0_PRE	0x400400	*Table 14: WPC.WP_{IA/OA/IV/WPC_PERF}x_PRE register definition on page 38*
	WPC.WP_IV0_ACTION	0x400408	*Table 18: WPC.WP_{IA/OA/IV}x_ACTION register definition on page 49*
	WPC.WP_IV0_MATCH_VALUE	0x400410	*Table 56: WPC.WP_IVx_MATCH_VALUE register definition on page 153*
	WPC.WP_IV0_MATCH_MASK	0x400418	*Table 57: WPC.WP_IVx_MATCH_MASK register definition on page 153*
	WPC.WP_IV1_PRE	0x400440	See cross references for channel WP_IV0.
	WPC.WP_IV1_ACTION	0x400448	
	WPC.WP_IV1_MATCH_VALUE	0x400450	
	WPC.WP_IV1_MATCH_MASK	0x400458	
WPC.PERF	WPC.WP_WPC_PERF0_PRE	0x400200	*Table 14: WPC.WP_{IA/OA/IV/WPC_PERF}x_PRE register definition on page 38*
	WPC.WP_WPC_PERF0_MATCH_TYPE	0x400208	*Table 60: WPC.WP_WPC_PERFx_MATCH_TYPE register definition on page 166*
	WPC.WP_WPC_PERF1_PRE	0x400210	*Table 14: WPC.WP_{IA/OA/IV/WPC_PERF}x_PRE register definition on page 38*
	WPC.WP_WPC_PERF1_MATCH_TYPE	0x400218	*Table 60: WPC.WP_WPC_PERFx_MATCH_TYPE register definition on page 166*

Table 94: Register map and references

	Register name	Offset	References
DM version control/error rags			
	DM.VCR	0x0	*Section 4.1.11: DM.VCR register on page 250*.
DM chain latches			
	DM.CHAIN_0	0x100000	*Section 1.2.8: Chain latches on page 21* and *Table 5: {WPC/DM}_CHAIN_x definition on page 25*
	DM.CHAIN_TRIG_IN	0x100008	
DM event counters			
	DM.ECOUNT_VALUE_0	0x200000	*Table 7: {WPC/DM}.ECOUNT_VALUE_x register definition on page 27*.
DM performance counters			
	DM.PCOUNT_VALUE_0	0x200008	*Section 1.2.10: Performance counters on page 28*.
	DM.PCOUNT_VALUE_1	0x200010	

Table 94: Register map and references

	Register name	Offset	References
DM misc.			
	DM.EXP_CAUSE	0x100010	*Table 11: DM.FORCE_DEBUGINT register definition on page 34*.
	DM.FPF	0x100018	*Section 1.8.9: DM.FPF register definition on page 97*.
	DM.PC	0x100020	*Section 1.8.13: DM.FIFO_0/ DM.FIFO_1/DM.FIFO_2 (FIFO port register) on page 112*.
	DM.WP_BR_FILTER	0x100028	*Table 58: DM.WP_BR_FILTER register definition on page 157*
	DM.OA_MATCH_DATAVAL	0x100030	*Section 1.12.3: Data match registers on page 147*
	DM.OA_MATCH_ DATAMASK	0x100038	
	DM.FORCE_DEBUGINT	0x100088	*DM.FORCE_DEBUGINT register definition on page 34*.

Table 94: Register map and references

	Register name	Offset	References
DM trace/FIFO control			
	DM.TRCTL	0x100040	*Table 31: DM.TRCTL definition on page 97*.
	DM.TRBUF	0x100048	*Table 32: DM.TRBUF definition on page 107*.
	DM.TRPTR	0x100050	*Table 33: DM.TRPTR definition on page 111*.
	DM.FIFO_0	0x100058	*Table 35: DM.FIFO_{0/1/2} definition on page 114*
	DM.FIFO_1	0x100060	
	DM.FIFO_2	0x100068	
	DM.FIFO_REQ	0x100070	*Table 36: DM.FIFO_REQ definition on page 115*.
	DM.FIFO_ACK	0x100078	*Table 37: DM.FIFO_ACK definition on page 116*.
	DM.CLKOUTDIV	0x100090	*Table 74: DM.CLKOUTDIV definition on page 211*.

Table 94: Register map and references

	Register name	Offset	References
SuperHyway bus analyzer			
PL	DM.WP_PL0_PRE	0x800200	*Table 16 on page 44*
	DM.WP_PL0_ACTION	0x800208	*Table 20 on page 60*
	DM.WP_PL0_CTRL	0x400000	*Table 67 on page 190*.
	DM.WP_PL0_EXCTRL	0x400008	*Table 68 on page 193*.
	DM.WP_PL0_MATCH_START	0x400010	*Table 65 on page 188*.
	DM.WP_PL0_MATCH_END	0x400018	*Table 66 on page 189*.
	DM.WP_PL0_CBHDR	0x400020	*Table 69 on page 194*.
	DM.WP_PL0_CBDATA	0x400028	*Table 70 on page 196*.
	DM.WP_PL1_PRE	0x800240	See cross references for channel WP_PL0.
	DM.WP_PL1_ACTION	0x800248	
	DM.WP_PL1_CTRL	0x400080	
	DM.WP_PL1_EXCTRL	0x400088	
	DM.WP_PL1_MATCH_START	0x400090	
	DM.WP_PL1_MATCH_END	0x400098	
	DM.WP_PL1_CBHDR	0x4000A0	
	DM.WP_PL1_CBDATA	0x4000A8	
	DM.PL_FRZ	0x100080	*Table 64 on page 186*.

Table 94: Register map and references

	Register name	Offset	References
DM channel-specific registers			
IA	DM.WP_IA0_PRE	0x800000	*Table 15: DM.WP_{IA/OA/IV}x_PRE register definition on page 42*
	DM.WP_IA0_ACTION	0x800008	*Table 19: DM.WP_{IA/OA/IV}x_ACTION register definition on page 54*
	DM.WP_IA1_PRE	0x800040	See cross references for channel WP_IA0.
	DM.WP_IA1_ACTION	0x800048	
	DM.WP_IA2_PRE	0x800080	
	DM.WP_IA2_ACTION	0x800088	
	DM.WP_IA3_PRE	0x8000C0	
	DM.WP_IA3_ACTION	0x8000C8	
OA	DM.WP_OA0_PRE	0x800100	See cross references for channel WP_IA0.
	DM.WP_OA0_ACTION	0x800108	
	DM.WP_OA1_PRE	0x800140	
	DM.WP_OA1_ACTION	0x800148	
IV	DM.WP_IV0_PRE	0x800180	See cross references for channel WP_IA0.
	DM.WP_IV0_ACTION	0x800188	
	DM.WP_IV1_PRE	0x8001C0	
	DM.WP_IV1_ACTION	0x8001C8	
FPF	DM.WP_FPF_PRE	0x800280	*Table 17: DM.WP_FPF_PRE register definition on page 46*

Table 94: Register map and references

Index

D

E

F

I

J

K

L

M

N

O

P

R

S

T

U

V

W