SuperH^{TM} (SH) 64-Bit RISC Series

SH-5 System Architecture, Volume 1: System

Last updated 18 March 2002

SuperH, Inc.

This publication contains proprietary information of SuperH, Inc., and is not to be copied in whole or part.

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

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 Overview

1.1 Introduction

The SH-5 architecture forms the common centre of a family of products. This document describes the infrastructure built to support the development of this family. To aid in this description, the STB1 evaluation device is used as an example. This device is a technology demonstrator enabling product focused systems to be developed rapidly.

Eval chip features	
SH-5 RISC CPU	Flash/ROM interface
Single-issue 64-bit core	8/16/32-bit data, 26-bit address
400 MHz internal clock speed	Write cycle for flash memory support
On-chip separate i&d caches and TLB's	Wait pin for slow devices
PCI interface	Five decoded CS_N signals
PCI 2.1, 32-bit, 66 MHz	Target part: Intel flash P/N: 28F032SA
Support bus mastering to main memory with multiple pipe-lined transactions	LVTTL I/O
Support four external bus masters	DMA controller
PCI to system memory is cache coherent	Four channels
Support configuration as PCI peripheral (non-host)	Clock controller with S/W programmable ratios for internal / external clocks
3.3 V PCI	

Table 1: Features

Eval chip features	
SDRAM interface SDR/DDR 32/64-bit, 133 MHz, 64 Mbyte, 256 Mbyte Support four open banks Support both PC100 and 133 MHz DDR SDRAM Support power saving idle / standby states Target parts: Hitachi P/N: 54S64XXD2 & Intel PC100 SDRAM standard Multiple timers 32-bit timer with auto reload, interrupts Configurable input clocks	Debug support High speed debug interface (100 MHz), independent of JTAG Development host can access entire internal address space, non-intrusively CPU can boot from debug adapter interface CPU core has code address, operand and opcode watchpoints plus branch trace and fast printf functions Watchpoints and trace on internal bus (SuperHyway) All watchpoints can send trace packets to debug link (non-intrusively) or generate CPU debug trap Performance counters for CPU core and bus parameters
Real time clock On-chip oscillator circuit Built-in clock, calendar, alarm, IRQ	SCIF UART FIFO serial interface with DMA 16 byte FIFO S/W configured baud clock generator
Interrupt controller Configurable priorities Internal peripheral interrupts	Power management unit Control unit and peripheral power saving modes Watchdog timer fuction (reset hard and soft)

Table 1: Features

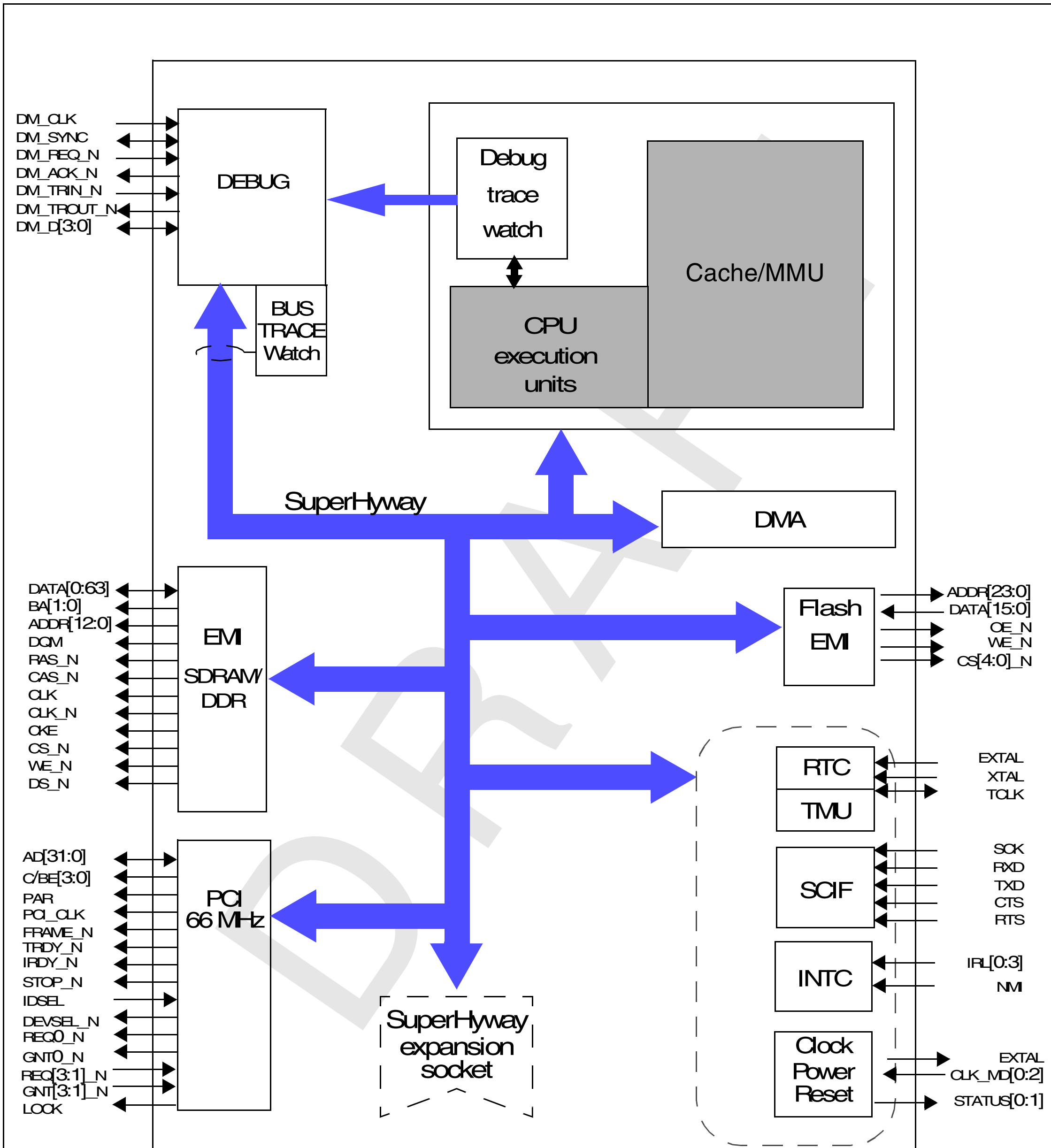


Figure 1: Eval chip block diagram

The organization of interconnects in the system illustrated in *Figure 1* is guided by the principle of optimizing each interconnect for its specific purpose.

1 The SuperHyway system interconnect facilitates the integration of several different types of sub-systems. It is used for closely coupled subsystems which have stringent memory latency/bandwidth requirements.

2 The PCI bus provides a standard interface which is used to expand the capabilities of the Eval chip and Reference board to provide a variety of product demonstrators.

3 The SuperHyway socket is an expansion port which supports the rapid integration of application modules without changing the eval chip core.

1.1.1 Address map

The CPU accesses a single 32-bit flat address space in which all of the external memory and device register are accessible. The entire address space is accessible from all memory requesters. *Table 3 on page 22* illustrates the memory map.

1.1.2 Interrupt architecture

The system uses a conventional interrupt architecture. A programmable interrupt controller INTC which is responsible for multiplexing and filtering interrupt requests onto the **irq** and **nmi** lines of the CPU. The INTC implements priority and route interrupts.

1.1.3 DMA architecture

A multiple channel DMA controller permits autonomous data transfers along multiple channels.

1.2 Debug architecture

SH-5 has a dedicated high-speed debug interface to connect the target system to a development host. This is independent of the JTAG interface and is capable of operating at clock speeds of up to 100 MHz.

The CPU core has eight watchpoints: four instruction address, two memory write[1], two instruction opcode. Each watchpoint has a control register field which defines the action when a hit occurs. Possible actions include:

- sending a trace packet to the debug link for writing into external debug adapter memory,
- writing the trace packet into an area of target system memory,
- generating a CPU debug trap which invokes a monitor program,
- incrementing a performance counter,
- generating a pulse on a TRIGGER_OUT pin.

All actions except CPU debug trap are non-intrusive.

The CPU core also supports branch trace and fast printf functions which send trace packets to the debug link for writing into external debug adapter memory.

The SuperHyway bus has a bus analyzer for monitoring selected bus transactions. The bus analyzer has watchpoints, a trace buffer and control register fields which define the action when a hit occurs. Possible actions are the same as for CPU watchpoints.

The debug link operates as a reduced speed extension to the SuperHyway bus with both bus master and bus slave capability. By using the bus slave capability, the CPU (or any other bus master) can access memory in an external debug adapter simply by using the appropriate address range. This allows the CPU, or any other bus master, to fetch boot code instructions over the debug link or write to data memory in the external debug adapter.

1. Future implementation may include 2 memory read watchpoints.

By using the SuperHyway bus master capability of the debug link, debug software running on a development host can access the whole SuperHyway address space, including all watchpoint control registers, without involving the CPU. Special memory-mapped registers allow debug software running on a development host to directly control the CPU using suspend, resume, change reset vector, soft reset commands.

The debug link supports several different price/performance options for connecting to a development host. At the simplest, a $50 signal converter connects to the parallel port of a personal computer but does not support real-time tracing. A higher performance option requires an external debug adapter containing SRAM, for use as the trace buffer, plus a processor for managing the debug link protocol.

This architecture is illustrated in *Figure 2*.

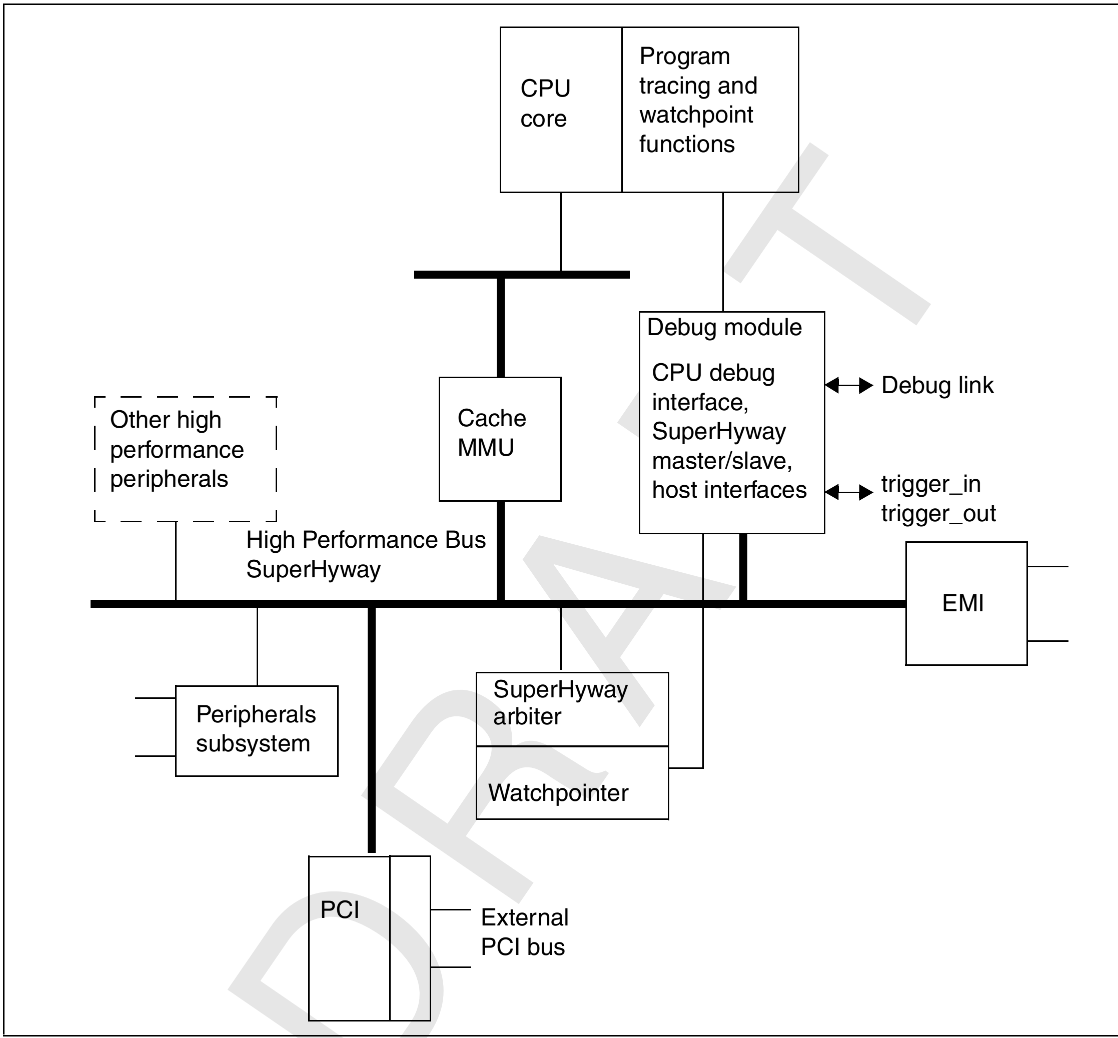


Figure 2: SH-5 debug architecture

2 System organization

2.1 Introduction

The SH-5 system architecture is modular. An SH-5 implementation consists of a number of modules which communicate using one or more interconnects. The interconnect to which the CPU core is connected provides the main path to external memory. This interconnect provides a memory-mapped packet routing mechanism between modules. It is known as the SuperHyway and forms the backplane of highly integrated systems which use the SH-5. The SuperHyway specifies a protocol which defines how packets are represented and propagated. The name SuperHyway is given to the family of implementations of this protocol on SH-5 chips.

A typical SH-5 implementation is a single chip which contains one or more CPU cores, one or more product-specific SH-5 modules, an interconnect, a peripheral subsystem, an external memory interface and a module dedicated to supporting debug of the core and system.

2.2 SuperHyway architecture

The SuperHyway architecture provides the 'glue' that binds together a set of SH-5 modules. A connection between the SuperHyway and an SH-5 module is called a port (sometimes referred to as a p-port). A SuperHyway port supports a bi-directional flow of packets between the interconnect and an SH-5 module.

The distinction between the SuperHyway architecture and implementation is important. This section, *SuperHyway architecture*, defines the abstractions that are used to build implementations containing a SuperHyway interconnect. The architecture includes an abstract view of the packets, the SuperHyway, the port, a SH-5 module and the protocol.

The implementation determines how the SuperHyway, the ports and the required modules are physically represented. It also defines how many SH-5 modules are implemented and how these are connected to the SuperHyway.

In all SH-5 systems there is at least one interconnect path and at least one SH-5 module. Each SH-5 module is connected to the SuperHyway using at least one port. The SuperHyway provides complete connectivity between SH-5 modules.

The architectural relationship between the SuperHyway packet-router, the port and the SH-5 module is illustrated in *Figure 3*.

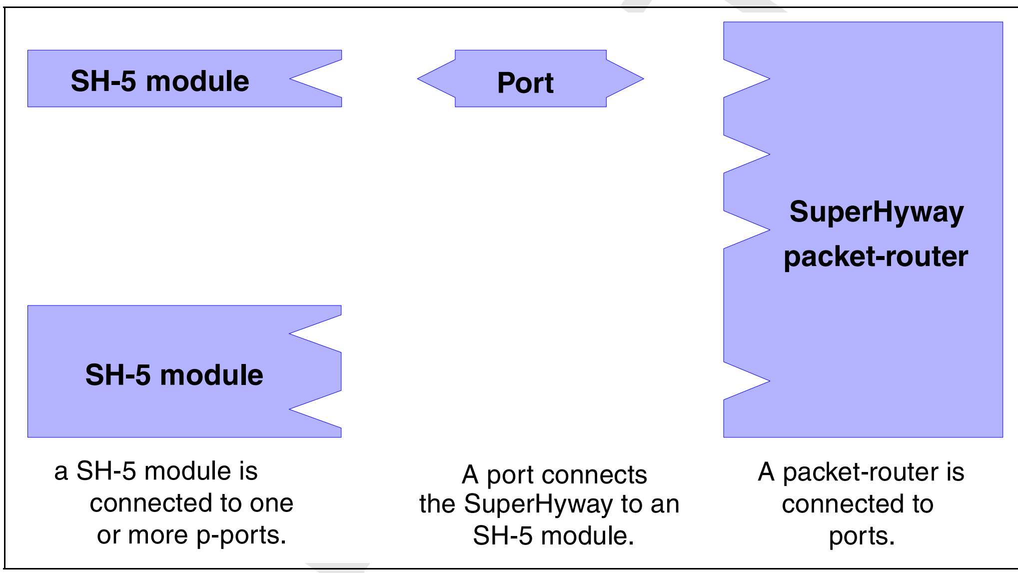


Figure 3: SuperHyway packet-router, port and SH-5 module architecture

A simple implementation containing a packet-router, two single-ported SH-5 modules and one double-ported SH-5 module is illustrated in *Figure 4*.

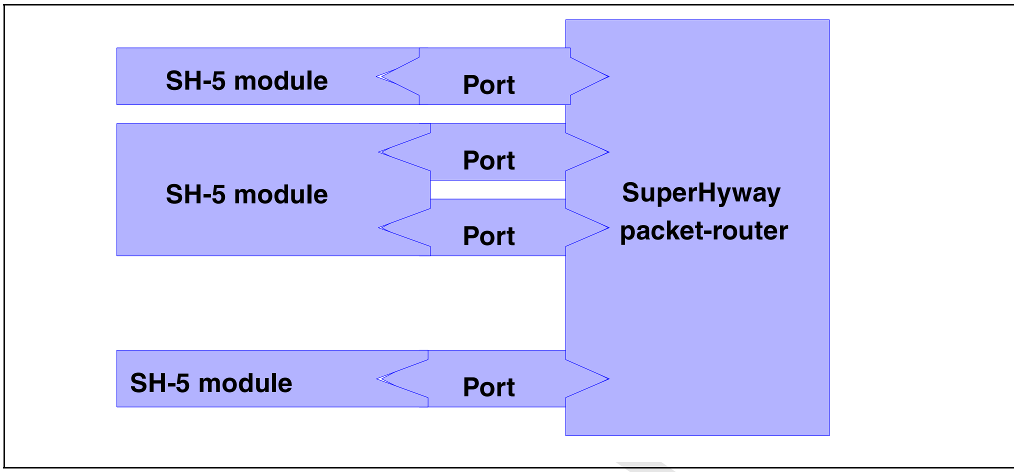


Figure 4: A simple implementation

The SH-5 eval chip implementation of the SuperHyway packet-router architecture is described in *Section 2.5: SH-5 SuperHyway implementation on page 22*.

2.2.1 Packets

The packet is the unit of data transfer through the SuperHyway. Communication between SH-5 modules is achieved by the exchange of packets between those SH-5 modules.

A packet is composed of fields. Each field has a number of possible values to characterize that packet. Every packet contains a destination field which is used to determine which SH-5 module the packet should be routed to. Further information on packets is given in *Section 2.2.6: SuperHyway protocol on page 16*. In particular, packets contain a field that indicates the type of access made by that packet.

Each packet journey is associated with a source SH-5 module and a destination SH-5 module. The source sends a packet over a port into the packet-router. The packet-router arranges for the packet to be routed to a port connected to the destination. The destination then receives this packet over that port from the packet-router. It is possible for the source and destination to be the same SH-5 module.

A packet route from a source to a destination is illustrated in *Figure 5*. In packet routing diagrams, such as *Figure 5*, the vertical direction represents time with time flowing forward down the page.

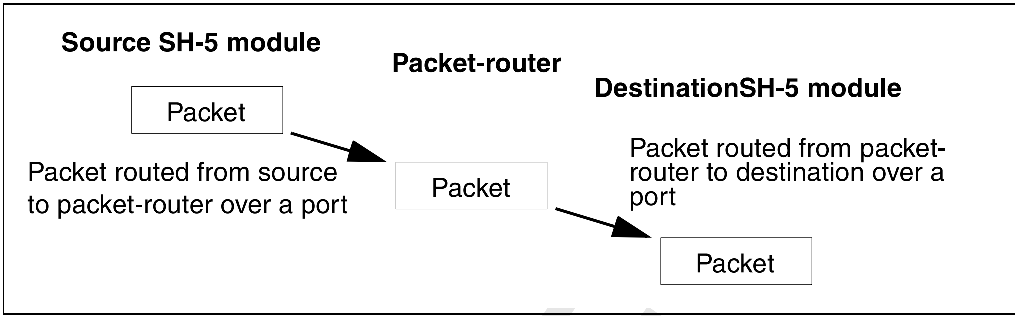


Figure 5: A packet route

2.2.2 Transactions

A transaction is an exchange of packets that allows an SH-5 module to access the state of another module using the SuperHyway protocol. A transaction consists of the transfer of a request packet from a requesting module to a responding module, followed by the transfer of a response packet from that responding module back to the requesting module. The request packet initiates the transaction and its contents determine the access to be made. The response packet completes the transaction and its contents indicate the result of the access.

This style of communication is called split phase. The separation between the request packet and the response packet allows systems to be constructed which are tolerant of high latency SH-5 modules. A requesting module can send multiple requests into the SuperHyway packet-router before any responses are received. This is called request pipe-lining and allows the latencies of those transactions to be overlapped.

There is a causal relationship between a request packet and its corresponding response packet since the request packet must be received before the response packet can be sent. Additionally, there is a one-to-one correspondence between request packets and response packets.

When a response packet is received by the SH-5 module that sent the corresponding request, the transaction is complete. It is guaranteed that the access associated with the response has been committed to by the destination module. This means that, apart from internal latency inside the destination module, the access is completed as viewed through all ports to that module. Any subsequent requests to that destination module will therefore act after that access.

This guarantee means that time-ordering of accesses at a destination can be imposed by waiting for the corresponding response.

A response packet also indicates whether the request was valid or not. The response packet is called an ordinary response if the request was valid, or an error response if the request was invalid.

The following sections elaborate on the actions comprising a single transaction.

Request

A request packet is constructed by a requesting SH-5 module when that module needs to make an access to a particular target module. This target module is recorded in the request packet's destination or address field. The requesting module is the source of the request packet and sends that packet into the packet-router. The packet-router arranges for that request packet to be routed from its source to its destination. The destination receives the request packet from the packet-router and services that access according to the information in the received request packet. The destination is known as the responding module because it replies to the request packet using a response packet.

Response

A response packet is constructed by a responding module in order to reply to a previous request. The module that originated that request packet is recorded in the response packet's destination field. The responding module is the source of the response packet and sends that packet into the packet-router. The packet-router arranges for that response packet to be routed from its source to its destination. The destination receives the response packet from the packet-router and matches that response to the original request in order to complete the transaction.

A complete transaction

A packet routing diagram showing a complete transaction is given in *Figure 6*.

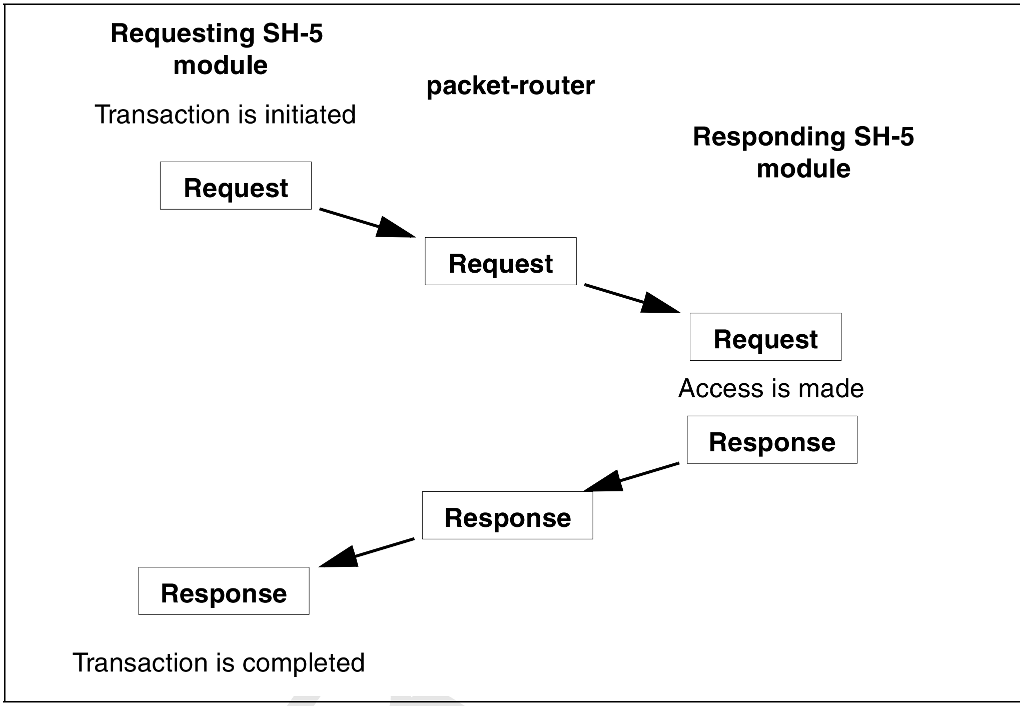


Figure 6: A SuperHyway transaction

2.2.3 Packet-router

The packet-router provides a packet routing interconnect for communication between SH-5 modules. The packet-router arranges for packets to be routed from their source module to their destination module. A variety of packet-router implementations are possible. Implementations include, but are not limited to, a bus, a crossbar and a packet routing network.

All packets passed into the packet-router contain a destination field which is used to route the packet. The packet-router contains a mapping from all possible destination field values to an appropriate p-port. The mechanism by which this mapping is established and the mapping itself are defined by a packet-router implementation.

The packet-router needs to interpret only a few fields of a packet. It must inspect the destination field to route the packet. The bulk of the packet does not need to be interpreted by the routing mechanism and is used to convey information between the requesting module and the responding module.

2.2.4 SuperHyway ports

A port provides bi-directional packet-level communication between the packet-router and an SH-5 module. A module may have multiple port connections to the packet-router. Multiple ports may be used to increase the bandwidth between a module and the packet-router. Multiple ports may also be used to decouple logically separate functional units within that module.

2.2.5 SH-5 modules

SH-5 modules (abbreviated to modules) communicate using packets routed via the packet-router. The interpretation placed on packets of different types by an SH-5 module depends on the implementation of that module.

Example modules include a CPU, a memory or a device. A CPU module will typically generate request packets to fetch instructions and to access data. A memory module will typically service request packets and generate response packets to return the results of those memory accesses. An example device module might service request packets and generate response packets to access the memory-mapped state of the device. It might also have a DMA engine to allow the device to access memory by generating request packets.

2.2.6 SuperHyway protocol

The protocol is a memory-mapped packet routing protocol.

Packet routing

Packets are associated with a physical address. A physical address is an unsigned integral value that indicates a location in physical memory. Physical memory is byte-addressed.

Physical addresses are split into two parts as illustrated in *Figure 7*.

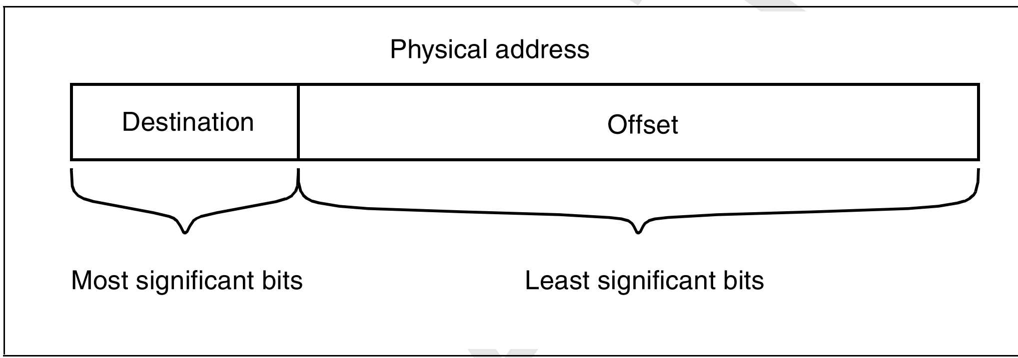


Figure 7: Physical address decomposition

The most significant bits of a physical address identify the destination to which a packet is to be sent. The least significant bits of a physical address are an offset that identifies a location within that destination. The offset information is present in request packets to identify the location at which the request is targeted. The offset information is not present in response packets.

The size of a physical address, the number of destination bits and the number of offset bits are defined by the implementation. The packet-router is arranged so that every packet-router access is associated with a single destination.

The packet-router uses the destination field to perform routing. Each possible destination field value is uniquely associated with a p-port and hence is uniquely associated with a module (the module connected to that p-port). When the packet-router routes a packet, it inspects the destination field, determines the appropriate p-port, and routes the packet to that p-port and hence onto the destination module.

A particular module may be able to handle requests for multiple destination fields. Multiple different destination field values may map to the same p-port. It is common for memory modules to handle requests for a contiguous range of destination field values. This allows a memory module to provide a physical address space, larger than that possible with a single destination field value, which is viewed as contiguous physical memory when addressed by the p-protocol.

Packet classification

A packet has a class and a type.

A packet's class is either a request packet or a response packet. The response packet class is subdivided into two different kinds of response packet: ordinary response packets and error response packets. The term 'response packet' refers to either an ordinary response packet or an error response packet, unless the surrounding context makes it clear that a particular kind of response packet is intended.

A packet's type indicates the memory transaction associated with that packet. The SH-5 implementation uses four basic kinds of memory operation: **read**, **write**, atomic **read-modify-write** and **cache-coherency** operations.

These are shown in the table below:

	Transaction name	Request parameters[a]	Response parameters
Memory read	**Load8**	$addr_8$, byte-mask	word[b]
	Load16	$addr_{16}$	2 words
	Load32	$addr_{32}$	4 words
Memory write	**Store8**	$addr_8$, byte-mask, word	-
	Store16	$addr_{16}$, byte-mask, 2 words	-
	Store32	$addr_{32}$, byte-mask, 4 words	-
Atomic read-modify-write	**Swap8**	$addr_8$, byte-mask, word	word

Table 2: SH-5 p-router transactions

	Transaction name	Request parameters[a]	Response parameters
Cache coherency	**Flush**	dest[c], $addr_{32}$,	-
	Purge	dest[c], $addr_{32}$	-

Table 2: SH-5 p-router transactions

a. $addr_n$ is used to denote an n-byte aligned address.

b. A word in the SH-5 system corresponds to eight bytes.

c. The dest field here is the port identifier of the cache controller. The address is a 32-bit address.

The packet class and packet type are combined to form a packet opcode.

Memory transaction descriptions

The transactions supported by the p-protocol are described below. In this description a word is 8 bytes of memory.

The **Load8** transaction reads up to 8 bytes of data from an 8-byte aligned location in a destination. The transaction is qualified by an 8-bit mask; each bit of this mask indicates whether a particular byte in the location is to be accessed or not. If all 8 bytes of data are read then it is a whole-word transaction. If less than 8 bytes of data are read then it is a subword transaction.

The **Load16** transaction reads 16 bytes of data from a 16-byte aligned location in a destination.

The **Load32** transaction reads 32 bytes of data from a 32-byte aligned location in a destination. The requestor may indicate which word within the four words being accessed should be returned first to implement critical word first fetches.

The **Store8** transaction writes up to 8 bytes of data to an 8-byte aligned location in a destination. The transaction is qualified by an 8-bit mask; each bit of this mask indicates whether a particular byte in the location is to be accessed or not. If all 8 bytes of data are written then it is a whole-word transaction. If less than 8 bytes of data are written then it is a subword transaction.

The **Store16** transaction writes 16 bytes of data to a 16-byte aligned location in a destination. This is qualified by a 16-bit mask.

The **Store32** transaction writes 32 bytes of data to a 32-byte aligned location in a destination. This is qualified by a 32-bit mask.

The **Swap8** transaction allows up to 8-bytes of data to be read from and then the same quantity of data written to a word location in memory. The read and write are performed atomically. The transaction is qualified by an 8-bit mask. Each bit of the mask indicates whether a particular byte is to be accessed or not. The data is word aligned.

Cache coherency transactions are described in *Section 2.3*.

2.3 Cache coherency support

Cache coherency transactions are provided primarily to support the integration of the PCI bridge into the system. However the coherency support is general and can be used by any modules attached to the system interconnect.

There are two cache control transactions: **Flush** and **Purge**. They are defined in the following sections.

2.3.1 Flush

The flush transaction has a single operand which is the physical address to be flushed from the cache:

Flush <physical address>

When a flush transaction is received from the interconnect, by the CPU cache controller, it causes the cache controller to lookup the address in the cache. If the lookup yields a miss, or a hit to a line which is unmodified with regard to main memory, then the cache controller will issue a response to the flush request immediately following the lookup. If the lookup yields a hit to a line which is modified with regard to main memory then the cache controller causes a writeback of the line to main memory. Following the writeback the cache controller issues a response to the flush request.

Responses to flush requests are simple acknowledgments; they do not carry any data.

The <physical address> should be 32-bit aligned. The low order 2 bits, if not zero, are ignored.

2.3.2 Purge

The purge transaction has a single operand which is the physical address which is to be purged from the cache:

Purge <physical address>

When a purge transaction is received from the interconnect, by the CPU cache controller, it causes the cache controller to lookup the address in the cache. If the lookup yields a miss, then the cache controller will issue a response to the flush request immediately, following the lookup. If the lookup yields a hit then the cache controller causes a writeback of the line to main memory (if the line has been modified in the cache) and then invalidates the line. Following the invalidation the cache controller issues a response to the flush request.

Responses to purge requests are simple acknowledgments; they do not carry any data.

The <physical address> should be 32-bit aligned. The low order 2 bits, if not zero, are ignored.

2.3.3 Coherency maintenance

The use of flush and purge by a module in association with appropriate cache behavior provides a level of cache coherency. In particular it guarantees two properties:

1 That a read operation by module to an address in shared system memory will receive the value last written to that address. The time of the access is given as the time at which the flush is received by the cache controller. The module read operation is guaranteed to get a data value coherent with the value of system memory no earlier than the time of access.

2 That a write operation by a module to an address in shared system memory will be completed such that the data written is readable by all memory users after the time of access. The time of access is given as the time at which the write operation is performed to system memory following the purge of the data cache(s).

See *Chapter 3: SH-5 CPU on page 53* for details of cache controller behavior.

2.3.4 Use of coherency transactions

When a module wishes to make a coherent request to shared memory, the module performs the following routine:

1 Splits the memory request into a number of non-cache-line straddling system interconnect requests.

For each of these requests it does the following:

2 For a read a flush request is sent to the data cache port, for a write a purge request is sent to the data cache port.

3 The module waits until it receives a response from the cache controller.

4 For a read, a load request is then sent to the main memory. For a write, a store request is sent to main memory.

5 The memory's response indicates the completion of the coherent access.

2.4 Other features

The SH-5 SuperHyway implementation also contains a number of additional features to enhance the system functionality. This includes support for module powerdown, module freeze and visibility of transaction traffic.

2.4.1 Module powerdown

The SH-5 supports partial and complete system powerdown by stopping clocks to various parts of the system. This is achieved under software control by accesses to the clock, power and reset controller (CPRC) logic. Once a module powerdown request has been received, the SuperHyway ensures requests to that module are responded to with an error response, so that no further new requests are routed to that module. This keeps the system live, and allows it to be debugged. Details may be found in *Chapter 10: Clock, power and reset controller on page 259*.

2.4.2 Debug features

Module freeze

To aid in system analysis or control over critical execution areas, the SH-5 supports a module freeze mechanism. The SuperHyway, under debug module control, is able to isolate a module from the system by stopping that module generating any new requests to the system. This may be used to simplify the system behavior when the user has specific requirements. Details may be found in *Volume 3 Debug, Chapter 3 External Debug Interfaces*.

Transaction tracing

In the SH-5 SuperHyway, all traffic is made visible to a bus analyzer associated with the debug module. This is able to log or capture any traffic across the SuperHyway[1]. To reduce the amount of traffic captured, the triggering event may be based on one or more of the following; address range, opcode, source identity, transaction identity. Details may be found in *Volume 3 Debug, Chapter 3 External Debug Interfaces*.

2.5 SH-5 SuperHyway implementation

SH-5 modules share a common physical address space for memory. The SuperHyway provides point-to-point connectivity between all SH-5 modules based on this address map.

The SH-5 eval chip contains several direct ports. One of these are CPU's and the remainder support external memory, peripherals, debug and peripherals. The modules which are directly connected to the packet-router are shown below. Some SH-5 modules may have more than one connection to the packet router. The full list of p-modules and p-ports for SH-5 is given in *Table 3*.

SH-5 module P-ports name	P-port name (abbreviation)
Peripheral subsystem	PERIPH
Debug Module	DEBUG
External Memory Interface	EMI

Table 3: SH-5 modules and p-ports

1. Subject to bandwidth limitations.

Flash ROM interface	FEMI
CPU core	CPU
PCI Interface	PCI
DMA controller	DMAC
SuperHyway expansion Socket	SOCKET

Table 3: SH-5 modules and p-ports

2.5.1 Supported transactions

SH-5 designs support the following transaction types

Load8 - read 8 bytes (1 * 64 bit word)

Load16 - read 16 bytes (2 * 64 bit word)

Load32 - read 32 bytes (4 * 64 bit words)

Store8 - write 8 bytes (1* 64 bit word)

Store16 - write 16 bytes (2* 64 bit word)

Store32 - write 32 bytes (4 * 64 bit words)

Swap8 - swap 8 bytes (1*64 bit word)

Flush (address)

Purge (address)

2.5.2 Implementation

All transactions on the SH-5 eval chip are implemented on a non-multiplexed 64-bit interface. Each transaction is constructed from a request packet and a response packet. Each packet is constructed from a series of cells or tokens framed using an EOP (end of packet) signal. These cells are defined by the SH-5 SuperHyway interface.

Each request cell can carry the following information:

ADDRESS[31:3]

OPCODE[7:0]

MSK[7:0]

SRC[7:0]

TID[7:0]

DATA[63:0]

If a request packet contains more information than a single cell may hold, it is constructed from a sequence of cells framed using EOP.

Each response packet carries the following information:

R_OPCODE[7:0]

R_SRC[7:0]

R_TID[7:0]

R_DATA[63:0]

Again if the response packet contains more information than a single cell, it is constructed from a sequences of cells framed using the R_EOP signal.

2.5.3 Data organization

For simple memory accesses (load or store) the MSK field indicates the bytes involved in the transaction. Other bytes are invalid. The relationship between the target ADDRESS, MSK and DATA fields is as follows:

Each transaction has a defined ADDRESS[31:3] which specifies an aligned 64-bit quadword. For operations transferring a quadword or less, the MSK field validates the data lanes and specifies which bytes are to be accessed.

The important aspect to understand is that the SuperHyway byte lanes are labelled by the significance of the byte carried within a quadword. The significance of the data carried is invariant in little and big endian modes but the address which a particular physical byte lane is associated with depends on whether the system is in little or big endian mode. For example, in little endian mode the byte lane of lowest significance always corresponds to a byte having an address 8n. This same byte lane (that is, the least significant) in big endian mode corresponds to a byte having an address 8n+7.

For accesses larger than a quadword, data is transferred as a sequence of quadwords starting at the addressed quadword and incrementing by a quadword until the number of quadwords indicated by the transaction opcode have been transferred. The increment will wrap around if the addressed quadword is not aligned to the size of the transfer. So that, for example, for a 32-byte load request whose (byte) address is 8 the sequence of transfers will be the quadwords at (byte) addresses 8, 16, 24 and finally 0.

This is described further in *Section 2.8: SH-5 endianess and data mapping on page 43*.

2.5.4 SH-5 physical memory organization

An SH-5 physical address is 32 bits wide. The eight most significant bits of this physical address identify a destination. The 24 least significant bits of this physical address are an offset that identifies a location within that destination. This is illustrated in *Figure 8*.

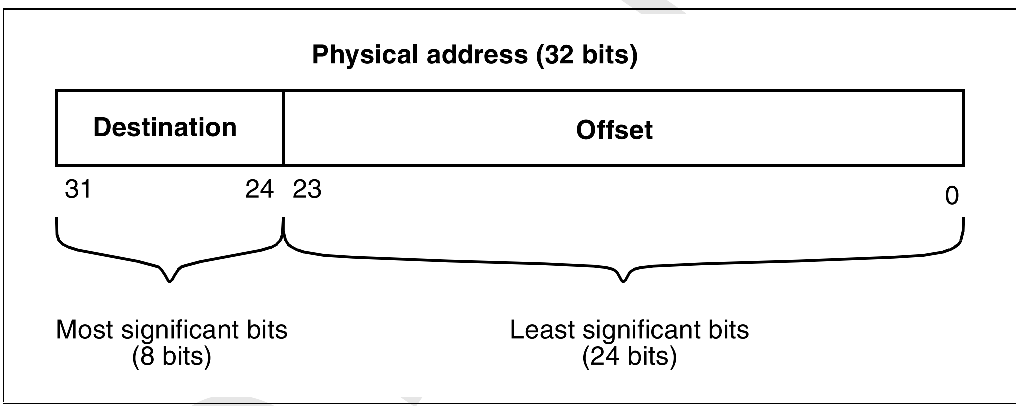


Figure 8: SH-5 physical address decomposition

The 2^{24} bytes of address space associated with a particular destination is called a memory block (MB). There are 256 memory blocks in the SH-5 physical address space and each memory block is 16 Mbytes in size.

There are three types of memory block: control blocks (CB), data blocks (DB) and undefined blocks (UB). These blocks are described in *Section 2.7.1: Memory blocks on page 29*.

2.6 SH-5 physical address map

Each memory block is associated with a particular p-port and hence with a particular module. The mapping from memory blocks to p-ports is not programmable on SH-5. The mapping defines the SH-5 physical address map and is given in *Table 4*.

P-port acronym (or RESERVED)	Block type	Destination range	Physical address range	Physical address space
FEMI_db	DB	0x00 to 0x07	0x00000000 to 0x07FFFFFF	128 Mbyte
FEMI_cb	CB	0x08	0x08000000 to 0x08FFFFFF	16 Mbyte
PERIPHERAL_cb	CB	0x09 to 0x0A	0x09000000 to 0x0AFFFFFF	32 Mbyte
Debug_link	DB	0x0B	0x0B000000 to 0x0BFFFFFF	16 Mbyte
Debug_cb	CB	0x0C	0x0C000000 to 0x0CFFFFFF	16 Mbyte
CPU	CB	0x0D	0x0D000000 to 0x0DFFFFFF	16 Mbyte
DMAC	CB	0x0E	0x0E000000 to 0x0EFFFFFF	16 Mbyte
RESERVED	UB	0x0F to 0x3F	0x0F000000 to 0x3FFFFFFF	784 Mbyte
PCI_db	DB	0x40-0x5F	0x40000000 to 0x5FFFFFFF	512 Mbyte
PCI_cb	CB	0x60	0x60000000 to 0x60FFFFFF	16 Mbyte
RESERVED	UB	0x61 to 0x6F	0x61000000 to 0x6FFFFFFF	240 Mbyte
SHwy Socket	UB	0x70 to 0x7F	0x70000000 to 0x7FFFFFFF	256 Mbyte
EMI_DRAM	DB	0x80 to 0xFE	0x80000000 to 0xFEFFFFFF	2032 Mbyte
EMI_cb	CB	0xFF	0xFF000000 to 0xFFFFFFFF	16 Mbyte

Table 4: SH-5 physical address map

The physical address map is organized so that each p-port deals with a contiguous range of blocks and with a set of blocks of the same type. The only exception to this organization is the treatment of accesses to undefined blocks. All accesses to undefined blocks are routed to the Debug_cb p-port where they are handled as errors.

The SH-5 packet-router implementation is illustrated in *Figure 9*.

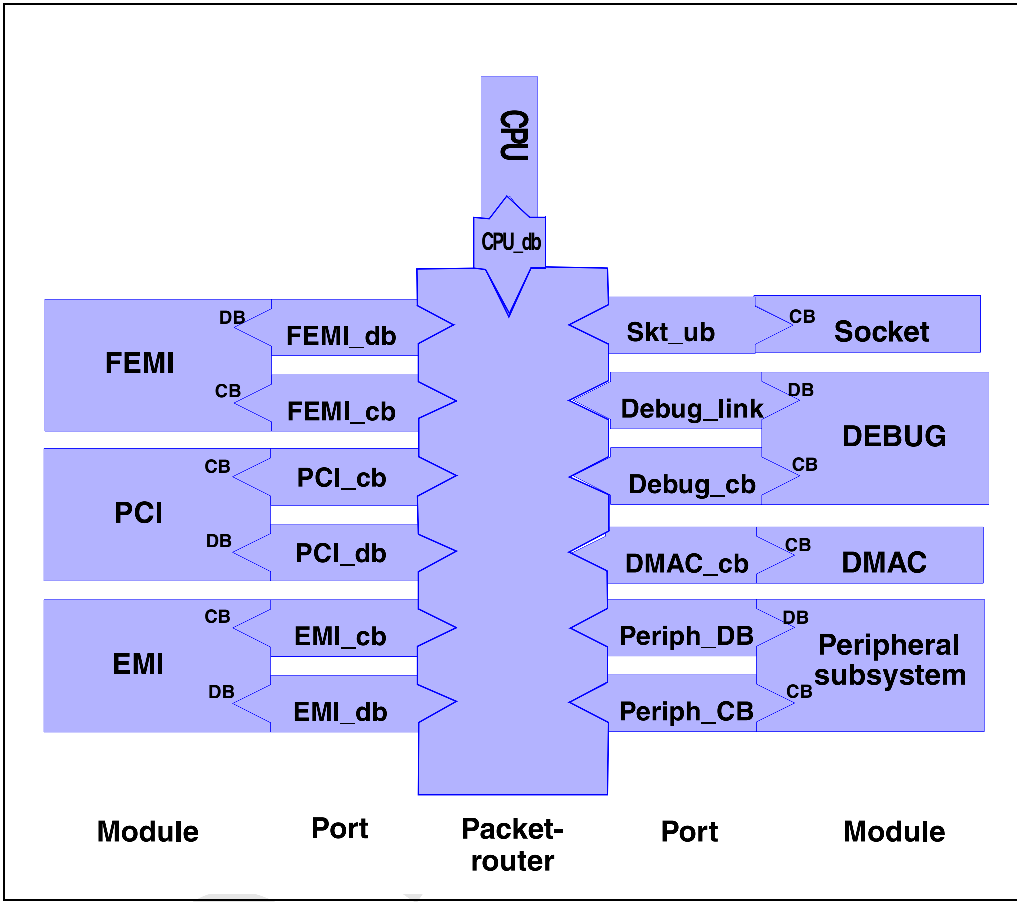


Figure 9: SH-5 SuperHyway implementation

Address	Region
0xFF00 0000	
	EMI High performance DRAM
0x8000 0000	
0x7F00 0000	
	Reserved
0x6100 0000	
0x6000 0000	
	PCI
0x4000 0000	
	Reserved
0x0F00 0000	
0x0E00 0000	DMAC
0x0D00 0000	CPU
0x0C00 0000	
0x0B00 0000	DEBUG
	Peripheral subsystem (Timers, clocks, serial ports, interrupt controller, power management controller)
0x0900 0000	
0x0800 0000	
	FEMI (FLASH,SRAM,ROM,MPX, companion chip interfacing)
0x0000 0000	

Control Registers   Data Space

Table 5: SH-5 eval chip physical address map

2.6.1 SH-5 debug link

The SH-5's debug module provides features for debugging the system. It interacts with the debug facilities in the CPU, to provide, among other things tracing, and bus monitors. A principal debug support takes the form of the debug link. The debug link uses SH-5 pins to implement a bit-serial communication port which is typically used to connect the SH-5 to a host development system (abbreviated to host).

The debug link protocol contains packet formats that allow the host to engage in transactions with SH-5 modules. There are debug link packets that correspond to packet-router request packets and response packets of each packet-router transaction. This allows the host to access the SH-5's physical address space via the packet-router. Additionally, the SH-5 can engage in transactions with the host by making accesses to a memory block that maps onto the debug link.

These are powerful features which can be used, for example, to:

- boot-strap the SH-5 through memory provided by the host,
- support interactive or post-mortem debugging of the SH-5,
- provide host-based input and output facilities to software running on the SH-5.

2.7 SH-5 conventions

The SH-5 follows additional conventions which present a consistent interface to SH-5 modules.

2.7.1 Memory blocks

There are three types of memory block: control blocks (CB), data blocks (DB) and undefined blocks (UB).

A control block contains a collection of memory-mapped registers holding a variety of status and control information for an SH-5 module. There is a one-to-one association between SH-5 modules and control blocks. All control blocks contain a Version Control Register (VCR). A VCR identifies the module associated with that control block, the version number of that module, the memory blocks associated with that module (if any) and the module's error status.

A data block is a contiguous range of data memory blocks associated with a module. These blocks are not control blocks and do not contain a VCR. Data blocks are typically used to provide access to memory. A module may be associated with zero, one or more data blocks. The set of data blocks associated with a module must be contiguous in physical address space. Each data block is associated with exactly one module.

All blocks which are neither control blocks nor data blocks are undefined blocks. Undefined blocks do not provide useful functionality and all accesses to undefined blocks are errors. Packets destined for an undefined block are routed to the Debug module where an error is recorded.

2.7.2 Control registers

A control register is a memory-mapped register held in a control block. Control registers are 64-bit wide and allocated on addresses that are 8-byte aligned.

Each control register has a unique name. Control register names are composed hierarchically by concatenating subnames, separated by a period ('.'), together. The left-most subname indicates the module that implements that control register. Succeeding subnames repeatedly refine the classification of the control register. A control register can be refined to a field by concatenating the control register name with the field name separated by a period ('.'). A field can be refined to a single bit by concatenating the field with the bit name separated by a period ('.').

An example control register is DEBUG.VCR. This name refers to the VCR register within the DEBUG module. An example field is DEBUG.VCR.PERR_FLAGS. This name refers to the PERR_FLAGS field within the DEBUG.VCR control register. An example bit is DEBUG.VCR.PERR_FLAGS.ERR_RCV. This name refers to the ERR_RCV bit within the DEBUG.VCR.PERR_FLAGS field.

Control registers are accessed using the p-protocol. The set of transactions that are supported by a control register depends on the implementation of that control register. Control registers are typically read using a whole-word **LoadWord** transaction and written using a whole-word **StoreWord** transaction. It is possible to have control registers that support other transactions, though these are much less common.

Each control block is fully populated by an array of control registers. For SH-5 there are 2^{21} control registers in each control block. Typically, however, each module will only implement a very small proportion of the available control registers.

The semantics of a control register are, in general, specific to that control register and defined by the architecture of the module that implements that control register.

However, there are conventions which all control registers adhere to. Additionally, there is a standard table format for describing the layout of control registers. These are described in the following sections.

Note: Each SH-5 CPU contains a control block and debug control registers which are memory mapped and so can be accessed from the debug link.

Register conventions

Each control register is one of 'reserved', 'undefined' or 'defined'.

'RESERVED' control registers

'reserved' control registers are used to reserve parts of the control block address space. A read from a 'reserved' control register always returns zero. Writes to a 'reserved' control register are always ignored.

If a control register is 'reserved', it is possible that this control register will have a different implementation in future components in the SH-5 family.

'Undefined' control registers

'undefined' control registers are used to identify parts of the control block address space where the behavior of accesses is not well defined by the architecture. The specification of a particular 'undefined' control register may elaborate on the actual behavior.

Typically, an access causing a request to an 'undefined' control register will result in an error flag being set in the VCR of the module that deals with the request. Additionally, a response from an access to an 'undefined' control register might result in an error flag being set in the VCR of the module that deals with the response.

A read from an 'undefined' control register will typically return an undefined value or an implementation defined value. A write to an 'undefined' control register will typically be ignored or lead to behavior that is architecturally undefined.

If a control register is 'undefined', it is possible that this control register will have a different implementation in future components in the SH-5 family.

‘Defined’ control registers

‘Defined’ control registers are implemented and the behavior of accesses is well defined. A ‘defined’ control register is composed of one or more fields. Each field is a collection of bits in the control register. All bits in a ‘defined’ control register belong to a field. Further categorization of a ‘defined’ control register is performed at the field level.

Field conventions

Each field in a ‘defined’ control register is one of ‘reserved’ or ‘defined’.

‘RESERVED’ fields

‘Reserved’ fields are used to reserve parts of a control register. A read from a ‘reserved’ field always returns zero. Writes to a ‘reserved’ field are always ignored.

If a field is ‘reserved’, it is possible that this control field will have a different implementation in future components in the SH-5 family.

‘Defined’ fields

A ‘defined’ field is one of ‘read-only’, ‘read-write’ or ‘other’.

A ‘read-only’ field indicates that the value of the field cannot be changed by software. A read from a ‘read-only’ field returns the value associated with that field. A write to a ‘read-only’ field is ignored.

A ‘read-only’ field indicates that the value of the field has conventional read and write behavior. A read from a ‘read-write’ field returns the value of that field. A write to a ‘read-write’ field sets the value of the field.

An ‘other’ field indicates that the field has atypical semantics. The specification of an ‘other’ field describes the actual semantics.

In addition to the above defined field types, a field is also either volatile or non-volatile. A non-volatile field is never changed autonomously by hardware, while a volatile field may be changed autonomously by hardware. When a field is volatile, its specification describes the actual semantics. A non-volatile ‘read-only’ field has an immutable value.

When the value of a field is not architecturally defined, the field is said to have an undefined value. For example, a writable field might have an undefined value between hard reset and the first time that it is written.

A field may have some values which are reserved. These values must not be written into that field, otherwise the behavior of the access is not well defined by the architecture. The specification of a particular writable field which has reserved values will enumerate the reserved values and may elaborate on the actual behavior. It is possible that all values apart from one specific value may be reserved. In this case, the field must be programmed with that specific value.

Control register layout

The standard table format for describing the layout of a control register is illustrated in *Table 6*.

<table>
<tr><th colspan="4">REGISTER</th><th colspan="2">OFFSET</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">FIELD</td><td>Bits</td><td>Size</td><td>Volatile</td><td>Synopsis</td><td>Type</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Operation</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Read</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Write</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Hard_reset</td></tr>
</table>

Table 6: Standard table format for describing a control register layout

The capitalized fields in this table are place-holders for the following information:

- Register - the name of the register.
- Offset - the byte-offset of the register in the control block of the module containing the register.
- Field - the name of the field.
- Bits - the bit numbers occupied by this field. The least significant bit in a control register is bit 0; the most significant bit in a control register is bit 63. A single number indicates a single bit. The notation [x:y] represents the inclusive contiguous range of bits starting at bit x and ending at bit y.
- Size - the number of bits occupied by this field.
- Volatile - a '✓' symbol indicates that the field is volatile, while '-' indicates that the field is not volatile.
- Synopsis - a summary of the purpose of this field.

- Type - the type of this field. This can be 'RES' to indicate a 'reserved' field, 'RO' to indicate a 'read-only' field, 'RW' to indicate a 'read-write' field or 'other' to indicate an 'other' field.
- Operation - defines the operation of this field.
- Read - defines the behavior of this field for a valid read access.
- Write - defines the behavior of this field for a valid write access.
- Hard_reset - defines the value of this field after a hard reset.

The set of rows used to describe a field are repeated for each field in the control register.

2.7.3 Version control registers

The control register at offset 0 of every control block is the version control register of the module that implements that control block. Thus, the physical address of a control block is the same as the physical address of the VCR in that control block.

Every VCR uses the same layout. This allows software to parse a VCR value without knowledge of the implementation of the module that provided the VCR. It is architecturally guaranteed that no VCR will ever have a value of zero.

The VCR of each module contains the following fields:

- PERR_FLAGS contains the packet error flags (p-error flags) which report the error status of the interface between this module and the packet-router. The set of supported flags and their standard semantics are described in *Section 2.7.4: P-error flags on page 37*. Further information on the supported p-error flags of the module is given in the VCR description for that module.
- MERR_FLAGS contains module specific error flags (m-error flags). The set of supported flags (if any) and their semantics are given in the VCR description for that module.
- MOD_VERS is provided to allow software to distinguish different versions of a module. This allows software to take appropriate action if there are differences between module versions.
- MOD_ID is provided to allow software to identify and distinguish different modules.
- BOT_MB indicates the number of data blocks associated with this module.

- TOP_MB - if there is one or more data blocks associated with this module then TOP_MB is the number of control blocks associated with the module. If there are no data blocks associated with this module then TOP_MB is the offset of the last block associated with this module.

  If a module is associated with data blocks, then these data blocks will be contiguous in the address space. This allows ranges of data blocks to be described as the inclusive range from the value of BOT_MB to the value of TOP_MB.

The VCR format is illustrated in *Table 7*.

MODULE.VCR				**0x000000**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PERR_FLAGS	[7:0]	8	✓	P-port error flags	Varies
	Operation		See *Section 2.7.4: P-error flags on page 37*		
	When read		See *Section 2.7.4: P-error flags on page 37*		
	When written		See *Section 2.7.4: P-error flags on page 37*		
	HARD reset		0		
MERR_FLAGS	[15:8]	8	✓	P-module error flags (module specific)	Varies
	Operation		See *Section 2.7.5: M-error flags on page 41*		
	When read		See *Section 2.7.5: M-error flags on page 41*		
	When written		See *Section 2.7.5: M-error flags on page 41*		
	HARD reset		0		
MOD_VERS	[31:16]	16	—	Module version	RO
	Operation		Used to indicate module version number		
	When read		Returns *MOD_VERS*		
	When written		Ignored		
	HARD reset		*MOD_VERS*		

Table 7: Standard VCR format

MODULE.VCR				**0x000000**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MOD_ID	[47:32]	16	—	Module identity	RO
	Operation		Used to identify module		
	When read		Returns *MOD_ID*		
	When written		Ignored		
	HARD reset		*MOD_ID*		
BOT_MB	[55:48]	8	—	Bottom memory block	RO
	Operation		Used to locate bottom memory block in address space		
	When read		Returns *BOT_MB*		
	When written		Ignored		
	HARD reset		*BOT_MB*		
TOP_MB	[63:56]	8	—	Top memory block	RO
	Operation		Used to identify top memory block		
	When read		Returns *TOP_MB*		
	When written		Ignored		
	HARD reset		*TOP_MB*		

Table 7: Standard VCR format

The italicized fields in this table are place-holders for the following information:

- MODULE - the name of the module that contains this VCR. The module name will be one of the modules provided by the SH-5.
- MOD_VERS - the version number of this module.
- MOD_ID - the identity of this module.
- BOT_MB - the relative value of the bottom memory block of this module.
- TOP_MB - the relative value for the top memory block of this module.

The values of these fields for the SH-5 modules is given in the register descriptions of each module chapter in this manual.

2.7.4 P-error flags

The p-error flags are a set of eight flags in a SH-5 module's VCR which indicate errors in the interface between that module and the packet-router. Two of these error flags are reserved and always read zero. The remaining six error flags are used in a standard way by all SH-5 modules, though not all modules implement all of these flags. If a module does not implement a particular p-error flag, then that flag has reserved behavior.

All p-error flags are zero after hard reset. Implemented p-error flags are volatile and are set to 1 by hardware whenever the associated error condition arises. The p-error flags will be cleared autonomously by hardware only at hard reset. Software can read and write these flags at any time using appropriate accesses.

It is possible for multiple error conditions to be triggered by a single request. The actual behavior in such cases is specified by the module receiving that request.

The complete set of supported p-error flags is given in *Table 8*.

Bit name	Bit	Size	Volatile?	Synopsis	Type
ERR_RCV	0	1	✓	An error response has been received	RW or RES
	Operation		This bit is set by the module hardware if an error response is received by that module from the packet-router. It indicates that an earlier request from that module was invalid. This bit will be cleared autonomously by hardware only at hard reset.		
	When read		Returns current value		
	When written		If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.		
	HARD reset		0		

Table 8: P-error flags

<table>
<tr><th>Bit name</th><th>Bit</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">ERR_SNT</td><td>1</td><td>1</td><td>✓</td><td>An error response has been sent</td><td>RW or RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the module hardware if an error response is sent by that module to the packet-router. It indicates that an earlier request to that module was invalid.This bit will be cleared autonomously by hardware only at hard reset.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">BAD_ADDR</td><td>2</td><td>1</td><td>✓</td><td>A request for an ‘UNDEFINED’ control register has been received</td><td>RW or RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the module hardware if the module receives a request for an ‘UNDEFINED’ control register. This bit will be cleared autonomously by hardware only at hard reset.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 8: P-error flags

<table>
<tr><th>Bit name</th><th>Bit</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">UNSOL_RESP</td><td>3</td><td>1</td><td>✓</td><td>An unsolicited response has been received</td><td>RW or RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the module hardware if the module detects that it has received an unsolicited response. This bit will be cleared autonomously by hardware only at hard reset.

A response is unsolicited if it does not match an outstanding request sent by that module. If a module is incapable of generating requests then all responses to that module are unsolicited. All responses with illegal destinations are routed to the DEBUG where they are signalled as unsolicited responses.

It is possible that a module may receive an unsolicited response which is not detected as unsolicited. This will cause that module to exhibit architecturally undefined behavior.

Software can attempt to clear this flag by writing a zero to it. However, this does not necessarily remove the cause of this error condition. If the error condition persists then this flag will continue to be set until a subsequent hard reset.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 8: P-error flags

Bit name	Bit	Size	Volatile?	Synopsis	Type
BAD_DEST	4	1	✓	A request with an illegal destination has been received	RW or RES
	Operation		This bit is set by the module hardware if a request with an illegal destination is received by that module from the packet-router. This bit will be cleared autonomously by hardware only at hard reset. All requests with illegal destinations are routed to the DEBUG. The DEBUG is the only module that supports this bit.		
	When read		Returns current value		
	When written		If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.		
	HARD reset		0		
BAD_OPC	5	1	✓	A request with an unsupported opcode has been received	RW or RES
	Operation		This bit is set by the module hardware if a request with an unsupported opcode is received by that module from the packet-router. This error can arise because not all modules support all packet-router opcodes. This bit will be cleared autonomously by hardware only at hard reset.		
	When read		Returns current value		
	When written		If this bit is implemented by this module, writes update the current value. If this bit is not implemented by this module, writes are ignored. Software may write to this bit at any time.		
	HARD reset		0		
—	[7:6]	2	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 8: P-error flags

2.7.5 M-error flags

The m-error flags are a set of eight flags in a module's VCR which indicate errors specific to that module. If a module does not implement a particular m-error flag, then that flag has reserved behavior.

All m-error flags are zero after hard reset. Implemented m-error flags are volatile and are set to 1 by hardware whenever the associated error condition arises. The m-error flags will be cleared autonomously by hardware only after hard reset. Software can read and write these flags at any time using appropriate accesses.

2.7.6 Memory map conventions

The SH-5 physical memory map is organized so that each control block contains a VCR at offset 0. Additionally, the address range of each data block is described by the VCR of its associated control block. In general, each control block is allocated at a higher address than its data block. The one exception to this rule is the EMI: the EMI control block is at a lower address than the EMI data blocks. The memory map is also organized so that every memory block that is neither a control block nor a data block is an undefined block.

This organization allows software to scan the SH-5 memory map to locate control blocks and data blocks. The software can run on a SH-5 CPU, or it can run on the host and access SH-5 memory using the debug link protocol. The scanning can be achieved in a safe manner with accesses that do not cause major changes to the architectural state of the system. The scanning algorithm scans downwards through the memory map reading 8-byte words from addresses aligned to 2^{24} bytes.

This scan should start at the highest address aligned to 2^{24}, that is, 0xFF000000. The bot_mb and top_mb information in the VCR of each control block should be used to ensure that data blocks are skipped over in the scanning sequence. It is important to ensure that data blocks are not read from, since data blocks may correspond to areas of data that have to be configured carefully and to address space that is implemented using off-chip state. It is possible that reads from off-chip state could modify the state of external memory-mapped peripherals.

Each 8-byte word that is read will be to the first 8-byte word in a memory block. If the access is to an undefined block then the access will be to an illegal destination address. This will cause the following behavior:

- DEBUG.VCR.PERR_FLAGS.BAD_ADDR will be set since the access will cause a request to an illegal destination to be sent to the DEBUG module.
- DEBUG.VCR.PERR_FLAGS.ERR_SNT will be set since the access will cause an error response to be returned by the DEBUG.
- If the access is made by a SH-5 CPU, then the VCR of that CPU will have VCR.PERR_FLAGS.ERR_RCV set since the error response will be received by that CPU. If the access is made by the host, then the host will be returned an error response over the debug link.
- If the access is made by a SH-5 CPU, then the value loaded by the instruction causing that access will be undefined.

If the access is to a control block then the access will return the value of the VCR for that control block without setting any error bit nor generating an error response.

These different behaviors can be used to distinguish accesses to an undefined block from accesses to a control block. The scanning algorithm detects data blocks through the VCR values of control blocks. The scanning algorithm itself makes no accesses to locations within data blocks.

This algorithm can be used to classify each SH-5 memory block between 0x00000000 and 0xFF000000 (inclusive) as a control block, a data block or an undefined block. Since each VCR value contains a module identity and module version, it is possible for software to check for the presence of particular modules and to handle different module versions appropriately.

2.7.7 P-module specification standards

The specification of each module defines the functionality provided by that module. Each module defines the following:

- The memory map of each memory block associated with that module. In particular, the memory map of each control block associated with that module indicates whether each control register in that block has 'DEFINED', 'RESERVED' or 'UNDEFINED' behavior.
- The interactions of that module with the packet-router. This includes the set of transactions that can be initiated by that module, and the transactions that can be serviced by that module. The behavior and signalling of error cases is also fully described.

- The format of all control registers defined by that module and the behavior of all fields in each control register for different types of access.

2.8 SH-5 endianess and data mapping

The SH-5 system is able to process data accesses in a manner consistent with either a little endian or a big endian interpretation. At power-on reset the endian pin is sampled. The sampled value determines the system endianess until the system is next power-on reset. The Endianess is distributed to all on-chip modules which enables them to behave in a manner consistent with the endian mode selected.

Pin name	Abbreviation	I/O	Function
Endian Pin	BEN[a]	Input	CPU core is big endian mode if this pin is asserted at power-on reset. Otherwise The CPU is in little endian mode.

Table 9: System Endian pin

a. The actual name of the endian pin may be different from this please refer to the data sheet for the pin-out.

2.8.1 Accessing memory

The SH-5 system accesses memory consistent with either little or big endian mode depending on the mode at reset. Software written for the SH-4 should port to the SH-5 without modification for reasons of endian manipulation.

Details of the data format in memory in each endian format is given in *Volume 2 Peripherals, Chapter 1 External memory interface* and *Chapter 2 Flash external memory interface (FEMI)*.

2.8.2 Accessing device registers

Any CPU access to configuration registers made at the documented address and with the documented data width will preserve the significance of the data. This means that software does not require any byte swapping or endian manipulation in either endian mode. The register descriptions given in this manual is correct for both endian modes.

2.8.3 Accessing PCI memory space

The region of the SH-5 address space mapped to PCI memory space behaves uniquely. The PCI specification is defined to always be little endian. When the SH-5 is in big endian mode careful design of software is required to use this address space in order to give expected results. The endian data mapping is described in detail in *Volume 2 Peripherals, Chapter 3 PCI bus bridge*.

2.8.4 Using the SHdebug link

The SHdebug link is always little endian in all respects other than the correlation between addr/mask/data on DBUS transactions. When the SH-5 is in big endian mode careful attention should be given to using this link in order to give the expected results. The SHdebug link data mapping is described in detail in *Volume 3 Debug*.

2.8.5 SuperHyway byte lane mapping

The mapping of data bytes to SuperHyway byte lanes is shown in *Table 10* for little endian mode and *Table 11* for big endian mode.

Datum size (Bytes)	Mask	Byte lanes 7	6	5	4	3	2	1	0	Implied low order Address[a][2:0]
8	11111111	MSB							LSB	0
4	00001111					MSB			LSB	0
4	11110000	MSB			LSB					4
2	00000011							MSB	LSB	0
2	00001100					MSB	LSB			2
2	00110000			MSB	LSB					4
2	11000000	MSB	LSB							6
1	00000001									0
1	00000010									1
1	00000100									2

Table 10: Little endian SuperHyway mapping

Datum size (Bytes)	Mask	Byte lanes								Implied low order Address[a][2:0]
		7	6	5	4	3	2	1	0	
1	00001000									3
1	00010000									4
1	00100000									5
1	01000000									6
1	10000000									7

Table 10: Little endian SuperHyway mapping

a. Note that this is for explanation only; the SuperHyway doesn't transmit the low order 3 address bits as these are implied by the mask.

Datum size (Bytes)	Mask	Byte lanes								Implied low order Address[a][2:0]
		7	6	5	4	3	2	1	0	
8	11111111	MSB							LSB	0
4	00001111					MSB			LSB	4
4	11110000	MSB			LSB					0
2	00000011							MSB	LSB	6
2	00001100					MSB	LSB			4
2	00110000			MSB	LSB					2
2	11000000	MSB	LSB							0
1	00000001									7
1	00000010									6
1	00000100									5
1	00001000									4

Table 11: Big endian SuperHyway mapping

Datum size (Bytes)	Mask	Byte lanes								Implied low order Address[a][2:0]
		7	6	5	4	3	2	1	0	
1	00010000									3
1	00100000									2
1	01000000									1
1	10000000									0

Table 11: Big endian SuperHyway mapping

a. Note that this is for explanation only; the SuperHyway doesn't transmit the low order 3 address bits as these are implied by the mask.

For transactions involving more than one 8-byte quadword (that is, LD16, ST16, LD32 and ST32) the mask field is not used. The sequence of access is that the addressed quadword is accessed first then the address is incremented to the next quadword until the last quadword is transferred. The incrementing is performed modulo the access size so that this will involve address wrapping if the first address is not aligned to the access size. The sequence is the same in both big and little endian mode.

Table 12 and *Table 13* show the sequence of quadwords accessed for 16-byte and 32-byte operations.

Transaction address[3]	Second quadword address[3]
0	1
1	0

Table 12: 16-byte load/store quadword sequence

Transaction address[4:3]	Second quadword address[4:3]	Third quadword address[4:3]	Fourth quadword address[4:3]
00	01	10	11
01	10	11	00

Table 13: 32-byte load/store quadword sequence

Transaction address[4:3]	Second quadword address[4:3]	Third quadword address[4:3]	Fourth quadword address[4:3]
10	11	00	01
11	00	01	10

Table 13: 32-byte load/store quadword sequence

2.9 SH-5 undefined behavior

There are conditions which result in the SH-5 entering states that exhibit architecturally undefined behavior. When the SH-5 enters such a state the architecture does not define how the SH-5 will behave. The SH-5 implementation attempts to provide reasonable limits, where practical, to the effects of undefined behavior.

It is always possible to use a hard reset to return the SH-5 to an architecturally defined state. A hard reset can be applied from the host using the NOT_RST_IN pin or may be caused by the SH-5's watchdog timer. Hard reset can be used as a last resort to reboot an unresponsive SH-5.

Hard reset re-initializes and destroys large amounts of SH-5 state and this will typically hinder debugging. After a hard reset, each piece of SH-5 state will take either its hard-reset-defined value or an undefined value. Volatile external memory that requires refreshing may lose its value across a hard reset. External components that are chained to the SH-5's NOT_RST_OUT pin may also lose their pre-reset state. It is therefore important to be able to inspect SH-5 state from a host, in as many cases as possible, without using hard reset.

2.9.1 SH-5 chip-level architecturally undefined behavior

Driving the SH-5 outside of its specified operating range will result in the behavior of the whole SH-5 being undefined. This can be caused, for example, by violating the SH-5's electrical specification.

An example where software can directly cause this kind of failure is by programming the SH-5's clock controller with reserved values. This can lead to the generation of an inappropriate clock signal resulting in the behavior of the whole SH-5 being undefined.

These cases may render the whole SH-5 inoperable or unresponsive. They may cause an unexpected reset of the SH-5. It might not be possible to inspect the SH-5 state without correcting the problem and causing a hard reset. It is possible that such problems can cause the operating life of a SH-5 component to be degraded.

2.9.2 SH-5 module-level architecturally undefined behavior

Most instances of architecturally undefined behavior cause the behavior of a particular module to become architecturally undefined. An example is a write to an undefined part of the address space in a particular module. It may be architecturally undefined as to whether such a write is ignored or whether that write can cause a state change in some other part of the address space of that module due to aliasing. After such an access the state of that module is not defined by the architecture and the behavior of that module is not architecturally defined.

When a particular module exhibits architecturally undefined behavior, the function of that module may be compromised. For example, a CPU might stop executing instructions, a DMA engine might stop making accesses or an output port might stop producing data. It is possible that such a module could interact with other modules in a manner which causes those modules to exhibit architecturally undefined behavior as well.

In severe cases, a module exhibiting architecturally undefined behavior could cause the whole chip to exhibit architecturally undefined behavior. An example is that a CPU exhibiting architecturally undefined behavior could start executing arbitrary instructions. This could cause reads or writes to arbitrary parts of the memory system. It is possible that such writes could result in one of the conditions described in *Section 2.9.1*. These cases are considered to be extremely rare except in contrived examples. The rest of this section assumes that chip-level architecturally undefined behavior is avoided.

The SH-5 implementation attempts to provide a reasonable limit, where practical, to the effect of module-level architecturally undefined behavior. The motivation is to allow a host-based development system to access as much SH-5 state as possible in order to diagnose system problems even when one or more modules is exhibiting undefined behavior. This requires that the debug link continues to propagate packets and continues to have access to the SH-5 physical address space.

The SH-5 achieves this by ensuring that the packet-router and the p-ports can continue to propagate packets regardless of whether individual modules are in an architecturally undefined state. The packet-router implementation is always capable of routing packets. If a sending p-port is capable of sending a packet and a receiving p-port is capable of receiving that packet, then the packet-router will route that packet in a finite amount of time. In general, each p-port implementation is always capable of receiving a packet from the packet-router. There are exceptions to this, however, see *Section 2.9.3: Unresponsive modules on page 50*. A p-port implementation is capable of sending a packet providing that the associated module is not in an architecturally undefined state.

If a module is in an architecturally defined state then that module will interact with the packet-router according to its architectural specification. If a module is in an architecturally undefined state, then the interactions may differ from the architectural specification.

- The module might not generate request and response packets according to its architectural specification. For example, the module might discard outgoing packets without sending them, it might propagate undefined data in those packets or it might generate spurious packets. These actions may cause other modules to exhibit architecturally undefined behavior.
- The module will remove request and response packets from the packet-router within a finite amount of time. There are exceptions to this, however, see *Section 2.9.3: Unresponsive modules on page 50*. The module might not deal with those removed packets according to its architectural specification. For example, the module might service requests incorrectly, it might discard incoming request packets without servicing or responding to them, it might discard incoming response packets without completing the transaction or it might propagate undefined data.

The architecture guarantees that the packet-router and the p-ports continue to propagate packets. The architecture does not guarantee that a module exhibiting architecturally undefined behavior will generate and service those packets. In practice, however, the implementation of each SH-5 module is capable, in many architecturally undefined circumstances, of receiving request packets, servicing them and sending an appropriate response packet.

In summary, once a module has become architecturally undefined, there is no architectural guarantee that it will respond correctly to requests but it is likely that it will.

2.9.3 Unresponsive modules

There are certain circumstances under which some modules can indefinitely refuse to remove packets from the packet-router. These cases arise where the completion of transactions serviced by a module is dependent upon the external environment of the SH-5. If the external environment behaves in a way that causes a transaction never to complete, then that module may be forced to refuse to remove packets from the packet-router indefinitely.

The canonical example is a module that relies upon external memory to satisfy memory access requests. Each request typically causes an access to the external memory. If an access to external memory has infinite latency, then the request causing that access will be blocked indefinitely. This will typically cause further requests to also block. The amount of request buffering provided by any module must be finite, so eventually the module will have to refuse to accept request packets from the packet-router. It is also possible that a module in this state may also be unable to accept response packets from the packet-router.

The situations that can cause this behavior on SH-5 are:

- an external device accesses the EMI bus directly by acquiring ownership of the EMI bus and never returns ownership to the SH-5,
- the EMI timing parameters are mis-programmed in such a way that accesses to the EMI never complete,
- an external device accesses the PERIPHERAL bus directly by acquiring ownership of the PERIPHERAL bus and never returns ownership to the SH-5,
- an external device being accessed over the PERIPHERAL bus by the SH-5 inserts an infinite series of wait states,
- the PERIPHERAL bus timing parameters are mis-programmed in such a way that accesses to the PERIPHERAL bus never complete,
- the SH-5 attempts to send packets over the debug link to the host but the host refuses to read them.

These behaviors can cause the module to refuse packets from the packet-router indefinitely. The affected p-port or p-ports will have infinite latency for receiving packets as viewed from the packet-router. Such cases are exceptions to the rule that p-ports are always capable of receiving packets from the packet-router.

A single unresponsive module may cause any transaction involving that module to be blocked indefinitely. In particular, packets sent from the host to that module over the debug link may block. This may also block any packet that is subsequently sent down the debug link regardless of the destination module of that packet. Thus a single unresponsive module can cause the entire SH-5 to become unresponsive as viewed from the host.

The only way for the SH-5 to become responsive again is to remove the condition that is causing the blockage. In some cases, the only practical way to achieve this may be by applying an external hard reset.

The best approach is prevention: careful system design can prevent the SH-5 from entering one of these states. This places constraints on board design and on software design. Abuse of these constraints may result in a system where the SH-5 can enter an unresponsive state.

3 SH-5 CPU

3.1 Introduction

This chapter contains a bus functional description of the SH-5 CPU together with information regarding how external requests may affect the internal state of the processor.

3.2 CPU port

The CPU is a module which contains a single port onto the system bus. Both instruction cache refills and all external data accesses use this port. The CPU implements a single memory control block. The control block of the CPU memory-maps the version control registers and some debug registers[1].

3.2.1 Instruction fetch

The CPU port is used to fetch instructions for the CPU to execute. A CPU implementation typically uses prefetching to hide memory latency. The port fetches instructions by issuing a series of load accesses. These accesses may include any combination of **Load8**, **Load16**, and **Load32** transactions. The exact sequences used are dependent upon the CPU implementation.

1. See *Volume 3 Debug, Chapter 1 Debug/trace architecture* for details.

Instruction fetch with MMU disabled

If the CPU is executing with the MMU disabled then the value of the program counter directly indicates the effective address from which instructions are fetched. Since the Instruction cache is not used when the MMU is disabled, instruction fetches exclusively use **Load8** transactions.

Instruction fetch with MMU enabled

If the CPU executes with the MMU enabled the value of the program counter is translated to an effective address. Instruction fetches may be either cached or uncached depending on the PTE of the page from which instructions are fetched. Therefore an implementation may include any combination of **Load8**, **Load16**, and **Load32** transactions in order to effect instruction fetch.

3.2.2 Data access

The data accesses made by the CPU port depend upon the instruction sequence executed by the CPU, whether the execution occurs with the MMU enabled or disabled and, if the MMU is enabled, the PTE entry corresponding to the data access.

Only memory instructions and cache control instructions may directly cause the CPU to make data accesses. In addition, it is permissible for a cache implementation to access data independently of instruction execution to increase performance. The CPU may issue **Load8** **Load16**, **Load32**, **Store8**, **Store16**, **Store32** and **Swap8** transactions during the course of its operation.

Data access with MMU disabled

When the MMU is disabled all data accesses made by the CPU are the direct result of execution of a memory instruction. Furthermore only the **Load8** and **Store8** transactions are used for accesses. The **Load8/Store8** physical address is simply the effective address of the instruction (truncated for implementations where the effective address space is larger than the physical).

Data access with MMU enabled

When the MMU is enabled the data accesses made by the CPU during execution of memory and cache control instruction and by the cache operation closely depends on the implementation of the load/store unit of the CPU and will not be described here.[1]

1. They should be described here later.

For accesses to addresses on pages which have the PTEL.CB attribute of "device" only **Load8/Store8** transactions are used.

The most common mode of use of the cache would see the **Load32** and **Store32** transactions most frequently used for cache refills and writebacks.

3.3 Cache coherency support

Cache coherency transactions are provided primarily to support the integration of the PCI bridge into the system. However the coherency support is general and can be used by any module attached to the system interconnect.

3.3.1 Operand cache snooping

In order to support coherency of access for memory shared between the operand cache and another memory user (for example, a PCI bridge), the CPU allows the contents of the operand cache to be flushed or purged by external requestors on the system interconnect.

The semantics of the cache snooping transactions are described below.

flush Following the completion of a flush transaction to a physical address, the value in the CPU operand cache and the value at the physical address (as accessed by other memory users) is the same. The execution of a **flush** transaction may involve the operand cache issuing a **Store32** transaction to the physical address.

purge Following the completion of a purge transaction to a physical address, the CPU operand cache no longer holds a copy of data at that address. This execution of a **purge** instruction may involve the operand cache issuing a **Store32** transaction to the physical address.

The two cache control operations **flush** and **purge** are described more fully below.

3.3.2 Flush

The flush transaction has a single operand which is the physical address which is to be flushed from the cache:

```
flush <target address of CPU><data = physical address to be flushed>
```

When a flush transaction is received from the interconnect, by the CPU cache controller, it causes the cache controller to lookup the address in the cache. If the lookup yields a miss, or a hit to a line which is unmodified with regard to main memory, then the cache controller will issue a response to the flush request immediately following the lookup. If the lookup yields a hit to a line which is modified with regard to main memory then the cache controller causes a writeback of the line to main memory. Following the writeback the cache controller issues a response to the flush request.

Responses to flush requests are simple acknowledgments; they do not carry any data.

The *<target address of CPU><data = physical address to be flushed>* should be 32-bit aligned. The low order 2 bits, if not zero, are ignored.

3.3.3 Purge

The purge transaction has a single operand which is the physical address which is to be purged from the cache:

purge *<target address of CPU><data = physical address to be flushed>*

When a purge transaction is received from the interconnect by the CPU cache controller, it causes the cache controller to lookup the address in the cache. If the lookup yields a miss, then the cache controller will issue a response to the flush request immediately, following the lookup. If the lookup yields a hit then the cache controller causes a writeback of the line to main memory (if the line has been modified in the cache) and then invalidates the line. Following the invalidation the cache controller issues a response to the flush request.

Responses to purge requests are simple acknowledgments; they do not carry any data.

The *<target address of CPU><data = physical address to be purged>* should be 32-bit aligned. The low order 2 bits, if not zero, are ignored.

3.3.4 Coherency maintenance

The use of flush and purge by a module in association with appropriate cache behavior provides a level of cache coherency. In particular, it guarantees two properties.

- A read operation by module to an address in shared system memory will receive the value last written to that address. The time of the access is given as the time at which the flush is received by the cache controller. The module read operation is guaranteed to get a data value coherent with the value of system memory no earlier than the time of access.
- A write operation by a module to an address in shared system memory will be completed such that the data written is readable by all memory users after the time of access. The time of access is given as the time at which the write operation is performed to system memory following the purge of the data cache(s).

Note: These instructions are sufficient to support coherency within a concurrent read exclusive write software sharing network.

3.3.5 Cache controller behaviour

Assuming that the cache implements a valid bit and a dirty bit per line and that it has a means for performing a lookup on a physical address, *Figure 10* shows the state transitions necessary to support cache coherency on receipt of a snoop transaction. Note that the dirty state has a choice of transitions depending on the implementation option taken. *Table 14* describes the actions that the cache controller has to take when responding to a snoop transaction.

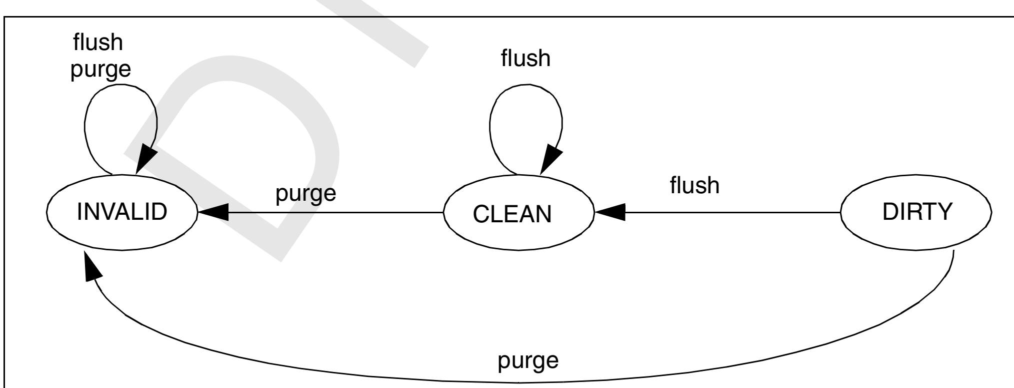


Figure 10: Cache state diagram

	Cache State		
	Invalid (miss)	**Dirty**	**Clean**
flush	respond	writeback line respond	respond
purge	respond	writeback line invalidate line respond	invalidate line respond

Table 14: Cache controller action table

3.3.6 Use of coherency transactions

When a module wishes to make a coherent request to shared memory the module performs the following routine:

1 Splits up the memory request into a number of non-cache-line straddling system interconnect requests.

For each of these requests it does the following:

2 For a read a **flush** request is sent to the datacache port, for a write a **purge** request is sent to the datacache port.

3 The module waits until it receives a response from the cache controller.

4 For a read, a load request is then sent to the main memory for a write a store request is sent to main memory.

5 The EMI's response indicates the completion of the coherent access.

3.4 Bi-endian support

The CPU supports both big and little endian modes. The endian modes affect how software views the results of memory accesses.

The physical address map is the same in either endian mode. On the system interconnect, addresses and byte enables are interpreted the same way by all devices attached to the system interconnect independent of the endian mode of the CPU.

3.5 Memory mapped registers

The behaviour of the ‘defined’ control registers implemented by the CPU are described in the following sections.

The base address of the CPU control registers, CPUBASE is given in the system interconnect chapter.

The watchpoint controller (WPC) has a number of memory-mapped registers which allow an external tool to control CPU behavior. These registers are described in *Volume 3: Debug*.

In addition the CPU has a version control register described below.

3.5.1 CPU.VCR

There is a version control register (VCR) for each major module on the chip. See *Section 2.7.3* for a general description of the of the VCR register.

The CPU.VCR control register is specified in *Table 15*. The PERR_FLAGS field of this control register reports errors conditions arising in the port.

CPU.VCR				**CPUBASE + 0x00**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PERR_FLAGS	[7:0]	8	3	P-port error flags	Varies
	Operation		See *Table 16: CPU.VCR.PERR_FLAGS on page 61*		
	When read		See *Table 16: CPU.VCR.PERR_FLAGS*		
	When written		See *Table 16: CPU.VCR.PERR_FLAGS*		
	HARD reset		0		
MERR_FLAGS	[15:8]	8	—	Module error flags (module specific)	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 15: CPU.VCR

CPU.VCR				CPUBASE + 0x00	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MOD_VERS	[31:16]	16	—	Module version	RO
	Operation		Used to indicate module version number		
	When read		0x0000		
	When written		Ignored		
	HARD reset		0x0000		
MOD_ID	[47:32]	16	—	Module identity	RO
	Operation		Used to identify module		
	When read		0x51E2		
	When written		Ignored		
	HARD reset		0x51E2		
BOT_MB	[55:48]	8	—	Bottom memory block	RO
	Operation		Used to identify bottom memory block		
	When read		For CPU0: 0x00		
	When written		Ignored		
	HARD reset		For CPU0: 0x00		
TOP_MB	[63:56]	8	—	Top memory block	RO
	Operation		Used to identify top memory block		
	When read		For CPU0: returns 0x00		
	When written		Ignored		
	HARD reset		For CPU0: 0x00		

Table 15: CPU.VCR

The set of supported p-error flags in CPU.VCR is given in *Table 16*.

Bit name	Bit	Size	Volatile?	Synopsis	Type
ERR_RCV	0	1	Yes	An error response has been received	RW
	Operation		This bit is set by the CPU hardware if an error response is received by the CPU port or CPU from the packet-router. It indicates that an earlier request from either of these p-ports was invalid.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
ERR_SNT	1	1	Yes	An error response has been sent	RW
	Operation		This bit is set by the CPU hardware if an error response is sent by the CPU to the packet-router. It indicates that an earlier request to the CPU was invalid. This will occur if: • The CPU port receives an unsupported request opcode. • The CPU port receives any request packet.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
BAD_ADDR	2	1	Yes	A request for an 'UNDEFINED' control register has been received	RW
	Operation		This bit is set by the CPU hardware if the CPU receives a **LoadWord** or **StoreWord** request for an 'undefined' control register in the CPU memory map.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 16: CPU.VCR.PERR_FLAGS

Bit name	Bit	Size	Volatile?	Synopsis	Type
UNSOL_RESP	3	1	Yes	An unsolicited response has been received	RW
	Operation		This bit is set by the CPU hardware if an unsolicited response is detected by the CPU p-port or CPU p-port.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
BAD_OPC	5	1	Yes	A request with an unsupported opcode has been received	RW
	Operation		This bit is set by the cpu hardware if a request with an unsupported opcode is received by the cpu from the packet-router.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
—	4, 6, 7	3	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 16: CPU.VCR.PERR_FLAGS

4 DMA controller

SH-5 integrates an on-chip 4-channel direct memory access controller (DMAC). The DMAC can be used in place of the CPU to perform high-speed data transfers between memory-mapped internal devices modules and main memory and to perform memory to memory transfers.

4.1 Features

The DMA controller has the following features:

- Four independent channels. Throughout this document the channels 0 to 3 of the DMAC are referred to as DMA[n], where n can be 0, 1, 2 or 3. The registers of the channels, for example, the SAR of DMA channel 0 is referred to as DMAC.SAR[0].
- Selection of 1-, 2-, 4-, 8-, 16- or 32-byte transfer unit.
- Programmable increment/decrement of source/destination addresses.
- Automatic or peripheral module transfer request modes
  - Auto request: the transfer request is generated automatically within the DMAC.
  - Requests from peripherals modules: Transfer requests from modules such as the SCIF, and TMU. The association between requester and channel is programmable.
- Two types of DMAC channel priority ordering:
  - Fixed priority mode: Channel priorities are permanently fixed.
  - Round robin mode: Sets the lowest priority for the channel that last executed a transfer.

- DMA channel suspension/resume.
- DMA channels can be globally or independently suspended. This allows Programmable Interrupt generation on normal or abnormal transfer completion.
- Error detection and error interrupt generation.

4.2 Address map

The DMAC comprises two general registers the VCR and COMMON register and five registers which are specific to each of the four DMA channels. A DMA channel N is characterized by the registers: the source address register DMAC.SAR[N], the destination address register DMAC.DAR[N], the transfer count register DMAC.COUNT[N], the channel control register DMAC.CTRL[N] and the channel status register DMAC.STATUS[N].

Register name	Description	Address offset
DMAC.VCR	Version control	0x00
DMAC.COMMON	DMA operation	0x08
DMAC.SAR[N]	channel N source address	0x10 + (0x28 * N)
DMAC.DAR[N]	channel N destination address	0x18 + (0x28 * N)
DMAC.COUNT[N]	channel N transfer count	0x20 + (0x28 * N)
DMAC.CTRL[N]	channel N control register	0x28 + (0x28 * N)
DMAC.STATUS[N]	channel N status register	0x30 + (0x28 * N)
DMAC.DMAEXG	external request control	0xc0

Table 17: DMAC registers

4.3 Operation

When there is a DMA transfer request, the DMAC starts the transfer according to the predetermined channel priority order. It ends the transfer when the transfer end conditions are satisfied. Transfers can be requested in two modes: auto-request, and on-chip peripheral module request. In either mode the transfer is dual address; the DMAC uses a load transaction to fetch data from the source address, buffers the data in the DMAC itself then uses a store transaction to write data to the destination address.

4.3.1 DMA basic transfer procedure

After the desired transfer parameters have been configured in the DMA source address registers (DMAC.SAR[N]), destination address registers (DMAC.DAR[N]), transfer count registers (DMAC.COUNT[N]), channel control registers (DMAC.CTRL[N]), and common registers (DMAC.COMMON[N]), the DMAC transfers data on the configured channels according to the following procedure:

1 The DMAC checks to see on which channels transfer is enabled (DMAC.CTRL[N].TRANSFER_ENABLE=1, DMAC.COMMON.MASTER_ENABLE=1, DMAC.COMMON.NMI_FLAG=0).

2 For each enabled channel, when a transfer request is issued, the DMAC transfers one transfer unit of data (determined by the setting of DMAC.CTRL[N].TRANSFER_SIZE). In auto-request mode, the transfer begins automatically when the DMAC.CTRL[N].TRANSFER_ENABLE bit and DMAC.COMMON.MASTER_ENABLE bit are set to 1. The DMAC.COUNT[N] value is decremented by 1 for each transfer.

3 When the specified number of transfers have been completed (that is, when DMAC.COUNT[N] reaches 0), the transfer on that channel ends normally. The DMAC engine checks to see if the DMAC.CTRL[N].INTERRUPT_ENABLE bit is set to 1, and if set, an interrupt DMTE[n] (DMAC Transfer End) is issued to the interrupt controller for that channel. The bit DMAC.STATUS[N].TRANSFER_END is set to 1 during the same state.

4.3.2 Configuring a DMA channel

A DMA channel is configured as follows:

1. Disable the channel (i) by ensuring that DMAC.CTRL[i].TRANSFER_ENABLE=0 and ensure that any error flags relating to the channel in DMAC.COMMON are clear.
2. Program the channel parameters by writing DMAC.SAR[i], DMAC.DAR[i], DMAC.COUNT[i] and DMAC.CTRL[i] registers.
3. Enable the channel by writing DMAC.CTRL[i].TRANSFER_ENABLE=1.

Provided the DMAC module is enabled (DMAC.COMMON.MASTER_ENABLE=1) the channel will be ready to start transferring data subject to DMAC.CTRL[i].RESOURCE_SELECT.

The only permitted write to the SAR, DAR, COUNT, or CTRL registers of an enabled channel is to disable that channel. Attempting to reconfigure the parameters of an enabled channel is undefined.

4.3.3 Errors and suspended channels

Once a channel has been configured and enabled it may

- complete normally as described above in *Section 4.3.1*,
- complete abnormally which means that an error has arisen which causes data transfer to stop before normal completion,
- be suspended partway through the configured number of transfers.

DMAC errors

An error on a channel may occur any time after the channel has been enabled and before it completes normally. This results in abnormal completion.

The DMAC can detect two types of errors:

- misalignment of the SAR or DAR addresses with respect to the transfer size chosen,
- a transfer load or store request has produced an error response for example, when access is attempted to an undefined or reserved address or does not support the chosen transfer size.

In the case of a misalignment DMAC sets:

DMAC.STATUS[N].ADDRESS_ALIGN_ERROR = 1 of the corresponding channel DMAC.COMMON.ADDRESS_ALIGNMENT_ERROR field bit for this channel.

In the case of receiving an error response the DMAC sets:

DMAC.COMMON.ERROR_RESPONSE field bit for this channel.
DMAC.VCR.PERR_FLAGS.ERR_RCV = 1.

In addition to the above in both error cases the DMAC does the following:

- it terminates further accesses by that channel,
- it clears the DMAC.CTRL[N].TRANSFER_ENABLE bit to '0',
- it flags an interrupt DERR(DMAC error), see *Section 4.3.7 on page 70*),
- the status of the SAR, DAR and COUNT of the errant channel are left in their state immediately prior to detection of the error.

Transfers on other channels will progress as programmed.

Suspension

Channel suspension occurs when an enabled DMA channel is temporarily halted due to the action of software or the occurrence of non-maskable interrupt (NMI).

- When an NMI interrupt occurs, transfer on all the channels is suspended and the DMAC does the following
  - Sets the DMAC.COMMON.NMI_FLAG to 1.
  - The DMAC retains the control register values and the transfer data.
  - DMAC resumes when NMI is no longer asserted and the software reads the DMAC.COMMON.NMI_FLAG then clears the DMAC.COMMON.NMI_FLAG to 0.
- Transfer may be suspended on all otherwise enabled channels when DMAC.COMMON.MASTER_ENABLE bit is cleared to 0 by software. Writing 0 to the DMAC.COMMON.MASTER_ENABLE bit starts the process of suspending the channels. When the DMAC.COMMON.MASTER_ENABLE bit is read as 0 by software this confirms that suspension has occurred.

  In this case, the channels stop issuing loads and stores to service the channels when the current transfer(s) (that is, load store pair) completes. The SAR, DAR and COUNT will reflect the state of the suspended channels. The DMAC resumes when DMAC.COMMON.MASTER_ENABLE is set to 1.

4 Enabled channels may be suspended individually by clearing the DMAC.CTRL[N].TRANSFER_ENABLE bit to 0. Writing 0 to a DMAC.CTRL[N].TRANSFER_ENABLE bit starts the process of suspending the channel(s). When a DMAC.CTRL[N].TRANSFER_ENABLE bit is read as 0 by software this confirms that suspension(s) has occurred.

In this case, the channel stops issuing loads and stores to service the channel when the current transfer (that is, load store pair) completes. The SAR, DAR and COUNT will reflect the state of the suspended channel.

DMAC resumes operation on each channel when its TRANSFER_ENABLE bit is set to 1.

4.3.4 DMA channel completion status

The completion status of the DMAC's enabled channels can be identified by either using the interrupt enable or by polling the channel transfer count until it reaches zero.

- DMAC.CTRL[N].INTERRUPT_ENABLE bit is set to 0. The DMAC.COUNT[N] for that particular channel can be checked for a zero value for request completion. Upon completion of all the channels transfers the DMAC hardware sets a 1 on the DMAC.STATUS[N].TRANSFER_END field. The DMAC does not interrupt the CPU in this case.
- DMAC.CTRL[N].INTERRUPT_ENABLE bit is set to 1. The DMAC hardware sends an interrupt to the cpu on completion of all transfers on a channel. Also, this is flagged by the DMAC setting a 1 in the field DMAC.STATUS[N].TRANSFER_END.

4.3.5 Request modes

Transfer requests may be generated by devices associated with the data transfer source or destination, but they can also be issued by the CPU or on-chip peripheral modules that are not associated with the source nor the destination addresses.

Transfers can be requested in two modes: auto-request or on-chip peripheral module request. The transfer request mode is selected for each channel by means of the RESOURCE_SELECT field in the DMAC.CTRL[N] register for the channel.

Auto Request Mode

When there is no transfer request signal from an on-chip peripheral module to request a transfer, the auto-request mode allows the DMAC to automatically generate a transfer request signal internally. This mode can be used by the CPU to set up memory to memory moves. When the TRANSFER_ENABLE bit in DMAC.CTRL[N] register for the channel and the MASTER_ENABLE bit in the DMAC.COMMON register are set to 1, the transfer begins (provided the NMI_FLAG in the DMAC.COMMON register and the ADDRESS_ALIGN_ERROR bit in DMAC.STATUS[N] register for the channel are all 0 and the DMAC.COUNT[N] is non zero).

On-Chip Peripheral Module Request Mode

The DMAC supports a transfer to be requested by any of seven peripherals that may be added onto the system. It treats all the peripherals same. DMAC hardware receives transfer requests from any of these peripherals, depending on the configuration of DMAC.SAR[N], DMAC.DAR[N] and DMAC.CTRL[N].RESOURCE_SELECT, and executes a transfer from SAR to DAR.

However when the transfer request is set to a particular peripheral, the transfer source/destination is normally that peripheral's memory mapped source/destination register respectively. For example, the peripheral SCIF on the SH-5 system falls under the category of peripherals supported by the DMAC. If the transfer request is set to the REQ signal coming off the signal RXI (transfer request by SCI receive-data-full interrupt) from SCIF, the transfer source must be the SCIF's receive data register (SCRDR). In the same vein when the transfer request REQ is coming off the signal TXI (transfer request by SCIF transmit-data-empty interrupt), the transfer destination must be the SCIF's transmit data register (SCTDR).

4.3.6 Channel priorities

If the DMAC receives simultaneous transfer requests on two or more channels, it selects a channel according to a pre-determined priority system, either in a fixed mode or round robin mode. The mode is selected with PRIORITY field in the DMA operation register (DMAC.COMMON).

Fixed Mode

In this mode, the relative channel priorities remain fixed. The following priority order is available in fixed mode:

CH0 $\rightarrow$ CH1 $\rightarrow$ CH2 $\rightarrow$ CH3

All the channels in the fixed priority mode operate in a steal mode meaning the lower priority channel can steal control from the higher priority channel if the channel is idle. The higher priority channel regains control or loses control depending on the speed at which the serviced unit requests the DMAC. For example, if channel 0 is programmed to service in auto request mode then channel 1 gets control only when transfer on channel 0 is completed.

Note: *A channel in auto request mode will perform all of its transfers contiguously to completion without the possibility of pre-emption from other channels.*

If channel 0 is programmed to service in on-chip peripheral request mode, control moves to channel 1 if a request is not issued for channel 0 peripheral as soon as the DMAC is ready to issue a new transfer. This is to ensure that the full capacity of having multiple outstanding requests to the SuperHyway is able to be utilized.

Round Robin Mode

In round robin mode, each time the transfer of one transfer unit (1-, 2-, 4-, 8-, 16- or 32-byte) ends on a given channel, that channel is assigned the lowest priority level. The order of priority in round robin mode immediately after a reset is 0 > 1 > 2 > 3.

There are four channel priority states:

Priority ordering state	Entering conditions
0 > 1 > 2 > 3	Reset or last transfer was on channel 3
1 > 2 > 3 > 0	last transfer was on channel 0
2 > 3 > 0 > 1	last transfer was on channel 1
3 > 0 > 1 > 2	last transfer was on channel 2

Table 18: Round robin priority changes

In round robin mode, the DMAC gives each channel in turn an opportunity to perform a transfer.

If the channel with the highest priority has a DMAC.COUNT.RESOURCE_SELECT field which specifies a peripheral that is not asserting a request, the DMAC transfer will occur from the channel of the next highest priority that either has an asserted request from its peripheral or is in auto-request mode. The priority ordering state is then adjusted as shown in *Table 18*.

4.3.7 Interrupts

Interrupts may be raised when an enabled DMAC channel completes. Interrupts may be enabled or disabled (that is, masked) for normal completion using the DMAC.CTRL[i].INTERRUPT_ENABLE bit. Interrupts for abnormal completion may not be masked by the DMAC.

Refer to the *Chapter 6: Interrupt controller on page 107* for the interrupt codes which are presented to the CPU when the DMAC raises an interrupt.

After an interrupt is raised it is continuously asserted to the interrupt controller as long as the channel causing the interrupt remains enabled. Clearing an interrupt may be achieved by clearing the TRANSFER_ENABLE bit to 0 in the DMAC.CTRL register for the channel concerned. Disabling all channels by clearing the MASTER_ENABLE bit to 0 in the DMAC.COMMON register will clear all interrupts.

Before software re-enables a DMAC channel following abnormal completion any alignment error should be rectified and status bits indicating the reason for the DERR interrupt should be cleared so that subsequent channel completion is unambiguous.

If a channel i is enabled with DMAC.CTRL[i].COUNT = 0 (and with DMAC.SAR[i] and DMAC.DAR[i] aligned with respect to DMAC.CTRL[i].TRANSFER_SIZE) it will instantly complete normally without any transfers having taken place.

	Interrupt name	Conditions for raising interrupt	Clearing the interrupt
Normal completion	DMTE[*i*] *i*=0,1,2,3	Transfer on channel i is enabled AND DMAC.COUNT[i] = 0 AND DMAC.CTRL[i].INTERRUPT_ENABLE = 1	DMAC.CTRL[i].TRANSFER_ENABLE = 0 OR DMAC.COMMON.MASTER_ENABLE = 0
Abnormal completion	DERR	(DMAC.COMMON.ERROR_RESPONSE OR DMAC.COMMON.ADDRESS_ALIGNMENT) <> 0	Remove the condition OR OR DMAC.COMMON.MASTER_ENABLE = 0

Table 19: Interrupt states

4.3.8 Behavior of SAR, DAR and count

While the DMA is in action the values of the DMAC.SAR[N], DMAC.DAR[N], and DMAC.COUNT[N] registers are volatile, that is, software can only reliably observe these when the DMA is disabled (due to completion, suspension or error). In general, the state of the DMA when interrupted is:

- the SAR indicates the next source address to be read,
- the DAR indicates the next destination address to be written,
- the COUNT indicates the number of bytes remaining to be transferred from source to destination to complete the DMA.

The SAR is incremented when the load request is sent. DAR is incremented when the store request is sent. COUNT is decremented when the store response is received. Due to pipelining and buffering the SAR and DAR can increment multiple units ahead of the COUNT up to the amount of pipelining in the DMAC implementation. But, the DAR cannot move further ahead than SAR. COUNT should be considered the reference value as the amount of pipe-lining is implementation-specific and its timing is non-deterministic. In the case of a non-error suspension of a DMA (that is, by channel disable, DMA disable or NMI) then the DMA will be suspended in a clean state where SAR/DAR/COUNT are each consistent with the amount remaining to be transferred. It is straightforward for software to restart the DMA (and there will be no data loss).

In the case of an error suspension of a DMA (that is, through receipt of an error response from the source or destination), SAR and DAR can be observed to be ahead of the COUNT. If software remembers the original values of SAR and DAR, then software can determine the range of SAR and DAR values which caused the error response. It is difficult to pin-point the error exactly based on SAR/DAR because the timing of the pipe-lining will vary. However, typically there are only a few buffers in the whole of the DMA this should be accurate enough for debug purposes (for example, within 2 or 3 packets, < 100 bytes).

If software wishes to restart the DMA (perhaps by repeating or stepping over the failed transfer) then software should re-program SAR and DAR based on the information in COUNT since the SAR and DAR could be out of step with each other.

Note: The above is for incrementing DMA's. For non-incrementing modes obviously SAR *and* DAR *are not changing so have no issue.*

Potential for losing data

In the case of a successful read but a failed write, the read data will be lost. If the read data is from a peripheral, then it may not be possible to recover the lost data.

4.4 Register descriptions

4.4.1 DMAC.VCR

DMAC.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PERR_FLAGS	[7:0]	8	✓	P-port error flags	Varies
	Operation		See *Table 21 on page 75*		
	When read				
	When written				
	HARD reset		0		
MERR_FLAGS	[15:8]	8	✓	Module error flags	RO
	Operation		Not used for DMAC		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
MOD_VERS	[31:16]	16	—	Module version	RO
	Operation		Used to indicate module version number		
	When read		Returns 0x0000		
	When written		Ignored		
	HARD reset		0x0000		

Table 20: DMAC.VCR

DMAC.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MOD_ID	[47:32]	16	—	Module identity	RO
	Operation		Used to identify module		
	When read		Returns 0x0183		
	When written		Ignored		
	HARD reset		0x0183		
BOT_MB	[55:48]	8	—	Bottom memory block	RO
	Operation		Used to identify bottom memory block		
	When read		Returns 0x00		
	When written		Ignored		
	HARD reset		0x00		
TOP_MB	[63:56]	8	—	Top memory block	RO
	Operation		Used to identify top memory block		
	When read		Returns 0x00		
	When written		Ignored		
	HARD reset		0x00		

Table 20: DMAC.VCR

The set of supported p-error flags in DMAC.VCR is given in *Table 21*. The bit positions in this table are relative to the start of the DMAC.VCR.PERR_FLAGS field; this field starts at bit 0 of DMAC.VCR.

<table>
<tr><th>Bit name</th><th>Bit</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">ERR_RCV</td><td>0</td><td>1</td><td>✓</td><td>An error response has been received</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the module hardware if an error response is received by the module port or module from the SuperHyway. It indicates that an earlier request from either of these ports was invalid.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">ERR_SNT</td><td>1</td><td>1</td><td>✓</td><td>An error response has been sent</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the DMA hardware if an error response is sent by the DMA to the SuperHyway. It indicates that an earlier request to the DMA was invalid.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">BAD_ADDR</td><td>2</td><td>1</td><td>✓</td><td>A request for an ‘UNDEFINED’ control register has been received</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the DMA hardware if the DMA receives a request for an ‘UNDEFINED’ control register.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 21: DMAC.VCR.perr_flags

Bit name	Bit	Size	Volatile?	Synopsis	Type
UNSOL_RESP	3	1	✓	An unsolicited response has been received	RW
	Operation		This bit is set by the DMA hardware if an unsolicited response is received by the DMA from the SuperHyway.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
—	4	1	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
BAD_OPC	5	1	✓	A request with an unsupported opcode has been received	RW
	Operation		This bit is set by the DMA hardware if a request with an unsupported opcode is received by that module from the SuperHyway.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
—	[7:6]	2	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 21: DMAC.VCR.perr_flags

4.4.2 DMAC.COMMON

DMAC.COMMON				0x000008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PRIORITY	0	1	—	priority of active DMA channels	RW
	Operation		0: 0 ⟶ 1 ⟶ 2 ⟶ 3 1: Round Robin		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[2:1]	2	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		
MASTER_ENABLE	3	1	—	Enables / Disables operation of DMAC	RW
	Operation		0: All channels are disabled 1: channels are enabled as per DMAC.CTRL[N].ENABLE for channel N.		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 22: DMA common register

<table>
<tr><th colspan="4">DMAC.COMMON</th><th colspan="2">0x000008</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">NMI_FLAG</td><td>4</td><td>1</td><td>✓</td><td>Indicates NMI input.</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set regardless of whether or not the DMAC is operating. If this bit is set during data transfer, transfers on all channels are suspended. The CPU cannot write a 1 to DMAC.COMMON.NMI_FLAG. Clearing is performed by reading the flag when set to 1, then writing 0 to the flag</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Cleared when written with 0</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Undefined</td></tr>
<tr><td rowspan="5">—</td><td>[6:5]</td><td>2</td><td>—</td><td>RESERVED</td><td>RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">RESERVED</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns 0</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Ignored</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">ERROR_
RESPONSE</td><td>[10:7]</td><td>4</td><td>✓</td><td>Indicates the channel on which error response occurred.</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">These to be used in conjunction with the DERR interrupt, from the DMAC to the INTC, to specify the channel on which the error response has occurred.
When set to ‘1’, bit i of this field indicates whether channel i has received an error response.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Undefined</td></tr>
</table>

Table 22: DMA common register

DMAC.COMMON				0x000008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDRESS_ALIGNMENT_ERROR	[14:11]	4	✓	Indicates the channel on which an address alignment error occurred.	RW
	Operation		These bits are to be used in conjunction with the DERR interrupt, from the DMAC to the INTC, to specify the channel on which the alignment error has occurred. When set to ‘1’, bit *i* of this field indicates whether channel *i* ‘s mis-alignment has caused the error.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
—	[63:15]	49	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 22: DMA common register

4.4.3 DMAC.SAR[n]

DMAC.SAR[n] where n is in the range [3:0]				0x000010 + (n * 0x28)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDRESS	[31:0]	32	✓	Source address register for DMA channel n	RW
	Operation		This register indicates the address from which the next transfer unit will be fetched. This address will be incremented/decremented as DMA channel n proceeds in accordance with the setting of DMAC.CTRL[n].SOURCE_INCREMENT. The increment/decrement occurs immediately following the response from the fetch of the transfer unit. This address should be aligned with the data size specified in the DMAC.CTRL[n].TRANSFER_SIZE field.		
	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 23: DMAC.SAR[n] register

4.4.4 DMAC.DAR[n]

DMAC.DAR[n] where n is in the range [3:0]				0x000018 + (n * 0x28)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADDRESS	[31:0]	32	✓	Destination address register for DMA channel n	RW
	Operation		This register indicates the address from which the next transfer unit will be stored. This address will be incremented/decremented as DMA channel n proceeds in accordance with the setting of DMAC.CTRL[n].DESTINATION_INCREMENT. The increment/decrement occurs immediately following the response from the store of the transfer unit. This address should be aligned with the data size specified in the DMAC.CTRL[n].TRANSFER_SIZE field.		
	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 24: DMAC.DAR[n] register

4.4.5 DMAC.COUNT[n]

DMAC.COUNT[n] where n is in the range [3:0]				0x000020 + (n * 0x28)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
COUNT	[31:0]	32	✓	Transfer count for DMA channel n	RW
	Operation		Contains a count of the number of transfers remaining on this channel. Each transfer involves the move of a datum of size indicated by DMAC.CTRL[N].TRANSFER_SIZE field.		
	When read		Returns current value		
	When written		Updates current value Enabling a channel with DMAC.COUNT[n] = 0 will mean that it will complete immediately in the normal way but without any transfers taking place. The count is decremented by 1 on the completion of each transfer which does not generate an error.		
	HARD reset		Undefined		
—	[63:32]	32	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 25: DMAC.COUNT[n] register

4.4.6 DMAC.CTRL[n]

DMAC.CTRL[n] where n is in the range [3:0]				0x000028 + (n * 0x28)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TRANSFER_SIZE	[2:0]	3	—	Transfer data size	RW
	Operation	0: Transfer size is 1 byte 1: Transfer size is 2 bytes 2: Transfer size is 4 bytes 3: Transfer size is 8 bytes 4: Transfer size is 16 bytes 5: Transfer size is 32 bytes 6: Field is reserved 7: Field is reserved			
	When read	Returns current value			
	When written	Updates current value			
	HARD reset	Undefined			
SOURCE_ INCREMENT	[4:3]	2	—	Increment mode of source address	RW
	Operation	0: source address incremented (by transfer.size bytes) 1: source address decremented (by transfer.size bytes) 2: source address is neither incremented or decremented. All DMA channel n fetches will be from the same address. 3: Field is reserved.			
	When read	Returns current value			
	When written	Updates current value			
	HARD reset	Undefined			

Table 26: DMAC.CTRL[n] register

<table>
<tr><th colspan="4">DMAC.CTRL[n] where n is in the range [3:0]</th><th colspan="2">0x000028 + (n * 0x28)</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">DESTINATION_ INCREMENT</td><td>[6:5]</td><td>2</td><td>—</td><td>Increment mode of destination address</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">0: Destination address incremented (by transfer.size bytes)
1: Destination address decremented (by transfer.size bytes)
2: Destination address is neither incremented or decremented. All DMA channel n stores will be to the same address.
3: Field is reserved.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Undefined</td></tr>
<tr><td rowspan="5">RESOURCE_ SELECT</td><td>[10:7]</td><td>4</td><td>—</td><td>Selects transfer request source</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">0: Auto request
1: Peripheral 0 transfer request
2: Peripheral 1 transfer request
3: Peripheral 2 transfer request
4: Peripheral 3 transfer request
5: TMU transfer request
6: SCIF transmit transfer request
7: SCIF receive transfer request
[15:8]: Values reserved.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 26: DMAC.CTRL[n] register

<table>
<tr><th colspan="4">DMAC.CTRL[n] where n is in the range [3:0]</th><th colspan="2">0x000028 + (n * 0x28)</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">INTERRUPT_ ENABLE</td><td>11</td><td>1</td><td>—</td><td>Controls if an interrupt is sent after normal completion of this channel.</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">0: Interrupt not enabled for channel n
1: Interrupt enabled for channel n</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">TRANSFER_ ENABLE</td><td>12</td><td>1</td><td>—</td><td>Enables/disables DMA channel n</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">0: DMA not enabled for channel n
1: DMA enabled for channel n</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">—</td><td>[63:15]</td><td>51</td><td>—</td><td>RESERVED</td><td>RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">RESERVED</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns 0</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Ignored</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 26: DMAC.CTRL[n] register

4.4.7 DMAC.STATUS[n]

DMAC.STATUS[n] where n is in the range [3:0]				0x000030 + (n * 0x28)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TRANSFER_END	0	1	✓	DMA channel n has ended normally	RO
	Operation				
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
ADDRESS_ALIGN_ ERROR	1	1	—	DMA channel n address is mis-aligned	RO
	Operation		RESERVED		
	When read		Returns current value For enabled channels this is just a read-only copy of bit i of DMAC.COMMON.ADDRESS_ALIGNMENT: This field is generated from the SAR, DAR and transfer unit for the channel. It is not qualified by DMAC.COMMON.MASTER_ENABLE nor by DMAC.COMMON.TRANSFER_ENABLE.		
	When written		Ignored		
	HARD reset		0		
—	[63:2]	62	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 27: DMAC.STATUS[n] register

4.4.8 DMA external pin control (DMAEX)

This register is provided to support an extended feature for the SH-5 DMA operation. The feature is implemented as a separate block in the design called the DMAEXG (DMA external glue) block. The DMAEXG register, however is integrated with the DMA controller logic. The main functions of this register are:

- to control the external request/acknowledge signals,
- to control the request queue for the signals,
- to arbitrate the data available to the external requesting peripherals.

The attributes are valid only for peripherals connected external to the chip.

In the current eval chip implementation, the DMAC supports four external requesters. The external requesters receive their external request/acknowledge (or other required control signals) from a glue logic block that is functionally partitioned outside the logic definition of the DMAC. Software control of this glue logic block is achieved by the DMAEXG register.

The behavior of the external requests is not anticipated and hence there might not be a one-to-one match between the external glue logic issuing the request acknowledges and the DMAC issuing the request acknowledges. This problem arises from the fact the external peripheral may not wait long enough for issuing the next request.

This implementation of the external glue logic supports a request queue. Several off-chip peripherals may be using the DMA simultaneously so the implementation supports a queue for each requester. The queue is programmable to have a maximum depth of one or two requests.

Two request modes are handled by the DMAEXG block:

- cycle steal mode,
- burst mode.

In cycle steal mode, the external request may be asserted continuously while the DMAEXG block generates the appropriate requests to the DMAC.

In burst mode, the DMAEXG block receives only one request for a cycle and generates the requests by itself.

DMAC.DMAEXG				0x0000c0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DACK_READWRITE	[3:0]	4	—	Dack generated on read or write transaction detection	RW
	Operation		0: DACK asserted read transaction detection 1: DACK asserted on write transaction detection		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0 (implementation-specific)		
DREQ_ATTRIBUTE	[7:4]	4	—	dreq level- or edge- based	RW
	Operation		0: level mode 1: edge mode (negative edge)		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
DRACK_ATTRIBUTE	[11:8]	4	—	DRACK active high or low	RW
	Operation		0: active high 1: active low		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0 (implementation-specific)		

Table 28: DMAC.DMAEXG register

DMAC.DMAEXG				0x0000c0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DACK_ATTRIBUTE	[15:11]	4	—	dack active high or low	RW
	Operation		0: active high 1: active low		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
REQUEST_MODE	[19:16]	4	—	Requests in cycle steal or burst mode	RW
	Operation		0: cycle steal mode 1: burst mode		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
REQUEST_QUEUE_ DEPTH	[23:20]	4	—	Depth of request queue	RW
	Operation		External module request queues reside inside the DMAC external glue logic. This field controls the depth of the request queues. 0: one queue 1: two queues		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 28: DMAC.DMAEXG register

DMAC.DMAEXG				0x0000c0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
REQUEST_QUEUE_CLEAR_ENABLE	[27:24]	4	—	Request queue clear enable	RW
	Operation		External module request queues reside inside the DMAC external glue logic. This field is used to automatically clear the queues when the DMA transfer is completed or stopped. 0: If the enable signal from the DMAC is low (the DMA transfer is completed or stopped), the request queue is enabled and the DMAEXG continues receiving the external requests. 1: If the enable signal from the DMAC is low (the DMA transfer is completed or stopped), the request queue is disabled and the DMAEXG no longer recognizes any requests.			
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
—	[63:28]	36	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 28: DMAC.DMAEXG register

4.5 DMAC SuperHyway transactions

This section describes the different types of transactions that involve the DMAC module. In particular the treatment of requests received and the types of requests that are generated in response to its programming.

4.5.1 DMAC as a request target

The DMAC supports only quadword (that is, 64-bit) load and store requests to the registers listed in *Table 17 on page 64*. All other accesses will generate an error response and set the appropriate bit in the VCR register.

- If the request is of any other type than a **load8** or **store8** the BAD_OPC field of the DMAC.VCR will be set and an error response will be sent to the initiator. A list of the operations available on the SuperHyway are given in *Table 2 on page 17*
- If the request is to an address mapped to this module but not listed in *Table 17 on page 64* then the BAD_ADDR field of the DMAC.VCR will be set and an error response will be sent to the initiator. The range of addresses allocated to this module is specified in the system address map given in *Table 4 on page 26*.

4.5.2 DMAC as a request initiator

The DMAC initiates requests only as a result of enabled DMA channels. *Table 29 on page 91* shows the correspondence between the transfer size field (which is $\log_2$ of the transfer size in bytes) and the SuperHyway transaction used to implement the transfer. The largest transaction type able to be used for a transfer is used exclusively, so for example, a DMA transfer configured with a TRANSFER.SIZE field of 4 will be performed only using **load16** and **store16** transactions on the SuperHyway.

DMAC.CTRL.TRANSFER_SIZE	SHWY transaction used
0	Load8/Store8 (with 1 bit set in the mask)
1	Load8/Store8 (with 2 bits set in the mask)
2	Load8/Store8 (with 4 bits set in the mask)

Table 29: DMAC SHWY transactions

DMAC.CTRL.TRANSFER_SIZE	SHWY transaction used
3	Load8/Store8 (with 8 bits set in the mask)
4	Load16/Store16
5	Load32/Store32

Table 29: DMAC SHWY transactions

4.6 Power down

A safe procedure for entering a power down mode is for software to disable the DMAC and so suspend any active channels prior to putting the DMAC into a low power state.

This may be achieved by disabling the DMAC which leads to any active channels moving to the suspended state on a well-defined transfer boundary. As described in *Suspension on page 67* this is done by clearing the DMAC.COMMON.MASTER_ENABLE flag to 0. Because it may take an implementation-dependent number of cycles before the DMAC will be disabled. Software should wait until DMAC.COMMON.MASTER_ENABLE flag is read as 0 before proceeding.

Following exit from the powerdown state software can (if necessary) restore the setting of the DMAC.COMMON.MASTER_ENABLE flag which will cause the DMAC to resume its previous activity.

SuperH

5

Peripheral bridge

5.1 Introduction

The peripheral bridge manages accesses between the high performance SuperHyway interconnect and low data rate peripherals.

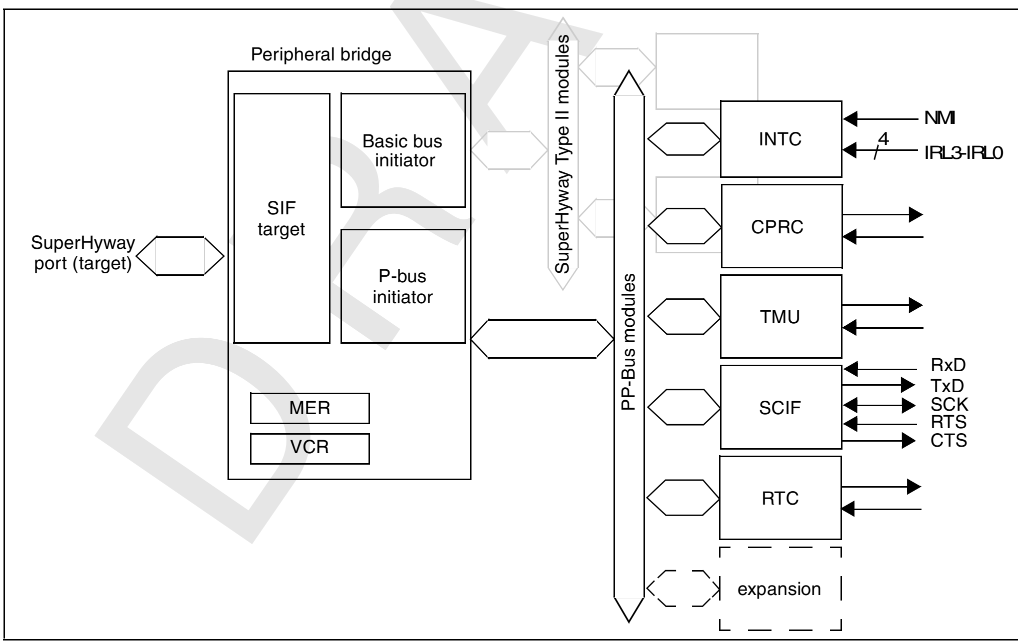


Figure 11: Peripherals subsystems architecture

This is achieved by grouping the modules into a single port on the SuperHyway, and handling any data size, data rate and endianness issues transparently. This ensures that future and existing modules can maintain both architectural and design compatibility with existing SuperH, ST and Hitachi design libraries[1] whilst providing a simple integration strategy for SH-5 systems in the future.

5.2 Functionality

5.2.1 Overview

The system views the majority of the SH-5 peripherals as a single SuperHyway module. The bridge maps these onto a single contiguous 32 Mbyte area of memory.

The peripheral bridge allocates a segment of this memory space to each device, and ensures that the view of the peripheral visible to the system is consistent. This includes responsibility for ensuring that all modules have the same representation of address and data for their registers, and that features such as unmapped areas of memory, and/or addition functionality is handled consistently.

For the Eval implementation, the peripheral subsystem divides it's memory space into a number of contiguous slots as below:

Slot 0 is reserved for the bridge status

Slots S1 to S15 are reserved for SuperHyway type 2 peripherals

Slots P0 to P15 and PEX are reserved for PP-Bus peripherals

The peripheral subsystem also maintains the state associated with the peripheral subsystem as a whole, handles the protocol and format changes required to map SuperHyway operations onto those required by the peripheral bus, and maintains the status associated with erroneous accesses to these devices.

1. SH7750, SH-4 CORE, XX-ST20, ST40

5.2.2 Address map

The memory map of the peripheral bridge is organized as follows:

Subsystem	Description	Address range[a]
Bridge state	B0 - peripheral bridge status	0x0000000 to 0x000FFFFF
External	S1 to S15: reserved for peripherals	0x0100000 to 0x0FFFFFF
INTC	P0: interrupt control	0x1000000 to 0x100FFFF
CPRC	P1: clock power reset controller	0x1010000 to 0x101FFFF
TMU	P2: timer management unit	0x1020000 to 0x102FFFF
SCIF	P3: serial control interface with fifo	0x1030000 to 0x103FFFF
RTC	P4: real time clock	0x1040000 to 0x104FFFF
External 5-15	P5 to P15: reserved PP-Bus areas	0x1050000 to 0x10FFFFF
External	P16: 15 Mbyte external area	0x1100000 to 0x1FFFFFF

Table 30: Peripheral bridge memory map

a. The address is given as an offset from the peripheral bridge base address as defined in the system organization.

Detailed descriptions of each peripheral are described in the associated chapter.

5.3 Operation

The system presents request to the peripheral bridge when it wishes to access peripheral devices. It uses the address to determine which region or device is being accessed.

5.3.1 Bridge registers

If the address of the operation is the bridge area of the peripheral subsystem, it is trapped locally and used to access information stored within the bridge.

If the access is to the bridge status area, it is not passed to the external peripherals and is used to access information about the bridge implementation, past erroneous operations and other status information associated with peripherals attached to this system.

The following operations are supported to this region:

- load eight bytes,
- store eight bytes.

If a mask bit is not asserted, or, any other operation is presented to this area an error is returned and the bit BAD_OPC is asserted in the VCR register.

If an access occurs to an reserved address within this area, an error is returned and the bit BAD_ADDR is asserted in the VCR register.

5.3.2 SuperHyway type 2 area

Operations onto the SuperHyway type 2 area are mapped onto the external interface. This is organized as a single region whose address decode is completed externally to the peripheral bridge.

If the interface is disabled (as indicated by the associated pin) then all accesses to this interface are treated as if the area is reserved, an error is returned and the bit BAD_ADDR is asserted in the VCR register.

If the interface is enabled, then the information is passed across the interface using the standard protocols.

5.3.3 PP-Bus area

Operations to the PP-Bus area are mapped onto one of a number of possible subregions. There are 17 subregions are currently defined, the first 16 allocating a 64 Kbytes area of memory, and the 17th allocates a 16 Mbyte region of memory.

Each subregion is pre-decoded by the peripheral bridge and it's own associated signal set, as follows:

- padrerr_xxx_n[1]
- pms_xxx_n
- pdouble_xxx_n
- pwait_xxx_n

These are used to determined if an access to that address and region is valid, the access width to that module and the number of cycles required to access that module.

If the PP-Bus peripheral signals an address error, an error is returned to the SuperHyway, and the bit BAD_ADDR is asserted in the VCR register.

The peripheral bridge supports the following PP-Bus operations:

- read/write byte,
- read/write double byte,
- read/write four byte.

The mapping between SuperHyway operations and the PP-Bus is shown in *Table 31*.

1. xxx corresponds to the module name or number. These signals are active low.

CPU operation	SuperHyway operation	SuperHyway mask	PP-Bus operation
load/store byte	load/store byte load/store eight bytes	“10000000” | “01000000” “00100000” | “00010000” “00001000” | “00000100” “00000010” | “00000001”	read/write byte
load/store two bytes	load/store two byte load/store eight bytes	“11000000” | “00110000” “00001100” | “00000011”	read/write two bytes
load/store four bytes	load/store four bytes load/store eight bytes	“11110000” | “”00001111”	read/write four bytes
load/store eight bytes	load/store eight bytes	“11111111”	two * read/write four bytes[a]

Table 31: Mapping SuperHyway operations and the PP-Bus

a. An 8-byte quantity considered as two packed 4-byte quantities, that is, in a little endian system, the 4 least significant bytes map onto the 32-bit register at address 0, the 4 most significant at address 4, whilst in a big endian system the 4 most significant bytes map to the register at address 0, and the 4 least significant bytes to address 4.

All other SuperHyway operations and mask combinations are not supported for accesses to PP-Bus peripherals and such requests will lead to an error being returned to the system. The peripheral subsystem will also assert the BAD_OPC flag in the VCR register.

For SuperHyway accesses which do not match the address and size of a single valid peripheral register an error response will be returned that may set either or both the BAD_ADDR | BAD_OPC error bits in the VCR register. This means, for example, that it is not possible to access multiple peripheral registers with a single SuperHyway transaction.

The SH-5 CXXX core pre-allocates the first five PP-Bus ports to core peripherals. These being the INTC, PMU, TMU, SCIF and RTC modules. Details on these blocks and their register definitions can be found later in this document.

Ports 5 to 16 and port 17 are mapped onto an external bus available for SOC integration. Unused ports must terminate the control signals as follows:

padrerr_xxx_n <= '0'

pwait_xxx_n <= '1'

This effectively disables port xxx and all accesses to that port will be considered as errors, returning an error to the SuperHyway, and asserting the BAD_ADDR flag in the VCR register.

5.3.4 Peripheral bridge registers

Address map

Subsystem	Offset	Register description
VCR	0x00000	Version control register
MER	0x00008	Module enable register
Reserved	0x00010 to 0xFFFFF	Reserved

Table 32: Peripheral bridge registers address map

Register definitions

PERIPH.VCR

This control register is specified in *Table 33*.

PERIPH.VCR				**0x000000**	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PERR_FLAGS	[7:0]	8	✓	P-port error flags	Varies
	Operation		Indicates a communication error has occurred between the system and a module.		
	When read		Error status		
	When written		Reset error status		
	HARD reset		0		

Table 33: PERIPH.VCR

PERIPH.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MERR_FLAGS	[15:8]	8	✓	Module error flags	Varies
	Operation		Indicates a communication error has occurred between the system and a module.		
	When read		Error status		
	When written		Reset error status		
	HARD reset		0		
MOD_VERS	[31:16]	16	—	Module version	RO
	Operation		Used to indicate module version number		
	When read		Returns 0x0000		
	When written		Ignored		
	HARD reset		0x0000		
MOD_ID	[47:32]	16	—	Module identity	RO
	Operation		Used to identify module		
	When read		0x448F		
	When written		Ignored		
	HARD reset		0x448F		
BOT_MB	[55:48]	8	—	Bottom memory block	RO
	Operation		Used to identify bottom memory block		
	When read		0x00		
	When written		Ignored		
	HARD reset		0x00		

Table 33: PERIPH.VCR

PERIPH.VCR				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TOP_MB	[63:56]	8	—	Top memory block	RO
	Operation		Used to identify top memory block		
	When read		0xFF		
	When written		Ignored		
	HARD reset		0xFF		

Table 33: PERIPH.VCR

The set of supported p-error flags in PERIPH.VCR is given in *Table 34*. The bit positions in this table are relative to the start of the PERIPH.VCR.PERR_FLAGS field; this field starts at bit 0 of PERIPH.VCR.

Bit name	Bit	Size	Volatile?	Synopsis	Type
ERR_SNT	1	1	✓	An error response has been sent	RW
	Operation		This bit is set by the bridge hardware if an error response is sent by the bridge to the SuperHyway. It indicates that an earlier request to the bridge was invalid.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
BAD_ADDR	2	1	✓	A request for an ‘UNDEFINED’ control register has been received	RW
	Operation		This bit is set by the bridge hardware if the bridge receives a request for an ‘UNDEFINED’ control register.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 34: PERIPH.VCR.PERR_FLAGS

Bit name	Bit	Size	Volatile?	Synopsis	Type
BAD_OPC	5	1	✓	A request with an unsupported opcode has been received	RW
	Operation		This bit is set by the bridge hardware if a request with an unsupported opcode is received by that module from the packet-router.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
—	[0] [4:3] [7:6]	1 2 2	—	RESERVED	RES
	Operation		RESERVED		
	When read		Undefined		
	When written		Write 0		
	HARD reset		Undefined		

Table 34: PERIPH.VCR.PERR_FLAGS

The set of supported m-error flags in PERIPH.VCR is given in *Table 35*. The bit positions in this table are relative to the start of the PERIPH.VCR.MERR_FLAGS field; this field starts at bit 8 of PERIPH.VCR.

Bit name	Bit	Size	Volatile?	Synopsis	Type
PBR_ERR	15	1	✓	Peripheral bridge registers area error	RW
	Operation		This bit is set by the bridge hardware if p-error flags is caused by the area access factor of peripheral bridge registers.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
STL_ERR	14	1	✓	SuperHyway bus area error	RW
	Operation		This bit is set by the bridge hardware if p-error flags is caused by the area access factor of SuperHyway bus.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
PPB_ERR	13	1	✓	PP-bus area error	RW
	Operation		This bit is set by the bridge hardware if p-error flags is caused by the area access factor of PP-Bus.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 35: PERIPH.VCR.MERR_FLAGS

<table>
<tr><th>Bit name</th><th>Bit</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">PBE_STATUS</td><td>[12:8]</td><td>5</td><td>✓</td><td>PP-bus area error status</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This bit is set by the bridge hardware if modules issue the signals of padrerr_xxx_n.
Bit 12 is 1’b0: P0 - P15 return PADRERR signal
Bit 12 is 1’b1: PEX return PADRERR signal
Bit [11:8]: copy of address [19:16]. This condition is also true if an access is made to a reserved part of the PPBUS area.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 35: PERIPH.VCR.MERR_FLAGS

PERIPH.MER

This control register is specified in *Table 36*.

<table>
<tr><th colspan="4">PERIPH.MER</th><th colspan="2">0x000000</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">PME_FLAGS</td><td>[16:0]</td><td>17</td><td>No</td><td>P-port module enable flags</td><td>RO</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">These bits are reset by module standby mode.
bit 0: P0
: :
bit 16: PEX</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">module enable status</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Ignored</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0x01F</td></tr>
</table>

Table 36: PERIPH.MER

PERIPH.MER				0x000000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SME_FLAG	17	1	No	Type 2 expansion port module enable flag	RO
	Operation		—		
	When read		0		
	When written		Ignored		
	HARD reset		0x0		
reserved	[63:18]	46	—	Reserved	RO
	Operation		Reserved		
	When read		Returns 0x0000		
	When written		Ignored		
	HARD reset		0x0000		

Table 36: PERIPH.MER

6 Interrupt controller

The interrupt controller (INTC) is responsible for performing the following functions:

- detecting the existence of (and reporting the cause of) an interrupt,
- prioritizing interrupts when more than one interrupt occurs simultaneously,
- indicating to the CPU the priority and cause of interrupts.

Software may use the information received from the interrupt controller to associate particular service routines with specific causes and to control when these routines may be called. The INTC module is only responsible for dealing with standard interrupts referred to as EXTINT in the CPU documentation and NMI. The Debug interrupt (that is, DEBUGINT) is described in *Section 6.2.5 on page 114*.

6.1 Features

- 16 levels of interrupt priority can be set for each normal interrupt source[1]. In addition there is a priority level for the NMI and DEBUGINT interrupts.
- INTC can receive up to 64 interrupt request signals.
- Priority level of each interrupt request is programmable.
- All 64 interrupt requests are maskable individually.
- Interrupt event code is provided to CPU to identify cause.
- The interrupt controller can be cascaded; it enables easy connection to external logic for off-chip interrupt control.

1. This is the number of priorities for enabled EXTINT interrupts

6.1.1 Block diagram

Figure 12 shows a block diagram of the INTC.

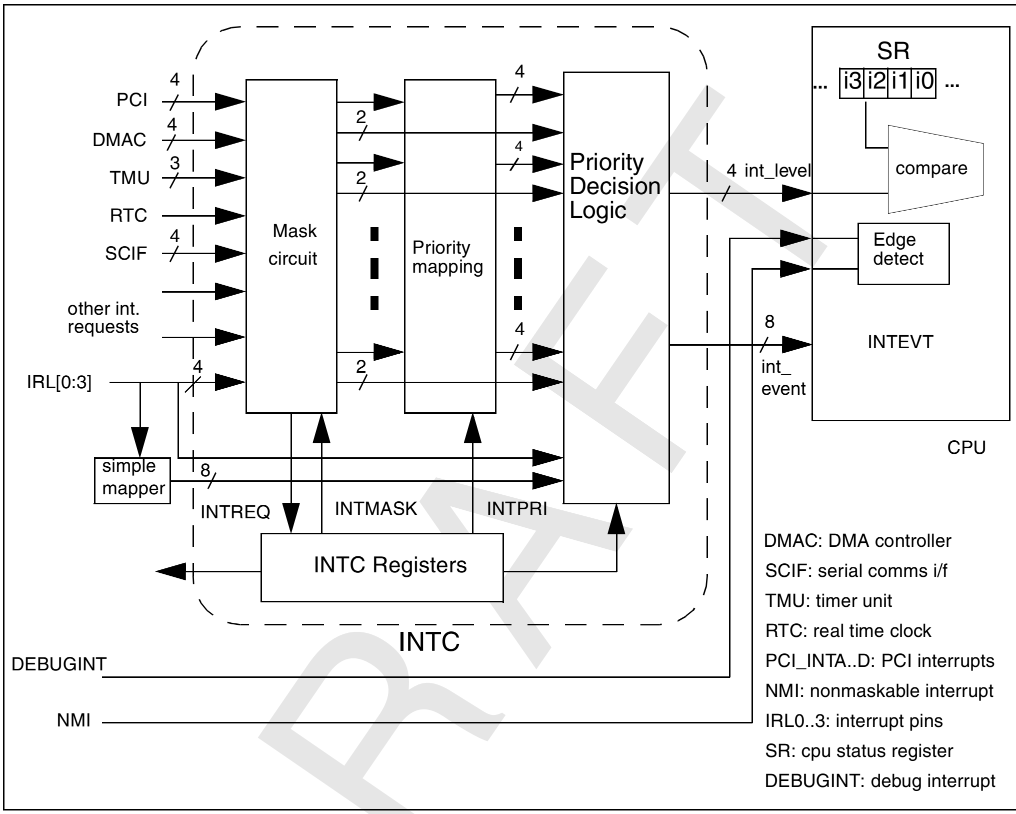


Figure 12: INTC block diagram

6.1.2 Pin configuration

Table 37 shows the INTC related pin configuration. Nonmaskable interrupt directly goes to CPU.

Name	Abbreviation	I/O	Description
Nonmaskable interrupt input pin	NMI	I	Input of non-maskable interrupt request signal
Interrupt input pins	IRL3 to IRL0	I	Input of interrupt request signals

Table 37: Pin configuration

6.1.3 Register configuration

The INTC has seventeen 32-bit registers summarized in *Table 38*.

Name	Abbreviation	Offset
Interrupt control register set	ICR.SET	0x00
Interrupt control register clear	ICR.CLEAR	0x08
Interrupt priority registers n where n can be 0 to 7	INTPRI[n]	0x10 to 0x48
Interrupt source status register n where n can be 0 or 1	INTSRC[n]	0x50 or 0x58
Interrupt request status register n where n can be 0 or 1	INTREQ[n]	0x60 or 0x68
Interrupt enable registers n where n can be 0 or 1	INTENB[n]	0x70 or 0x78
Interrupt disable register n where n can be 0 or 1	INTDSB[n]	0x80 or 0x88

Table 38: Register summary

The registers are offset from base address INTCBASE whose value is given in *Section 5.2.2: Address map on page 95*.

6.2 Interrupt sources

Interrupts are an asynchronous class of event handled by the CPU.

The CPU categorizes interrupts as shown in *Figure 39*. For concurrently asserted interrupts, the CPU orders the launching of interrupts according to these categories, and then, for EXTINT they are ordered by priority. The location of the event handling routine is determined by adding the offset associated with the event to the vector base address.

Event handle	Event name	Ordering	Vector register	Offset	EXPEVT/INTEVT
NMI	Non-maskable interrupt	↓	VBR	0x600	0x1C0 (INTEVT)
DEBUGINT	Debug interrupt		DBRVEC/ RESVEC[a]	0x200	Various (INTEVT)
EXTINT	External interrupt (IRL on-chip modules)		VBR	0x600	Various (INTEVT)

Table 39: CPU interrupt categories

a. Depending on debug mode. Refer to the event handling chapter in the CPU architecture document.

The interrupt event code is provided by the source of the interrupt signal for debug and external interrupts.

The INTC module only deals with EXTINT interrupts. NMI and DEBUGINT interrupts are described here for completeness.

The types of interrupt sources are:

- NMI,
- IRL,
- On-chip peripheral modules,
- Debug.

Each interrupt has a priority level (16 to 0) with 16 being the highest priority and 1 the lowest. A priority level of 0 means that the CPU will never be interrupted by this interrupt.

EXTINT interrupts source are level sensitive and are detected when source signals from on or off chip are asserted. The interrupts should continue to assert the interrupt line until the interrupt has been dealt with by software, otherwise there is no guarantee that the interrupt won't be lost.

6.2.1 NMI interrupts

A nonmaskable interrupt (NMI) is caused by detection of the rising edge of the signal on the NMI pin. An NMI can neither be masked (by SR.IMASK) nor blocked (by SR.BL) in the CPU. For details of how the NMI is handled by the CPU see the CPU architecture specification.

INTC does not generate INTEVT for NMI.

NMI noise cancellation

There is a noise cancellation function on the NMI pin to prevent spurious NMIs.

In normal operation, the NMI signal is sampled only on the rising edge of the peripheral bus clock. In order for an NMI to be detected the NMI signal should be low for at least three peripheral bus clock cycles (that is, rising edges) before the NMI is asserted then the NMI needs to be asserted for at least two peripheral bus clock cycles (that is, rising edges).

In standby mode the RTC (real time clock at 32.768 kHz) is used for sampling the NMI. So that in order for an NMI to be detected in standby, the NMI signal should be low for at least three RTC cycles (that is, rising edges) before the NMI is asserted then the NMI needs to be asserted for at least two RTC cycles (that is, rising edges). When the RTC is not used, interruption by means of NMI cannot be performed in standby mode.

6.2.2 IRL interrupts

Off chip interrupts are indicated by the level on the pins IRL0, IRL1, IRL2 and IRL3. The assertion of the interrupts on these should not be de-asserted until the interrupt is serviced. These interrupts are subject to masking (by SR.IMASK) and blocking (by SR.BL) by the CPU in addition to enabling and masking by the INTC module.

Processing of these interrupts occur in one of two modes:

- level-encoded interrupts,
- independently encoded interrupts.

Level encoded interrupts enable off-chip hardware to explicitly control the priority of an interrupt. This also allows interrupts to be cascaded through an off-chip interrupt controller.

Independently encoded interrupts treats each interrupt line separately.

Level encoded interrupts

Level encoded IRL interrupts are input by as the “interrupt” level at pins NOT_IRL3 to NOT_IRL0. The priority level are shown in *Table 40*.

The priority level of the IRL interrupt, once asserted, must not be lowered until the interrupt handling starts and the software can remove the reason for the interrupt. However, the priority level can be changed to a higher one.

For the eval chip implementation, there is a simple code mapper to generate proper INTEVT code shown as *Table 40 on page 112*. When interrupts are configured as level encoded. Then the source and request status of external interrupts is represented solely as interrupt number 0 in the source, request, enable and disable registers.

Interrupt level (NOT_IRL3-NOT_IRL0)	Priority (fixed)	INTEVT code
NOT_IRL3–NOT_IRL0 = (1110)	1	0x3C0
NOT_IRL3–NOT_IRL0 = (1101)	2	0x3A0
NOT_IRL3–NOT_IRL0 = (1100)	3	0x380
NOT_IRL3–NOT_IRL0 = (1011)	4	0x360
NOT_IRL3–NOT_IRL0 = (1010)	5	0x340
NOT_IRL3–NOT_IRL0 = (1001)	6	0x320
NOT_IRL3–NOT_IRL0 = (1000)	7	0x300
NOT_IRL3–NOT_IRL0 = (0111)	8	0x2E0
NOT_IRL3–NOT_IRL0 = (0110)	9	0x2C0

Table 40: Interrupt level and INTEVT code (IRLM=0)

Interrupt level (NOT_IRL3-NOT_IRL0)	Priority (fixed)	INTEVT code
NOT_IRL3–NOT_IRL0 = (0101)	10	0x2A0
NOT_IRL3–NOT_IRL0 = (0100)	11	0x280
NOT_IRL3–NOT_IRL0 = (0011)	12	0x260
NOT_IRL3–NOT_IRL0 = (0010)	13	0x240
NOT_IRL3–NOT_IRL0 = (0001)	14	0x220
NOT_IRL3–NOT_IRL0 = (0000)	15	0x200

Table 40: Interrupt level and INTEVT code (IRLM=0)

Independently encoded interrupts

Setting the IRLM bit to 1 in the ICR register, enables pins IRL0 to IRL3 to be used for four independent interrupt requests.

Interrupt pin	Priority	INTEVT code
IRL0	Programmable	0x240
IRL1	Programmable	0x2A0
IRL2	Programmable	0x300
IRL3	Programmable	0x360

Table 41: Interrupt pin INTEVT codes (IRLM=1)

To prevent erroneous acceptance of an interrupt from an IRL source which should have already been serviced, the appropriate field(s) in the INTSRC0 may be observed to guarantee that the interrupt is no longer being asserted by the INTC.

IRL noise cancellation

The INTC block includes a noise cancellation function to prevent spurious IRL interrupts being detected. An IRL interrupt is not detected unless the levels sampled on the rising edge of every PBC (peripheral bus clock) cycle remain unchanged for two consecutive cycles so that no transient level change on the IRL pins is detected. In standby mode, as the PBC is stopped, noise cancellation is performed using the 32.768 kHz clock for the RTC instead. When the RTC is not used, interruption by means of IRL interrupts cannot be performed in standby mode.

6.2.3 On-chip peripheral module interrupts

On-chip peripheral module interrupts are generated by the following modules:

- DMA controller (DMAC),
- Timer unit (TMU),
- Real-time clock (RTC),
- Serial communication interface (SCIF),
- Watchdog timer (WDT),
- PCI bus controller.

These interrupts are subject to masking (by SR.IMASK) and blocking (by SR.BL) inside the CPU in addition to enabling and masking by the INTC module.

On chip peripheral module interrupts are level sensitive and, once asserted, a module will continue to assert an interrupt until the cause has been removed and the relevant flag cleared. To prevent acceptance of an erroneous interrupt from an interrupt source which should have been serviced, first read the on-chip peripheral registers containing the relevant interrupt flag as 0 (to confirm that it is de-asserted) then read the INTSRC[1:0] register as 0 before re-enabling the interrupt and clearing the SR.BL bit to 0. This will normally[1] secure the necessary timing internally.

6.2.4 Reserved interrupts

INTC has reserved interrupt input for future extension. Therefore the input must always be 0.

6.2.5 DEBUG interrupt

There is a single source of debug interrupts (DEBUGINT). Debug interrupts are blocked when the SR.BL bit in the CPU is set, but cannot be masked.

DEBUGINT has a priority level of 16. NMI has a higher priority than DEBUGINT, but DEBUGINT has a higher priority than all other interrupts. This priority level is also greater than any CPU priority level, and a debug interrupt is accepted regardless of the value of SR.IMASK.

1. There may be an additional delay required before unblocking the interrupt depending on the clock mode employed. Please consult the datasheet for details.

There is no event code associated with a debug interrupt, and the value of INTEVT is not changed during the launch sequence for a debug interrupt. The base register and offset used for calculating the handler address for debug interrupts are not used by other events. This allows debug interrupts to be distinguished from other all events without relying on an event code.

The use of the DEBUGINT is further describe in the debug chapter of the SH-5 system architecture manual.

6.3 Interrupt exception handling and priority

There are three attributes which apply to all interrupts.

Priority - The priority is used to determine which of multiple concurrent interrupts is forwarded to the CPU. The priority is programmable for EXTINT interrupts and non-programmable for DEBUGINT and NMI.

Interrupt number - The interrupt number is used to sequence concurrent interrupts having the same priority. The interrupt having the lower interrupt number taking precedence. The interrupt number is also used to uniquely identify the interrupt within the INTC module.

INTEVT - The CPU's INTEVT register holds interrupt code which the INTC supplies to the CPU when an interrupt is launched. The INTEVT is fixed by hardware and is non-programmable. The INTC supplies an 8-bit code which is left shifted 5 bits before being visible to software in the INTEVT register. For the INTEVT values the low order 5 bits are always zero.

Interrupt handling software can unambiguously determine the source of the interrupt from the INTEVT.

Table 41 lists the codes for the interrupt event register (INTEVT), and the order of interrupt priority. Each interrupt source is assigned a unique code. The start address of the interrupt handler may be common to each interrupt source. The value of INTEVT can be used as a branch offset for the interrupt service routine.

When the priorities for multiples interrupt sources are set to the same level and such interrupts are generated simultaneously, they are handled according to the Interrupt numbers in *Table 40: Interrupt level and INTEVT code (IRLM=0) on page 112* with those having lower interrupt number in the table taking precedence over those having higher interrupt number.

Interrupt causes (in order of default precedence)[a]		Interrupt number	INTEVT code	Interrupt priority	
				Reset value	
IRL0		0	0x240	0	Programmable
IRL1		1	0x2A0	0	
IRL2		2	0x300	0	
IRL3		3	0x360	0	
PCI	INTA	4	0x800	0	
	INTB	5	0x820	0	
	INTC	6	0x840	0	
	INTD	7	0x860	0	
Reserved		8 to 11	0x020 to 0x080	0	
PCI	SERR#	12	0xA00	0	
	ERR	13	0xA20	0	
	Pwr3	14	0xA40	0	
	Pwr2	15	0xA60	0	
	Pwr1	16	0xA80	0	
	Pwr0	17	0xAA0	0	
DMAC	DMTE0	18	0x640	0	
	DMTE1	19	0x660		
	DMTE2	20	0x680		
	DMTE3	21	0x6A0		
	DAERR	22	0x6C0	0	
Reserved		23 to 31	0x0A0 to 0x1A0	0	

Table 42: Interrupt causes and priorities

Interrupt causes (in order of default precedence)[a]		Interrupt number	INTEVT code	Interrupt priority	
				Reset value	
TMU	TUNI0	32	0x400	0	Programmable
	TUNI1	33	0x420	0	
	TUNI2	34	0x440	0	
	TICPI2	35	0x460		
RTC	ATI	36	0x480	0	
	PRI	37	0x4A0		
	CUI	38	0x4C0		
SCIF	ERI	39	0x700	0	
	RXI	40	0x720		
	BRI	41	0x740		
	TXI	42	0x760		
Reserved		43	0x1C0	0	
		44	0x1E0		
		45 to 62	0xC00 to 0xE20		
WDT	ITI	63	0x560	0	

Table 42: Interrupt causes and priorities

a. TUNI0 to TUNI2: Underflow interrupts, see TMU section.
TICPI2: Input capture interrupt, see TMU section
ATI: Alarm interrupt, see RTC section
PRI: Periodic interrupt, see RTC section
CUI: Carry-up interrupt, see RTC section
ERI: Receive error interrupt, see SCIF section
RXI: Receive-data-full interrupt, see SCIF section
TXI: Transmit-data-empty interrupt, see SCIF section.
DMTE0 to DMTE3: DMAC transfer end interrupts
DAERR: DMAC address error interrupt
ITI: Interval timer interrupt

6.3.1 Interrupt masking

An interrupt is masked if it is prevented from being passed to the CPU when the interrupt source becomes active. Each interrupt source has a corresponding bit in the mask. If a bit in the interrupt mask is set then the corresponding interrupt is disabled. Otherwise if the bit is clear then the interrupt is enabled. See *Figure 13*. The interrupt controller uses a separate register location for setting the mask bits (the DISABLE register) and for clearing the mask bits (the ENABLE register).

This mechanism allows mask bits to be set or cleared independently and atomically without visibility of the other bits in the mask.

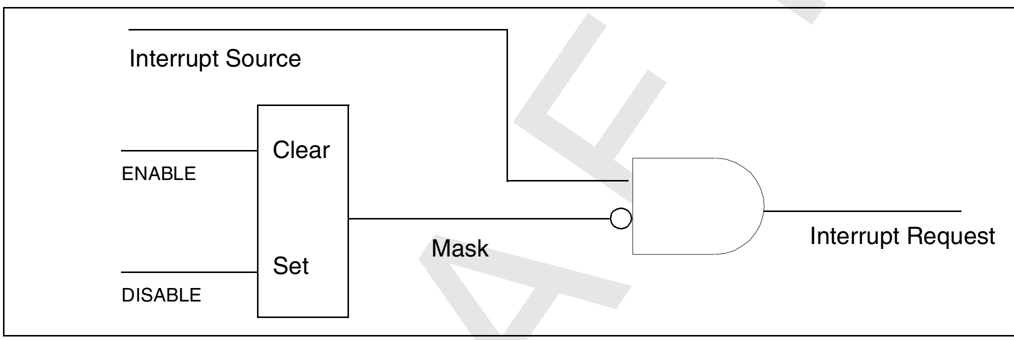


Figure 13: Interrupt masking

Level-encoded external interrupts all use the mask bit which corresponds to IRL0. So that when (ICR.IRLM=0) writing a 1 to the ENABLE register in the position corresponding to IRL0 will enable all level encoded interrupts and writing a 1 to the DISABLE register will disable all level encoded interrupts.

6.4 Register descriptions

The following registers govern the behaviour of the interrupt controller module.

Interrupt control register set/clear (ICR.SET and ICR.CLEAR)

These registers are used to control the operating mode of the IRL pins; whether they are treated as four independent interrupts or level encoded to allow external devices to set the interrupt priority. These registers form a pair which can be used to independently set or clear the ICR register. Writing to the ICR.SET register is used to set individual bits; writing to the ICR.CLEAR register is used to clear individual bits of the register. Reading either register returns the same value which is the value of the ICR register. The format of these registers are shown in *Table 43* and *Table 44*.

Interrupt control register set (ICR.SET)				INTCBASE + 0x00	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IRLM	0	1	No	IRL pin mode	RW
	Operation		0: IRL pins are used for level-encoded interrupt requests 1: IRL pins are used as four independent interrupt requests		
	When read		Returns current value		
	When written		Write 1 makes bit set. Write 0 is ignored.		
	HARD reset		0		
—	[15:1]	15	—	ICR15 to ICR1	RW
	Operation		INTC provides those bits as output signals. External glue logic can use those bits to impliment additional feature in future chip integration.		
	When read		Returns current value		
	When written		Write 1 makes bit set. Write 0 is ignored.		
	HARD reset		0		

Table 43: Interrupt control register set

Interrupt control register set (ICR.SET)				INTCBASE + 0x00	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[31:16]	16	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 43: Interrupt control register set

Interrupt control register clear (ICR.CLEAR)				INTCBASE + 0x08	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IRLM	0	1	No	IRL pin mode	RW
	Operation		0: IRL pins are used for level-encoded interrupt requests 1: IRL pins are used as four independent interrupt requests		
	When read		Returns current value		
	When written		Write 1 clears the bit to 0. Write 0 is ignored.		
	HARD reset		0		
—	[15:1]	15	—	ICR15 to ICR1	RW
	Operation		INTC provides those bits as output signals. External glue logic can use those bits to implement additional feature in future chip integration.		
	When read		Returns current value		
	When written		Write 1 clears the bit to 0. Write 0 is ignored.		
	HARD reset		0		

Table 44: Interrupt control register clear

Interrupt control register clear (ICR.CLEAR)				INTCBASE + 0x08	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
—	[31:16]	16	—	RESERVED	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 44: Interrupt control register clear

Priority registers 0 to 7 (INTPRI n)

These eight registers control the priority associated with each interrupt source. These are 32-bit registers with 4 bits per priority. Register 0 is used to specify the priorities of interrupt numbers 0 through 7, register 1 for interrupt numbers 8 through 15 and so on. The format of these registers is shown in *Table 45*.

Interrupt priority register n (INTPRI n) n={0..7}				INTCBASE + 0x10 +8*n	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 8n	[3:0]	4	—	Priority of interrupt 8n	RW
	Operation		Contains the priority of the interrupt		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
INTERRUPT 8n + 1	[7:4]	4	—	Priority of interrupt 8n + 1	RW
	As for interrupt 8n				
INTERRUPT 8n + 2	[11:8]	4	—	Priority of interrupt 8n + 2	RW
	As for interrupt 8n				

Table 45: Interrupt priority register n (INTPRIn) [a]

Interrupt priority register n (INTPRI n) n={0..7}				INTCBASE + 0x10 +8*n	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 8n + 3	[15:12]	4	—	Priority of interrupt 8n + 3	RW
	As for interrupt 8n				
INTERRUPT 8n + 4	[19:16]	4	—	Priority of interrupt 8n + 4	RW
	As for interrupt 8n				
INTERRUPT 8n + 5	[23:20]	4	—	Priority of interrupt 8n + 5	RW
	As for interrupt 8n				
INTERRUPT 8n + 6	[27:24]	4	—	Priority of interrupt 8n + 6	RW
	As for interrupt 8n				
INTERRUPT 8n + 7	[31:28]	4	—	Priority of interrupt 8n + 7	RW
	As for interrupt 8n				

Table 45: Interrupt priority register n (INTPRIn) [a]

a. When ICR.IRLM=0 (that is, IRL pins are level encoded) interrupt numbers 0 to 3 cannot be assigned priorities using the INTPRI0 register. In this case, values written to Interrupt 0, Interrupt 1, Interrupt 2 and Interrupt 3 are ignored.

Interrupt enable registers 0 and 1 (INTENB0 and INTENB1)

This pair of 32-bit registers are used to examine and clear bits in the 64-bit interrupt mask. When either register is read it will return the value of the mask. For each bit where the mask is clear the corresponding interrupt is enabled. When written, each data bit which is ‘1’ will clear the corresponding bit in the mask; each data bits which is ‘0’ is ignored.The format of these registers is shown in *Table 46* and *Table 47*.

Interrupt enable register 0 (INTENB0)				INTCBASE + 0x70	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 0	0	1	—	Interrupt 0 enable	RW
	Operation[a]		Enables interrupt 0 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 0 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 1	1	1	—	Interrupt 1 enable	RW
	Operation[b]		Enables interrupt 1 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 1 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 2	2	1	—	Interrupt 2 enable	RW
	Operation[b]		Enables interrupt 2 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 2 of mask 0: Ignored		
	HARD reset		0		
...					

Table 46: Interrupt enable register 0 (INTENB0)

Interrupt enable register 0 (INTENB0)				INTCBASE + 0x70	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 31	31	1	—	Interrupt 31 enable	RW
	Operation		Enables interrupt 31 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 31 of mask 0: Ignored.		
	HARD reset		0		

Table 46: Interrupt enable register 0 (INTENB0)

a. When ICR.IRLM=0 this bit is used to enable all level encoded interrupts.

b. When ICR.IRLM=0 this bit is ignored.

Interrupt enable register 1 (INTENB1)				INTCBASE + 0x78	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 32	0	1	—	Interrupt 32 enable	RW
	Operation		Enables interrupt 32 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 32 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 33	1	1	—	Interrupt 33 enable	RW
	Operation		Enables interrupt 33 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 33 of mask 0 ignored		
	HARD reset		0		

Table 47: Interrupt enable register 1 (INTENB1)

Interrupt enable register 1 (INTENB1)				INTCBASE + 0x78	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 34	2	1	—	Interrupt 34 enable	RW
	Operation		Enables interrupt 34 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 34 of mask 0: Ignored		
	HARD reset		0		
...					
INTERRUPT 63	31	1	—	Interrupt 63 enable	RW
	Operation		Enables interrupt 63 to be passed to CPU		
	When read		Returns current value		
	When written		1: Clears bit 63 of mask 0: Ignored		
	HARD reset		0		

Table 47: Interrupt enable register 1 (INTENB1)

Interrupt disable register 0 and 1 (INTDSB0 and INTDSB1)

This pair of 32-bit registers are used to examine and set bits in the 64-bit interrupt mask. For each bit where the mask is set the corresponding interrupt is disabled. When either register is read, it will return the value of the mask, When written, each data bit which is '1' will set the corresponding bit in the mask; each data bits which is '0' is ignored. The format of these registers is shown in *Table 48* and *Table 49*.

Interrupt disable register 0 (INTDSB0)				INTCBASE + 0x80	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 0	0	1	—	Interrupt 0 disable	RW
	Operation[a]		Disables interrupt 0 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 0 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 1	1	1	—	Interrupt 1 disable	RW
	Operation[b]		Disables interrupt 1 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 1 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 2	2	1	—	Interrupt 2 disable	RW
	Operation[b]		Disables interrupt 2 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 2 of mask 0: Ignored		
	HARD reset		0		
...					

Table 48: Interrupt disable register 0 (INTDSB0)

Interrupt disable register 0 (INTDSB0)				INTCBASE + 0x80	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 31	31	1	—	Interrupt 31 disable	RW
	Operation		Disables interrupt 31 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 31 of mask 0: Ignored		
	HARD reset		0		

Table 48: Interrupt disable register 0 (INTDSB0)

a. When ICR.IRLM=0 this bit is used to disable all level encoded interrupts.

b. When ICR.IRLM=0 this bit is ignored.

Interrupt disable register 1 (INTDSB1)				INTCBASE + 0x88	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 32	0	1	—	Interrupt 32 disable	RW
	Operation		Disables interrupt 32 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 32 of mask 0: Ignored		
	HARD reset		0		
INTERRUPT 33	1	1	—	Interrupt 33 disable	RW
	Operation		Disables interrupt 33 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 33 of mask 0 ignored		
	HARD reset		0		

Table 49: Interrupt disable register 1 (INTDSB1)

Interrupt disable register 1 (INTDSB1)				INTCBASE + 0x88	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 34	2	1	—	Interrupt 34 disable	RW
	Operation		Disables interrupt 34 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 34 of mask 0: Ignored		
	HARD reset		0		
...					
INTERRUPT 63	31	1	—	Interrupt 63 disable	RW
	Operation		Disables interrupt 63 to be passed to CPU		
	When read		Returns current value		
	When written		1: Sets bit 63 of mask 0: Ignored		
	HARD reset		0		

Table 49: Interrupt disable register 1 (INTDSB1)

Interrupt source status register 0 and 1 (INTSRC0 and INTSRC1)

This pair of registers provide the status of the interrupt sources to the interrupt controller. A ‘1’ bit indicates that the corresponding interrupt request is active prior to masking. The format of these registers is shown in *Table 50* and *Table 51*.

Interrupt source status register 0 (INTSRC0)				INTCBASE + 0x50	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 0	0	1	—	Interrupt 0 source status	RO
	Operation[a]		Indicates status of interrupt prior to masking 0: Inactive 1: Active		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 1	1	1	—	Interrupt 1 source status	RO
	Operation[b]		Indicates status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 2	2	1	—	Interrupt 2 source status	RO
	Operation[b]		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
...					

Table 50: Interrupt source status register 0 (INTSRC0)

Interrupt source status register 0 (INTSRC0)				INTCBASE + 0x50	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 31	31	1	—	Interrupt 31 source status	RO
	Operation		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 50: Interrupt source status register 0 (INTSRC0)

a. When ICR.IRLM=0 this bit indicates the or-ed status of all the IRLs; Interrupts 0, 1, 2 and 3.

b. When ICR.IRLM=0 this bit is undefined.

Interrupt source status register 1 (INTSRC1)				INTCBASE + 0x58	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 32	0	1	—	Interrupt 32 source status	RO
	Operation		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 33	1	1	—	Interrupt 33 source status	RO
	Operation		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 51: Interrupt source status register 1 (INTSRC1)

Interrupt source status register 1 (INTSRC1)				INTCBASE + 0x58	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 34	2	1	—	Interrupt 34 source status	RO
	Operation		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
...					
INTERRUPT 63	31	1	—	Interrupt 63 source status	RO
	Operation		Holds active status of interrupt prior to masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 51: Interrupt source status register 1 (INTSRC1)

Interrupt request status register 0 and 1

This pair of registers provide the status of the interrupt sources after masking. A ‘1’ bit indicates that the corresponding interrupt request is active and will generate an interrupt to the CPU.The format of these registers is shown in *Table 52* and *Table 53*.

Interrupt request status register 0 (INTREQ0)				INTCBASE + 0x60	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 0	0	1	—	Interrupt 0 request status	RO
	Operation[a]		Indicates status of interrupt after masking 0: Inactive 1: Active		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 1	1	1	—	Interrupt 1 request status	RO
	Operation[b]		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 2	2	1	—	Interrupt 2 request status	RO
	Operation[b]		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
...					

Table 52: Interrupt request status register 0 (INTREQ0)

Interrupt request status register 0 (INTREQ0)				INTCBASE + 0x60	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 31	31	1	—	Interrupt 31 request status	RO
	Operation		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 52: Interrupt request status register 0 (INTREQ0)

a. When ICR.IRLM=0 this bit indicates the or-ed status of all the IRLs; Interrupts 0, 1, 2 and 3.

b. When ICR.IRLM=0 this bit is undefined.

Interrupt request status register 1 (INTREQ1)				INTCBASE + 0x68	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 32	0	1	—	Interrupt 32 request status	RO
	Operation		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
INTERRUPT 33	1	1	—	Interrupt 33 request status	RO
	Operation		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 53: Interrupt request status register 1 (INTREQ1)

Interrupt request status register 1 (INTREQ1)				INTCBASE + 0x68	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
INTERRUPT 34	2	1	—	Interrupt 34 request status	RO
	Operation		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		
...					
INTERRUPT 63	31	1	—	Interrupt 63 request status	RO
	Operation		Holds active status of interrupt request after masking		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 53: Interrupt request status register 1 (INTREQ1)

6.4.1 INTC operation

The sequence of interrupt operations is explained below.

1 The interrupt request sources send interrupt request signals to the interrupt controller (INTC).

2 The interrupt controller masks interrupt request according to value of the mask registers (INTENB0, INTENB1, INTDSB0 and INTDSB1) then selects the unmasked interrupt having the highest priority. The priority of each interrupt is determined by the contents of the INTPRI registers. Lower priority interrupts are held pending. If two of these interrupts have the same priority level the one having the lower interrupt number (as shown in *Table 42: Interrupt causes and priorities on page 116*) is selected.

3 Further processing is performed by the CPU. The priority level of the interrupt selected by the interrupt controller is compared with the interrupt mask bits (I3 to I0) in the status register (SR) of the CPU. If the request priority level is higher[1] than the IMASK (that is, bits 7 to 4 of the SR) and the SR.BL bit is not set then, the CPU accepts the interrupt. If the interrupt priority is lower than the IMASK or the SR.BL bit is set then the interrupt is not accepted.

4 When an interrupt is accepted the CPU allows its execution pipeline to drain then launches an interrupt handler. The INTEVT value accompanying the interrupt (see *Table 40: Interrupt level and INTEVT code (IRLM=0) on page 112*) is set in the CPU's INTEVT register.

5 The interrupt source flag should be cleared in the exception handling routine. To ensure that an interrupt request that should have been cleared is not inadvertently accepted again, read the interrupt source flag after it has been cleared, then read the appropriate INTSRC[n] bit as '0' before clearing the SR.BL bit or executing an **rte** instruction[2].

For further details please refer to *Core Architecture Volume 1, Chapter 16 Event Handling*.

6.4.2 Transactions

Correct access to the INTC registers is by long word (32-bit) loads and stores to the documented addresses only. For the handling of other accesses types and of bad addresses see *Chapter 5: Peripheral bridge on page 93*.

1. That is, interrupt-priority < SR[7:4] see CPU for a full description of this.
2. An implementation dependent wait time may additionally be required, see the datasheet for details.

7 Real-time clock (RTC)

7.1 Overview

The system includes an on-chip real-time clock (RTC) and a 32.768 kHz crystal oscillator for use by the RTC.

7.1.1 Features

The RTC has the following features.

- Clock and calendar functions (BCD display). It counts seconds, minutes, hours, day-of-week, days, months and years.
- Timer (binary display). The current count state within the RTC frequency divider is indicated in register RTC.R64CNT.
- Start/stop function.
- 30-second adjustment function.
- Alarm interrupts. Comparison with second, minute, hour, day-of-week, day, or month can be selected as the alarm interrupt condition.
- Periodic interrupts. An interrupt period of 1/256 second, 1/64 second, 1/16 second, 1/4 second, 1/2 second, 1 second, or 2 seconds can be selected.
- Carry interrupt indicates a second counter carry, or an increment in RTC.R64CNT when this register is read.
- Automatic leap year adjustment.

7.1.2 Block diagram

Figure 14 shows a block diagram of the RTC.

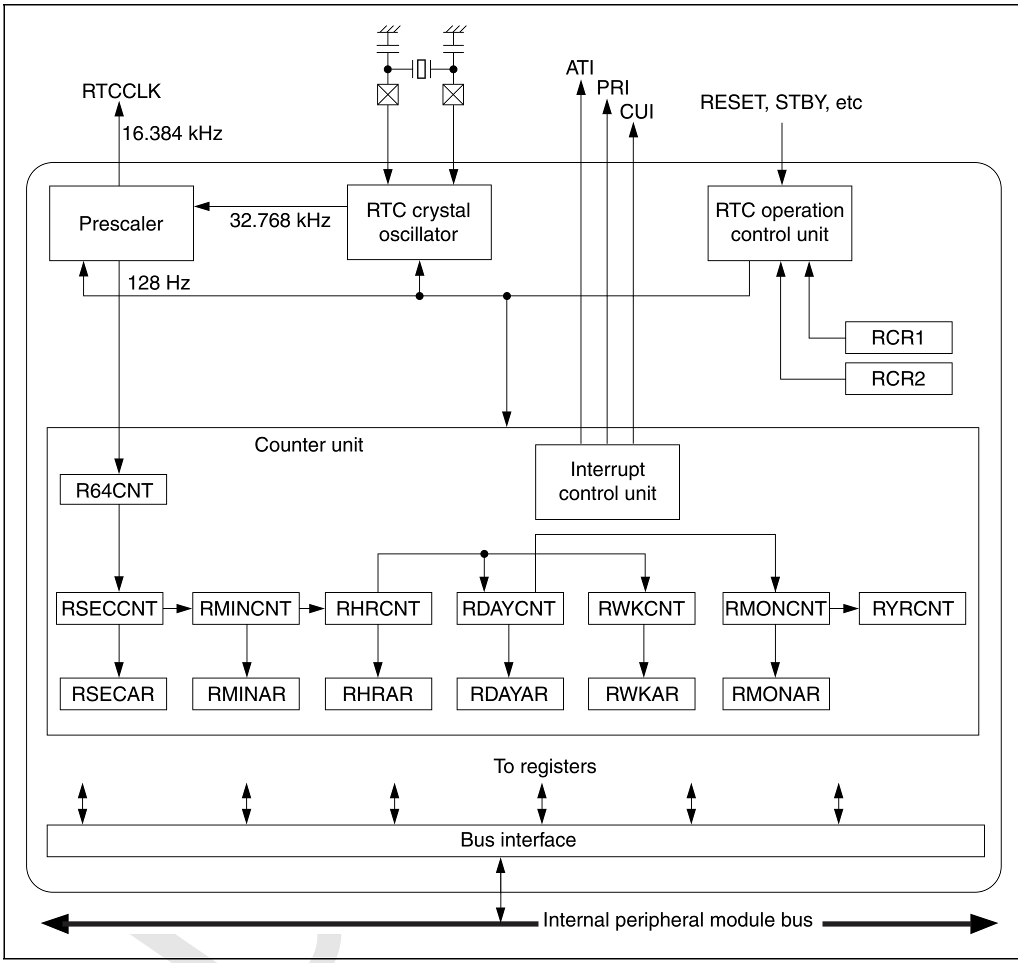


Figure 14: Block diagram of RTC

7.1.3 Pin configuration

Table 54 shows the RTC pins.

Pin name	Abbreviation	I/O	Function
RTC oscillator crystal pin	EXTAL2	Input	Connects crystal to RTC oscillator
RTC oscillator crystal pin	XTAL2	Output	Connects crystal to RTC oscillator
Clock input/clock output	TCLK	I/O	External clock input pin/input capture control input pin/RTC output pin (shared with TMU)
Dedicated RTC power supply	V_{CC} (RTC)	-	RTC oscillator power supply pin[a]
Dedicated RTC GND pin	V_{SS} (RTC)	-	RTC oscillator GND pin[a]

Table 54: RTC Pins

a. Power must always be supplied to the RTC power supply pins even when the RTC is not used. Power should be supplied to all power supply pins when in Standby mode.

7.1.4 Register configuration

Table 55 summarizes the RTC registers.

The register addresses are offset from RTCBASE. Refer to the system address map for the value of RTCBASE.

Name	Abbreviation	RW	Power-on reset	Initialization			Address Offset	Access size
				Manual reset	Standby mode	Initial value		
Freq. divider counter	RTC.R64CNT	R	Counts	Counts	Counts	Undefined	0x00	8
Second counter	RTC.RSECCNT	RW	Counts	Counts	Counts	Undefined	0x04	8
Minute counter	RTC.RMINCNT	RW	Counts	Counts	Counts	Undefined	0x08	8

Table 55: RTC registers

Name	Abbreviation	RW	Power-on reset	Initialization			Address Offset	Access size
				Manual reset	Standby mode	Initial value		
Hour counter	RTC.RHRCNT	RW	Counts	Counts	Counts	Undefined	0x0C	8
Day-of-week counter	RTC.RWKCNT	RW	Counts	Counts	Counts	Undefined	0x10	8
Day counter	RTC.RDAYCNT	RW	Counts	Counts	Counts	Undefined	0x14	8
Month counter	RTC.RMONCNT	RW	Counts	Counts	Counts	Undefined	0x18	8
Year counter	RTC.RYRCNT	RW	Counts	Counts	Counts	Undefined	0x1C	16
Second alarm register	RTC.RSECAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x20	8
Minute alarm register	RTC.RMINAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x24	8
Hour alarm register	RTC.RHRAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x28	8
Day-of-week alarm register	RTC.RWKAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x2C	8
Day alarm register	RTC.RDAYAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x30	8
Month alarm register	RTC.RMONAR	RW	Initialized[a]	Held	Held	Undefined[a]	0x34	8

Table 55: RTC registers

Name	Abbreviation	RW	Power-on reset	Initialization			Address Offset	Access size
				Manual reset	Standby mode	Initial value		
RTC control register 1	RTC.RCR1	RW	Initialized	Initialized	Held	0x00[b]	0x38	8
RTC control register 2	RTC.RCR2	RW	Initialized	Initialized[c]	Held	0x09[d]	0x3C	8

Table 55: RTC registers

a. The ENB bit in each register is initialized.

b. The value of the CF bit and AF bit is undefined.

c. Bits other than the RTCEN bit and START bit are initialized.

d. The value of the PEF bit is undefined.

7.2 Register descriptions

7.2.1 Frequency divider counter (RTC.R64CNT)

RTC.R64CNT is an 8-bit read-only register that indicates the current count state within the RTC frequency divider. RTC.R64CNT is driven by a 128 Hz signal. It generates a carry (once per second) to increment the second counter RTC.RSECCNT.

If this register is read when a carry is generated from the 128 Hz frequency division stage, bit 7 (CF) in RTC control register 1 (RTC.RCR1) is set to 1, indicating the simultaneous occurrence of the carry and the RTC.R64CNT read. In this case, the read value is not valid, and so RTC.R64CNT must be read again after first writing 0 to the CF bit in RTC.RCR1 to clear it.

When the RESET bit or ADJ bit in RTC control register 2 (RTC.RCR2) is set to 1, the RTC frequency divider is initialized and RTC.R64CNT is initialized to 0x00.

RTC.R64CNT is not initialized by a power-on or manual reset, or in standby mode.

Bit 7 is always read as 0 and cannot be modified.

RTC.R64CNT				0x0	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
R64CNT	[6:0]	7	✓	Frequency divider counter	RO
	Operation		Holds a counter value which increments at 128 Hz. Generates a carry for the second counter once per second.		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Undefined		
Reserved	7	1	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 56: RTC.R64CNT

7.2.2 Second counter (RTC.RSECCNT)

RTC.RSECCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded second value in the RTC. It counts on the carry generated once per second by the RTC.R64CNT counter.

The setting range is decimal 00 to 59. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RSECCNT is not initialized by a power-on or manual reset, or in standby mode.

Bit 7 is always read as 0. A write to this bit is invalid, but the write value should always be 0.

RTC.RSECCNT				0x4	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	✓	Second Counter	RW
	Operation		Counts up 1 second units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[6:4]	3	✓	Second Counter	RW
	Operation		Counts up 10 seconds		
	When read		Reads current value		
	When written		Only values		
	HARD reset		Undefined		
RESERVED	7	1	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 57: RTC.RSECCNT

7.2.3 Minute counter (RTC.RMINCNT)

RTC.RMINCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded minute value in the RTC. It counts on the carry generated once per minute by the second counter.

The setting range is decimal 00 to 59. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RMINCNT is not initialized by a power-on or manual reset, or in standby mode.

Bit 7 is always read as 0. A write to this bit is invalid, but the write value should always be 0.

RTC.RMINCNT				0x8	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	✓	Minute counter	RW
	Operation		Counts up 1 minute units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[6:4]	3	✓	10 minute counter	RW
	Operation		Counts up 10 minutes		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
RESERVED	7	1	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 58: RTC.RMINCNT

7.2.4 Hour counter (RTC.RHRCNT)

RTC.RHRCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded hour value in the RTC. It counts on the carry generated once per hour by the minute counter.

The setting range is decimal 00 to 23. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RHRCNT is not initialized by a power-on or manual reset, or in standby mode.

Bits 7 and 6 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

<table>
<tr><th colspan="4">RTC.RHRCNT</th><th colspan="2">0x0C</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">UNITS</td><td>[3:0]</td><td>4</td><td>✓</td><td>Hour counter</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Counts up 1 hour units</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Reads current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Undefined</td></tr>
<tr><td rowspan="5">TENS</td><td>[5:4]</td><td>2</td><td>✓</td><td>Hour Counter</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Counts up 10 hours</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Reads current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current values</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Undefined</td></tr>
<tr><td rowspan="5">RESERVED</td><td>[7:6]</td><td>2</td><td>-</td><td>Reserved</td><td>RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Reserved</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">0</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Ignored</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 59: RTC.RHRCNT

7.2.5 Day-of-week counter (RTC.RWKCNT)

RTC.RWKCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded day-of- week value in the RTC. It counts on the carry generated once per day by the hour counter.

The setting range is decimal 0 to 6. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RWKCNT is not initialized by a power-on or manual reset, or in standby mode.

Bits 7 to 3 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

RTC.RWKCNT				0x10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[2:0]	3	✓	Day of week	RW
	Operation		Indicates Day of week		
	When read		Returns Day-of-week code (0=Sun; 1=Mon; 2=Tue; 3=Wed; 4=Thu; 5=Fri; 6=Sat)		
	When written		Updates current value		
	HARD reset		Undefined		
RESERVED	[7:3]	5	-	Reserved	RES
	Operation				
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 60: RTC.RWKCNT

7.2.6 Day counter (RTC.RDAYCNT)

RTC.RDAYCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded day value in the RTC. It counts on the carry generated once per day by the hour counter.

The setting range is decimal 01 to 31. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag. RTC.RDAYCNT is not initialized by a power-on or manual reset, or in standby mode.

The setting range for RTC.RDAYCNT depends on the month and whether the year is a leap year, so care is required when making the setting. Bits 7 and 6 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

RTC.RDAYCNT				0x14	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	✓	Day counter	RW
	Operation		Counts up 1 day units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[5:4]	2	✓	10 day counter	RW
	Operation		Counts up 10 days		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 61: RTC.RDAYCNT

RTC.RDAYCNT				0x14	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESERVED	[7:6]	2	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 61: RTC.RDAYCNT

7.2.7 Month counter (RTC.RMONCNT)

RTC.RMONCNT is an 8-bit readable/writable register used as a counter for setting and counting the BCD-coded month value in the RTC. It counts on the carry generated once per month by the day counter.

The setting range is decimal 01 to 12. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RMONCNT is not initialized by a power-on or manual reset, or in standby mode.

Bits 7 to 5 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

RTC.RMONCNT				0x18	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	✓	Month Counter	RW
	Operation		Counts up 1 Month units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 62: RTC.RMONCNT

RTC.RMONCNT				0x18	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TENS	4	1	✓	10 Month Counter	RW
	Operation		Counts up 10 Months		
	When read		Reads current value		
	When written		only values		
	HARD reset		Undefined		
RESERVED	[7:5]	3	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 62: RTC.RMONCNT

7.2.8 Year counter (RTC.RYRCNT)

RTC.RYRCNT is a 16-bit readable/writable register used as a counter for setting and counting the BCD-coded year value in the RTC. It counts on the carry generated once per year by the month counter.

The setting range is decimal 0000 to 9999. The RTC will not operate normally if any other value is set. Write processing should be performed after stopping the count with the START bit in RTC.RCR2, or by using the carry flag.

RTC.RYRCNT is not initialized by a power-on or manual reset, or in standby mode.

RTC.RYRCNT				0x1C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	✓	Year counter	RW
	Operation		Counts up 1 year units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[7:4]	4	✓	10 year counter	RW
	Operation		Counts up 10 years		
	When read		Reads current value		
	When written		only values		
	HARD reset		Undefined		
HUNDREDS	[11:8]	4	✓	100 year counter	RW
	Operation		Counts up 100 year units		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
THOUSANDS	[15:12]	4	✓	1000 year counter	RW
	Operation		Counts up 1000 years		
	When read		Reads current value		
	When written		only values		
	HARD reset		Undefined		

Table 63: RTC.RYRCNT

7.2.9 Second alarm register (RTC.RSECAR)

RTC.RSECAR is an 8-bit readable/writable register used as an alarm register for the RTC's BCD-coded second value counter, RTC.RSECCNT. When the ENB bit is set to 1, the RTC.RSECAR value is compared with the RTC.RSECCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR, and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 00 to 59 + ENB bit. The RTC will not operate normally if any other value is set.

The ENB bit in RTC.RSECAR is initialized to 0 by a power-on reset. The other fields in RTC.RSECAR are not initialized by a power-on or manual reset, or in standby mode.

RTC.RSECAR				0x20	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	-	Second Value	RW
	Operation		1 second value for comparison		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[6:4]	3	-	10 Second Value	RW
	Operation		10 second value for comparison		
	When read		Reads current value		
	When written		only values		
	HARD reset		Undefined		

Table 64: RTC.RSECAR

RTC.RSECAR				0x20	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable second comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 64: RTC.RSECAR

7.2.10 Minute alarm register (RTC.RMINAR)

RTC.RMINAR is an 8-bit readable/writable register used as an alarm register for the RTC’s BCD-coded minute value counter, RTC.RMINCNT. When the ENB bit is set to 1, the RTC.RMINAR value is compared with the RTC.RMINCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 00 to 59 + ENB bit. The RTC will not operate normally if any other value is set.

The ENB bit in RTC.RMINAR is initialized by a power-on reset. The other fields in RTC.RMINAR are not initialized by a power-on or manual reset, or in standby mode.

RTC.RMINAR				0x24	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	-	Minute value	RW
	Operation		1 minute comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 65: RTC.RMINAR

RTC.RMINAR				0x24	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TENS	[6:4]	3	-	10 minute value	RW
	Operation		1 minute comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 65: RTC.RMINAR

7.2.11 Hour alarm register (RTC.RHRAR)

RTC.RHRAR is an 8-bit readable/writable register used as an alarm register for the RTC’s BCD-coded hour value counter, RTC.RHRCNT. When the ENB bit is set to 1, the RTC.RHRAR value is compared with the RTC.RHRCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 00 to 23 + ENB bit. The RTC will not operate normally if any other value is set.

The ENB bit in RTC.RHRAR is initialized by a power-on reset. The other fields in RTC.RHRAR are not initialized by a power-on or manual reset, or in standby mode.

Bit 6 is always read as 0. A write to this bit is invalid, but the write value should always be 0.

RTC.RHRAR				0x28	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	-	Hour value	RW
	Operation		1 Hour comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[5:4]	2	-	10 hour value	RW
	Operation		10 Hour comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
RESERVED	6	1	✓	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable hour comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 66: RTC.RHRAR

7.2.12 Day-of-week alarm register (RTC.RWKAR)

RTC.RWKAR is an 8-bit readable/writable register used as an alarm register for the RTC's BCD-coded day-of-week value counter, RTC.RWKCNT. When the ENB bit is set to 1, the RTC.RWKAR value is compared with the RTC.RWKCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 0 to 6 + ENB bit. The RTC will not operate normally if any other value is set.

The ENB bit in RTC.RWKAR is initialized by a power-on reset. The other fields in RTC.RWKAR are not initialized by a power-on or manual reset, or in standby mode.

Bits 6 to 3 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

RTC.RWKAR				0x2C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[2:0]	3	-	Week value	RW
	Operation		Day of week comparison value		
	When read		Returns Day-of-week code (0=Sun; 1=Mon; 2=Tue; 3=Wed; 4=Thu; 5=Fri; 6=Sat)		
	When written		Updates current value		
	HARD reset		Undefined		
RESERVED	[6:3]	4	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 67: RTC.RWKAR

RTC.RWKAR				0x2C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable Day of week comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 67: RTC.RWKAR

7.2.13 Day alarm register (RTC.RDAYAR)

RTC.RDAYAR is an 8-bit readable/writable register used as an alarm register for the RTC’s BCD-coded day value counter, RTC.RDAYCNT. When the ENB bit is set to 1, the RTC.RDAYAR value is compared with the RTC.RDAYCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 01 to 31 + ENB bit. The RTC will not operate normally if any other value is set. The setting range for RTC.RDAYAR depends on the month and whether the year is a leap year, so care is required when making the setting.

The ENB bit in RTC.RDAYAR is initialized by a power-on reset. The other fields in RTC.RDAYAR are not initialized by a power-on or manual reset, or in standby mode.

Bit 6 is always read as 0. A write to this bit is invalid, but the write value should always be 0.

RTC.RDAYAR				0x30	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	-	Day alarm value	RW
	Operation		1 Day comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	[5:4]	2	-	10 day value	RW
	Operation		10 Day comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
RESERVED	6	1	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable Day comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 68: RTC.RDAYAR

7.2.14 Month alarm register (RTC.RMONAR)

RTC.RMONAR is an 8-bit readable/writable register used as an alarm register for the RTC's BCD-coded month value counter, RTC.RMONCNT. When the ENB bit is set to 1, the RTC.RMONAR value is compared with the RTC.RMONCNT value. Comparison between the counter and the alarm register is performed for those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR, and RTC.RMONAR in which the ENB bit is set to 1, and the RTC.RCR1 alarm flag is set when the respective values all match.

The setting range is decimal 01 to 12 + ENB bit. The RTC will not operate normally if any other value is set.

The ENB bit in RTC.RMONAR is initialized by a power-on reset. The other fields in RTC.RMONAR are not initialized by a power-on or manual reset, or in standby mode.

Bits 6 and 5 are always read as 0. A write to these bits is invalid, but the write value should always be 0.

RTC.RMONAR				0x34	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNITS	[3:0]	4	-	Month alarm value	RW
	Operation		1 month comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		
TENS	4	1	-	10 month alarm value	RW
	Operation		10 month comparison value		
	When read		Reads current value		
	When written		Updates current value		
	HARD reset		Undefined		

Table 69: RTC.RMONAR

RTC.RMONAR				0x34	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESERVED	[6:5]	2	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		
ENB	7	1	-	Comparison enable bit	RW
	Operation		Set this bit to ‘1’ to enable month comparison for alarm		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 69: RTC.RMONAR

7.2.15 RTC control register 1 (RTC.RCR1)

RTC.RCR1 is an 8-bit readable/writable register containing a carry flag and alarm flag, plus flags to enable or disable interrupts for these flags.

The CIE and AIE bits are initialized to 0 by a power-on or manual reset; the value of bits other than CIE and AIE is undefined. In standby mode RTC.RCR1 is not initialized, and retains its current value.

RTC.RCR1				0x38	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
AF	0	1	✓	Alarm flag	RW
	Operation		Set to 1 when the alarm time set in those registers among RTC.RSECAR, RTC.RMINAR, RTC.RHRAR, RTC.RWKAR, RTC.RDAYAR and RTC.RMONAR in which the ENB bit is set to 1 matches the respective counter values		
	When read		0: Alarm registers and counter values do not match 1: When alarm registers in which the ENB bit is set to 1 and counter values match		
	When written		0: Clears the alarm condition 1: Ignored		
	HARD reset		0		
RESERVED	[2:1]	2	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 70: RTC.RCR1

RTC.RCR1				0x38	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
AIE	3	1	✓	Alarm interrupt enable flag	RW
	Operation		Enables or disables interrupt generation when the alarm flag (AF) is set to 1		
	When read		Returns current value		
	When written		0: Alarm interrupt is not generated when AF flag is set to 1 (Initial value) 1: Alarm interrupt is generated when AF flag is set to 1		
	HARD reset		0		
CIE	4	1	✓	Carry interrupt enable flag	RW
	Operation		Enables or disables interrupt generation when the carry flag (CF) is set to 1.		
	When read		Returns current value		
	When written		0: Alarm interrupt is not generated when AF flag is set to 1 (Initial value) 1: Carry interrupt is generated when CF flag is set to 1		
	HARD reset		0		
RESERVED	[6:5]	2	-	Reserved	RES
	Operation		Reserved		
	When read		Undefined		
	When written		Ignored		
	HARD reset		Undefined		

Table 70: RTC.RCR1

RTC.RCR1				0x38	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CF	7	1	✓	Carry flag	RW
	Operation		This flag is set to 1 on generation of a second counter carry, or a RTC.R64CNT counter carry when RTC.R64CNT is read. The count register value read at this time is not guaranteed, and so the count register must be read again		
	When read		0: No second counter carry, or R64CNT carry when the counter RTC.R64CNT is read 1: Second counter carry, or R64CNT carry when the counter RTC.R64CNT is read		
	When written		0: Clears the CF 1: Generation of a second counter carry, or a R64CNT carry when the counter RTC.R64CNT is read		
	HARD reset		Undefined		

Table 70: RTC.RCR1

7.2.16 RTC control register 2 (RTC.RCR2)

RTC.RCR2 is an 8-bit readable/writable register used for periodic interrupt control, 30-second adjustment, and frequency divider RESET and RTC count control.

RTC.RCR2 is basically initialized to 0x09 by a power-on reset, except that the value of the PEF bit is undefined. In a manual reset, bits other than RTCEN and START are initialized, while the value of the PEF bit is undefined. In standby mode RTC.RCR2 is not initialized, and retains its current value.

RTC.RCR2				0x3C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
START	0	1	✓	Start bit	RW
	Operation		Stops and restarts counter (clock) operation		
	When read		Returns current value		
	When written		0: Second, minute, hour, day, day-of-week, month, and year counters are stopped[a] 1: Second, minute, hour, day, day-of-week, month, and year counters operate normally (Initial value)		
	HARD reset		1		
RESET	1	1	✓	Reset bit	RW
	Operation		The frequency divider circuits are initialized by writing 1 to this bit. When 1 is written to the RESET bit, the frequency divider circuits (RTC prescaler and RTC.R64CNT) are reset and the RESET bit is automatically cleared to 0 (that is, does not need to be written with 0)		
	When read		Returns current value		
	When written		0: Normal clock operation (Initial value) 1: Frequency divider circuits are reset		
	HARD reset		0		

Table 71: RTC.RCR2

RTC.RCR2				0x3C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ADJ	2	1	✓	Second adjustment	RW
	Operation		Used for 30-second adjustment. When 1 is written to this bit, a value up to 29 seconds is rounded down to 00 seconds, and a value of 30 seconds or more is rounded up to 1 minute. The frequency divider circuits (RTC prescaler and RTC.R64CNT) are also reset at this time. This bit always returns 0 if read			
	When read		Returns current value		
	When written		0: Normal clock operation (Initial value) 1: 30-second adjustment performed		
	HARD reset		0		
RTCEN	3	1	✓	Oscillator enable	RW
	Operation		Controls the operation of the RTC's crystal oscillator		
	When read		Returns current value		
	When written		0: NRTC crystal oscillator is halted 1: RTC crystal oscillator is operated (Initial value)		
	HARD reset		1		

Table 71: RTC.RCR2

RTC.RCR2				0x3C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PESn where n is 2, 1, 0	[6:4]	3	✓	Periodic interrupt enable	RW
	Operation		These bits specify the period for periodic interrupts		
	When read		Returns current value		
	When written		000: No periodic interrupt generation (Initial value) 001: Periodic interrupt generated at 1/256 second intervals 010: Periodic interrupt generated at 1/64 second intervals 011: Periodic interrupt generated at 1/16 second intervals 100: Periodic interrupt generated at 1/4 second intervals 101: Periodic interrupt generated at 1/2 second intervals 110: Periodic interrupt generated at 1 second intervals 111: Periodic interrupt generated at 2 second intervals		
	HARD reset		0		
PEF	7	1	✓	Periodic interrupt flag	RW
	Operation		Indicates interrupt generation at the interval specified by bits PES2–PES0. When this flag is set to 1, a periodic interrupt is generated		
	When read		Returns current value		
	When written		0: Interrupt is not generated at interval specified by bits PES2 to PES0 1: Interrupt is generated at interval specified by bits PES2 to PES0		
	HARD reset		Undefined		

Table 71: RTC.RCR2

a. The counter RTC.R64CNT continues to operate unless stopped by means of the RTCEN bit.

7.3 Operation

Examples of the use of the RTC are shown below.

7.3.1 Time setting procedures

Figure 15 shows examples of the time setting procedures.

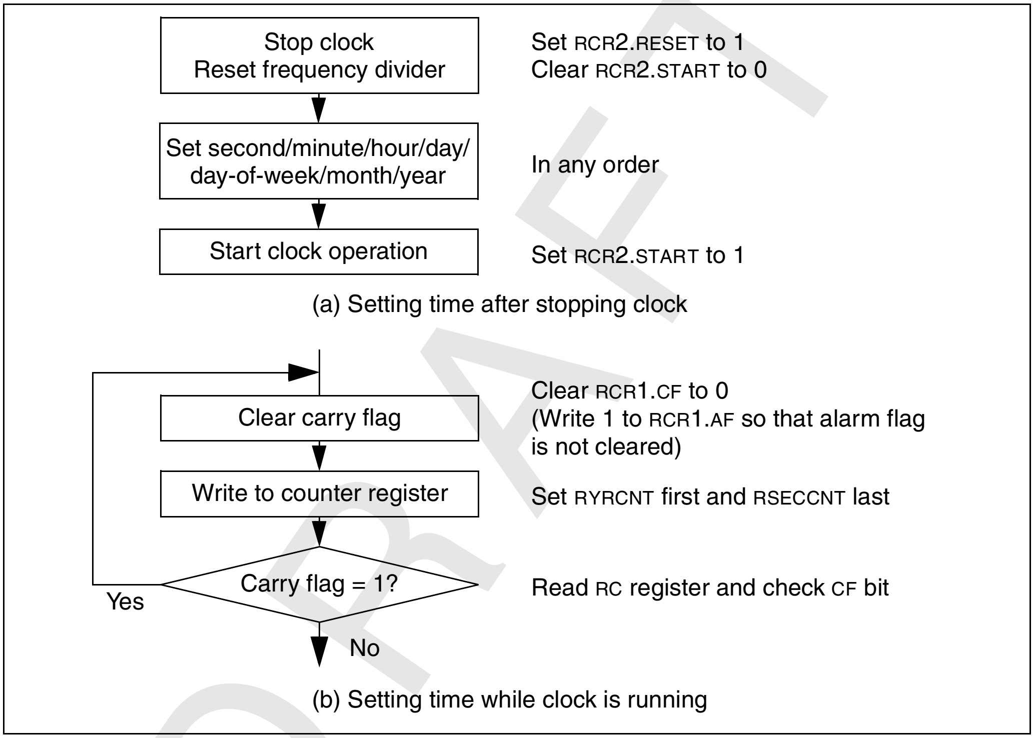


Figure 15: Examples of time setting procedures

The procedure for setting the time after stopping the clock is shown in (a). The programming for this method is simple, and it is useful for setting all the counters, from second to year.

The procedure for setting the time while the clock is running is shown in (b). This method is useful for modifying only certain counter values (for example, only the second data or hour data). If a carry occurs during the write operation, the write data is automatically updated and there will be an error in the set data. The carry flag should therefore be used to check the write status. If the carry flag (RTC.RCR1.CF) is set to 1, the write must be repeated.

The interrupt function can also be used to determine the carry flag status.

7.3.2 Time reading procedures

Figure 16 shows examples of the time reading procedures.

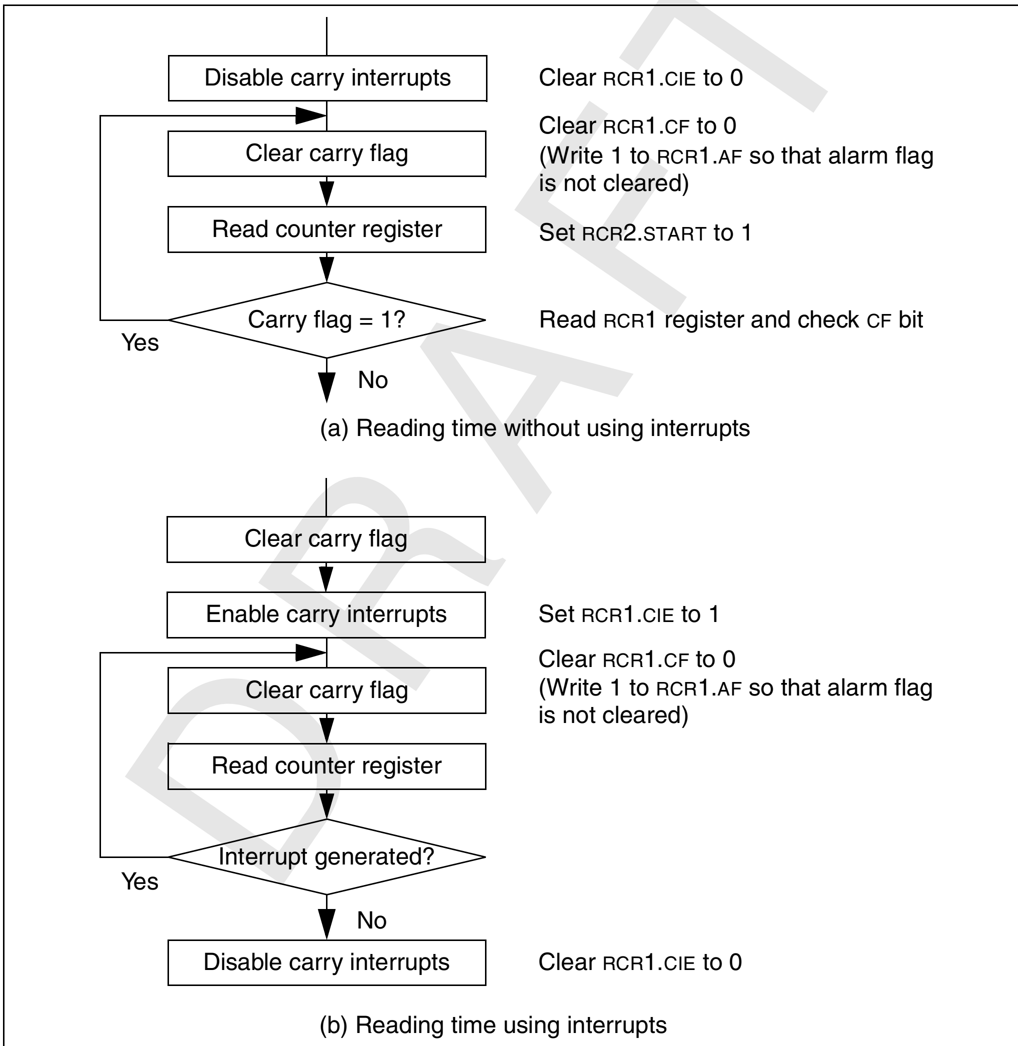


Figure 16: Examples of time reading procedures

If a carry occurs while the time is being read, the correct time will not be obtained and the read must be repeated. The procedure for reading the time without using interrupts is shown in (a), and the procedure using carry interrupts in (b). The method without using interrupts is normally used to keep the program simple.

7.3.3 Alarm function

The use of the alarm function is illustrated in *Figure 17*.

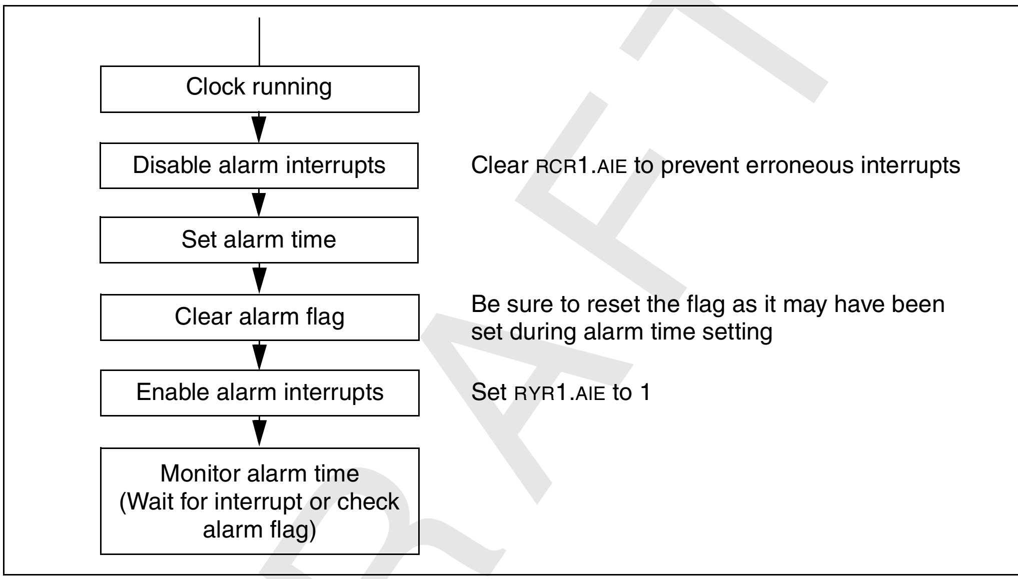


Figure 17: Example of use of alarm function

An alarm can be generated by the second, minute, hour, day-of-week, day, or month value, or a combination of these. Write 1 to the ENB bit in the alarm registers involved in the alarm setting, and set the alarm time in the lower bits. Write 0 to the ENB bit in registers not involved in the alarm setting.

When the counter and the alarm time match, RTC.RCR1.AF is set to 1. Alarm detection can be confirmed by reading this bit, but normally an interrupt is used. If 1 has been written to RTC.RCR1.AIE, an alarm interrupt is generated in the event of alarm, enabling the alarm to be detected.

The alarm flag remains set while the counter and alarm time match. If the alarm flag is cleared by writing 0 during this period, it will therefore be set again immediately afterward. This needs to be taken into consideration when writing the program.

7.4 Interrupts

There are three kinds of RTC interrupt: alarm interrupts, periodic interrupts, and carry interrupts.

An alarm interrupt request (ATI) is generated when the alarm flag (AF) in RTC.RCR1 is set to 1 while the alarm interrupt enable bit (AIE) is also set to 1.

A periodic interrupt request (PRI) is generated when the periodic interrupt enable bits (PES2 to PES0) in RTC.RCR2 are set to a value other than 000 and the periodic interrupt flag (PEF) is set to 1.

A carry interrupt request (CUI) is generated when the carry flag (CF) in RTC.RCR1 is set to 1 while the carry interrupt enable bit (CIE) is also set to 1.

7.5 Usage notes

7.5.1 Register initialization

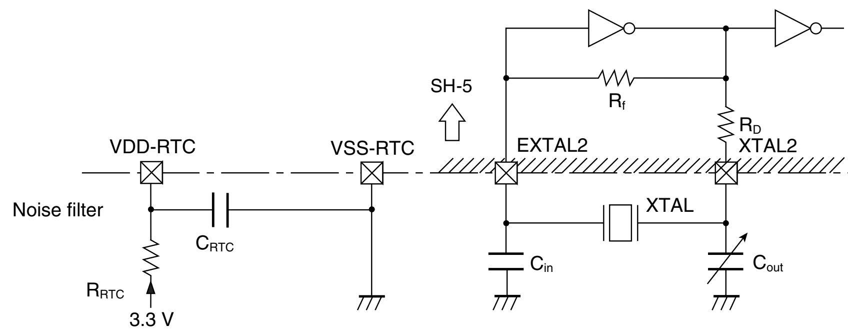
After powering on and making the RTC.RCR1 register settings, reset the frequency divider (by setting RTC.RCR2.RESET to 1) and make initial settings for all the other registers.

7.5.2 Crystal oscillator circuit

Crystal oscillator circuit constants (recommended values) are shown in *Table 72*, and the RTC crystal oscillator circuit in *Figure 18*.

f_{osc}	C_{in}	C_{out}
32.768 kHz	10-22 pF	10-22 pF

Table 72: Crystal oscillator circuit constants (recommended values)



Notes:
1. Select either the C_{in} or C_{out} side for the frequency adjustment variable capacitor according to requirements such as the adjustment range, degree of stability, etc.
2. Built-in resistance value R_f (typ. value) = 10 MΩ, R_D (typ. value) = 400 kΩ
3. C_{in} and C_{out} values include floating capacitance due to the wiring. Take care when using a solid-earth board.
4. The crystal oscillation stabilization time depends on the mounted circuit constants, floating capacitance, etc., and should be decided after consultation with the crystal resonator manufacturer.
5. Place the crystal resonator and load capacitors C_{in} and C_{out} as close as possible to the chip. (Correct oscillation may not be possible if there is externally induced noise in the EXTAL2 and XTAL2 pins.)
6. Ensure that the crystal resonator connection pin (EXTAL2 and XTAL2) wiring is routed as far away as possible from other power lines (except GND) and signal lines.
7. Insert a noise filter in the RTC power supply.
   The values of C_{RTC} and R_{RTC} depend on the bus and CPU frequency.

Figure 18: Example of crystal oscillator circuit connection

8

Timer unit (TMU)

8.1 Overview

This chapter describes the on-chip 32-bit timer unit (TMU) module comprising three 32-bit timer channels (channels 0 to 2).

8.1.1 Features

The TMU has the following features.

- Auto-reload type 32-bit down-counter provided for each channel.
- Input capture function provided in channel 2.
- Selection of rising edge or falling edge as external clock input edge when external clock is selected or input capture function is used.
- 32-bit timer constant register for auto-reload use, readable/writable at any time, and 32-bit down- counter provided for each channel.
- Selection of seven counter input clocks for each channel.
- External clock (TCLK), on-chip RTC output clock, five internal clocks (P/4, P/16, P/64, P/256, P/1024) (P is the peripheral module clock).
- Each channel can also operate in module standby mode when the on-chip RTC output clock is selected as the counter input clock; that is, timer operation continues even when the clock has been stopped for the TMU.

  Timer count operations using an external or internal clock are only possible when a clock is supplied to the timer unit.

- Synchronous read operation.

  As the timer counters (TCNT) are serially modified 32- bit registers and the internal peripheral module bus is 16 bits wide, there is a time difference when reading the upper 16 bits and lower 16 bits of TCNT. To prevent counter read value drift due to this time difference, a synchronization circuit is provided that allows simultaneous reading of all 32 bits of the TCNT data.

- Two interrupt sources.

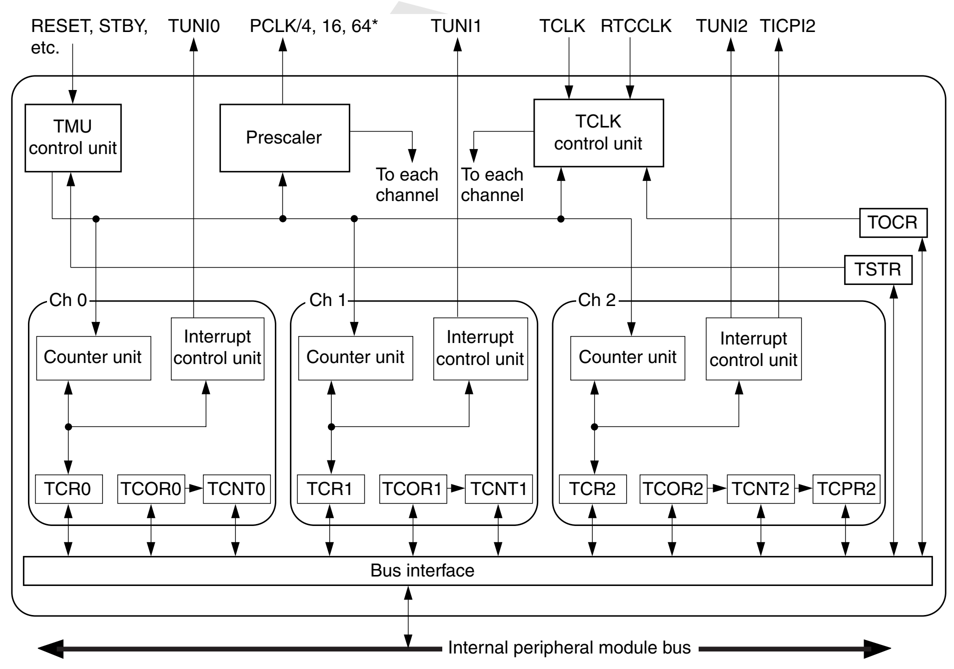
  One underflow source (channels 0 to 2) and one input capture source (channel 2).

- DMAC data transfer request capability.

  On channel 2, a data transfer request is sent to the DMAC when an input capture interrupt is generated.

8.1.2 Block diagram

Figure 19 shows a block diagram of the TMU.



Note: * Signals with 1/4, 1/16, and 1/64 the Pϕ frequency, supplied to the on-chip peripheral functions.

Figure 19: Block diagram of TMU

8.1.3 Pin configuration

Table 73 shows the TMU pin.

Pin name	Abbreviation	I/O	Function
Clock input/clock output	TCLK	I/O	External clock input pin/input capture control input pin/RTC output pin (shared with RTC)

Table 73: TMU pins

8.1.4 Register configuration

Table 74 summarizes the TMU registers. All Addresses are given as offsets to the TMU base address. Refer to the system address map for the value of TMUBASE.

Channel	Name	Abbreviation	RW	Initialization			Initial value	Offset	Access size
				Power-on reset	Manual reset	Standby mode			
Common	Timer output control register	TMU.TOCR	RW	Initialized	Initialized	Held	0x00	0x00	8
	Timer start register	TMU.TSTR	RW	Initialized	Initialized	Initialized	0x00	0x04	8
0	Timer constant register 0	TMU.TCOR0	RW	Initialized	Initialized	Held	0xFFFFFFFF	0x08	32
	Timer counter 0	TMU.TCNT0	RW	Initialized	Initialized	Held*2	0xFFFFFFFF	0x0C	32
	Timer control register 0	TMU.TCR0	RW	Initialized	Initialized	Held	0x0000	0x10	16

Table 74: TMU registers

Channel	Name	Abbreviation	RW	Initialization			Initial value	Offset	Access size
				Power-on reset	Manual reset	Standby mode			
1	Timer constant register 1	TMU.TCOR1	RW	Initialized	Initialized	Held	0xFFFFFFFF	0x14	32
	Timer counter 1	TMU.TCNT1	RW	Initialized	Initialized	Held*2	0xFFFFFFFF	0x18	32
	Timer control register 1	TMU.TCR1	RW	Initialized	Initialized	Held	0x0000	0x1C	16
2	Timer constant register 2	TMU.TCOR2	RW	Initialized	Initialized	Held	0xFFFFFFFF	0x20	32
	Timer counter 2	TMU.TCNT2	RW	Initialized	Initialized	Held*2	0xFFFFFFFF	0x24	32
	Timer control register 2	TMU.TCR2	RW	Initialized	Initialized	Held	0x0000	0x28	16
	Input capture register	TMU.TCPR2	R	Held	Held	Held	Undefined	0x2C	32

Table 74: TMU registers

Note: 1 Not initialized in module standby mode when the input clock is the on-chip RTC output clock.

2 Counts in module standby mode when the input clock is the on-chip RTC output clock.

8.2 Register descriptions

8.2.1 Timer output control register (TMU.TOCR)

TMU.TOCR is an 8-bit readable/writable register that specifies whether external pin TCLK is used as the external clock or input capture control input pin, or as the on-chip RTC output clock output pin.

TMU.TOCR is initialized to 0x00 by a power-on or manual reset, but is not initialized in standby mode.

<table>
<tr><th colspan="4">TMU.TOCR</th><th colspan="2">0x00</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">TCOE</td><td>[0]</td><td>1</td><td>-</td><td>Timer Output Control</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies whether timer clock pin TCLK is used as the external clock or input capture control input pin, or as the on-chip RTC output clock output pin</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Timer clock pin (TCLK) is used as external clock input or input capture control input pin
1: Timer clock pin (TCLK) is used as on-chip RTC output clock output pin</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">RESERVED</td><td>[7:1]</td><td>7</td><td>✓</td><td>Reserved</td><td>RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Reserved</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">0</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Invalid</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 75: TMU.TOCR

8.2.2 Timer start register (TMU.TSTR)

TMU.TSTR is an 8-bit readable/writable register that specifies whether the channel 0 to channel 2 timer counters (TCNT) are operated or stopped.

TMU.TSTR is initialized to 0x00 by a power-on or manual reset. In module standby mode, TMU.TSTR is not initialized when the input clock selected by each channel is the on-chip RTC output clock (RTCCLK), and is initialized only when the input clock is the external clock (TCLK) or internal clock (Pø)

TMU.TSTR				0x04	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
STR0	[0]	1	-	Counter 0 start	RW
	Operation		Specifies whether timer counter 0 (TMU.TCNT0) is operated or stopped		
	When read		Returns current value		
	When written		0: TMU.TCNT0 count operation is stopped 1: TMU.TCNT0 performs count operation		
	HARD reset		0		
STR1	[1]	1	-	Counter 1 start	RW
	Operation		Specifies whether timer counter 1 (TMU.TCNT1) is operated or stopped		
	When read		Returns current value		
	When written		0: TMU.TCNT1 count operation is stopped 1: TMU.TCNT1 performs count operation		
	HARD reset		0		

Table 76: TMU.TSTR

TMU.TSTR				0x04	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
STR2	[2]	1	-	Counter 2 start	RW
	Operation		Specifies whether timer counter 2 (TMU.TCNT2) is operated or stopped		
	When read		Returns current value		
	When written		0: TMU.TCNT2 count operation is stopped 1: TMU.TCNT2 performs count operation		
	HARD reset		0		
-	[7:3]	5	✓	Reserved	RES
	Operation				
	When read		0		
	When written		Invalid.		
	HARD reset		0		

Table 76: TMU.TSTR

8.2.3 Timer constant registers (TMU.TCOR)

The TCOR registers are 32-bit readable/writable registers. There are three TCOR registers, one for each channel.

When a TCNT counter underflows while counting down, the TCOR value is set in that TCNT, which continues counting down from the set value.

The TCOR registers are initialized to 0xFFFFFFFF by a power-on or manual reset, but are not initialized and retain their contents in standby mode.

<table>
<tr><th colspan="4">TMU.TCOR[n] where n=[0,2]</th><th colspan="2">0x08 +(n*0x0C)</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5"></td><td>[31:0]</td><td>32</td><td>-</td><td>Timer constant</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">This value is used to reload TCNT[n] when it underflows</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0xFFFFFFFF</td></tr>
</table>

Table 77: TMU.TCOR registers

8.2.4 Timer Counters (TMU.TCNT)

The TCNT registers are 32-bit readable/writable registers. There are three TCNT registers, one for each channel.

Each TCNT counts down on the input clock selected by TPSC2 to TPSC0 in the timer control register (TCR).

When a TCNT counter underflows while counting down, the underflow flag (UNF) is set in the corresponding timer control register (TCR). At the same time, the timer constant register (TCOR) value is set in TCNT, and the count-down operation continues from the set value.

As the TCNT registers are serially modified 32-bit registers and the internal peripheral module bus is 16 bits wide, there is a time difference when reading the upper 16 bits and lower 16 bits of TCNT. To prevent counter read value drift due to this time difference, a synchronization circuit is provided. When the upper 16 bits are read, the lower 16 bits are simultaneously stored in a buffer register. After the upper 16 bits are read, the lower 16 bits are read from the buffer register.

The TCNT registers are initialized to 0xFFFFFFFF by a power-on or manual reset, but are not initialized and retain their contents in standby mode.

TMU.TCNT[n] where n=[0,2]				0x0C+(n*0x0C)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
	[31:0]	32	✓	Timer counter	RW
	Operation		32-bit down counter; counts on the clock selected by TMU.TCR		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0xFFFFFFFF		

Table 78: TMU.TCNT registers

When the input clock is the on-chip RTC output clock (RTCCLK), TCNT counts even in module standby mode (that is, when the clock for the TMU is stopped). When the input clock is the external clock (TCLK) or internal clock (P), TCNT contents are retained in standby mode.

8.2.5 Timer control registers (TMU.TCR)

The TCR registers are 16-bit readable/writable registers. There are three TCR registers, one for each channel.

Each TCR selects the count clock, specifies the edge when an external clock is selected, and controls interrupt generation when the flag indicating timer counter (TCNT) underflow is set to 1. TMU.TCR2 is also used for channel 2 input capture control, and control of interrupt generation in the event of input capture.

The TCR registers are initialized to 0x0000 by a power-on or manual reset, but are not initialized in standby mode. Note that there are two register formats one used by TCR[0] and TCR[1] and another by TCR[2].

TMU.TCR[n] where n=[0,1]				0x10 + (n*0x0C)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TPSC	[2:0]	3	-	Timer prescaler	RW
	Operation		Specifies the TCNT count clock for channel n		
	When read		Returns current value		
	When written		000: Counts on Pϕ/4 001: Counts on Pϕ/16 010: Counts on Pϕ/64 011: Counts on Pϕ/256 100: Counts on Pϕ/1024 101: Reserved (do not set) 110: Counts on-chip RTC output clock 111: Counts on external clock		
	HARD reset		000		
CKEG	[4:3]	2	-	Clock edge	RW
	Operation		Select the external clock input edge when an external clock is selected or the input capture function is used		
	When read		Returns current value		
	When written		00: Count/input capture register set on rising edge 01: Count/input capture register set on falling edge 1X: Count/input capture register set on both rising and falling edges		
	HARD reset		00		

Table 79: TMU.TCR[n]

TMU.TCR[n] where n=[0,1]				0x10 + (n*0x0C)	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
UNIE	[5]	1	-	Underflow interrupt control	RW
	Operation		Controls enabling or disabling of interrupt generation when the UNF status flag is set to 1, indicating TCNT underflow		
	When read		Returns current value		
	When written		0: Interrupt due to underflow (TUNI) is not enabled 1: Interrupt due to underflow (TUNI) is enable		
	HARD reset		0		
RESERVED	[7:6]	2	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored.		
	HARD reset		0		
UNF	[8]	1	✓	Underflow flag	RW
	Operation		Status flag that indicates the occurrence of underflow		
	When read		0: TCNT has not underflowed 1: TCNT has underflowed		
	When written		0: Clears flag to 0 1: Write Ignored.		
	HARD reset		0		
RESERVED	[15:9]	7	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 79: TMU.TCR[n]

Timer Channel 2 Notes

When the input capture function is used, a data transfer request is sent to the DMAC in the event of input capture.

When using the input capture function, the TCLK pin must be designated as an input pin with the TCOE bit in the TMU.TOCR register. The CKEG bits specify whether the rising edge or falling edge of the TCLK signal is used to set the TMU.TCNT2 value in the input capture register (TMU.TCPR2).

The TMU.TCNT2 value is set in TMU.TCPR2 only when the TMU.TCR2.ICPF bit is 0. When the TMU.TCR2.ICPF bit is 1, TMU.TCPR2 is not set in the event of input capture. When input capture occurs, a DMAC transfer request is generated regardless of the value of the TMU.TCR2.ICPF bit. However, a new DMAC transfer request is not generated until processing of the previous request is finished.

TMU.TCR[2]				0x28	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TPSC	[2:0]	3	-	Timer prescaler	RW
	Operation		Specifies the TCNT count clock for channel 2		
	When read		Returns current value		
	When written		000: Counts on Pϕ/4 001: Counts on Pϕ/16 010: Counts on Pϕ/64 011: Counts on Pϕ/64 100: Counts on Pϕ/1024 101: Reserved (Do not set) 110: Counts on-chip RTC output clock 111: Counts on external clock		
	HARD reset		000		

Table 80: TMU.TCR[2]

TMU.TCR[2]				0x28	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CKEG	[4:3]	2	-	Clock edge	RW
	Operation		Select the external clock input edge when an external clock is selected or the input capture function is used		
	When read		Returns current value		
	When written		00: Count/input capture register set on rising edge 01: Count/input capture register set on falling edge 1X: Count/input capture register set on both rising and falling edges		
	HARD reset		00		
UNIE	[5]	1	-	Underflow interrupt control	RW
	Operation		Controls enabling or disabling of interrupt generation when the UNF status flag is set to 1, indicating TCNT underflow		
	When read		Returns current value		
	When written		0: Interrupt due to underflow (TUNI) is not enabled 1: Interrupt due to underflow (TUNI) is enable		
	HARD reset		0		

Table 80: TMU.TCR[2]

TMU.TCR[2]				0x28	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ICPE	[7:6]	2	-	Input capture control	RW
	Operation		Specify whether the input capture function is used, and control enabling or disabling of interrupt generation when the function is used		
	When read		Returns current value		
	When written		00: Input capture function is not used 01: Reserved (Do not set) 10: Input capture function is used, but interrupt due to input capture (TICPI2) is not enabled. Data transfer request is sent to DMAC in the event of input capture 11: Input capture function is used, and interrupt due to input capture (TICPI2) is enabled. Data transfer request is sent to DMAC in the event of input capture		
	HARD reset		0		
UNF	[8]	1	✓	Underflow flag	RW
	Operation		Status flag that indicates the occurrence of underflow		
	When read		0: TCNT has not underflowed 1: TCNT has underflowed		
	When written		0: Clears flag to 0 1: Write Ignored.		
	HARD reset		0		

Table 80: TMU.TCR[2]

TMU.TCR[2]				0x28	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ICPF	[9]	1	✓	Input capture interrupt flag	RW
	Operation		Status flag, provided in channel 2 only, that indicates the occurrence of input capture		
	When read		0: Input capture has not occurred 1: Input capture has occurred		
	When written		0: Clear flag to 0 1: Write ignored.		
	HARD reset		0		
RESERVED	[15:10]	6	-	Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 80: TMU.TCR[2]

8.2.6 Input capture register (TMU.TCPR2)

TMU.TCPR2 is a 32-bit read-only register for use with the input capture function, provided only in channel 2.

The input capture function is controlled by means of the input capture control bits (ICPE1, ICPE0) and clock edge bits (CKEG1, CKEG0) in TMU.TCR2. When input capture occurs, the TMU.TCNT2 value is copied into TMU.TCPR2. The value is set in TMU.TCPR2 only when the ICPF bit in TMU.TCR2 is 0.

TMU.TCPR2 is not initialized by a power-on or manual reset, or in standby mode.

TMU.TCPR2				0x2C)	
Field	Bits	Size	Volatile?	Synopsis	Type
	[31:0]	32	-	Input capture value	RO
	Operation		This value of TMU.TCNT2 when capture occurs		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0xFFFFFFFF		

Table 81: TMU.TCPR2 registers

8.3 Operation

Each channel has a 32-bit timer counter (TCNT) that performs count-down operations, and a 32-bit timer constant register (TCOR). The channels have an auto-reload function that allows cyclic count operations, and can also perform external event counting. Channel 2 also has an input capture function.

8.3.1 Counter operation

When one of bits STR0 to STR2 is set to 1 in the timer start register (TMU.TSTR), the timer counter (TCNT) for the corresponding channel starts counting. When TCNT underflows, the UNF flag is set in the corresponding timer control register (TCR). If the UNIE bit in TCR is set to 1 at this time, an interrupt request is sent to the CPU. At the same time, the value is copied from TCOR into TCNT, and the count-down continues (auto-reload function).

Example of count operation setting procedure

Figure 20 shows an example of the count operation setting procedure.

1 Select the count clock with bits TPSC2 to TPSC0 in the timer control register (TCR). When an external clock is selected, set the TCLK pin to input mode with the TCOE bit in TMU.TOCR, and select the external clock edge with bits CKEG1 and CKEG0 in TCR.

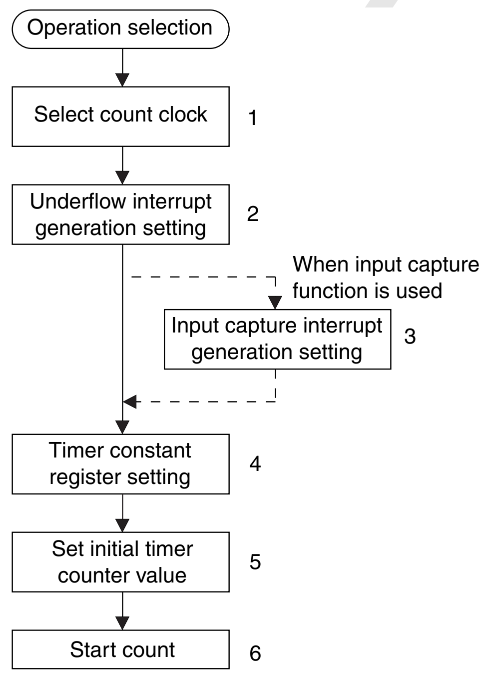
2 Specify whether an interrupt is to be generated on TCNT underflow with the UNIE bit in TCR.

3 When the input capture function is used, set the ICPE bits in TCR, including specification of whether the interrupt function is to be used.

4 Set a value in the timer constant register (TCOR).

5 Set the initial value in the timer counter (TCNT).

6 Set the STR bit to 1 in the timer start register (TMU.TSTR) to start the count.



Note: When an interrupt is generated, clear the source flag in the interrupt handler. If the interrupt enabled state is set without clearing the flag, another interrupt will be generated.

Figure 20: Example of count operation setting procedure

Auto-reload count operation

Figure 21 shows the TCNT auto-reload operation.

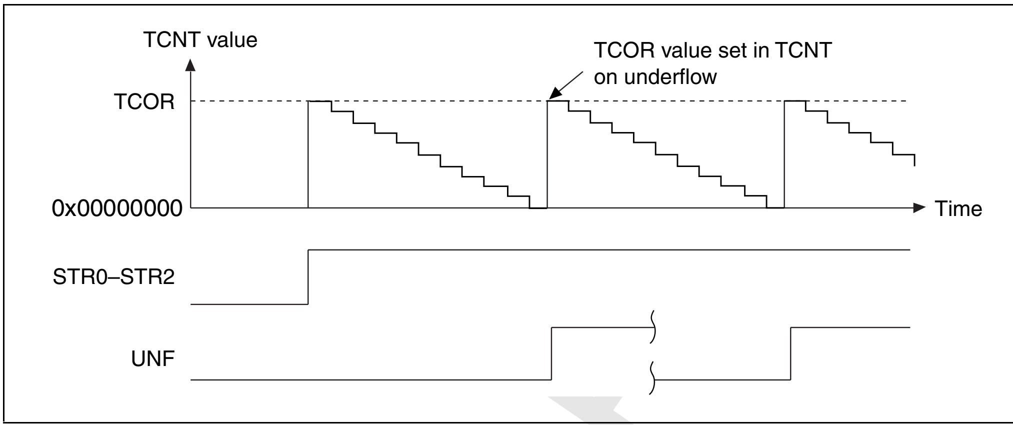


Figure 21: TCNT auto-reload operation

TCNT count timing

- Operating on internal clock

  Any of five count clocks (Pø/4, Pø/16, Pø/64, Pø/256, or Pø/1024) scaled from the peripheral module clock can be selected as the count clock by means of the TPSC2 to TPSC0 bits in TCR.

  Figure 22 shows the timing in this case.

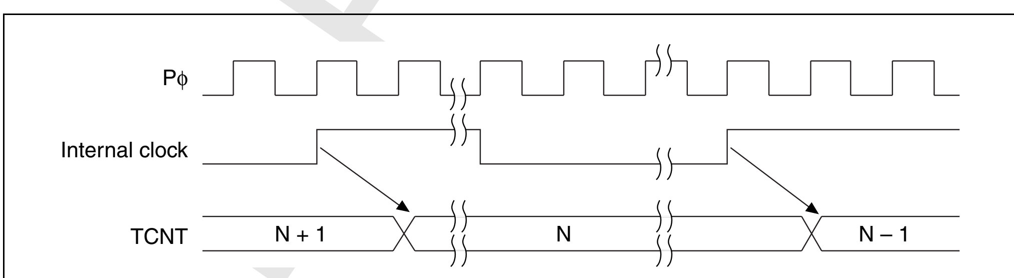


Figure 22: Count timing when operating on internal clock

- Operating on external clock

  External clock pin (TCLK) input can be selected as the timer clock by means of the TPSC2 to TPSC0 bits in TCR. The detected edge (rising, falling, or both edges) can be selected with the CKEG1 and CKEG0 bits in TCR. *Figure 23* shows the timing for both-edge detection.

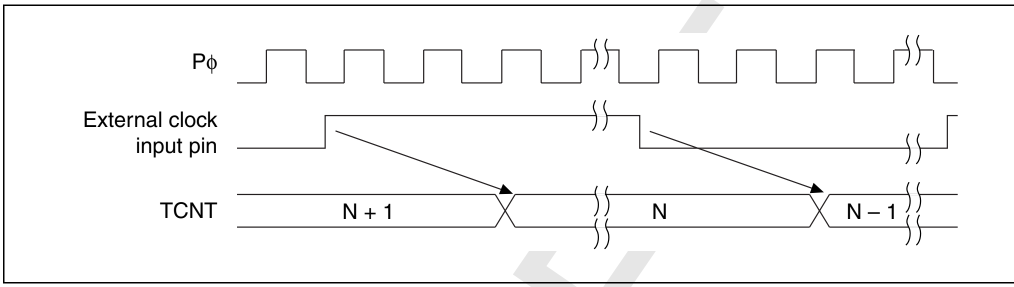


Figure 23: Count timing when operating on external clock

- Operating on-chip RTC output clock

  The on-chip RTC output clock can be selected as the timer clock by means of the TPSC2 to TPSC0 bits in TCR. *Figure 24* shows the timing in this case.

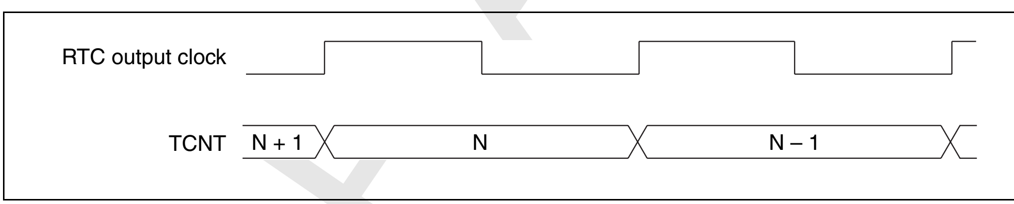


Figure 24: Count timing when operating on the on-chip RTC output clock

8.3.2 Input capture function

Channel 2 has an input capture function.

The procedure for using the input capture function is as follows:

1 Use the TCOE bit in the timer output control register (TMU.TOCR) to set the TCLK pin to input mode.

2 Use bits TPSC2 to TPSC0 in the timer control register (TCR) to set an internal clock or the on- chip RTC output clock as the timer operating clock.

3 Use bits IPCE1 and IPCE0 in TCR to specify use of the input capture function, and whether interrupts are to generated when this function is used.

4 Use bits CKEG1 and CKEG0 in TCR to specify whether the rising or falling edge of the TCLK signal is to be used to set the timer counter (TCNT) value in the input capture register (TMU.TCPR2).

This function cannot be used in standby mode.

When input capture occurs, the TMU.TCNT2 value is set in TMU.TCPR2 only when the ICPF bit in TMU.TCR2 is 0. Also, a new DMAC transfer request is not generated until processing of the previous request is finished.

Figure 25 shows the operation timing when the input capture function is used (with TCLK rising edge detection).

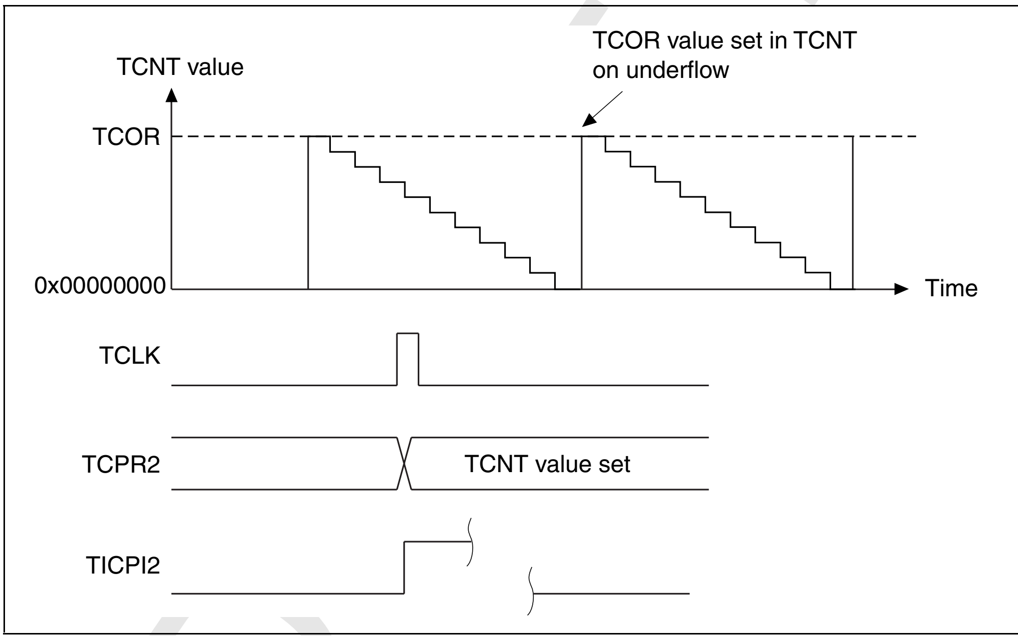


Figure 25: Operation timing when using input capture function

8.4 Interrupts

There are four TMU interrupt sources, comprising underflow interrupts and the input capture interrupt (when the input capture function is used). Underflow interrupts are generated on channels 0 to 2, and input capture interrupts on channel 2 only.

An underflow interrupt request is generated (for each channel) according to the AND of UNF and the interrupt enable bit (UNIE) in TCR.

When the input capture function is used and an input capture request is generated, an interrupt is requested if the input capture input flag (ICPF) in TMU.TCR2 is 1 and the input capture control bits (ICPE1, ICPE0) in TMU.TCR2 are 11.

The TMU interrupt sources are summarized in *Table 82*.

Channel	Interrupt source	Description	Priority
0	TUNI0	Underflow interrupt 0	High
1	TUNI1	Underflow interrupt 1	↑
2	TUNI2	Underflow interrupt 2	↓
	TICPI2	Input capture interrupt 2	Low

Table 82: TMU interrupt sources

8.5 Usage notes

8.5.1 Register writes

When performing a register write, timer count operation must be stopped by clearing the start bit (STR0 to STR2) for the relevant channel in the timer start register (TMU.TSTR).

8.5.2 TCNT register reads

When performing a TCNT register read, processing for synchronization with the timer count operation is performed. If a timer count operation and register read processing are performed simultaneously, the TCNT counter value prior to the count-down operation is read by means of the synchronization processing.

8.5.3 Resetting the RTC frequency divider

When the on-chip RTC output clock is selected as the count clock, the RTC frequency divider should be reset.

8.5.4 External clock frequency

Ensure that the external clock frequency for any channel does not exceed P/4.

Serial comms interface with FIFO (SCIF)

9

9.1 Overview

This chapter describes a single-channel serial communication interface module with built-in FIFO registers (serial communication interface with FIFO: SCIF). The SCIF can perform asynchronous serial communication.

Sixteen-stage FIFO registers are provided for both transmission and reception, enabling fast, efficient, and continuous communication.

9.1.1 Features

SCIF features are listed below.

- Asynchronous serial communication.

  Serial data communication is executed using an asynchronous system in which synchronization is achieved character by character. Serial data communication can be carried out with standard asynchronous communication chips such as a universal asynchronous receiver/transmitter (UART) or asynchronous communication interface adapter (ACIA).

  There is a choice of eight serial data transfer formats.

  - Data length: 7 or 8 bits
  - Stop bit length: 1 or 2 bits
  - Parity: Even/odd/none
  - Receive error detection: Parity, framing, and overrun errors

   - Break detection: If the receive data following that in which a framing error occurred is also at the space “0” level, and there is a frame error, a break is detected. When a framing error occurs, a break can also be detected by reading the RXD2 pin level directly from the serial port register (SCIF.SCSPTR2).

- Full-duplex communication capability

  The transmitter and receiver are independent units, enabling transmission and reception to be performed simultaneously.

  The transmitter and receiver both have a 16-stage FIFO buffer structure, enabling fast and continuous serial data transmission and reception.

- On-chip baud rate generator allows any bit rate to be selected.
- Choice of serial clock source: internal clock from baud rate generator or external clock from SCK2 pin
- Four interrupt sources.

  There are four interrupt sources (transmit-FIFO-data-empty, break, receive-FIFO-data-full, and receive-error) that can issue requests independently.

- The DMA controller (DMAC) can be activated to execute a data transfer by issuing a DMA transfer request in the event of a transmit-FIFO-data-empty or receive-FIFO-data-full interrupt.
- When not in use, the SCIF can be stopped by halting its clock supply to reduce power consumption.
- Modem control functions (RTS2 and CTS2) are provided.
- The amount of data in the transmit/receive FIFO registers, and the number of receive errors in the receive data in the receive FIFO register, can be ascertained.
- A time-out error (DR) can be detected during reception.

9.1.2 Block diagram

Figure 26 shows a block diagram of the SCIF.

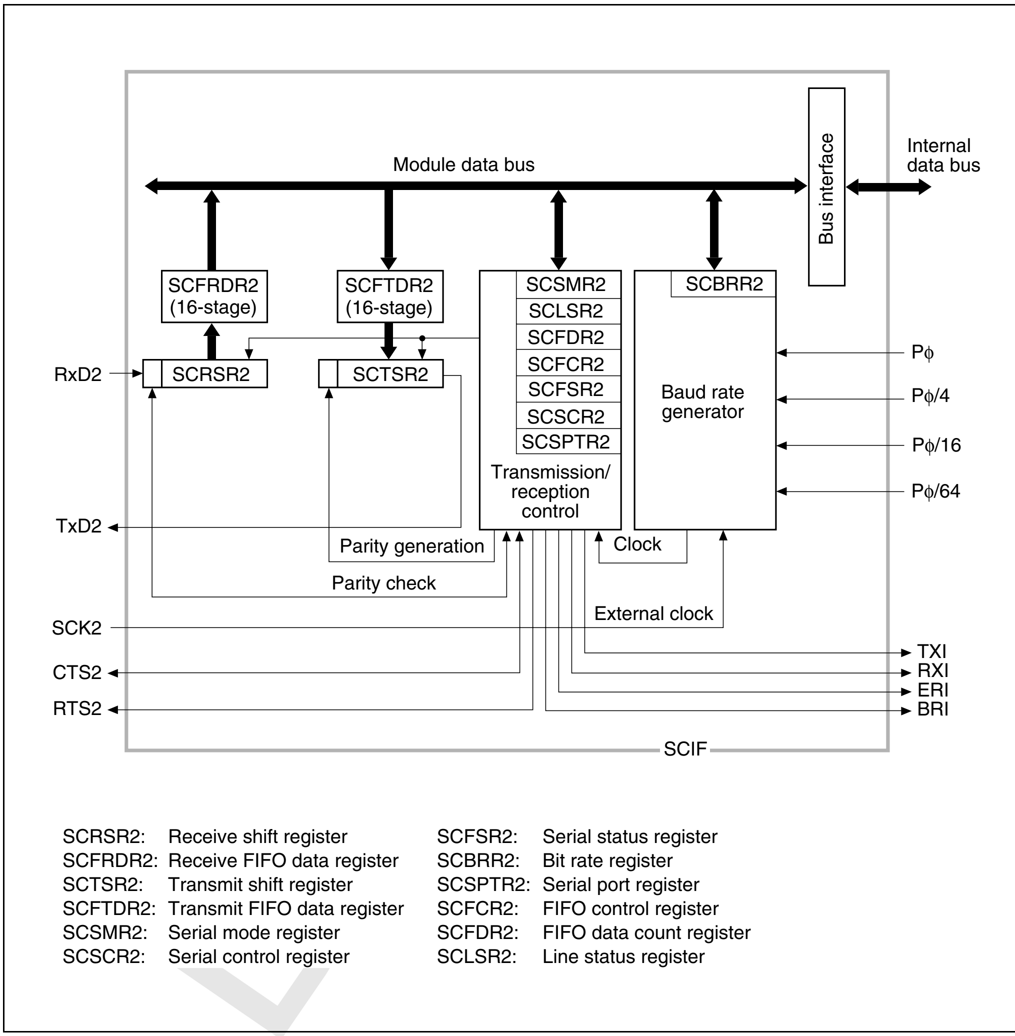


Figure 26: Block diagram of SCIF

9.1.3 Pin configuration

Table 83 shows the SCIF pin configuration.

Pin name	Abbreviation	I/O	Function
Serial clock pin	SCK2	I/O	Clock input/output or port input/output
Receive data pin	RXD2	Input	Receive data/port input
Transmit data pin	TXD2	Output	Transmit data/port output
Modem control pin	CTS2	I/O	Transmission enabled (Clear To Send) or port input/output
Modem control pin	RTS2	I/O	Transmission request (Request To Send) or port input/output

Table 83: SCIF Pins

Note: *These pins are made to function as serial pins by performing SCIF operation settings with the TE and RE bits in* SCIF.SCSCR2 *and the MCE bit in* SCIF.SCFCR2. *Break state transmission and detection can be set in the SCIF's* SCIF.SCSPTR2 *register.*

9.1.4 Register configuration

The SCIF has the internal registers shown in *Table 84*. These registers are used to specify the data format and bit rate, and to perform transmitter/receiver control. All addresses are given as offsets from the base address for this module. See the system address map for details.

Name	Abbreviation	RW	Initial value	Offset	Access size
Serial mode register	SCIF.SCSMR2	RW	0x0000	0x00	16
Bit rate register	SCIF.SCBRR2	RW	0xFF	0x04	8
Serial control register	SCIF.SCSCR2	RW	0x0000	0x08	16
Transmit FIFO data register	SCIF.SCFTDR2	WO	Undefined	0x0C	8
Serial status register	SCIF.SCFSR2	R/(W)[a]	0x0060	0x10	16

Table 84: SCIF registers

Name	Abbreviation	RW	Initial value	Offset	Access size
Receive FIFO data register	SCIF.SCFRDR2	R	Undefined	0x14	8
FIFO control register	SCIF.SCFCR2	RW	0x0000	0x18	16
FIFO data count register	SCIF.SCFDR2	R	0x0000	0x1C	16
Serial port register	SCIF.SCSPTR2	RW	0x0000[b]	0x20	16
Line status register	SCIF.SCLSR2	R/(W)[c]	0x0000	0x24	16

Table 84: SCIF registers

a. Only 0 can be written, to clear flags. Bits 15 to 8, 3, and 2 are read-only, and cannot be modified.

b. The value of bits 6, 4, and 0 is undefined.

c. Only 0 can be written, to clear flags. Bits 15 to 1 are read-only, and cannot be modified.

9.2 Register descriptions

This section describes all register state for the SCIF module. Note that all addresses are given as offsets from the base address for this module. See the system address map for details.

9.2.1 Receive shift register (SCIF.SCRSR2)

SCIF.SCRSR2 is the register used to receive serial data.

The SCIF sets serial data input from the RXD2 pin in SCIF.SCRSR2 in the order received, starting with the LSB (bit 0), and converts it to parallel data. When one byte of data has been received, it is transferred to the receive FIFO register, SCIF.SCFRDR2, automatically.

SCIF.SCRSR2 cannot be directly read or written to by the CPU.

9.2.2 Receive FIFO data register (SCIF.SCFRDR2)

SCIF.SCFRDR2 is a 16-stage FIFO register that stores received serial data.

When the SCIF has received one byte of serial data, it transfers the received data from SCIF.SCRSR2 to SCIF.SCFRDR2 where it is stored, and completes the receive operation. SCIF.SCRSR2 is then enabled for reception, and consecutive receive operations can be performed until the receive FIFO register is full (16 data bytes).

SCIF.SCFRDR2 is a read-only register, and cannot be written to by the CPU.

If a read is performed when there is no receive data in the receive FIFO register, an undefined value will be returned. When the receive FIFO register is full of receive data, subsequent serial data is lost. The contents of SCIF.SCFRDR2 are undefined after a power-on reset or manual reset.

SCIF.SCFRDR2				0x14	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SCIF.SCFRDR2	[7:0]	8	✓	16 byte receive FIFO data register	RO
	Operation		This register holds data transferred from the SCIF.SCRSR2 register		
	When read		The next data item in the FIFO is returned and removed from the FIFO. If the FIFO is empty and undefined value will be returned.		
	When written		Invalid		
	HARD reset		Undefined		

Table 85: SCIF.SCFRDR2

9.2.3 Transmit shift register (SCIF.SCTSR2)

SCIF.SCTSR2 is the register used to transmit serial data.

To perform serial data transmission, the SCIF first transfers transmit data from SCIF.SCFTDR2 to SCIF.SCTSR2, then sends the data to the TXD2 pin starting with the LSB (bit 0).

When transmission of one byte is completed, the next transmit data is transferred from SCIF.SCFTDR2 to SCIF.SCTSR2, and transmission started, automatically.

SCIF.SCTSR2 cannot be directly read or written to by the CPU.

9.2.4 Transmit FIFO data register (SCIF.SCFTDR2)

SCIF.SCFTDR2 is a 16-stage FIFO register that stores data for serial transmission.

If SCIF.SCTSR2 is empty when transmit data has been written to SCIF.SCFTDR2, the SCIF transfers the transmit data written in SCIF.SCFTDR2 to SCIF.SCTSR2 and starts serial transmission.

SCIF.SCFTDR2 is a write-only register, and cannot be read by the CPU.

The next data cannot be written when SCIF.SCFTDR2 is filled with 16 bytes of transmit data. Data written in this case is ignored. The contents of SCIF.SCFTDR2 are undefined after a power-on reset or manual reset.

SCIF.SCFTDR2				0x0C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SCFTDR2	[7:0]	8	✓	16-byte transmit FIFO data register	WO
	Operation		Stores data for serial transmission		
	When read		Invalid		
	When written		Appends data to the FIFO register to be copied to SCIF.SCTSR2 If the FIFO is full the write is ignored		
	HARD reset		Undefined		

Table 86: SCIF.SCFTDR2

9.2.5 Serial mode register (SCIF.SCSMR2)

SCIF.SCSMR2 is a 16-bit register used to set the SCIF's serial transfer format and select the baud rate generator clock source.

SCIF.SCSMR2 can be read or written to by the CPU at all times.

SCIF.SCSMR2 is initialized to 0x0000 by a power-on reset or manual reset. It is not initialized in standby mode or in the module standby state.

SCIF.SCSMR2				0x00	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CKS1, CKS0	[1:0]	2	-	Clock select 1 and 0	RW
	Operation		These bits select the clock source for the on-chip baud rate generator. See *Bits 1 and 0 - Clock Select 1 and 0 (CKS1, CKS0) on page 204*		
	When read		Returns current value.		
	When written		Updates current Value		
	HARD reset		0		
	[2]	1		Reserved	RES
	Operation		RESERVED		
	When read		0		
	When written		Should only be written with 0		
	HARD reset		0		
STOP	[3]	1	-	Stop bit length	RW
	Operation		Selects 1 or 2 bits as the stop bit length. See *Bit 3 - Stop bit length (STOP) on page 203*		
	When read		Returns current value		
	When written		Updates current value.		
	HARD reset		0		

Table 87: SCIF.SCSMR2

SCIF.SCSMR2				0x00	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
O/E	[4]	1	-	Parity mode	RW
	Operation		Selects either even or odd parity for use in parity addition and checking. This field is only used when the PE bit is set to 1. See *Bit 4 - Parity mode (O/E) on page 203*		
	When read		Returns the current value.		
	When written		Updates the current value		
	HARD reset		0		
PE	[5]	1	-	Parity Enable	RW
	Operation		Selects whether or not parity bit addition is performed in transmission, and parity bit checking in reception. See *Bit 5 - Parity enable (PE) on page 202*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
CHR	[6]	1	-	Character Length	RW
	Operation		Selects 7 or 8 bits as the asynchronous mode data length. See *Bit 6 - Character length (CHR) on page 202*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
	[15:7]	9		Reserved	RES
	Operation		RESERVED		
	When read		Returns 0		
	When written		These bits should only be written with 0		
	HARD reset		0		

Table 87: SCIF.SCSMR2

Bits 15 to 7 - Reserved

These bits are always read as 0, and should only be written with 0.

Bit 6 - Character length (CHR)

Selects 7 or 8 bits as the asynchronous mode data length.

Bit 6: CHR	Description
0	8-bit data (Initial value)
1	7-bit data[a]

a. When 7-bit data is selected, the MSB (bit 7) of SCIF.SCFTDR2 is not transmitted.

Bit 5 - Parity enable (PE)

Selects whether or not parity bit addition is performed in transmission, and parity bit checking in reception.

Bit 5: PE	Description
0	Parity bit addition and checking disabled (Initial value)
1	Parity bit addition and checking enabled[a]

a. When the PE bit is set to 1, the parity (even or odd) specified by the O/E bit is added to transmit data before transmission. In reception, the parity bit is checked for the parity (even or odd) specified by the O/E bit.

Bit 4 - Parity mode (O/E)

Selects either even or odd parity for use in parity addition and checking. The O/E bit setting is only valid when the PE bit is set to 1, enabling parity bit addition and checking. The O/E bit setting is invalid when parity addition and checking is disabled.

Bit 4: O/E	Description
0	Even parity[a] (Initial value)
1	Odd parity[b]

a. When even parity is set, parity bit addition is performed in transmission so that the total number of 1 bits in the transmit character plus the parity bit is even. In reception, a check is performed to see if the total number of 1 bits in the receive character plus the parity bit is even.

b. When odd parity is set, parity bit addition is performed in transmission so that the total number of 1 bits in the transmit character plus the parity bit is odd. In reception, a check is performed to see if the total number of 1 bits in the receive character plus the parity bit is odd.

Bit 3 - Stop bit length (STOP)

Selects 1 or 2 bits as the stop bit length.

Bit 3: STOP	Description
0	1 stop bit[a] (Initial value)
1	2 stop bits[b]

a. In transmission, a single 1 bit (stop bit) is added to the end of a transmit character before it is sent.

b. In transmission, two 1 bits (stop bits) are added to the end of a transmit character before it is sent.

In reception, only the first stop bit is checked, regardless of the STOP bit setting. If the second stop bit is 1, it is treated as a stop bit; if it is 0, it is treated as the start bit of the next transmit character.

Bit 2 - Reserved

This bit is always read as 0, and should only be written with 0.

Bits 1 and 0 - Clock Select 1 and 0 (CKS1, CKS0)

These bits select the clock source for the on-chip baud rate generator. The clock source can be selected from Pø, Pø/4, Pø/16, and Pø/64, according to the setting of bits CKS1 and CKS0. For the relation between the clock source, the bit rate register setting, and the baud rate, see *Section 9.2.8: Bit rate register (SCIF.SCBRR2) on page 221*.

Bit 1: CKS1	Bit 0: CKS0	Description
0	0	Pϕ clock (Initial value)
	1	Pϕ/4 clock
1	0	Pϕ/16 clock
	1	Pϕ/64 clock

Note: Pø is the Peripheral clock.

9.2.6 Serial control register (SCIF.SCSCR2)

The SCIF.SCSCR2 register performs enabling or disabling of SCIF transfer operations, and interrupt requests, and selection of the serial clock source.

SCIF.SCSCR2 can be read or written at any times.

SCIF.SCSCR2 is initialized to 0x0000 by a power-on reset or manual reset. It is not initialized in standby mode or in the module standby state.

SCIF.SCSCR2				0x08	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CKE0	[0]	1	-	Clock enable 0	RW
	Operation		Selects the SCIF clock source. See *Bits 1 and 0 - Clock enable (CKE0 and CKE1) on page 210*		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		
CKE1	[1]	1	-	Clock enable 1	RW
	Operation		Selects the SCIF clock source. See *Bits 1 and 0 - Clock enable (CKE0 and CKE1) on page 210*		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		
	[2]			Reserved	RES
	Operation		RESERVED		
	When read		0		
	When written		Ignored		
	HARD reset		0		
REIE	[3]	1	-	Receive error Interrupt enable	RW
	Operation		Enables or disables generation of receive-error interrupt (ERI) and break interrupt (BRI) requests. See *Bit 3 - Receive error interrupt enable (REIE) on page 209*.		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		

Table 88: SCIF.SCSCR2

SCIF.SCSCR2				0x08	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RE	[4]	1	-	Receive enable	RW
	Operation		Enables or disables the start of serial reception by the SCIF See *Bit 4 - Receive enable (RE) on page 209*		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		
TE	[5]	1	-	Transmit enable	RW
	Operation		Enables or disables the start of serial transmission by the SCIF. See *Bit 5 - Transmit enable (TE) on page 208*.		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		
RIE	[6]	1	-	Receive interrupt enable	RW
	Operation		Enables or disables generation of a receive interrupts. See *Bit 6 - Receive interrupt enable (RIE) on page 208*		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		
TIE	[7]	1	-	Transmit interrupt enable	RW
	Operation		Enables or disables transmit interrupts. See *Bit 7 - Transmit interrupt enable (TIE) on page 207*.		
	When read		Returns the current value		
	When written		Updates the current value		
	HARD reset		0		

Table 88: SCIF.SCSCR2

SCIF.SCSCR2				0x08	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
	[15:8]		-	Reserved	RES
	Operation		RESERVED		
	When read		0		
	When written		Ignored		
	HARD reset		0		

Table 88: SCIF.SCSCR2

Bits 15 to 8 and 2 - Reserved

These bits are always read as 0, and should only be written with 0.

Bit 7 - Transmit interrupt enable (TIE)

Enables or disables transmit-FIFO-data-empty interrupt (TXI) request generation when serial transmit data is transferred from SCIF.SCFTDR2 to SCIF.SCTSR2, the number of data bytes in the transmit FIFO register falls to or below the transmit trigger set number, and the TDFE flag in the serial status register (SCIF.SCFSR2) is set to 1.

Bit 7: TIE	Description
0	Transmit-FIFO-data-empty interrupt (TXI) request disabled[a] (initial value)
1	Transmit-FIFO-data-empty interrupt (TXI) request enabled

a. TXI interrupt requests can be cleared by writing transmit data exceeding the transmit trigger set number to SCIF.SCFTDR2, reading 1 from the TDFE flag, then clearing it to 0, or by clearing the TIE bit to 0.

Bit 6 - Receive interrupt enable (RIE)

Enables or disables generation of a receive-data-full interrupt (RXI) request when the RDF flag or DR flag in SCIF.SCFSR2 is set to 1, a receive-error interrupt (ERI) request when the ER flag in SCIF.SCFSR2 is set to 1, and a break interrupt (BRI) request when the BRK flag in SCIF.SCFSR2 or the ORER flag in SCIF.SCLSR2 is set to 1.

Bit 6: RIE	Description
0	Receive-data-full interrupt (RXI) request, receive-error interrupt (ERI) request, and break interrupt (BRI) request disabled[a] (Initial value)
1	Receive-data-full interrupt (RXI) request, receive-error interrupt (ERI) request, and break interrupt (BRI) request enabled

a. An RXI interrupt request can be cleared by reading 1 from the RDF or DR flag, then clearing the flag to 0, or by clearing the RIE bit to 0. ERI and BRI interrupt requests can be cleared by reading 1 from the ER, BRK, or ORER flag, then clearing the flag to 0, or by clearing the RIE and REIE bits to 0.

Bit 5 - Transmit enable (TE)

Enables or disables the start of serial transmission by the SCIF.

Bit 5: TE	Description
0	Transmission disabled (Initial value)
1	Transmission enabled[a]

a. Serial transmission is started when transmit data is written to SCIF.SCFTDR2 in this state.
Serial mode register (SCIF.SCSMR2) and FIFO control register (SCIF.SCFCR2) settings must be made, the transmission format decided, and the transmit FIFO reset, before the TE bit is set to 1.

Bit 4 - Receive enable (RE)

Enables or disables the start of serial reception by the SCIF.

Bit 4: RE	Description
0	Reception disabled[a] (Initial value)
1	Reception enabled[b]

a. Clearing the RE bit to 0 does not affect the DR, ER, BRK, RDF, FER, PER, and ORER flags, which retain their states.

b. Serial transmission is started when a start bit is detected in this state. Serial mode register (SCIF.SCSMR2) and FIFO control register (SCIF.SCFCR2) settings must be made, the reception format decided, and the receive FIFO reset, before the RE bit is set to 1.

Bit 3 - Receive error interrupt enable (REIE)

Enables or disables generation of receive-error interrupt (ERI) and break interrupt (BRI) requests. The REIE bit setting is valid only when the RIE bit is 0.

Bit 3: REIE	Description
0	Receive-error interrupt (ERI) and break interrupt (BRI) requests disabled[a] (Initial value)
1	Receive-error interrupt (ERI) and break interrupt (BRI) requests enabled

a. Receive-error interrupt (ERI) and break interrupt (BRI) requests can be cleared by reading 1 from the ER, BRK, or ORER flag, then clearing the flag to 0, or by clearing the RIE and REIE bits to 0. When REIE is set to 1, ERI and BRI interrupt requests will be generated even if RIE is cleared to 0. In DMAC transfer, this setting is made if the interrupt controller is to be notified of ERI and BRI interrupt requests.

Bit 2 - Reserved

This bit is always read as 0, and cannot be modified.

Bits 1 and 0 - Clock enable (CKE0 and CKE1)

These bits select the SCIF clock source and enable/disable clock output from the SCK2 pin. The combination of CKE1 and CKE0 determine whether the SCK2 pin functions as serial clock output pin or the serial clock input pin.

Note, however, that the setting of the CKE0 bit is valid only when CKE1 = 0 (internal clock operation). When CKE1 = 1 (external clock) CKE0 is ignored. These bits must be set before determining the SCIF's operating mode with SCIF.SCSMR2.

Bit 1: CKE1	Bit 0: CKE0	Description
0	0	Internal clock/SCK2 pin functions as input pin (input signal ignored)[a]
0	1	Internal clock/SCK2 pin functions as clock output[b]
1	0	External clock/SCK2 pin functions as clock input[c]
1	1	External clock/SCK2 pin functions as clock input[c]

a. Initial value.

b. Outputs a clock with a frequency 16 times the bit rate.

c. Inputs a clock with a frequency 16 times the bit rate.

9.2.7 Serial status register (SCIF.SCFSR2)

SCIF.SCFSR2 is a 16-bit register. The lower 8 bits consist of status flags that indicate the operating status of the SCIF, and the upper 8 bits indicate the number of receive errors in the data in the receive FIFO register.

SCIF.SCFSR2 can be read or written to by the CPU at all times. However, 1 cannot be written to flags ER, TEND, TDFE, BRK, RDF and DR. Also note that in order to clear these flags they must be read as 1 beforehand. The FER flag and PER flag are read-only flags and cannot be modified.

SCIF.SCFSR2 is initialized to 0x0060 by a power-on reset or manual reset. It is not initialized in standby mode or in the module standby state.

SCIF.SCFSR2				0X10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DR	[0]	1	✓	Receive data ready	RW*
	Operation		Indicates that there are fewer than the receive trigger set number of data bytes in SCIF.SCFRDR2, and no further data has arrived for at least 15 ETU after the stop bit of the last data received. See *Bit 0 - Receive data ready (DR) on page 221*.		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		0		
RDF	[1]	1	✓	Receive FIFO data full	RW*
	Operation		Indicates that the received data has been transferred from SCIF.SCRSR2 to SCIF.SCFRDR2, and the number of receive data bytes in SCIF.SCFRDR2 is equal to or greater than the receive trigger number set by bits RTRG1 and RTRG0 in the FIFO control register (SCIF.SCFCR2). See *Bit 1 - Receive FIFO data full (RDF) on page 220*		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		0		
PER	[2]	1		Parity error	RO
	Operation		Indicates a parity error in the data read from SCIF.SCFRDR2.See *Bit 2 - Parity error (PER) on page 219*		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		0		

Table 89: SCIF.SCFSR2

SCIF.SCFSR2				0X10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FER	[3]	1		Framing error	RO
	Operation		Indicates a framing error in the data read from SCIF.SCFRDR2.See *Bit 3 - Framing error (FER) on page 218*.		
	When read		Returns current value		
	When written		Invalid		
	HARD reset		0		
BRK	[4]	1		Break detect	RW*
	Operation		Indicates that a receive data break signal has been detected.*Bit 4 - Break detect (BRK) on page 218*		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		0		
TDFE	[5]	1		Transmit FIFO data empty	RW*
	Operation		Indicates that data has been transferred from SCIF.SCFTDR2 to SCIF.SCTSR2, the number of data bytes in SCIF.SCFTDR2 has fallen to or below the transmit trigger data number set by bits TTRG1 and TTRG0 in the FIFO control register (SCIF.SCFCR2), and new transmit data can be written to SCIF.SCFTDR2. See *Bit 5 - Transmit FIFO data empty (TDFE) on page 217*		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		1		

Table 89: SCIF.SCFSR2

SCIF.SCFSR2				0X10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TEND	[6]	1		Transmit end	RW*
	Operation		Indicates that there is no valid data in SCIF.SCFTDR2 when the last bit of the transmit character is sent, and transmission has been ended.See *Bit 6 - Transmit end (TEND) on page 216*.		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		1		
ER	[7]	1		Receive error	RW*
	Operation		Indicates that a framing error or parity error occurred during reception.See *Bit 7 - Receive error (ER) on page 215*.		
	When read		Returns current value		
	When written		*Only 0 can be written. This clears the flag		
	HARD reset		0		
FER3 to FER0	[11:8]	4		Number of framing errors	RO
	Operation		These bits indicate the number of data bytes in which a framing error occurred in the receive data stored in SCIF.SCFRDR2.See *Bits 11 to 8 - Number of framing errors (FER3 to FER0) on page 214*		
	When read		Returns current value		
	When written		Invalid		
	HARD reset		0		

Table 89: SCIF.SCFSR2

SCIF.SCFSR2				0X10	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PER3 to PER0	[15:12]	4		Number of parity errors	RO
	Operation		These bits indicate the number of data bytes in which a parity error occurred in the receive data stored in SCIF.SCFRDR2.See *Bits 15 to 12 - Number of parity errors (PER3 to PER0) on page 214*.		
	When read		Returns current value		
	When written		Invalid		
	HARD reset		0		

Table 89: SCIF.SCFSR2

Bits 15 to 12 - Number of parity errors (PER3 to PER0)

These bits indicate the number of data bytes in which a parity error occurred in the receive data stored in SCIF.SCFRDR2. After the ER bit in SCIF.SCFSR2 is set, the value indicated by bits 15 to 12 is the number of data bytes in which a parity error occurred. If all 16 bytes of receive data in SCIF.SCFRDR2 have parity errors, the value indicated by bits PER3 to PER0 will be 0.

Bits 11 to 8 - Number of framing errors (FER3 to FER0)

These bits indicate the number of data bytes in which a framing error occurred in the receive data stored in SCIF.SCFRDR2. After the ER bit in SCIF.SCFSR2 is set, the value indicated by bits 11 to 8 is the number of data bytes in which a framing error occurred. If all 16 bytes of receive data in SCIF.SCFRDR2 have framing errors, the value indicated by bits FER3 to FER0 will be 0.

Bit 7 - Receive error (ER)

Indicates that a framing error or parity error occurred during reception.

Bit 7: ER	Description
0	No framing error or parity error occurred during reception (Initial value) [Clearing conditions] Power-on reset or manual reset[a] When 0 is written to ER after reading ER = 1
1	A framing error or parity error occurred during reception [Setting conditions] When the SCIF checks whether the stop bit at the end of the receive data is 1when reception ends, and the stop bit is 0[b] When, in reception, the number of 1 bits in the receive data plus the parity bit does not match the parity setting (even or odd) specified by the O/E bit in SCIF.SCSMR2

a. The ER flag is not affected and retains its previous state when the RE bit in SCIF.SCSCR2 is cleared to 0. When a receive error occurs, the receive data is still transferred to SCIF.SCFRDR2, and reception continues. The FER and PER bits in SCIF.SCFSR2 can be used to determine whether there is a receive error in the data read from SCIF.SCFRDR2.

b. In 2-stop-bit mode, only the first stop bit is checked for a value of 1; the second stop bit is not checked.

Bit 6 - Transmit end (TEND)

Indicates that there is no valid data in SCIF.SCFTDR2 when the last bit of the transmit character is sent, and transmission has been ended.

Bit 6: TEND	Description
0	Transmission is in progress [Clearing conditions] When transmit data is written to SCIF.SCFTDR2, and 0 is written to TEND after reading TEND = 1 When data is written to SCIF.SCFTDR2 by the DMAC
1	Transmission has been ended (Initial value) [Setting conditions] Power-on reset or manual reset When the TE bit in SCIF.SCSCR2 is 0 When there is no transmit data in SCIF.SCFTDR2 on transmission of the last bit of a 1-byte serial transmit character

Bit 5 - Transmit FIFO data empty (TDFE)

Indicates that data has been transferred from SCIF.SCFTDR2 to SCIF.SCTSR2, the number of data bytes in SCIF.SCFTDR2 has fallen to or below the transmit trigger data number set by bits TTRG1 and TTRG0 in the FIFO control register (SCIF.SCFCR2), and new transmit data can be written to SCIF.SCFTDR2.

Bit 5: TDFE	Description
0	A number of transmit data bytes exceeding the transmit trigger set number have been written to SCIF.SCFTDR2 [Clearing conditions] When transmit data exceeding the transmit trigger set number is written to SCIF.SCFTDR2, and 0 is written to TDFE after reading TDFE = 1 When transmit data exceeding the transmit trigger set number is written to SCIF.SCFTDR2 by the DMAC
1	The number of transmit data bytes in SCIF.SCFTDR2 does not exceed the transmit trigger set number (Initial value) [Setting conditions] Power-on reset or manual reset When the number of SCIF.SCFTDR2 transmit data bytes falls to or below the transmit trigger set number as the result of a transmit operation[a]

a. As SCIF.SCFTDR2 is a 16-byte FIFO register, the maximum number of bytes that can be written when TDFE = 1 is 16 - (transmit trigger set number). Data written in excess of this will be ignored. The number of data bytes in SCIF.SCFTDR2 is indicated by the upper bits of SCIF.SCFDR2.

Bit 4 - Break detect (BRK)

Indicates that a receive data break signal has been detected.

Bit 4: BRK	Description
0	A break signal has not been received (Initial value) [Clearing conditions] Power-on reset or manual reset When 0 is written to BRK after reading BRK = 1
1	A break signal has been received[a] [Setting condition] When data with a framing error is received, followed by the space "0" level (low level) for at least one frame length

a. When a break is detected, the receive data (0x00) following detection is not transferred to SCIF.SCFRDR2. When the break ends and the receive signal returns to mark "1", receive data transfer is resumed.

Bit 3 - Framing error (FER)

Indicates a framing error in the data read from SCIF.SCFRDR2.

Bit 3: FER	Description
0	There is no framing error in the receive data read from SCIF.SCFRDR2 (Initial value) [Clearing conditions] Power-on reset or manual reset When there is no framing error in SCIF.SCFRDR2 read data
1	There is a framing error in the receive data read from SCIF.SCFRDR2 [Setting condition] When there is a framing error in SCIF.SCFRDR2 read data

Bit 2 - Parity error (PER)

Indicates a parity error in the data read from SCIF.SCFRDR2.

Bit 2: PER	Description
0	There is no parity error in the receive data read from SCIF.SCFRDR2 (Initial value) [Clearing conditions] Power-on reset or manual reset When there is no parity error in SCIF.SCFRDR2 read data
1	There is a parity error in the receive data read from SCIF.SCFRDR2 [Setting condition] When there is a parity error in SCIF.SCFRDR2 read data

Bit 1 - Receive FIFO data full (RDF)

Indicates that the received data has been transferred from SCIF.SCRSR2 to SCIF.SCFRDR2, and the number of receive data bytes in SCIF.SCFRDR2 is equal to or greater than the receive trigger number set by bits RTRG1 and RTRG0 in the FIFO control register (SCIF.SCFCR2).

Bit 1: RDF	Description
0	The number of receive data bytes in SCIF.SCFRDR2 is less than the receive trigger set number (Initial value) [Clearing conditions] Power-on reset or manual reset When SCIF.SCFRDR2 is read until the number of receive data bytes in SCIF.SCFRDR2 falls below the receive trigger set number, and 0 is written to RDF after reading RDF = 1 When SCIF.SCFRDR2 is read by the DMAC until the number of receive data bytes in SCIF.SCFRDR2 falls below the receive trigger set number
1	The number of receive data bytes in SCIF.SCFRDR2 is equal to or greater than the receive trigger set number [Setting condition] When SCIF.SCFRDR2 contains at least the receive trigger set number of receive data bytes[a]

a. SCIF.SCFRDR2 is a 16-byte FIFO register. When RDF = 1, at least the receive trigger set number of data bytes can be read. If all the data in SCIF.SCFRDR2 is read and another read is performed, the data value will be undefined. The number of receive data bytes in SCIF.SCFRDR2 is indicated by the lower bits of SCIF.SCFDR2.

Bit 0 - Receive data ready (DR)

Indicates that there are fewer than the receive trigger set number of data bytes in SCIF.SCFRDR2, and no further data has arrived for at least 15 ETU (elementary time unit - time for transfer of 1 bit) after the stop bit of the last data received.

Bit 0: DR	Description
0	Reception is in progress or has ended normally and there is no receive data left in SCIF.SCFRDR2 (Initial value) [Clearing conditions] Power-on reset or manual reset When all the receive data in SCIF.SCFRDR2 has been read, and 0 is written to DR after reading DR = 1 When all the receive data in SCIF.SCFRDR2 has been read by the DMAC
1	No further receive data has arrived [Setting condition] When SCIF.SCFRDR2 contains fewer than the receive trigger set number of receive data bytes, and no further data has arrived for at least 15 ETU after the stop bit of the last data received[a]*

a. Equivalent to 1.5 frames with an 8-bit, 1-stop-bit format.

9.2.8 Bit rate register (SCIF.SCBRR2)

SCIF.SCBRR2 is an 8-bit register that sets the serial transfer bit rate in accordance with the baud rate generator operating clock selected by bits CKS1 and CKS0 in SCIF.SCSMR2.

SCIF.SCBRR2 can be read or written to by the CPU at all times.

SCIF.SCBRR2 is initialized to H'FF by a power-on reset or manual reset. It is not initialized in standby mode or in the module standby state.

SCIF.SCBRR2				0x04	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SCBRR2	[7:0]	8		Bit Rate Register	RW
	Operation		Specifies the serial transfer bit rate in accordance with the baud rate generator operating clock		
	When read		Returns current value		
	When written		Updates current Value		
	HARD reset		0xFF		

Table 90: SCIF.SCBRR2

The SCIF.SCBRR2 setting is found from the following equation.

Asynchronous mode

$$\frac{P_{\phi}}{64 \times 2^{2n-1} \times B} \times 10^{6} - 1 = N$$

Where:

B: Bit rate (bits/s)

N: SCIF.SCBRR2 setting for baud rate generator ($0 \le N \le 255$)

Pϕ**:** Peripheral module operating frequency (MHz)

n: Baud rate generator input clock (n = 0 to 3)

(See *Table 91* for the relation between n and the clock.

n	Clock	SCIF.SCSMR2 Setting	
		CKS1	CKS0
0	Pϕ	0	0
1	Pϕ/4	0	1
2	Pϕ/16	1	0
3	Pϕ/64	1	1

Table 91:

The bit rate error in asynchronous mode is found from the following equation:

$$\mathrm{Error}(\%) = \left\{ \frac{P_{\phi} \times 10^{6}}{(N+1) \times B \times 64 \times 2^{2n-1}} - 1 \right\} \times 100$$

9.2.9 FIFO control register (SCIF.SCFCR2)

SCIF.SCFCR2 performs data count resetting and trigger data number setting for the transmit and receive FIFO registers, and also contains a loopback test enable bit.

SCIF.SCFCR2 can be read or written at any time.

SCIF.SCFCR2 is initialized to 0x0000 by a power-on reset or manual reset. It is not initialized in standby mode or in the module standby state.

SCIF.SCFCR2				0x18	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
LOOP	[0]	1	-	Loopback test	RW
	Operation		Enables loopback testing. See *Bit 0 - Loopback test (LOOP) on page 229*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
RFRS	[1]	1	-	Received FIFO data register reset	RW
	Operation		Enables FIFO reset on a power-on or manual reset. See *Bit 1 - Receive FIFO data register reset (RFRST) on page 228*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
TFRST	[2]	1	-	Transmit FIFO data register reset	RW
	Operation		Enables FIFO reset on a power-on or manual reset. See *Bit 2 - Transmit FIFO data register reset (TFRST) on page 228*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
MCE	[3]	1	-	Modem control enable	RW
	Operation		Enables Modem control signals. See *Bit 3 - Modem control enable (MCE) on page 228*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 92: SCIF.SCFCR2

SCIF.SCFCR2				0x18	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
TTRG1, TTRG0	[5:4]	2	-	Transmit FIFO data number triggers	RW
	Operation		Sets the number of remaining transmit data bytes that sets the transmit FIFO data register empty (TDFE) flag. See *Bits 5 and 4 - Transmit FIFO data number trigger (TTRG1, TTRG0) on page 227*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
RTRG1, RTRG0	[7:6]	2	-	Receive FIFO data number triggers	RW
	Operation		Sets the number of receive data bytes that sets the receive data full (RDF) flag. See *Bits 7 and 6 - Receive FIFO data number trigger (RTRG1, RTRG0) on page 227*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
RSTRG2, RSTRG1, RSTRG0	[10:8]	3	-	RTS2 output active trigger	RO
	Operation		Sets the number of receive data bytes that sets the $\overline{\text{RTS2}}$ signal active. See *Bits 10, 9 and 8 - RTS2 output active trigger (RSTRG2, RSTRG1 and RSTRG0) on page 226*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 92: SCIF.SCFCR2

SCIF.SCFCR2				0x18	
Field	Bits	Size	Volatile?	Synopsis	Type
	[15:11]	5		Reserved	RES
	Operation		RESERVED.		
	When read		0		
	When written		Should only be written with 0		
	HARD reset		0		

Table 92: SCIF.SCFCR2

Bits 15 to 11 - Reserved

These bits are always read as 0, and should only be written with 0.

Bits 10, 9 and 8 - RTS2 output active trigger (RSTRG2, RSTRG1 and RSTRG0)

These bits set the NOT_RTS2 signal active when the number of received data stored in the receive FIFO data register (SCFRDR2) exceeds the trigger number, as shown in the table below:

Bit 10: RSTRG2	Bit 9: RSTRG1	Bit 8: RSTRG0	RTS2 Output Active Trigger
0	0	0	15 (Initial value)
0	0	1	1
0	1	0	4
0	1	1	6
1	0	0	8
1	0	1	10
1	1	0	12
1	1	1	14

Bits 7 and 6 - Receive FIFO data number trigger (RTRG1, RTRG0)

These bits are used to set the number of receive data bytes that sets the receive data full (RDF) flag in the serial status register (SCIF.SCFSR2).

The RDF flag is set when the number of receive data bytes in SCIF.SCFRDR2 is equal to or greater than the trigger set number shown in the following table.

Bit 7: RTRG1	Bit 6: RTRG0	Receive trigger number
0	0	1[a]
	1	4
1	0	8
	1	14

a. Initial value.

Bits 5 and 4 - Transmit FIFO data number trigger (TTRG1, TTRG0)

These bits are used to set the number of remaining transmit data bytes that sets the transmit FIFO data register empty (TDFE) flag in the serial status register (SCIF.SCFSR2). The TDFE flag is set when the number of transmit data bytes in SCIF.SCFTDR2 is equal to or less than the trigger set number shown in the following table.

Bit 5: TTRG1	Bit 4: TTRG0	Transmit trigger number
0	0	8 (8)[a]
	1	4 (12)
1	0	2 (14)
	1	1 (15)

a. Initial value. Figures in parentheses are the number of empty bytes in SCIF.SCFTDR2 when the flag is set.

Bit 3 - Modem control enable (MCE)

Enables the CTS2 and RTS2 modem control signals.

Bit 3: MCE	Description	
0	Modem signals disabled[a]	(Initial value)
1	Modem signals enabled	

a. CTS2 is fixed at active-0 regardless of the input value, and RTS2 output is also fixed at 0.

Bit 2 - Transmit FIFO data register reset (TFRST)

Invalidates the transmit data in the transmit FIFO data register and resets it to the empty state.

Bit 2: TFRST	Description	
0	Reset operation disabled[a]	(Initial value)
1	Reset operation enabled	

a. A reset operation is performed in the event of a power-on reset or manual reset.

Bit 1 - Receive FIFO data register reset (RFRST)

Invalidates the receive data in the receive FIFO data register and resets it to the empty state.

Bit 1: RFRST	Description	
0	Reset operation disabled[a]	(Initial value)
1	Reset operation enabled	

a. A reset operation is performed in the event of a power-on reset or manual reset.

Bit 0 - Loopback test (LOOP)

Internally connects the transmit output pin (TXD2) and receive input pin (RXD2), and the RTS2 pin and CTS2 pin, enabling loopback testing.

Bit 0: LOOP	Description	
0	Loopback test disabled	(Initial value)
1	Loopback test enabled	

9.2.10 FIFO data count register (SCIF.SCFDR2)

SCIF.SCFDR2 is a 16-bit register that indicates the number of data bytes stored in SCIF.SCFTDR2 and SCIF.SCFRDR2.

The upper bits show the number of transmit data bytes in SCIF.SCFTDR2, and the lower bits show the number of receive data bytes in SCIF.SCFRDR2.

SCIF.SCFDR2 can be read by the CPU at all times.

SCIF.SCFDR2				0x1C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
R4 to R0	[4:0]	5	✓	Received data count	RO
	Operation		These bits show the number of receive data bytes in SCIF.SCFRDR2		
	When read		Returns the current count. A value of 0x00 indicates that there is no receive data, and a value of 0x10 indicates that SCIF.SCFRDR2 is full of receive data.		
	When written		Invalid		
	HARD reset		0x00		

Table 93: SCIF.SCFDR2

SCIF.SCFDR2				0x1C	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
	[7:5]	3		Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Should only be written with 0		
	HARD reset		0		
T4 to T0	[12:8]	5	✓	Transmitted data count	RO
	Operation		These bits show the number of untransmitted data bytes in SCIF.SCFTDR2.		
	When read		Returns the current count. A value of 0x00 indicates that there is no transmit data, and a value of 0x10 indicates that SCIF.SCFTDR2 is full of transmit data		
	When written		Invalid		
	HARD reset		0x00		
RESERVED	[15:13]	3		Reserved	RES
	Operation		Reserved		
	When read		0		
	When written		Should only be written with 0		
	HARD reset		0		

Table 93: SCIF.SCFDR2

9.2.11 Serial port register (SCIF.SCSPTR2)

SCIF.SCSPTR2 is a 16-bit readable/writable register that controls input/output and data for the port pins multiplexed with the serial communication interface (SCIF) pins. Input data can be read from the RXD2 pin, output data written to the TXD2 pin, and breaks in serial transmission/reception controlled, by means of bits 1 and 0. Data can be read from, and output data written to, the CTS2 pin by means of bits 5 and 4. Data can be read from, and output data written to, the RTS2 pin by means of bits 6 and 7.

SCIF.SCSPTR2 can be read or written to at any time.

All SCIF.SCSPTR2 bits except bits 6, 4, and 0 are initialized to 0 by a power-on reset or manual reset; the value of bits 6, 4, and 0 is undefined. SCIF.SCSPTR2 is not initialized in standby mode or in the module standby state.

SCIF.SCSPTR2				0x20	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SPB2DT	[0]	1	-	Serial port break data	RW
	Operation		Specifies the serial port RXD2 pin input data and TxD2 pin output data. See *Bit 0 - Serial port break data (SPB2DT) on page 236*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
SPB2IO	[1]	1	-	Serial port break I/O	RW
	Operation		Specifies the serial port TXD2 pin output condition. See *Bit 1 - Serial port break I/O (SPB2IO) on page 236*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 94: SCIF.SCSPTR2

SCIF.SCSPTR2				0x20	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SCKDT	[2]	1	-	Serial port clock port data (SCKDT)	RW
	Operation		Specifies the I/O data for the SCK2 pin serial port. See *Bit 2 - Serial port clock port data (SCKDT) on page 235*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
SCKIO	[3]	1		Serial port clock port I/O	
	Operation		Sets the I/O for the SCK2 pin serial port. See *Bit 3 - Serial port clock port data (SCKIO) on page 235*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
CTSDT	[4]		-	Serial port CTS port data	RW
	Operation		Specifies the serial port CTS2 pin input/output data. See *Bit 4 - Serial port CTS port data (CTSDT) on page 235*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
CTSIO	[5]	1	-	Serial port CTS port I/O	RW
	Operation		Specifies the serial port CTS2 pin input/output condition.See *Bit 5 - Serial port CTS port I/O (CTSIO) on page 234*		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		

Table 94: SCIF.SCSPTR2

SCIF.SCSPTR2				0x20	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RTSDT	[6]		-	Serial port RTS port data	RW
	Operation		Specifies the serial port RTS2 pin input/output data.See *Bit 6 - Serial port RTS port data (RTSDT) on page 234*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		Undefined		
RTSIO	[7]	1	-	Serial port RTS port I/O	RW
	Operation		Specifies the serial port RTS2 pin input/output condition. See *Bit 7 - Serial port RTS port I/O (RTSIO) on page 234*.		
	When read		Returns current value		
	When written		Updates current value		
	HARD reset		0		
	[15:8]	8		Reserved	RES
	Operation		Reserved		
	When read		0x00		
	When written		Should only be written with 0		
	HARD reset		0		

Table 94: SCIF.SCSPTR2

Bits 15 to 8 - Reserved

These bits are always read as 0, and should only be written with 0.

Bit 7 - Serial port RTS port I/O (RTSIO)

Specifies the serial port RTS2 pin input/output condition. When the RTS2 pin is actually set as a port output pin and outputs the value set by the RTSDT bit, the MCE bit in SCIF.SCFCR2 should be cleared to 0.

Bit 7: RTSIO	Description
0	RTSDT bit value is not output to RTS2 pin (Initial value)
1	RTSDT bit value is output to RTS2 pin

Bit 6 - Serial port RTS port data (RTSDT)

Specifies the serial port RTS2 pin input/output data. Input or output is specified by the RTSIO bit (see *Bit 7 - Serial port RTS port I/O (RTSIO)* for details). In output mode, the RTSDT bit value is output to the RTS2 pin. The RTS2 pin value is read from the RTSDT bit regardless of the value of the RTSIO bit. The initial value of this bit after a power-on reset or manual reset is undefined.

Bit 6: RTSDT	Description
0	Input/output data is low-level
1	Input/output data is high-level

Bit 5 - Serial port CTS port I/O (CTSIO)

Specifies the serial port CTS2 pin input/output condition. When the CTS2 pin is actually set as a port output pin and outputs the value set by the CTSDT bit, the MCE bit in SCIF.SCFCR2 should be cleared to 0.

Bit 5: CTSIO	Description
0	CTSDT bit value is not output to CTS2 pin/ (Initial value)
1	CTSDT bit value is output to CTS2 pin

Bit 4 - Serial port CTS port data (CTSDT)

Specifies the serial port CTS2 pin input/output data. Input or output is specified by the CTSIO bit (see *Bit 5 - Serial port CTS port I/O (CTSIO)* for details). In output mode, the CTSDT bit value is output to the CTS2 pin. The CTS2 pin value is read from the CTSDT bit regardless of the value of the CTSIO bit. The initial value of this bit after a power-on reset or manual reset is undefined.

Bit 4: CTSDT	Description
0	Input/output data is low-level
1	Input/output data is high-level

Bit 3 - Serial port clock port data (SCKIO)

Sets the I/O for the SCK2 pin serial port. To actually set the SCK2 pin as the port output pin and output the value set in the SCKDT bit, set the CKE1 and CKE0 bits of the SCSCR2 register to 0.

Bit 3: SCKIO	Description
0	Shows that the value of the SCKDT bit is not output to the SCK2 pin (Initial value)
1	Shows that the value of the SCKDT bit is output to the SCK2 pin

Bit 2 - Serial port clock port data (SCKDT)

Specifies the I/O data for the SCK2 pin serial port. The SCKIO bit specifies input or output (see *Bit 3 - Serial port clock port data (SCKIO)* for details). When set for output, the value of the SCKDT bit is output to the SCK2 pin. Regardless of the value of the SCKIO bit, the value of the SCK2 pin is fetched from the SCKDT bit. The initial value after a power-on reset or manual reset is undefined.

Bit 2: SCKDT	Description
0	Shows I/O data level is LOW
1	Shows I/O level data is HIGH

Bit 1 - Serial port break I/O (SPB2IO)

Specifies the serial port TXD2 pin output condition. When the TXD2 pin is actually set as a port output pin and outputs the value set by the SPB2DT bit, the TE bit in SCIF.SCSCR2 should be cleared to 0.

Bit 1: SPB2IO	Description
0	SPB2DT bit value is not output to the TXD2 pin (Initial value)
1	SPB2DT bit value is output to the TXD2 pin

Bit 0 - Serial port break data (SPB2DT)

Specifies the serial port RXD2 pin input data and TXD2 pin output data. The TXD2 pin output condition is specified by the SPB2IO bit (see *Bit 1 - Serial port break I/O (SPB2IO)* for details). When the TXD2 pin is designated as an output, the value of the SPB2DT bit is output to the TXD2 pin. The RXD2 pin value is read from the SPB2DT bit regardless of the value of the SPB2IO bit. The initial value of this bit after a power-on reset or manual reset is undefined.

Bit 0: SPB2DT	Description
0	Input/output data is low-level
1	Input/output data is high-level

SCIF I/O port block diagrams are shown *Figure 27* in to *Figure 30*.

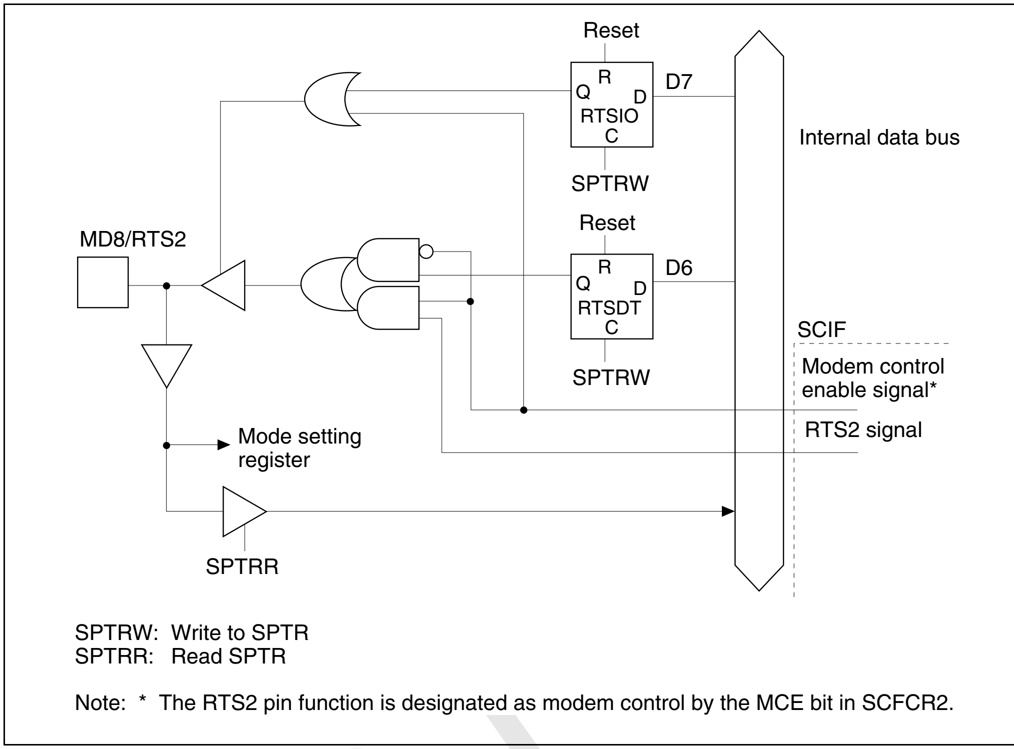


Figure 27: MD8/RTS2 pin

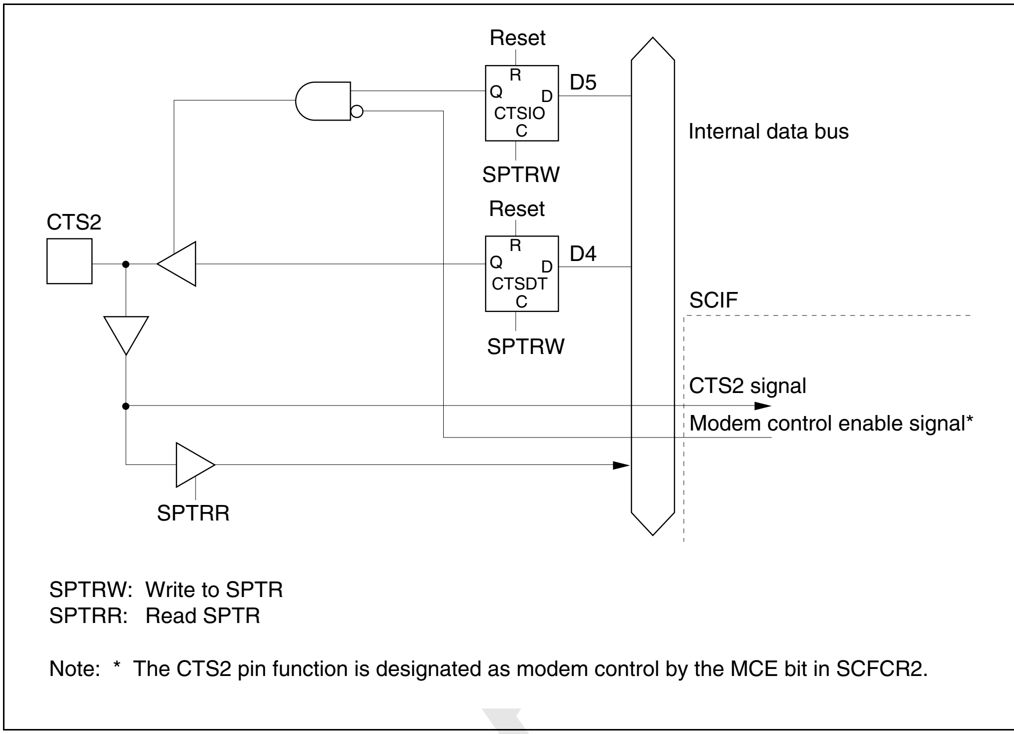


Figure 28: CTS2 pin

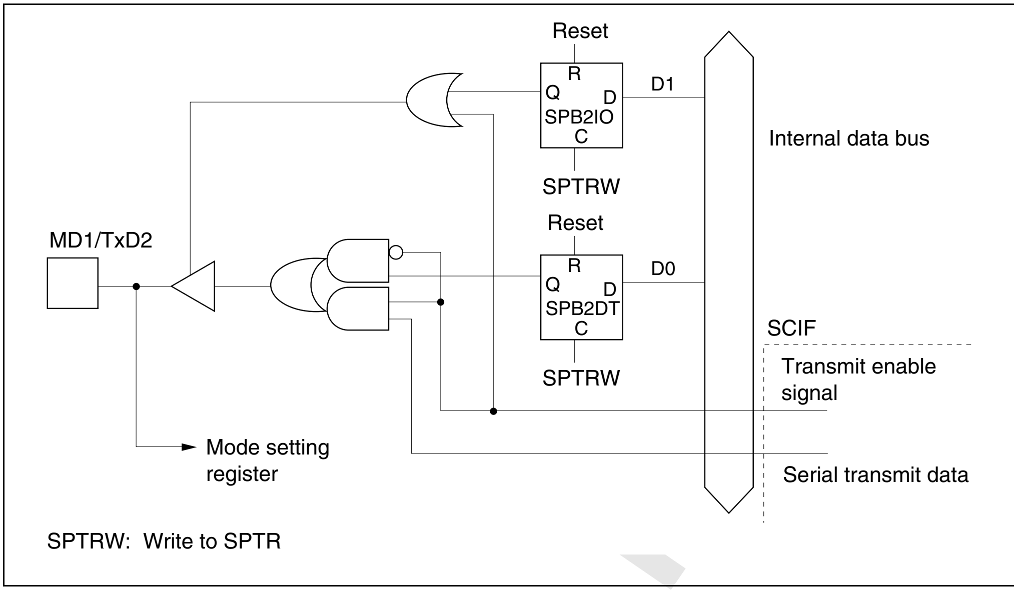


Figure 29: MD1/TxD2 pin

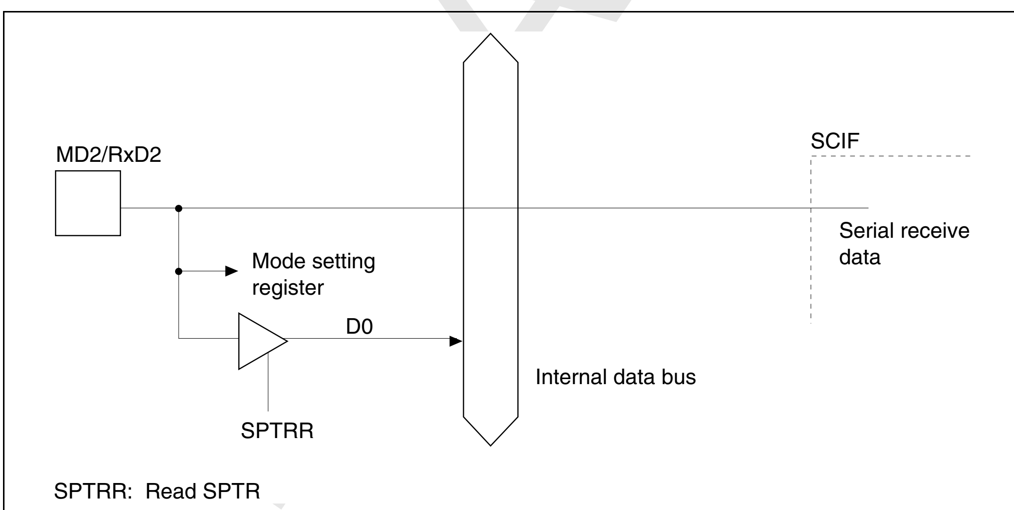


Figure 30: MD2/RxD2 pin

9.2.12 Line status register (SCIF.SCLSR2)

SCIF.SCLSR2 is a 16-bit register which contains the overrun error flag.

SCIF.SCLSR2				0X24	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
ORER	[0]	1	✓	Overrun error	RW*
	Operation	Indicates that an overrun error occurred. See *Bit 0 - Overrun error (ORER) on page 241*.			
	When read	Returns the current value			
	When written	*Only 0 can be written, to clear the flag.			
	HARD reset	0			
	[15:1]	15		Reserved	RES
	Operation	Reserved			
	When read	0			
	When written	Should only be written with 0			
	HARD reset	0			

Table 95: SCIF.SCLSR2

Bits 15 to 1 - Reserved

These bits are always read as 0, and should only be written with 0.

Bit 0 - Overrun error (ORER)

Indicates that an overrun error occurred during reception, causing abnormal termination.

Bit 0: ORER	Description
0	Reception in progress, or reception has ended normally[a] (Initial value) [Clearing conditions] Power-on reset or manual reset When 0 is written to ORER after reading ORER = 1
1	An overrun error occurred during reception[b] [Setting condition] When the next serial reception is completed while the receive FIFO is full

a. The ORER flag is not affected and retains its previous state when the RE bit in SCIF.SCSCR2 is cleared to 0.

b. The receive data prior to the overrun error is retained in SCIF.SCFRDR2, and the data received subsequently is lost. Serial reception cannot be continued while the ORER flag is set to 1.

9.3 Operation

9.3.1 Overview

The SCIF can carry out serial communication in asynchronous mode, in which synchronization is achieved character by character.

Sixteen stage FIFO buffers are provided for both transmission and reception, reducing the CPU overhead and enabling fast, continuous communication to be performed. RTS2 and CTS2 signals are also provided as modem control signals. The transmission format is selected using the serial mode register (SCIF.SCSMR2), as shown in *Table 96*. The SCIF clock source is determined by the CKE1 bit in the serial control register (SCIF.SCSCR2), as shown in *Table 97*.

- Data length: Choice of 7 or 8 bits.
- Choice of parity addition and addition of 1 or 2 stop bits (the combination of these parameters determines the transfer format and character length).
- Detection of framing errors, parity errors, receive-FIFO-data-full state, overrun errors, receive-data-ready state, and breaks, during reception.
- Indication of the number of data bytes stored in the transmit and receive FIFO registers.
- Choice of internal or external clock as SCIF clock source.

  When internal clock is selected: The SCIF operates on the baud rate generator clock.

  When external clock is selected: A clock with a frequency of 16 times the bit rate must be input (the on-chip baud rate generator is not used).

SCIF.SCSMR2 settings			**Mode**	**SCIF transfer format**			
Bit 6: CHR	**Bit 5: PE**	**Bit 3: STOP**		**Data length**	**Multiprocessor bit**	**Parity bit**	**Stop bit length**
0	0	0	Asynchronous mode	8-bit data	No	No	1-bit
		1					2 bits
	1	0				Yes	1 bit
		1					2 bits
1	0	0		7-bit data		No	1 bit
		1					2 bits
	1	0				Yes	1 bit
		1					2 bits

Table 96: SCIF.SCSMR2 settings for serial transfer format selection

SCIF.SCS CR2 setting		Mode	Clock source	SCK2 pin function
Bit 1:CKE1	**Bit 0:CKE0**			
0	0	Asynchronous mode	Internal	SCIF does not use SCK2 pin
0	1	Asynchronous mode	Internal	Output clock with frequency of 16 times the bit rate.
1	0	Asynchronous mode	External	Inputs clock with frequency of 16 times the bit rate
1	1	Asynchronous mode	External	Inputs clock with frequency of 16 times the bit rate

Table 97: SCIF.SCSCR2 settings for SCIF clock source selection

9.3.2 Serial operation

Data transfer format

Table 98 shows the data transfer formats that can be used. Any of the eight transfer formats can be selected according to the SCIF.SCSMR2 settings.

SCIF.SCSMR2 settings			Serial transfer format and frame length											
CHR	**PE**	**STOP**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	0	0	S	8-bit data								STOP		
0	0	1	S	8-bit data								STOP	STOP	

Table 98: Serial transfer formats

SCIF.SCSMR2 settings			Serial transfer format and frame length											
CHR	PE	STOP	1	2	3	4	5	6	7	8	9	10	11	12
0	1	0	S	8-bit data								P	STOP	
0	1	1	S	8-bit data								P	STOP	STOP
1	0	0	S	7-bit data							STOP			
1	0	1	S	7-bit data							STOP	STOP		
1	1	0	S	7-bit data							P	STOP		
1	1	1	S	7-bit data							P	STOP	STOP	

Table 98: Serial transfer formats

S: Start bit, STOP: Stop bit, P: Parity bit

Clock

Either an internal clock generated by the on-chip baud rate generator or an external clock input at the SCK2 pin can be selected as the SCIF's serial clock, according to the setting of the CKE1 bit in SCIF.SCSCR2. For details of SCIF clock source selection, see *Table 97*.

When an external clock is input at the SCK2 pin, the clock frequency should be 16 times the bit rate used.

Data transfer operations

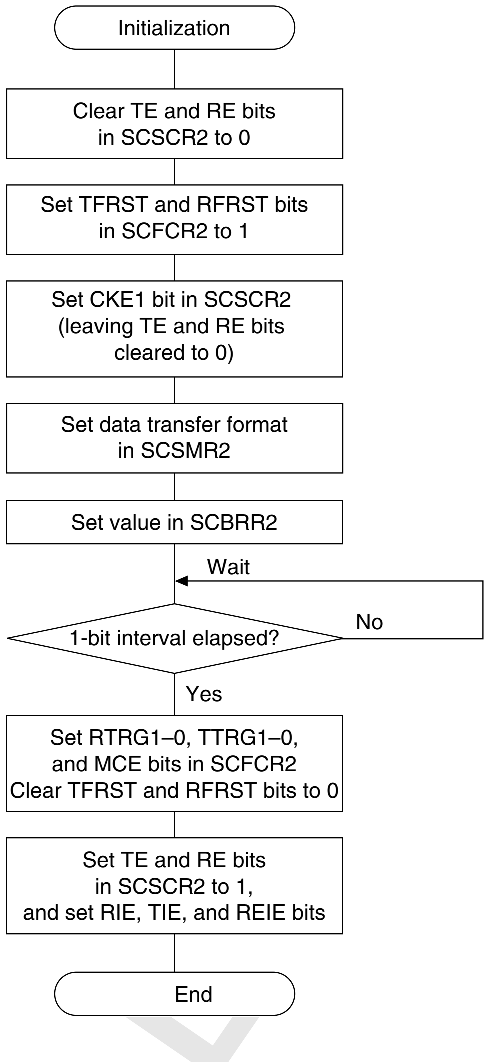
SCIF Initialization:

Before transmitting and receiving data, it is necessary to clear the TE and RE bits in SCIF.SCSCR2 to 0, then initialize the SCIF as described below.

When the transfer format is changed, the TE and RE bits must be cleared to 0 before making the change using the following procedure. When the TE bit is cleared to 0, SCIF.SCTSR2 is initialized. Clearing the TE and RE bits to 0 does not change the contents of SCIF.SCFSR2, SCIF.SCFTDR2, or SCIF.SCFRDR2. The TE bit should be cleared to 0 after all transmit data has been sent and the TEND flag in SCIF.SCFSR2 has been set. TEND can also be cleared to 0 during transmission, but the data being transmitted will go to the mark state after the clearance. Before setting TE again to start transmission, the TFRST bit in SCIF.SCFCR2 should first be set to 1 to reset SCIF.SCFTDR2.

When an external clock is used the clock should not be stopped during operation, including initialization, since operation will be unreliable in this case.

Figure 31 shows a sample SCIF initialization flowchart.



1. Set the clock selection in SCSCR2.

   Be sure to clear bits RIE and TIE, and bits TE and RE, to 0.
2. Set the data transfer format in SCSMR2.
3. Write a value corresponding to the bit rate into SCBRR2. (Not necessary if an external clock is used.)
4. Wait at least one bit interval, then set the TE bit or RE bit in SCSCR2 to 1. Also set the RIE, REIE, and TIE bits.

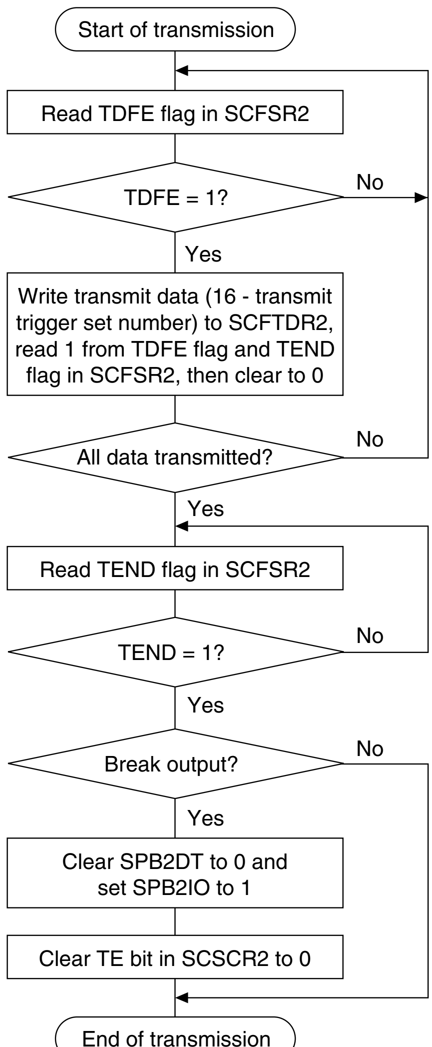
   Setting the TE and RE bits enables the TxD2 and RxD2 pins to be used. When transmitting, the SCIF will go to the mark state; when receiving, it will go to the idle state, waiting for a start bit.

Figure 31: Sample SCIF initialization flowchart

Serial Data Transmission:

Figure 32 shows a sample flowchart for serial transmission.

Use the following procedure for serial data transmission after enabling the SCIF for transmission.



1. SCIF status check and transmit data write:

   Read SCFSR2 and check that the TDFE flag is set to 1, then write transmit data to SCFTDR2, read 1 from the TDFE and TEND flags, then clear these flags to 0.

   The number of transmit data bytes that can be written is 16 - (transmit trigger set number).

2. Serial transmission continuation procedure:

   To continue serial transmission, read 1 from the TDFE flag to confirm that writing is possible, then write data to SCFTDR2, and then clear the TDFE flag to 0.

3. Break output at the end of serial transmission:

   To output a break in serial transmission, clear the SPB2DT bit to 0 and set the SPB2IO bit to 1 in SCSPTR2, then clear the TE bit in SCSCR2 to 0.

   In steps 1 and 2, it is possible to ascertain the number of data bytes that can be written from the number of transmit data bytes in SCFTDR2 indicated by the upper 8 bits of SCFDR2.

Figure 32: Sample serial transmission flowchart

In serial transmission, the SCIF operates as described below.

1 When data is written into SCIF.SCFTDR2, the SCIF transfers the data from SCIF.SCFTDR2 to SCIF.SCTSR2 and starts transmitting. Confirm that the TDFE flag in the serial status register (SCIF.SCFSR2) is set to 1 before writing transmit data to SCIF.SCFTDR2. The number of data bytes that can be written is at least 16 (transmit trigger setting).

2 When data is transferred from SCIF.SCFTDR2 to SCIF.SCTSR2 and transmission is started, consecutive transmit operations are performed until there is no transmit data left in SCIF.SCFTDR2. When the number of transmit data bytes in SCIF.SCFTDR2 falls to or below the transmit trigger number set in the FIFO control register (SCIF.SCFCR2), the TDFE flag is set. If the TIE bit in SCIF.SCSCR2 is set to 1 at this time, a transmit-FIFO-data-empty interrupt (TXI) request is generated. The serial transmit data is sent from the TXD2 pin in the following order.

2.1 Start bit: One 0-bit is output.

2.2 Transmit data: 8-bit or 7-bit data is output in LSB-first order.

2.3 Parity bit: One parity bit (even or odd parity) is output. (A format in which a parity bit is not output can also be selected.)

2.4 Stop bit(s): One or two 1-bits (stop bits) are output.

2.5 Mark state: 1 is output continuously until the start bit that starts the next transmission is sent.

3 The SCIF checks the SCIF.SCFTDR2 transmit data at the timing for sending the stop bit. If data is present, the data is transferred from SCIF.SCFTDR2 to SCIF.SCTSR2, the stop bit is sent, and then serial transmission of the next frame is started. If there is no transmit data, the TEND flag in SCIF.SCFSR2 is set to 1, the stop bit is sent, and then the line goes to the mark state in which 1 is output.

Figure 33 shows an example of the operation for transmission in asynchronous mode.

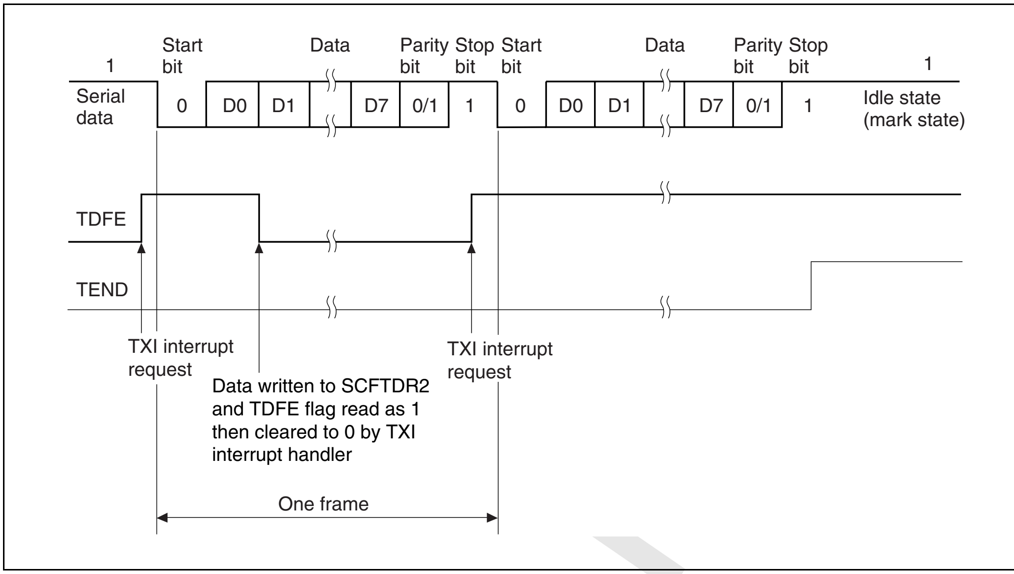


Figure 33: Example of transmit operation (Example with 8-Bit data, parity, one stop bit) 4

When modem control is enabled, transmission can be stopped and restarted in accordance with the CTS2 input value. When CTS2 is set to 1, if transmission is in progress, the line goes to the mark state after transmission of one frame. When CTS2 is set to 0, the next transmit data is output starting from the start bit. *Figure 34* shows an example of the operation when modem control is used.

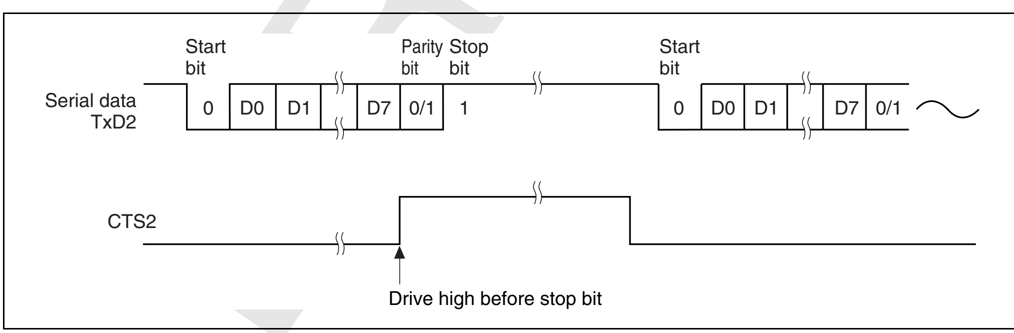
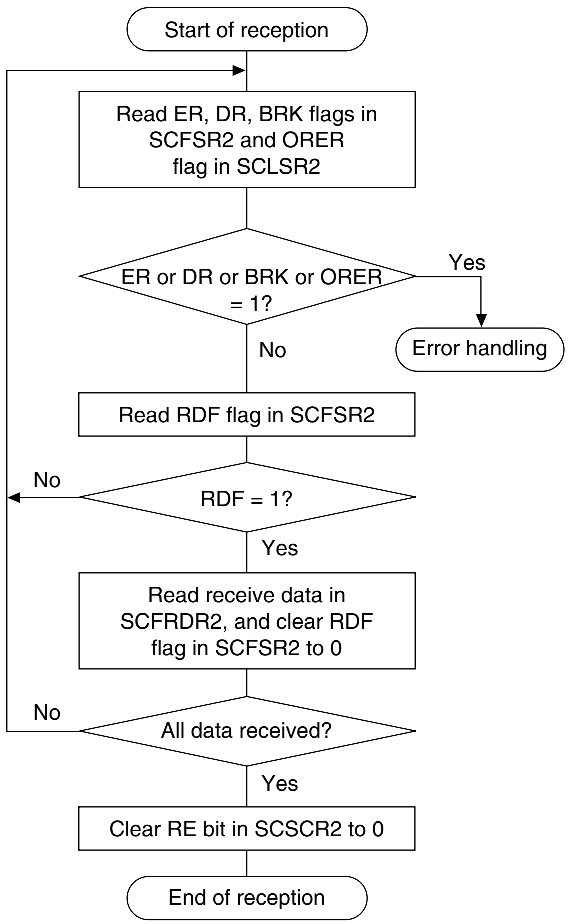


Figure 34: Example of operation using modem control (CTS2)

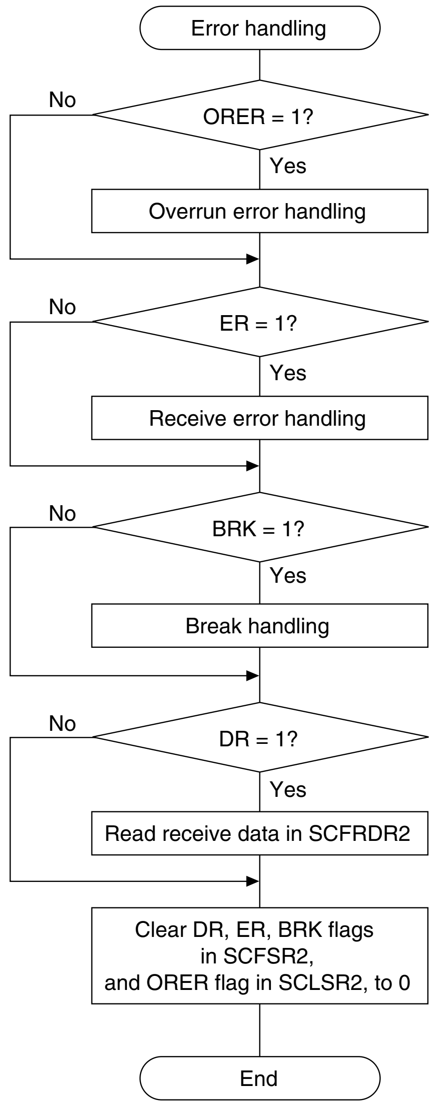
Serial Data Reception:

Figure 35 and *Figure 36* show a sample flowchart for serial reception. Use the following procedure for serial data reception after enabling the SCIF for reception.



1. Receive error handling and break detection: Read the DR, ER, and BRK flags in SCFSR2, and the ORER flag in SCLSR2, to identify any error, perform the appropriate error handling, then clear the DR, ER, BRK, and ORER flags to 0. In the case of a framing error, a break can also be detected by reading the value of the RxD2 pin.
2. SCIF status check and receive data read : Read SCFSR2 and check that RDF = 1, then read the receive data in SCFRDR2, read 1 from the RDF flag, and then clear the RDF flag to 0. The transition of the RDF flag from 0 to 1 can also be identified by an RXI interrupt.
3. Serial reception continuation procedure: To continue serial reception, read at least the receive trigger set number of receive data bytes from SCFRDR2, read 1 from the RDF flag, then clear the RDF flag to 0. The number of receive data bytes in SCFRDR2 can be ascertained by reading the lower bits of SCFDR2.

Figure 35: Sample serial reception flowchart (1)



1. Whether a framing error or parity error has occurred in the receive data read from SCFRDR2 can be ascertained from the FER and PER bits in SCFSR2.
2. When a break signal is received, receive data is not transferred to SCFRDR2 while the BRK flag is set. However, note that the last data in SCFRDR2 is 0x00 (the break data in which a framing error occurred is stored).

Figure 36: Sample serial reception flowchart (2)

In serial reception, the SCIF operates as described below.

1 The SCIF monitors the transmission line, and if a 0 start bit is detected, performs internal synchronization and starts reception.

2 The received data is stored in SCIF.SCRSR2 in LSB-to-MSB order.

3 The parity bit and stop bit are received. After receiving these bits, the SCIF carries out the following checks.

3.1 Stop bit check: The SCIF checks whether the stop bit is 1. If there are two stop bits, only the first is checked.

3.2 The SCIF checks whether receive data can be transferred from the receive shift register (SCIF.SCRSR2) to SCIF.SCFRDR2.

3.3 Overrun error check: The SCIF checks that the ORER flag is 0, indicating that no overrun error has occurred.

3.4 Break check: The SCIF checks that the BRK flag is 0, indicating that the break state is not set. If all the above checks are passed, the receive data is stored in SCIF.SCFRDR2.

Note: Reception continues when a receive error occurs.

4 If the RIE bit in SCIF.SCSCR2 is set to 1 when the RDF or DR flag changes to 1, a receive-FIFO-data-full interrupt (RXI) request is generated. If the RIE bit or REIE bit in SCIF.SCSCR2 is set to 1 when the ER flag changes to 1, a receive-error interrupt (ERI) request is generated. If the RIE bit or REIE bit in SCIF.SCSCR2 is set to 1 when the BRK or ORER flag changes to 1, a break reception interrupt (BRI) request is generated.

Figure 37 shows an example of the operation for reception.

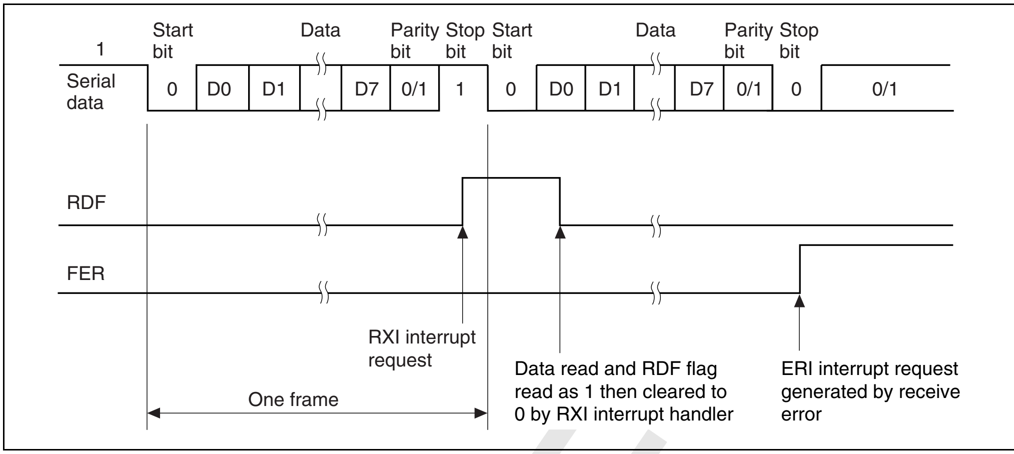


Figure 37: Example of SCIF receive operation (example with 8-Bit data, parity, one stop bit)

5 When modem control is enabled, the RTS2 signal is output when SCIF.SCFRDR2 is empty. When RTS2 is 0, reception is possible. When RTS2 is 1, this indicates that SCIF.SCFRDR2 contains 15 or more bytes of data, and there is no free space, reception is not possible. *Figure 38* shows an example of the operation when modem control is used.

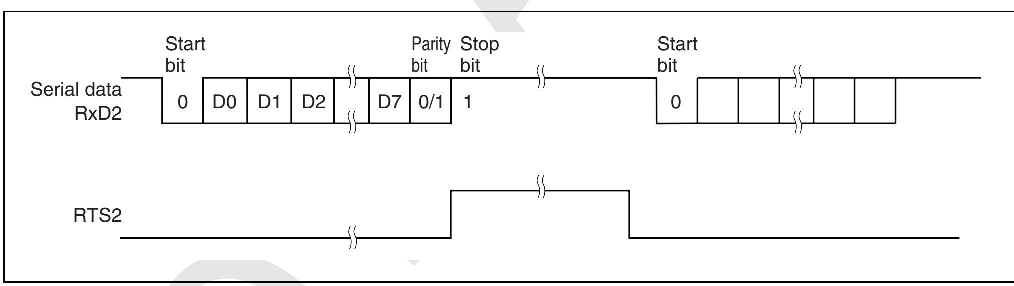


Figure 38: Example of operation using modem control (RTS2)

9.4 SCIF interrupt sources and the DMAC

The SCIF has four interrupt sources: transmit-FIFO-data-empty interrupt (TXI) request, receive-error interrupt (ERI) request, receive-FIFO-data-full interrupt (RXI) request, and break interrupt (BRI) request.

Table 99 shows the interrupt sources and their order of priority. The interrupt sources are enabled or disabled by means of the TIE, RIE, and REIE bits in SCIF.SCSCR2. A separate interrupt request is sent to the interrupt controller for each of these interrupt sources.

When transmission/reception is carried out using the DMAC, output of interrupt requests to the interrupt controller can be inhibited by clearing the RIE bit in SCIF.SCSCR2 to 0. By setting the REIE bit to 1 while the RIE bit is cleared to 0, it is possible to output ERI and BRI interrupt requests, but not RXI interrupt requests.

When the TDFE flag in the serial status register (SCIF.SCFSR2) is set to 1, a transmit-FIFO-data-empty request is generated separately from the interrupt request. A transmit-FIFO-data-empty request can activate the DMAC to perform data transfer.

When the RDF flag or DR flag in SCIF.SCFSR2 is set to 1, a receive-FIFO-data-full request is generated separately from the interrupt request. A receive-FIFO-data-full request can activate the DMAC to perform data transfer.

When using the DMAC for transmission/reception, set and enable the DMAC before making the SCIF settings. See the specification of the DMA controller module for details of the DMAC setting procedure.

When the BRK flag in SCIF.SCFSR2 or the ORER flag in the line status register (SCIF.SCLSR2) is set to 1, a BRI interrupt request is generated. The TXI interrupt indicates that transmit data can be written, and the RXI interrupt indicates that there is receive data in SCIF.SCFRDR2.

Interrupt source	Description	DMAC activation	Priority on reset release
ERI	Interrupt initiated by receive error flag (ER)	Not possible	High
RXI	Interrupt initiated by receive FIFO data full flag (RDF) or receive data ready flag (DR)	Possible	¦
BRI	Interrupt initiated by break flag (BRK) or overrun error flag (ORER)	Not possible	Ø
TXI	Interrupt initiated by transmit FIFO data empty flag (TDFE)	Possible	Low

Table 99: SCIF interrupt sources

See the chapter *Exceptions* in the *CPU Architecture* manual, for priorities and the relationship with nonSCIF interrupts.

9.5 Power down

The SCIF module may be put into a power down state either individually or by putting the chip into standby. See *Chapter 10: Clock, power and reset controller on page 259* for details.

In order to guarantee safe transition to a power down state software should first deactivate the SCIF. This will ensure that the state of the SCIF is architecturally defined.

The SCIF module can be deactivated by clearing the SCIF.SCSCR2.TE and SCIF.SCSCR2.RE flags to ‘0’.

Following exit from the power down state, software can re-enable the SCIF module operating by restoring the previous state of the SCIF.SCSCR2.TE and SCIF.SCSCR2.RE flags.

9.6 Usage notes

Note the following when using the SCIF.

SCIF.SCFTDR2 writing and the TDFE flag

The TDFE flag in the serial status register (SCIF.SCFSR2) is set when the number of transmit data bytes written in the transmit FIFO data register (SCIF.SCFTDR2) has fallen to or below the transmit trigger number set by bits TTRG1 and TTRG0 in the FIFO control register (SCIF.SCFCR2). After TDFE is set, transmit data up to the number of empty bytes in SCIF.SCFTDR2 can be written, allowing efficient continuous transmission.

However, if the number of data bytes written in SCIF.SCFTDR2 is equal to or less than the transmit trigger number, the TDFE flag will be set to 1 again after being read as 1 and cleared to 0. TDFE clearing should therefore be carried out when SCIF.SCFTDR2 contains more than the transmit trigger number of transmit data bytes.

The number of transmit data bytes in SCIF.SCFTDR2 can be found from the upper 8 bits of the FIFO data count register (SCIF.SCFDR2).

SCIF.SCFRDR2 reading and the RDF flag

The RDF flag in the serial status register (SCIF.SCFSR2) is set when the number of receive data bytes in the receive FIFO data register (SCIF.SCFRDR2) has become equal to or greater than the receive trigger number set by bits RTRG1 and RTRG0 in the FIFO control register (SCIF.SCFCR2). After RDF is set, receive data equivalent to the trigger number can be read from SCIF.SCFRDR2, allowing efficient continuous reception.

However, if the number of data bytes in SCIF.SCFRDR2 is equal to or greater than the trigger number, the RDF flag will be set to 1 again if it is cleared to 0. RDF should therefore be cleared to 0 after being read as 1 after all the receive data has been read.

The number of receive data bytes in SCIF.SCFRDR2 can be found from the lower 8 bits of the FIFO data count register (SCIF.SCFDR2).

Break detection and processing

Break signals can be detected by reading the RXD2 pin directly when a framing error (FER) is detected. In the break state the input from the RXD2 pin consists of all 0s, so the FER flag is set and the parity error flag (PER) may also be set. Although the SCIF stops transferring receive data to SCIF.SCFRDR2 after receiving a break, the receive operation continues.

Sending a break signal

The input/output condition and level of the TXD2 pin are determined by bits SPB2IO and SPB2DT in the serial port register (SCIF.SCSPTR2). This feature can be used to send a break signal.

After the serial transmitter is initialized, the TXD2 pin function is not selected and the value of the SPB2DT bit substitutes for the mark state until the TE bit is set to 1 (that is, transmission is enabled). The SPB2IO and SPB2DT bits should therefore be set to 1 (designating output and high level) beforehand.

To send a break signal during serial transmission, clear the SPB2DT bit to 0 (designating low level), then clear the TE bit to 0 (halting transmission). When the TE bit is cleared to 0, the transmitter is initialized, regardless of its current state, and 0 is output from the TXD2 pin.

Receive data sampling timing and receive margin

The SCIF operates on a base clock with a frequency of 16 times the bit rate. In reception, the SCIF synchronizes internally with the fall of the start bit, which it samples on the base clock. Receive data is latched at the rising edge of the eighth base clock pulse. The timing is shown in *Figure 39*.

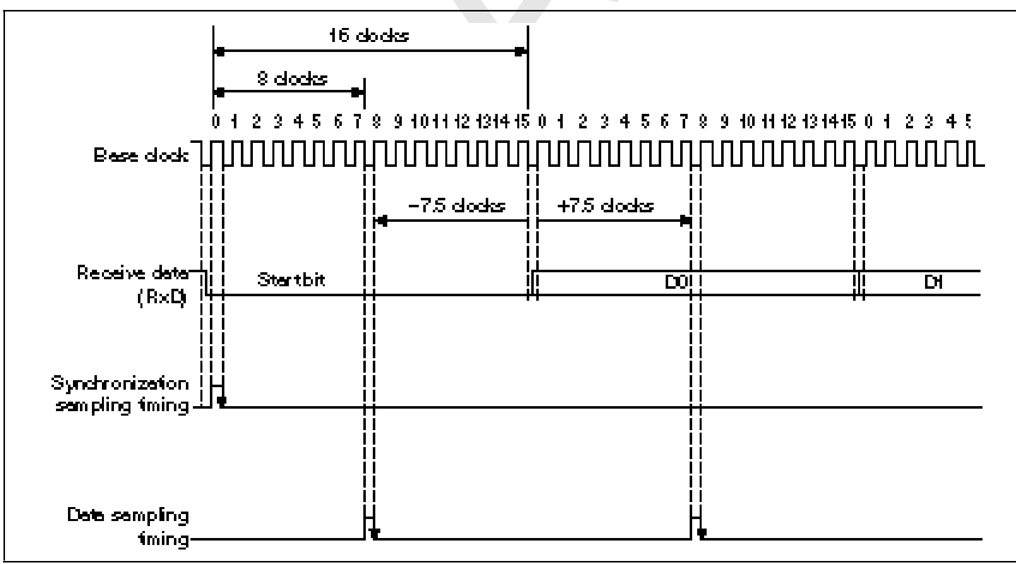


Figure 39: Receive data sampling timing in asynchronous mode

The receive margin in asynchronous mode can therefore be expressed as shown in equation (1).

$$M = \left| (0.5 - \frac{1}{2N}) - (L - 0.5) F - \frac{|D - 0.5|}{N} (1 + F) \right| \times 100\%$$

....(1)

M: Receive margin (%)

N: Ratio of clock frequency to bit rate (N = 16)

D: Clock duty cycle (D = 0 to 1.0)

L: Frame length (L = 9 to 12)

F: Absolute deviation of clock frequency

From equation (1), if F = 0 and D = 0.5, the receive margin is 46.875%, as given by equation (2).

When D = 0.5 and F = 0:

$M = (0.5 - 1 / (2 \times 16)) \times 100\% = 46.875\%$ ....(2)

This is a theoretical value. A reasonable margin to allow in system designs is 20% to 30%.

SCK2/MRESET

As the manual reset pin is multiplexed with the SCK2 pin, a manual reset must not be executed while the SCIF is operating in external clock mode.

When using the DMAC

When using the DMAC for transmission/reception, inhibit output of RXI and TXI interrupt requests to the interrupt controller. If interrupt request output is enabled, interrupt requests to the interrupt controller will be cleared by the DMAC without regard to the interrupt handler.

Serial ports

When the SCIF pin value is read using a serial port, the value read will be the value two peripheral clock cycles earlier.

10 Clock, power and reset controller

10.1 Overview

The SH-5 clock, power and reset controller (CPRC) comprises four parts:

- A clock generator[1] (CPG) which governs the supply of all clocks in the system via a clock tree to each synchronous device.
- A power management unit (PMU) which is used to control the state of the clock of each on-chip module.
- A watchdog timer (WDT) which can be used in changing clocking parameters, in waking up from power saving modes or for ensuring the software is active.
- A reset controller.

The relationship between these blocks is illustrated in *Figure 40*. The clock generator uses built in PLLs and external clocks to produce a clock signal for each of the clock domains. The power management unit is able to gate the clock to each module in a variety of modes and the watchdog timer is used both as a normal watchdog timer and to manage frequency changes when PLL's need to be re-synchronized.

1. Also known as CPG (clock pulse generator) in SH terminology.

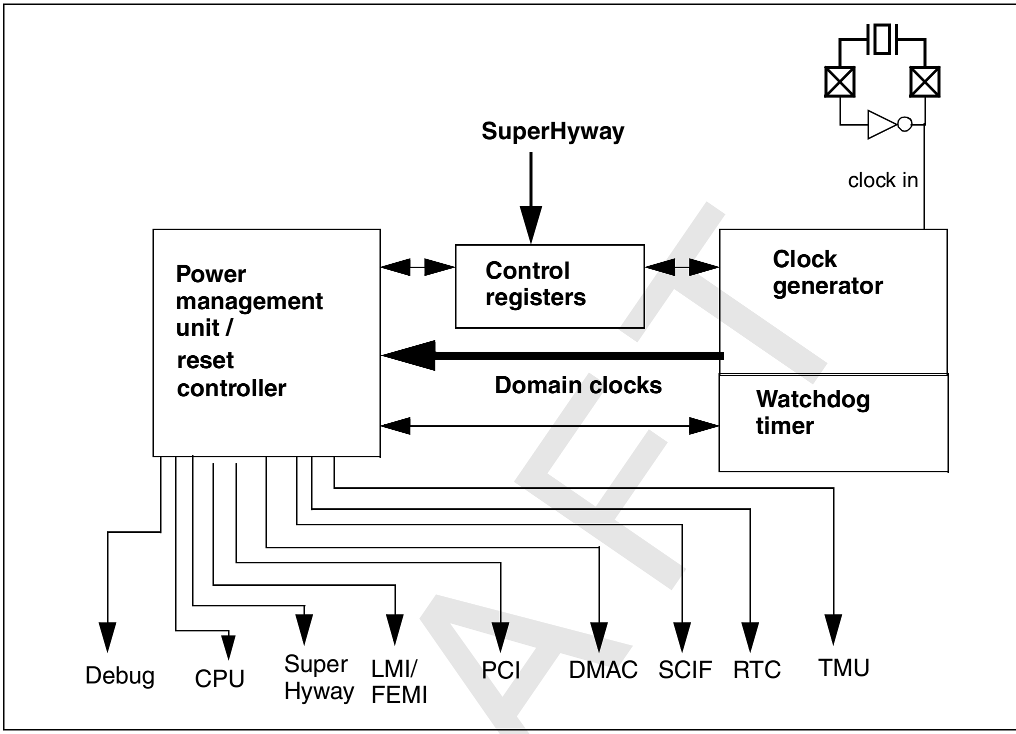


Figure 40: CPRC block diagram

10.1.1 Features

The CPRC has the following features:

- frequency change function,
- control of the ratio between domain clocks,
- power down mode control,
- on/off PLL control,
- PLL1 has programmable frequency multiply function,
- clock and clock mode output,
- management of reset.

10.2 Clock pulse generator (CPG)

The structure of the clock generator is illustrated in *Figure 41*.

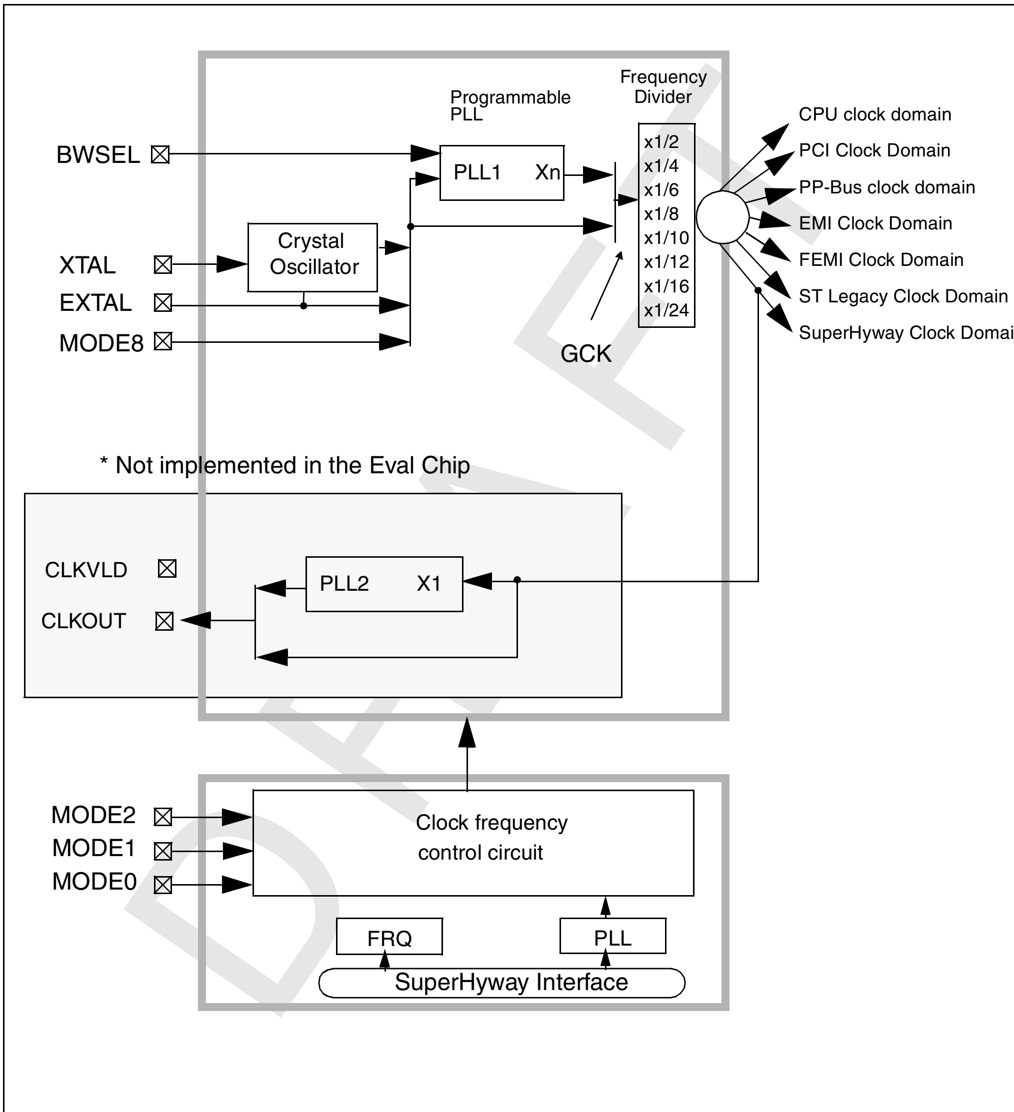


Figure 41: The clock generator

PLL circuit 1

PLL1 has a function for multiplying the clock frequency from the EXTAL pin or crystal oscillator by a programmable[1] amount. Starting and stopping is controlled by a frequency control register setting. Control is performed so that the internal clock rising edge phase matches the input clock rising edge phase.

PLL circuit 2

PLL circuit 2[2] coordinates the phases of the SuperHyway clock and the CLKOUT pin output clock. Starting and stopping is controlled by a frequency control register setting.

Crystal oscillator

This is the oscillator circuit used when a crystal resonator is connected to the XTAL and EXTAL pins. Use of the crystal oscillator can be selected with the MODE8 pin.

Frequency divider

Frequency divider generates the domain clocks from the master clock. The division ratios for each clock domain are set in the frequency control register.

Clock frequency control circuit

The clock frequency control circuit controls the clock frequency by means of the MODE pins and frequency control register.

Frequency control register (CPRC.FRQ)

The frequency control register contains control bits for clock output from the CLKOUT[2] pin, PLL circuit 1 and 2 on/off control, and the CPU clock, SuperHyway clock, and peripheral module clock frequency division ratios.

PLL control register (CPRC.PLL)

The PLL control register is used to program the frequency of the main PLL circuit.

1. Dependent on implementation. Some implementations may only offer a single programming. See product datasheet for permitted frequencies.
2. PLL2 and CLKOUT not implemented in the Eval Chip.

10.2.1 CPG pin configuration

Table 100 shows the CPG pins and their functions.

Pin name	Abbreviation	I/O	Function
Mode control pins	MODE0	Input	Set clock operating mode at power on reset
	MODE1		
	MODE2		
Crystal I/O pins (clock input pins)	XTAL	Output	Connects crystal resonator
	EXTAL	Input	Connects crystal resonator, or used as external clock input pin
	MODE8	Input	Selects use/non-use of crystal resonator When MODE8 = 0, external clock is input from EXTAL When MODE8 = 1, crystal resonator is connected directly to EXTAL and XTAL
	BWSEL	input	Selects the PLL bandwidth according to crystal resonator frequency BWSEL = 0 for a low frequency crystal BWSEL = 1 for a high frequency crystal
Clock output pin	CLKOUT[a]	Output	Used as external clock output pin Level can also be fixed
Clock valid pin	CLKVLD[a]	Output	0 when CLKOUT output clock is unstable

Table 100: CPG pins

a. Pins CLKOUT and CLKVLD not implemented in the Eval Chip.

10.2.2 CPG register configuration

Table 101 shows the CPG register configuration The addresses of registers are given as offset from CPRCBASE. See the system address map for its value.

Name	Abbreviation	RW	Initial value	Address offset	Access size (bits)
Frequency control register	CPRC.FRQ	RW	Defined by MODE0 to MODE2	0x0000	32
PLL1 control Register	CPRC.PLL	RW	Defined by MODE0 to MODE2	0x0008	32

Table 101: CPG registers

10.2.3 Clock operating modes

Table 102 shows the clock operating modes corresponding to various combinations of mode control pin (MODE2 to MODE0) settings. *Table 105* shows CRPC.FRQ settings and internal clock frequencies[1] for early versions of the silicon.

Clock operating mode	External pin combination			PLL1	PLL2[a]	Frequency ratios (cut 1.x only)			
	MODE2	MODE1	MODE0			CPU clock	SHWY clock	EMI clock	Other clock domains
0	0	0	0	On	On	X1/2	X1/8	X1/16	slowest clock
1	0	0	1	On	On	X1/2	X1/12	X1/24	slowest clock
2	0	1	0	On	On	X1/2	X1/6	X1/12	slowest clock
3	0	1	1	On	On	X1/2	X1/4	X1/8	slowest clock
4	1	0	0	On	On	X1/2	X1/2	X1/4	slowest clock
5	1	0	1	Off	On	X1/2	X1/2	X1/4	slowest clock
6	1	1	0	Off	On	X1/2	X1/4	X1/8	slowest clock
7	1	1	1	Reserved					

Table 102: Clock operating modes (cut 1 only)

a. PLL2 is not implemented in the Eval Chip

1. Dependent on implementation. See product datasheet for internal clock frequencies.

Clock operating mode	External pin combination			PLL1	Frequency ratios (cut 2 and later)			
	MODE 2	MODE 1	MODE 0		CPU clock	SHWY clock	FEMI clock	Other clock domains
0	0	0	0	On	X1/2	X1/4	X1/4	slowest clock
1	0	0	1	On	X1/6	X1/12	X1/24	slowest clock
2	0	1	0	On	X1/2	X1/6	X1/12	slowest clock
3	0	1	1	On	X1/2	X1/4	X1/8	slowest clock
4	1	0	0	On	X1/2	X1/8	X1/16	slowest clock
5	1	0	1	Off	X1/2	X1/4	X1/8	slowest clock
6	1	1	0	Off	X1	X1	X1	X1
7	1	1	1	Reserved for Test				

Table 103: Clock operating modes (later silicon versions)

Note: *The frequency range of the clock domains are found on the datasheet. Also, the reset configurations of PLL1.*

FRQ field value	Frequency division ratio
000	1/2
001	1/4
010	1/6
011	1/8
100	1/10
101	1/12
110	1/16
111	1/24

Note: Taking input clock value as 1.

The permitted ratios subset depend on the corresponding peripheral field in the FRQ register. Do not set values other than those following the relations:

IFC >= BFC >= EMC

BFC >= PCI

BFC >= PBC

10.2.4 Clock domains

Table 104 shows the correspondence between clock domains and system modules.

Clock domain	Dependent blocks
IFC	CPU, Debug
BFC	SuperHyway, DMAC, Socket
EMC	EMI
PBC	PP-BUS, RTC,TMU,SCIF,
FMC	FEMI

Table 104: CPRCC clock domains

Clock domain	Dependent blocks
SBC	SuperHyway type 2 bus
PCC	PCI clock

Table 104: CPRCC clock domains

10.2.5 Control registers

The addresses of registers are given as offset from CPRCBASE. See the system address map for its value.

CPRC.FRQ - Frequency control register				0x0000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
EMC	[2:0]	3	-	EMI clock frequency division ratio	RW
	Operation		Specifies the External Memory clock domain ratio with respect to the PLL circuit 1 output frequency.		
	When read		Returns current value		
	When written		Updates current value 001: X 1/4 010: X 1/6 011: X 1/8 100: X 1/10 101: X 1/12 110: X 1/16 111: X 1/24 Other values reserved. Changing the EMC may require for the EMI to be reset. See the datasheet or the EMI architecture specification for details.		
	HARD reset		Set by MODE0, MODE1 and MODE2 pins, see *Table 102 on page 264*		

Table 105: The FRQ CONTROL register

<table>
<tr><th colspan="4">CPRC.FRQ - Frequency control register</th><th colspan="2">0x0000</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">BFC</td><td>[5:3]</td><td>3</td><td>-</td><td>SuperHyway clock frequency divider</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies the SuperHyway clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value
000: X 1/2
001: X 1/4
010: X 1/6
011: X 1/8
101: X 1/12
110: X 1/16
Other values reserved</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Set by MODE0, MODE1 and MODE2 pins, see Table 102 on page 264</td></tr>
<tr><td rowspan="5">IFC</td><td>[5:6]</td><td>3</td><td>-</td><td>CPU clock frequency divider</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies the CPU clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">Updates current value
000: X 1/2
001: X 1/4
010: X 1/6
011: X 1/8
101: X 1/12
110: X 1/16
Other values reserved</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">Set by MODE0, MODE1 and MODE2 pins, see Table 102 on page 264</td></tr>
</table>

Table 105: The FRQ CONTROL register

CPRC.FRQ - Frequency control register				0x0000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PLL2EN	9	1	-	PLL Circuit 2 Enable[a]	RW
	Operation		Specifies whether PLL2 is on or off.		
	When read		Returns current value		
	When written		0: PLL2 is not used 1: PLL2 is used		
	HARD reset		1		
PLL1EN	10	1	-	PLL circuit 1 enable	RW
	Operation		Specifies whether PLL1 is used.		
	When read		Returns current value		
	When written		0: PLL1 is not used 1: PLL1 is used		
	HARD reset		1		
CKOEN	11	1	-	Clock output enable[b]	RW
	Operation		Specifies whether a clock is output from the CLKOUT pin or the CLKOUT pin has been put in the high impedance state. When the CLKOUT pin goes to the high impedance state, operation continues at the frequency prior to this state being entered. When the CLKOUT pin becomes high impedance it is pulled up.		
	When read		Returns current value		
	When written		0: CLKOUT pin goes to the high impedance state 1: Clock is output from the CLKOUT pin		
	HARD reset		1		

Table 105: The FRQ CONTROL register

CPRC.FRQ - Frequency control register				0x0000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
PBC	[14:12]	3	-	PP-bus clock frequency divider	RW
	Operation		Specifies the PP-bus bridge clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency		
	When read		Returns current value		
	When written		Updates current value 000: X 1/2 010: X 1/6 011: X 1/8 101: X 1/12 110: X 1/16 111: X 1/24 Other values reserved		
	HARD reset		Set by MODE0, MODE1 and MODE2 pins, see *Table 102 on page 264*		
PCC	[17:15]	3	-	PCI clock frequency divider	RW
	Operation		Specifies the PCI clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency		
	When read		Returns current value		
	When written		Updates current value 000: X 1/2 011: X 1/8 101: X 1/12 110: X 1/16 111: X 1/24 Other values reserved		
	HARD reset		Set by MODE0, MODE1 and MODE2 pins, see *Table 102 on page 264*		

Table 105: The FRQ CONTROL register

CPRC.FRQ - Frequency control register				0x0000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
FMC	[20:18]	3	-	FMI clock frequency divider	RW
	Operation		Specifies the flash memory interface clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency		
	When read		Returns current value		
	When written		Updates current value 011: X 1/8 100: X 1/10 101: X 1/12 110: X 1/16 111: X 1/24 Other values reserved		
	HARD reset		Set by MODE0, MODE1 and MODE2 pins, see *Table 102 on page 264*		
SBC	[23:21]	3	-	ST legacy bus clock frequency divider	RW
	Operation		Specifies the ST Legacy bus Interface clock domain frequency ratio with respect to the input clock of PLL circuit 1 output frequency.		
	When read		Returns current value		
	When written		Updates current value 011: X 1/8 100: X 1/10 101: X 1/12 110: X 1/16 111: X 1/24 Other values reserved		
	HARD reset		Set by MODE0, MODE1 and MODE2 pins, see *Table 102 on page 264*		

Table 105: The FRQ CONTROL register

CPRC.FRQ - Frequency control register				0x0000	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESERVED	[31:24]	8	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 105: The FRQ CONTROL register

a. PLL2 not implemented in the Eval Chip

b. Field CKOEN is not implemented in the Eval Chip.

CPRC.PLL - PLL1 control register1				0x0008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MDIV	[7:0]	8	—	Pre-divider	RW
	Operation		Parameter for programming PLL1		
	When read		Returns current value		
	When written		Updates current value This register may only be written when FRQ.PLL1EN is ‘0’. Otherwise any writes are ignored. Only certain values, as defined in the product datasheet, may be written. All other values are reserved and give undefined behavior.		
	HARD reset		1 in the Eval chip		

Table 106: CRP.CPLL control register

CPRC.PLL - PLL1 control register1				0x0008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
NDIV	[15:8]	8	—	Feedback divider	RW
	Operation		Parameter for programming PLL1		
	When read		Returns current value		
	When written		Updates current value This register may only be written when FRQ.PLL1EN is ‘0’. Otherwise any writes are ignored. Only certain values, as defined in the product datasheet, may be written. All other values are reserved and give undefined behavior.		
	HARD reset		32 in the eval chip		
PDIV	[18:16]	3	—	Post divider	RW
	Operation		Parameter for programming PLL1		
	When read		Returns current value		
	When written		Updates current value This register may only be written when FRQ.PLL1EN is ‘0’. Otherwise any writes are ignored. Only certain values, as defined in the product datasheet, may be written. All other values are reserved and give undefined behavior.		
	HARD reset		0 in the eval Chip		

Table 106: CRP.CPLL control register

CPRC.PLL - PLL1 control register1				0x0008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
SETUP	[27:19]	9	—	Loop characteristics	RW
	Operation		Parameter for programming PLL1		
	When read		Returns current value		
	When written		Updates current value This register may only be written when FRQ.PLL1EN is ‘0’. Otherwise any writes are ignored. Only certain values, as defined in the product datasheet, may be written. All other values are reserved and give undefined behavior.		
	HARD reset		0		
ENABLE	[29:28]	2	—	PLL1 enabling truth table	RW
	Operation		Parameter for programming PLL1		
	When read		Returns current value		
	When written		Updates current value This register may only be written when FRQ.PLL1EN is ‘0’. Otherwise any writes are ignored. Only certain values, as defined in the product datasheet, may be written. All other values are reserved and give undefined behavior.		
	HARD reset		0		
LOCK	30	1	✓	PLL circuit 1 Lock achieved[a]	RO
	Operation		Specifies whether PLL1 output has achieved lock and the output is stable.		
	When read		Returns current value		
	When written		0: PLL1 Lock not achieved 1: PLL1 is Locked		
	HARD reset		1		

Table 106: CRP.CPLL control register

CPRC.PLL - PLL1 control register1				0x0008	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
POWER	31	1	✓	PLL1 power control	RW
	Operation		Specifies the power state of PLL1		
	When read		Returns current value		
	When written		0: PLL1 is off and consuming no power. 1: PLL1 is on This field may only be cleared to '0'when FRQ.PLL1EN is '0'. Otherwise any writes are ignored. Hardware may set this field to '1' when FRQ.PLL1EN is set to '1'.		
	HARD reset		Depends on the clock operating mode after reset. Modes 0 to 4: Reset value = 1. Modes 5 - 6: Reset value = 0.		

Table 106: CRP.CPLL control register

a. Field LOCK not implemented in the Eval Chip (it is always set to 1).

10.2.6 Configuring PLL1

Configuring the programmable PLL1 is achieved using the CPRC. PLL control register. The frequency of PLL1 output (in Hz) may be calculated from the equation below:

$$\left(\frac{\mathrm{FRQ\_IN} \times \mathrm{NDIV}}{\mathrm{MDIV}}\right) / 2^{\mathrm{PDIV}}$$

The parameters in this expression are:

- FRQ_IN is the frequency in Hz of the SH-5's input clock
- MDIV is the pre-divider defined by PLL.MDIV
- NDIV is the feedback divider defined by PLL.NDIV
- PDIV is the post-divider defined by PLL.PDIV

If software can determine the value of FRQ_IN (for example, by statically defining that value), then software can determine the current clock speed of the SH-5. If the value of FRQ.PLL1EN is 0, the SH-5 has a clock speed of FRQ_IN. If its value is 1, the SH-5 has a clock speed given by the above equation where the values of NDIV, MDIV and PDIV can be determined by reading PLL.

The values of FRQ_IN, MDIV, NDIV, PDIV, SETUP and ENABLE which are supported by SH-5 are defined in the SH-5 Eval Device. Unsupported values give undefined behaviour.

The PLL registers cannot be modified when the FRQ.PLL1EN field contains the value '1'. This prevents the PLL1 configuration from being changed while it is in use. Any attempt to write to the PLL registers when FRQ.PLL1EN is '1' will be ignored. The PLL registers are always readable.

Note: In the Eval Chip implementation the dividers have fixed values: NDIV = 32, MDIV = 1, PDIV = 0.

10.2.7 Changing the frequency

There are three methods of changing the internal clock frequency:

- by changing the on/off state of PLL circuit1,
- by changing the frequency division ratio of each clock domain,
- by changing PLL circuit 1 configuration.

In all cases, control is performed by software by means of the FRQ register and in the third case, PLLs restart with default values defined in the product datasheet. These methods are described below.

Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is off)

When PLL circuit 1 is changed from the stopped to started state, a PLL stabilization time is required. The oscillation stabilization time count is performed by the on-chip watchdog timer (WDT).

1 Set a value in WDT to provide the specified oscillation stabilization time, and stop the WDT. The following settings are necessary: WTCSR register TME bit = 0: WDT stopped WTCSR register CKS2 to CKS0 bits: WDT count clock division ratio WTCNT counter: Initial counter value.

2 Set the PLL1EN bit to 1.

3 Internal processor operation stops temporarily, and the WDT starts counting up. The internal clock stops and an unstable clock is output to the CLKOUT[1] pin.

4 After the WDT count overflows, clock supply begins within the chip and the processor resumes operation. The WDT stops after overflowing.

Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is on)

When PLL circuit 2[2] is on, a PLL circuit 1 and PLL circuit 2 oscillation stabilization time is required.

1 Make WDT settings as in *Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is off)*.

2 Set the PLL1EN bit to 1.

3 Internal processor operation stops temporarily, PLL circuit 1 oscillates, and the WDT starts counting up. The internal clock stops and an unstable clock is output to the CLKOUT pin.

4 After the WDT count overflows, PLL circuit 2 starts oscillating. The WDT resumes its up-count from the value set in step 1 above. During this time, also, the internal clock is stopped and an unstable clock is output to the CLKOUT pin.

5 After the WDT count overflows, clock supply begins within the chip and the processor resumes operation. The WDT stops after overflowing.

Changing SuperHyway clock division ratio (when PLL circuit 2 is on)

If PLL circuit 2 is on when the SuperHyway clock frequency division ratio is changed, a PLL circuit 2 oscillation stabilization time is required.

1 Make WDT settings as in *Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is off)*.

2 Set the BFC field in the CPRC.FRQ register to the desired value.

3 Internal processor operation stops temporarily, and the WDT starts counting up. The internal clock stops and an unstable clock is output to the CLKOUT pin.

4 After the WDT count overflows, clock supply begins within the chip and the processor resumes operation. The WDT stops after overflowing.

---

1. CLKOUT not implemented in the Eval Chip.
2. PLL2 not implemented in the Eval Chip.

Changing SuperHyway clock division ratio (when PLL circuit 2 is off)

If PLL circuit 2 is off[1] when the SuperHyway clock frequency division ratio is changed, a WDT count is not performed.

1 Set the BFC field in the CPRC.FRQ register to the desired value.

2 The set clock is switched to after 120 CPU clock cycles[2].

Changing CPU or peripheral module clock division ratio

When the CPU or peripheral module clock frequency division ratio is changed, a WDT count is not performed.

1 Set the IFC or PBC field(s) in the CPRC.FRQ register to the desired value.

2 The set clock is switched to after 120 CPU clock cycles[2].

Changing PLL circuit 1 configuration

1 Set the PLL1EN bit to ‘0’; this stops PLL1.

2 Reprogram the PLL register to one of the supported configurations specified in the product datasheet.

3 Start PLL1. If PLL2 is off[3] then use the procedure described in *Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is off)*. If PLL2 is on then use the procedure described in *Changing PLL circuit 1 starting/stopping (when PLL circuit 2 is on)*.

10.2.8 Output clock control

The CLKOUT[4] pin can be switched between clock output and a fixed level setting by means of the CKOEN bit in the FRQ register.

---

1. In the Eval Chip, PLL2 is always off (not implemented).
2. 120 CPU cycles is the lowest common multiple of the clock ratios and represents the earliest point at which all clocks are guaranteed to be synchronized. This number is subject to change for each implementation. Please consult the product datasheet.
3. In the Eval Chip, PLL2 is always off (not implemented).
4. CLKOUT and bit CKOEN not implemented in the Eval Chip.

10.3 Watchdog timer

Figure 42 shows a block diagram of the WDT.

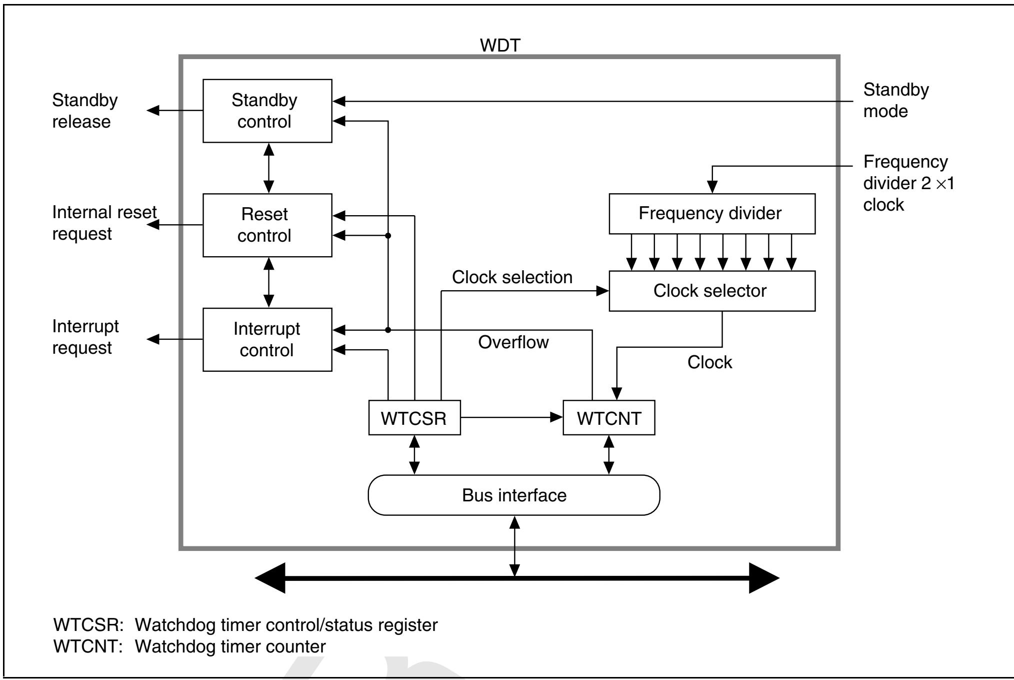


Figure 42: Block diagram of WDT

10.3.1 Register configuration

The WDT has the two registers summarized in *Table 107*. These registers control clock selection and timer mode switching.

Name	Abbreviation	RW	Initial value	Address offset	Access size (bits)
Watchdog timer counter	CPRC.WTCNT	RW*	0x00	0x0010	32
Watchdog timer control/ status register	CPRC.WTCSR	RW*	0x00	0x0018	32

Table 107: WDT registers

Note: *These registers can only be written in a specific manner that is, they are write-restricted see the register descriptions for details.*

10.3.2 WDT register descriptions

Watchdog timer counter (CPRC.WTCNT)

The watchdog timer counter (CPRC.WTCNT) is a 24-bit readable/writable counter that counts up on the selected clock. When CPRC.WTCNT overflows, a reset is generated in watchdog timer mode, or an interrupt in interval timer mode. CPRC.WTCNT is initialized to 0 only by a power-on reset via the NOT_RESETP pin.

To write to the CPRC.WTCNT counter, use a 4-byte -size access with the upper byte set to 0x5A. To read CPRC.WTCNT, use a 4-byte -size access and the counter value will be in the lower 3 bytes.

CPRC.WTCNT				0x0010	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
WTCNT	[23:0]	24	✓	Watchdog counter	RW
	Operation		Watchdog up counter; generates interrupt or reset on overflow		
	When read		Returns current value		
	When written		Updates current value. (See condition on field below.)		
	HARD reset		0		
---	[31:24]	8	—	Reserved	RES
	Operation		To write to the CPRC.WTCNT counter, a 4-byte-size access with this upper byte set to 0x5A must be used		
	When read		Returns 0		
	When written		This field must be 0x5A or the write to all fields is ignored.		
	HARD reset		0		

Table 108: CPRC.WTCNT

Watchdog timer control/status register (CPRC.WTCSR)

The watchdog timer control/status register (CPRC.WTCSR) is an 8-bit readable/writable register containing bits for selecting the count clock and timer mode, and overflow flags.

CPRC.WTCSR is initialized to 0x00 only by a power-on reset via the NOT_RESETP pin. It retains its value in an internal reset due to WDT overflow. When used to count the clock stabilization time when exiting standby mode, CPRC.WTCSR retains its value after the counter overflows.

To write to the CPRC.WTCSR register, use a 4-byte -size access with the upper bytes set to 0xA50000. To read CPRC.WTCSR, use a 4-byte-size access.

CPRC.WTCSR				0x0018	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
CKS	[2:0]	3	—	Clock select	RW
	Operation		Selects the clock frequency for the WTCNT.		
	When read		Returns current value		
	When written		Updates current value. The counter frequency is $2^{-(cks)}$ X (GCK / 24). GCK is the CPG clock rate before the frequency divider. This field is write restricted. See the final field of this register. The up-count may not be performed correctly if bits CKS2 to CKS0 are modified while the WDT is running. Always stop the WDT before modifying these bits.		
	HARD reset		0		

Table 109: CPRC.WTCSR

CPRC.WTCSR				0x0018	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
IOVF	3	1	—	Interval timer overflow flag	RW
	Operation		Indicates that the WTCNT has overflowed in interval timer mode. This flag is not set in watchdog timer mode.		
	When read		0: No overflow 1: WTCNT has overflowed in interval timer mode. This field is write restricted. See the final field of this register.		
	When written		Updates current value.		
	HARD reset		0		
WOVF	4	1	—	Watchdog timer overflow flag	RW
	Operation		Indicates that the WTCNT has overflowed in watchdog timer mode. This flag is not set in interval timer mode.		
	When read		0: No overflow 1: WTCNT has overflowed in watchdog timer mode. This field is write restricted. See the final field of this register.		
	When written		Updates current value.		
	HARD reset		0		
RSTS	5	1	—	Reset select	RW
	Operation		Specifies the kind of reset to be performed when WTCNT overflows in watchdog timer mode. This setting is ignored in interval timer mode.		
	When read		Returns current value 0: POWERON reset 1: MANUAL reset		
	When written		Updates current value. This field is write restricted. See the final field of this register.		
	HARD reset		0		

Table 109: CPRC.WTCSR

<table>
<tr><th colspan="4">CPRC.WTCSR</th><th colspan="2">0x0018</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">WT/IT</td><td>6</td><td>1</td><td>—</td><td>Timer mode select</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies whether the WDDT is used as a watchdog timer or interval timer.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Interval timer mode
1: Watchdog timer mode
The up-count may not be performed correctly if WT/IT is modified while the WDT is running
This field is write restricted. See the final field of this register.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">TME</td><td>7</td><td>1</td><td>—</td><td>Timer enable</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies starting and stopping of timer operation.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Up-count stopped, WTCNT value retained
1: Up-count enabled
This field is write restricted. See the final field of this register.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">—</td><td>[31:8]</td><td>24</td><td>—</td><td>Reserved</td><td>RES</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">To write to the CPRC.WTCSR register, use a 4-byte -size access with these upper bytes set to 0xA50000.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns 0</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">This field must be 0xA50000 or the write to all fields is ignored.</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 109: CPRC.WTCSR

Notes on register access

The watchdog timer counter CPRC.WTCNT and watchdog timer control/status register CPRC.WTCSR are write restricted to reduce the likelihood of accidental writes to these registers creating difficult-to-find bugs. These registers must be written to with an aligned 32-bit transfer instruction. They cannot be written to with any other type of access. When writing to WTCNT, perform the transfer with the upper byte set to 0x5A and the lower bytes containing the write data. When writing to WTCSR, perform the transfer with the upper byte set to 0xA50000 and the lowest byte containing the write data.

10.3.3 Using the WDT

Deep standby clearing procedure

The WDT is used when clearing deep standby mode by means of an NMI or other interrupt. The procedure is shown below. (As the WDT does not operate when standby mode is cleared with a reset, the NOT_RESETP pin should be held low until the clock stabilizes.)

1 Be sure to clear the TME bit in the WTCSR register to 0 before making a transition to standby mode. If the TME bit is set to 1, an inadvertent reset or interval timer interrupt may be caused when the count overflows.

2 Select the count clock to be used with the CKS field in the WTCSR register, and set the initial value in the WTCNT counter. Make these settings so that the time until the count overflows is at least as long as the clock oscillation stabilization time.

3 Make a transition to standby mode, and stop the clock, by executing a **sleep** instruction.

4 The WDT starts counting on detection of an NMI signal transition edge or an interrupt.

5 When the WDT count overflows, the CPG starts clock supply and the processor resumes operation. The WOVF flag in the CPR.WTCS register is not set at this time.

The counter stops at a value of 0x00 to 0x01. The value at which the counter stops depends on the clock ratio:

WTCNT stops at 1 if counter clock is not divided (that is, WTCS.CKS=0)

WTCNT stops at 0 otherwise.

Frequency changing procedure

The WDT is used in a frequency change using the PLL. It is not used when the frequency is changed simply by making a frequency divider switch.

1 Be sure to clear the TME bit in the WTCS register to 0 before making a frequency change. If the TME bit is set to 1, an inadvertent reset or interval timer interrupt may be caused when the count overflows.

2 Select the count clock to be used with bits CKS field in the WTCS register, and set the initial value in the WTCNT counter. Make these settings so that the time until the count overflows is at least as long as the clock oscillation stabilization time.

3 When the frequency control register (FRQ) is modified, the clock stops, and the standby state is entered temporarily. The WDT starts counting.

4 When the WDT count overflows, the CPG starts clock supply and the processor resumes operation. The WOVF flag in the WTCSR register is not set at this time.

The counter stops at a value of 0x00 to 0x01. The value at which the counter stops depends on the clock ratio:

WTCNT stops at 1 if counter clock is not divided (that is, WTCS.CKS=0)

WTCNT stops at 0 otherwise.

Using watchdog timer mode

1 Set the WT/IT bit in the WTCSR register to 1, select the type of reset with the RSTS bit, and the count clock with the WTCSR.CKS field, and set the initial value in the WTCNT counter.

2 When the TME bit in the WTCS register is set to 1, the count starts in watchdog timer mode.

3 During operation in watchdog timer mode, write 0 to the counter field periodically so that it does not overflow.

4 When the counter overflows, the WDT sets the WOVF flag in the WTCS register to 1, and generates a reset of the type specified by the RSTS bit. The counter then continues counting.

Using interval timer mode

When the WDT is operating in interval timer mode, an interval timer interrupt is generated each time the counter overflows. This enables interrupts to be generated at fixed intervals.

1 Clear the WT/IT bit in the WTCSR register to 0, select the count clock with the CKS field, and set the initial value in the WTCNT counter.

2 When the TME bit in the WTCSR register is set to 1, the count starts in interval timer mode.

3 When the counter overflows, the WDT sets the IOVF flag in the WTCSR register to 1, and sends an interval timer interrupt request to INTC. The counter continues counting.

10.4 Power management unit (PMU)

The SH-5 power management unit is responsible for controlling clock shutdown and startup for each of the on-chip modules. The power states are organized into a number of power down modes, each of which specify which modules are operating and which are halted.

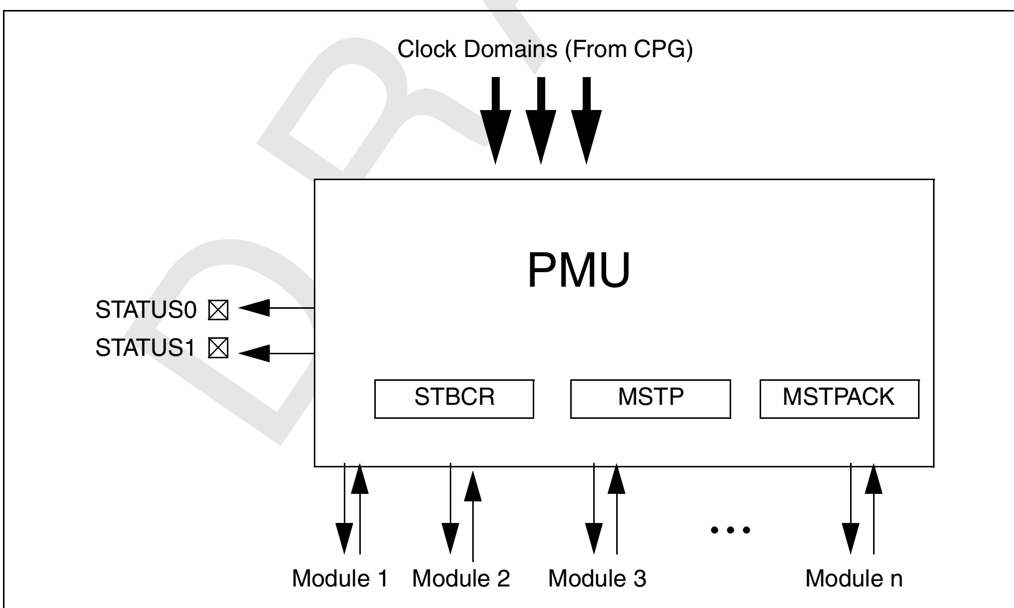


Figure 43: Power management module function

10.4.1 Types of power modes

The following power modes and functions are provided:

- Normal/economy mode
- Sleep mode
- Standby mode
  - Quick wakeup standby
  - Deep standby

Table 110 shows the conditions for entering these modes from the program execution state, the status of the CPU and peripheral modules in each mode, and the method of exiting each mode.

Power-down mode	Entering conditions	Status				Main exiting methods[a]
		Clock controller	CPU	On-chip peripheral modules	External memory refresh	
Economy	Setting any MSTP bit to 1	Operating	Operating	Specified modules halted	Depends on EMI MSTP bit.	Clearing MSTP bit(s) to 0 Reset[b]
Sleep	SLEEP instruction executed while STBCR.STBY= 0	Operating	Halted (registers held)	Specified modules halted	Depends on EMI MSTP bit	Interrupt[c] Reset[b]

Table 110: Status of CPU and peripheral modules in power-down modes

Power-down mode		Entering conditions	Status				Main exiting methods[a]
			Clock controller	CPU	On-chip peripheral modules	External memory refresh	
Standby	Quick Wakeup	SLEEP instruction executed while STBCR.STBY=1 and STBCR.QWU=1	Operating	Halted (registers held)	Halted	Halted[d]	Interrupt[c] Reset[b]
	Deep	SLEEP instruction executed while STBCR.STBY=1 and STBCR.QWU=0	Halted	Halted (registers held)	Halted	Halted[d]	Interrupt Reset[b]

Table 110: Status of CPU and peripheral modules in power-down modes

a. The debug system gives additional exiting methods, these are documented in *Section 10.5: Debug and power management on page 313*.

b. Any kind of Reset

c. Any kind of interrupt

d. Note most DRAMs can be put into self-refresh mode, so that data can be retained when EMI refresh is stopped.

Note: The RTC operates when the START bit in RCR2 is 1 (see Chapter 7: Real-time clock (RTC) on page 137).

10.4.2 Register configuration

Table 111 shows the registers used for power-down mode control.

Name	Abbreviation	RW	Initial value	Address offset	Access size (bits)
Power control register	CPRC.STBCR	RW	0	0x0030	32
Module Stop	CPRC.MSTP	RW	0	0x0020	32
Module Stop Acknowledge	CPRC.MSTPACK	RO	0	0x0028	32

Table 111: Power control register

10.4.3 Pin configuration

Table 112 shows the pins used for power-down mode control. These are further described in *Section 10.4.11: STATUS pin change timing on page 307*.

<table>
<tr><th>Pin name</th><th>Abbreviation</th><th>I/O</th><th colspan="3">Function</th></tr>
<tr><td rowspan="6">Processor status 1
Processor status 0</td><td rowspan="6">STATUS1
STATUS0</td><td rowspan="6">Output</td><td colspan="3">Indicate the processor’s operating status.
STATUS</td></tr>
<tr><td>1</td><td>0</td><td></td></tr>
<tr><td>H</td><td>H</td><td>Reset in progress</td></tr>
<tr><td>H</td><td>L</td><td>Sleep mode</td></tr>
<tr><td>L</td><td>H</td><td>Standby mode</td></tr>
<tr><td>L</td><td>L</td><td>Normal/economy operation</td></tr>
</table>

Table 112: Power-down mode pins

Note: *H: High level*
L: Low level

10.4.4 Overview

When the CPU is operating normally it is able to select which peripheral module(s) will go into a low-power state by setting to '1' the appropriate bit(s) in the MSTP register. When a selected module is completely powered down, the corresponding bit in the MSTPACK register is set by hardware.

When an MSTPACK bit is set the corresponding module will apparently respond with an error whenever any attempt is made to access it. Also, a powered down module cannot initiate any memory requests.

In most cases, modules are powered down by stopping the module's clock as soon as hardware has determined it is safe to do so. The latency between an MSTP bit being set to '1' and the corresponding bit in the MSTPACK being set to one depends on the module concerned and its state when the MSTP bit is set. This scheme provides a general mechanism which allows for complex modules to be powered down safely.

Software has the responsibility for quiescing and powering down modules in the correct sequence to avoid deadlock. In addition, software has the responsibility for ensuring that modules either powered down or in the process of powering down are not the target of accesses. Individual modules may be powered up by software clearing to '0' the appropriate MSTP bit. The module will be operating normally when the MSTPACK bit is cleared by hardware.

When the CPU executes a **sleep** instruction the CPU is put into a low power state. What else happens depends on the STBY bit in the STBCR. If the STBY bit is '0' the chip goes into sleep mode in which only the CPU and modules selected by the MSTP register are in a low power state. The clock controller, for example, remains active in sleep mode. If the STBY bit in the STBCR is set then the chip goes into standby mode in which all modules are in a low power state (regardless of their MSTP setting) and the clock controller is stopped.

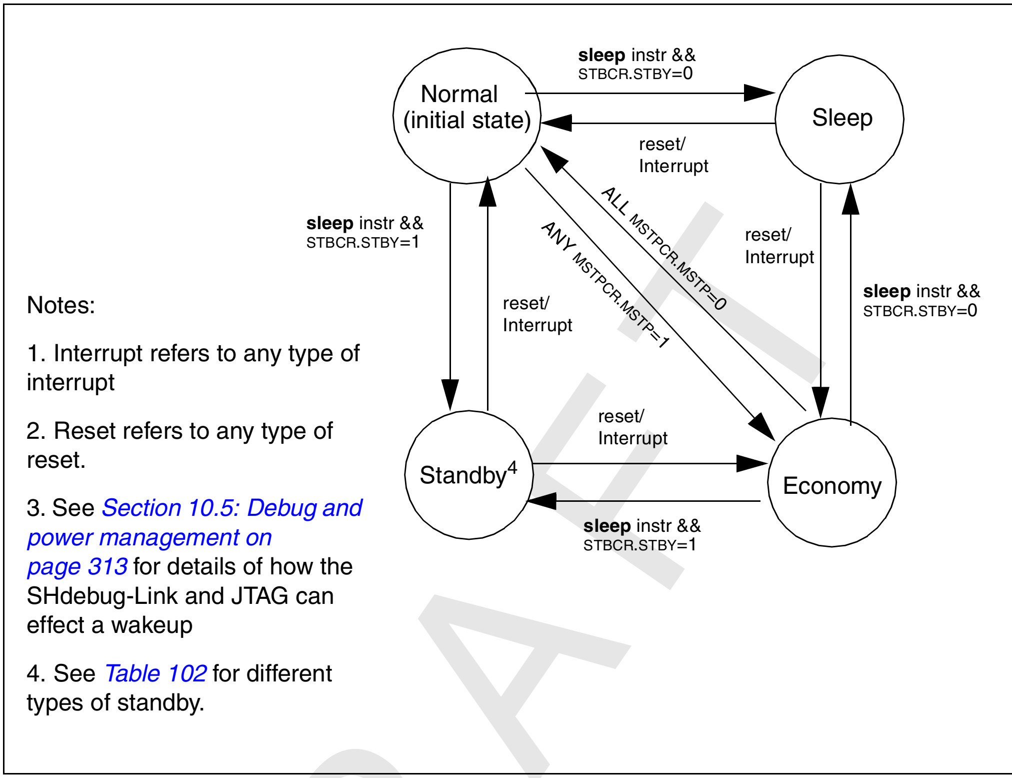


Figure 44: Power down state transitions

10.4.5 Register descriptions

Module power control register (CPRC.MSTP)

The module power control register (MSTP) is a 32 bit readable/writable register that specifies the power down mode status of each module. It is initialized to 0x00000000 by a POWERON reset via the NOT_RESETP pin or due to watchdog timer overflow.

Module number allocation

Number	Module
0	DMAC
1	SCIF
2	TMU
3	RTC
4	FEMI
5	PCI
6	EMI
7 to 31	Reserved

Table 113: Module numbers

CPRC.MSTP				0x0020	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTP0	0	1	—	DMAC module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		

Table 114: CPRC.MSTP

CPRC.MSTP				0x0020	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTP1	1	1	—	SCIF module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		
MSTP2	2	1	—	TMU module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		
MSTP3	3	1	—	RTC module stop bit	RW
	Operation		Controls whether the RTC register interface is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		

Table 114: CPRC.MSTP

CPRC.MSTP				0x0020	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTP4	4	1	—	FEMI module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		
MSTP5	5	1	—	PCI module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		
MSTP6	6	1	—	EMI module stop bit	RW
	Operation		Controls whether this module is operating normally or in low power mode.		
	When read		Returns current value.		
	When written		1: Puts module into power saving state and halts operation 0: Puts module into normal operating state		
	HARD reset		0		
—	[31:7]	25	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 114: CPRC.MSTP

Module power control acknowledge register (CPRC.MSTPACK)

CPRC.MSTPACK				0x0028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTPACK0	0	1	✓	DMAC module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		
MSTPACK1	1	1	✓	SCIF module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		
MSTPACK2	2	1	✓	TMU module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		

Table 115: CPRC.MSTPACK

CPRC.MSTPACK				0x0028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTPACK3	3	1	✓	SKT module stop bit acknowledge.	RO
	Operation		Indicates whether the SuperHighway expansion socket is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		
MSTPACK4	4	1	✓	FEMI module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		
MSTPACK5	5	1	✓	PCI module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		

Table 115: CPRC.MSTPACK

CPRC.MSTPACK				0x0028	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
MSTPACK6	6	1	✓	EMI module stop bit acknowledge.	RO
	Operation		Indicates whether this module is operating normally or in low power mode.		
	When read		Returns current value. 1: Module is currently in low power mode and is halted 0: Module is operating normally		
	When written		Ignored		
	HARD reset		0		
—	[31:7]	25	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 115: CPRC.MSTPACK

Power control register (CPRC.STBCR)

<table>
<tr><th colspan="4">CPRC.STBCR</th><th colspan="2">0x0030</th></tr>
<tr><th>Field</th><th>Bits</th><th>Size</th><th>Volatile?</th><th>Synopsis</th><th>Type</th></tr>
<tr><td rowspan="5">STBY</td><td>0</td><td>1</td><td>—</td><td>Standby bit</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Specifies a transition to standby mode.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Transition to sleep mode on execution of a sleep instruction
1: Transition to standby mode on execution of a sleep instruction</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">PHZ</td><td>1</td><td>1</td><td>—</td><td>Peripheral module pin high impedance control.</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Controls the state of peripheral module related pins in power saving modes.</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Peripheral module related pins are in normal state
1: Peripheral module related pins go to high-impedance state</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
<tr><td rowspan="5">PPU</td><td>2</td><td>1</td><td>—</td><td>Peripheral module pin pull-up control.</td><td>RW</td></tr>
<tr><td colspan="2">Operation</td><td colspan="3">Controls the state of peripheral module related pins</td></tr>
<tr><td colspan="2">When read</td><td colspan="3">Returns current value</td></tr>
<tr><td colspan="2">When written</td><td colspan="3">0: Peripheral module related pins pull-up resistors enabled
1: Peripheral module related pins pull-up resistors disabled</td></tr>
<tr><td colspan="2">HARD reset</td><td colspan="3">0</td></tr>
</table>

Table 116: CPRC.STBCR

CPRC.STBCR				0x0030	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
DEBUG	3	1	—	Indicates status of debug logic	RO
	Operation		Enables/disables the clock to the debug module and also Enables/disables the WPC logic.		
	When read		Returns current value 0: All Debug Logic is disabled and consuming least power 1: Debug module and WPC logic are enabled		
	When written		Ignored		
	HARD reset		The state of the DM_ENABLE multi-function pin is sampled at the end of the reset sequence and determines the state of this field.		
QWU	4	1	—	Quick wake up from standby	RW
	Operation		Setting this bit enables a quicker wake-up from standby mode.		
	When read		Returns current value		
	When written		0: Deep standby (lowest power consumption). The PLL and the CPG clock oscillator are stopped during standby. 1: Quick wakeup (fastest exit from standby). The PLL and CPG clock oscillator are kept running during Standby. This allows a faster wake-up from standby mode.		
	HARD reset		0		
—	[31:5]	27	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 116: CPRC.STBCR

Peripheral module pin high impedance control

When CPRC.STBCR.PHZ is set to 1, peripheral module related pins go to the high-impedance state when the system is in standby or, when the system is in economy mode and the associated module is in standby (as specified by the CPRC.MSTP register). The TMU and SCIF modules are involved.

Relevant pins

Applicable pins	Module
TCLK	TMU/RTC
SCK2	SCIF
CTS	SCIF
RTS	SCIF
TXD2	SCIF

Table 117: CPRC.STCR.PHZ pins

Peripheral module pin pull-up control

When CPRC.STBCR.PPU is cleared to 0, peripheral module related pins are pulled up when in the input or high-impedance state.

Relevant pins

Applicable pins	Module
TCLK	TMU/RTC
SCK2	SCIF
CTS	SCIF
RTS	SCIF
TXD2	SCIF

Table 118: CPRC.STCR.PPU pins

10.4.6 Sleep mode

Transition to sleep mode

If a **sleep** instruction[1] is executed when STBCR.STBY is cleared to 0, the chip switches from the program execution state to sleep mode. After execution of the **sleep** instruction, the CPU halts but its register contents are retained. The on-chip peripheral modules continue to operate, and the clock continues to be output from the CLKOUT[2] pin.

In sleep mode, a high-level signal is output at the STATUS1 pin, and a low-level signal at the STATUS0 pin.

Exit from sleep mode

Sleep mode is exited by means of an interrupt (NMI[3], IRL, or on-chip peripheral module) or a reset. When leaving sleep mode by means of an interrupt (other than NMI) an interrupt handler is launched if the interrupt is not blocked and not masked. If the interrupt is blocked or is masked, then the handler is not launched and execution continues with the next instruction after the **sleep** instruction.

Exit by interrupt

An asserted interrupt causes the CPU to exit sleep mode regardless of whether that interrupt causes the CPU to handle the exception. The requirements for exception handling to start (that is, launch an interrupt handler) are the same as when the CPU is executing normally. This means that the SR.BL bit is tested and the SR.IMASK is compared to the interrupt's priority in order to determine if exception handling should commence or execution should continue with the instruction following the **sleep** instruction. Software will typically arrange for SR.BL to be cleared or for SR.IMASK to be reduced, as appropriate, in order for the wake-up interrupt to be accepted.

Exit by reset

Sleep mode is exited by means of a power- on or manual reset via the NOT_RESETP or NOT_RESETM pins, or a power-on or manual reset executed when the watchdog timer overflows.

---

1. Refer to the CPU architecture manual for constraints on the use of the **sleep** instruction.
2. CLKOUT not implemented in the Eval Chip
3. System will always wake up regardless of IMASK and BL.

10.4.7 Standby mode

Transition to standby mode

If a **sleep** instruction[1] is executed when STBCRCR.STBY is set to 1, the chip switches from the program execution state to standby mode. In standby mode, the on-chip peripheral modules halt as well as the CPU. Clock output from the CLKOUT[2] pin is also stopped.

The CPU and cache register contents are retained. Some on-chip peripheral module registers are initialized. The state of the peripheral module registers in standby mode is shown in *Table 119*.

Module	Initialized registers	Registers that retain their contents
Interrupt controller	-	All registers
debug module	-	All registers
Clock module	-	All registers
Timer unit	TSTR register[a]	All registers except TSTR
Real-time clock	-	All registers
Direct memory access controller	-	All registers
Serial communication interface	-	All registers

Table 119: State of registers in standby mode

a. Not initialized when the real-time clock (RTC) is in use (see *Chapter 8: Timer unit (TMU) on page 171*).

DMA transfer should be terminated before making a transition to standby mode. Transfer results are not guaranteed if standby mode is entered during transfer.

1. Refer to the CPU architecture manual for constraints on the use of the SLEEP instruction.
2. CLKOUT not implemented in the Eval Chip

Standby mode has two sub modes of operation. Deep standby mode where the clock controller including the PLL is halted and Quick wake up mode where the clock controller and PLL are operational. Quick wakeup mode consumes the more power but exit from this mode does not require a PLL synchronization period and is therefore faster. Deep standby mode is the lowest power mode supported but exit does need the WDT to be programmed to allow a PLL synchronization period.

Transition to deep standby

The procedure for a transition to deep standby mode is described below.

1 If wakeup by interrupt is required then the RTC module should first be correctly configured.

2 Clear the TME bit in the WDT timer control register (WTCSR) to 0, and stop the WDT. Set the initial value for the up-count in the WDT timer counter (WTCNT), and set the clock to be used for the up-count in bits CKS2 to CKS0 in the WTCSR register.

3 Set STBCR.STBY=1 and STBCR.QWU=0

4 Execute a **sleep** instruction.

5 When standby mode is entered and the chip's internal clock stops, a low-level signal is output at the STATUS1 pin, and a high-level signal at the STATUS0 pin.

Transition to quick wakeup standby

The procedure for a transition to quick wakeup standby mode is shown below.

1 Set STBCR.STBY=1 and STBCR.QWU=1

2 Execute a **sleep** instruction.

When standby mode is entered and the chip's internal clock stops, a low-level signal is output at the STATUS1 pin, and a high-level signal at the STATUS0 pin.

Note: *It may be necessary to power down modules in a strict sequence prior to entering standby mode to ensure that recovery from standby is architecturally defined. See* *Transition to economy mode on page 307*.

10.4.8 Exit from standby mode

Standby mode can be exited by means of any interrupt (NMI, IRL, or on-chip peripheral module) or a reset via either the NOT_RESETP or NOT_RESETM pins.

Exit by interrupt

When leaving Standby mode by means of an interrupt (other than NMI) CPU execution can resume either with the launch of an interrupt handler or with the next instruction following the **sleep** instruction depending on whether an interrupt launch would have occurred if the CPU not executed the sleep instruction.

An interrupt handler is launched on exit from standby if the interrupt is not blocked (by SR.BL) and not masked (by SR.IMASK). If the interrupt is blocked or is masked, then the handler is not launched and execution continues with the next instruction after the **sleep** instruction.

This means that an asserted interrupt causes the CPU to exit standby mode regardless of whether that interrupt causes a launch. Software will typically arrange for SR.BL to be cleared or for SR.IMASK to be reduced, as appropriate, so that the wake-up interrupt can be accepted.

Deep standby

A hot start can be performed by means of the on-chip WDT. When an NMI, IRL (see 1 below), or other kind of interrupt (see 2 below) is detected, the WDT starts counting. After the count overflows, clocks are supplied to the entire chip, standby mode is exited, and the STATUS1 and STATUS0 pins both go low. If an interrupt to be launched, Interrupt exception handling is executed, and the code corresponding to the interrupt source is set in the INTEVT register.

The phase of the CLKOUT pin clock output may be unstable immediately after an interrupt is detected, until standby mode is exited.standby mode is exited, and the STATUS1 and STATUS0 pins both go low. Interrupt exception handling is then executed, and the code corresponding to the interrupt source is set in the INTEVT register.

Quick wakeup

When an NMI or other interrupt is detected in quick wakeup standby mode clocks are soon supplied to the entire chip.

1 Only when the RTC clock (32.768 kHz) is operating (see *Section 6.2.2: IRL interrupts on page 111*), can deep standby mode can be exited by means of IRL3 to IRL0 (when the IRL3 to IRL0 level is higher than the SR.MASK register mask level).

2 Standby mode can be exited by means of an RTC interrupt.

Exit by Reset

Standby mode is exited by means of a reset (power-on or manual) via either of the NOT_RESETP or NOT_RESETM pins. If the NOT_RESETP pin is used it should be held low until clock oscillation stabilizes. The internal clock continues to be output at the CLKOUT[1] pin.

10.4.9 Clock pause function

In deep standby mode, it is possible to stop or change the frequency of the clock input from the EXTAL pin. This function is used as follows.

1 Enter deep standby mode following the transition procedure described above.

2 When standby mode is entered and the chip's internal clock stops, a low-level signal is output at the STATUS1 pin, and a high-level signal at the STATUS0 pin.

3 The input clock is stopped, or its frequency changed, after the STATUS1 pin goes low and the STATUS0 pin high.

4 When the frequency is changed, input an NMI or IRL interrupt after the change. When the clock is stopped, input an NMI or IRL interrupt after applying the clock.

5 After the time set in the WDT, clock supply begins inside the chip, the STATUS1 and STATUS0 pins both go low, and operation is resumed.

1. CLKOUT not implemented in the Eval Chip

10.4.10 Economy mode

Transition to economy mode

Setting any of the MSTP bits in the CRPC.MSTP to 1 causes the module to be put into a low power state, this is normally achieved by disabling the clock supply to the module but may use other mechanisms depending on the module. Use of this function allows power consumption when the CPU is operating normally or in sleep mode to be further reduced.

Note that to allow the integration of complex modules the powering down of modules is subject to confirmation in the MSTPACK. For example when powering down memory interfaces it may be necessary to power down all DMA's first. The power down confirmation given by MSTPACK indicates to software when this has been achieved and so enabling power down to be performed in a recoverable manner.

Before using the MSTP bits to put a module into low power mode software should ensure:

1 The module is in suitable state for being stopped. This may involve configuring the module into disabled state. Refer to individual module specification for details of how to do this.

2 That the sequence in which modules are powered down is safe and does not lead to deadlock or loss of state. For example, DMA channels should be stopped before the memory interfaces which they use are stopped.

Exit from economy mode

The module standby function is exited by clearing all the MSTP bits to 0, or by a power-on or manual reset via the NOT_RESETP or NOT_RESETM pins or either reset caused by watchdog timer overflow.

10.4.11 STATUS pin change timing

The STATUS1 and STATUS0 pin change timing is shown below.

The meaning of the STATUS pin settings is as follows:

Reset: HH (STATUS1 high, STATUS0 high)

Sleep: HL (STATUS1 high, STATUS0 low)

Standby: LH (STATUS1 low, STATUS0 high)

Normal: LL (STATUS1 low, STATUS0 low)

The meaning of the clock units is as follows:

Bcyc: SuperHyway clock cycle

Pcyc: PP-bus clock cycle

Reset

In the figures in this section, NOT_RESET refers to either NOT_RESETP or NOT_RESETM depending on the figure title.

Power-on reset

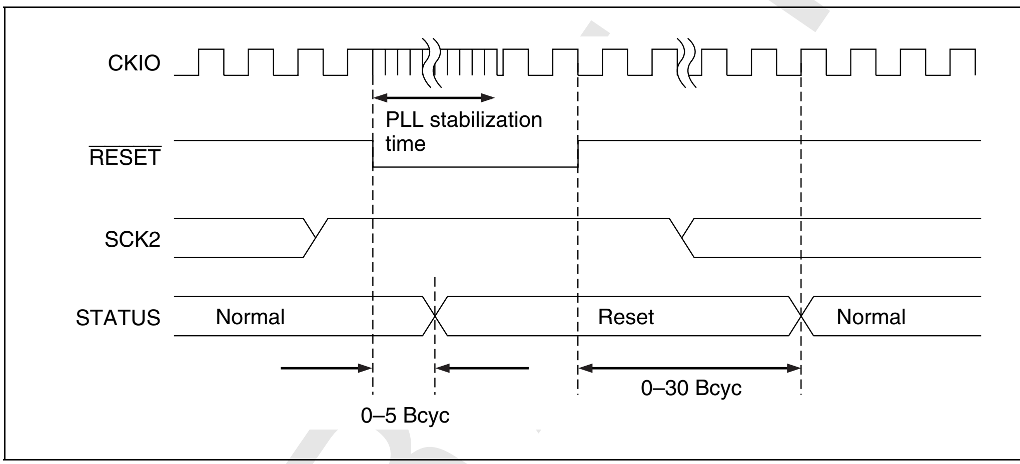
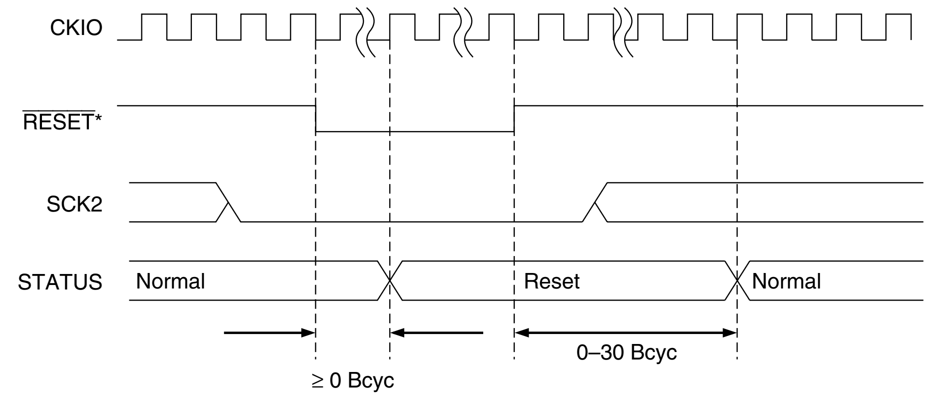


Figure 45: STATUS output in power-on reset

Manual reset



Note: * In a manual reset, STATUS = HH (reset) is set and an internal reset started after waiting until the end of the currently executing bus cycle.

Figure 46: STATUS output in manual reset

In exit from standby mode

Standby-interrupt

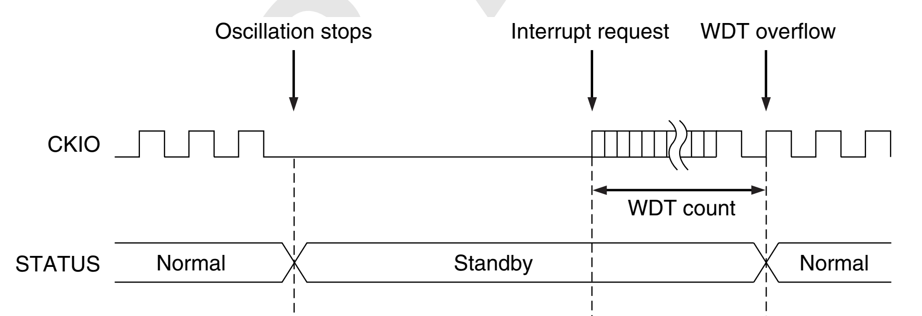
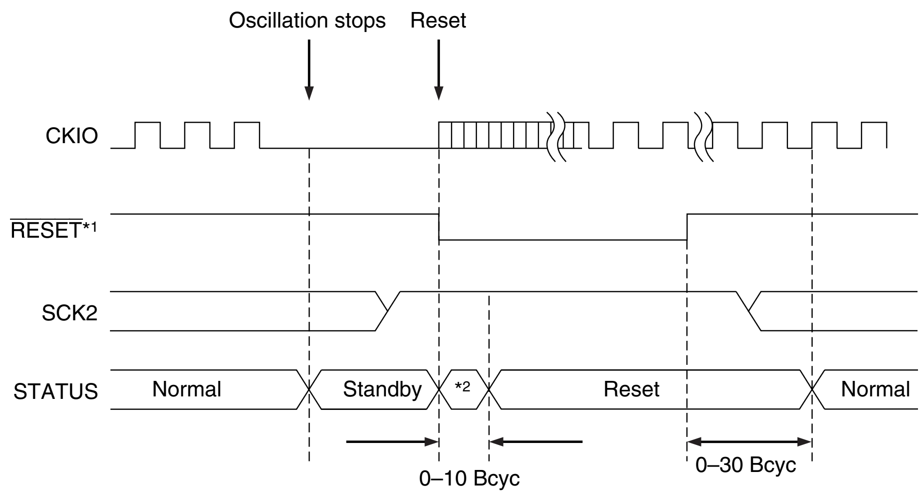


Figure 47: STATUS output in standby interrupt sequence

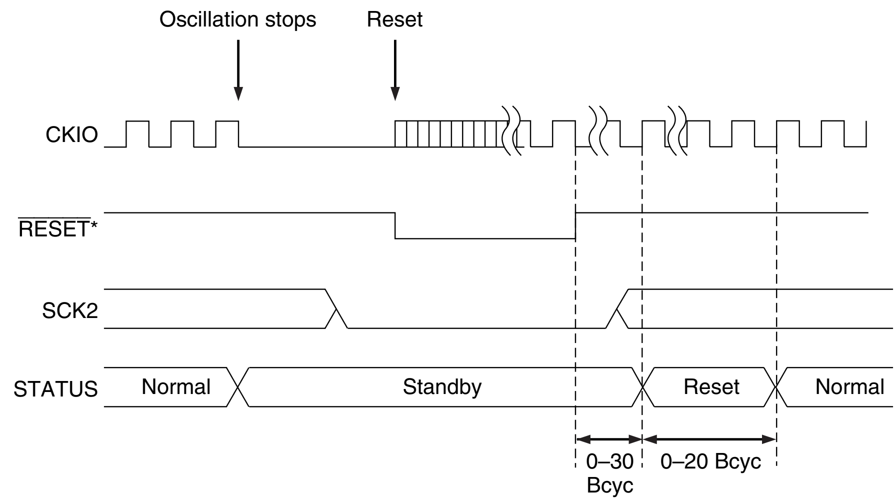
Standby-power-on reset



Notes: 1. When standby mode is exited by means of a power-on reset, a WDT count is not performed. Hold RESET low for the PLL oscillation stabilization time.
2. Undefined

Figure 48: STATUS output in standby power-on reset

Standby-manual reset



Note: * When standby mode is exited by means of a manual reset, a WDT count is not performed. Hold RESET low for the PLL oscillation stabilization time.

Figure 49: STATUS output in standby manual reset sequence

In exit from sleep mode

Sleep-interrupt

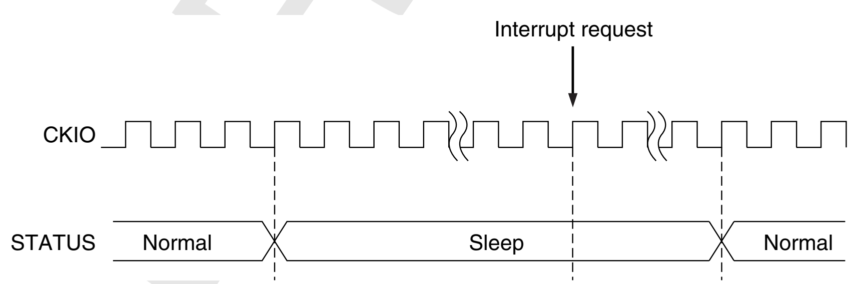
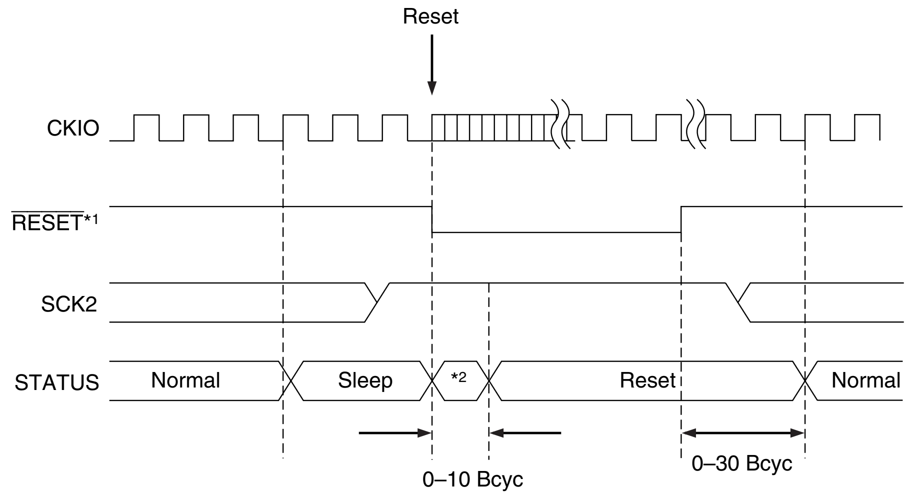


Figure 50: STATUS output in sleep interrupt sequence

Sleep-power-on reset



Notes: 1. When sleep mode is exited by means of a power-on reset, hold $\overline{\text{RESET}}$ low for the oscillation stabilization time.
2. Undefined

Figure 51: STATUS output in sleep power-on reset sequence

Sleep-manual reset

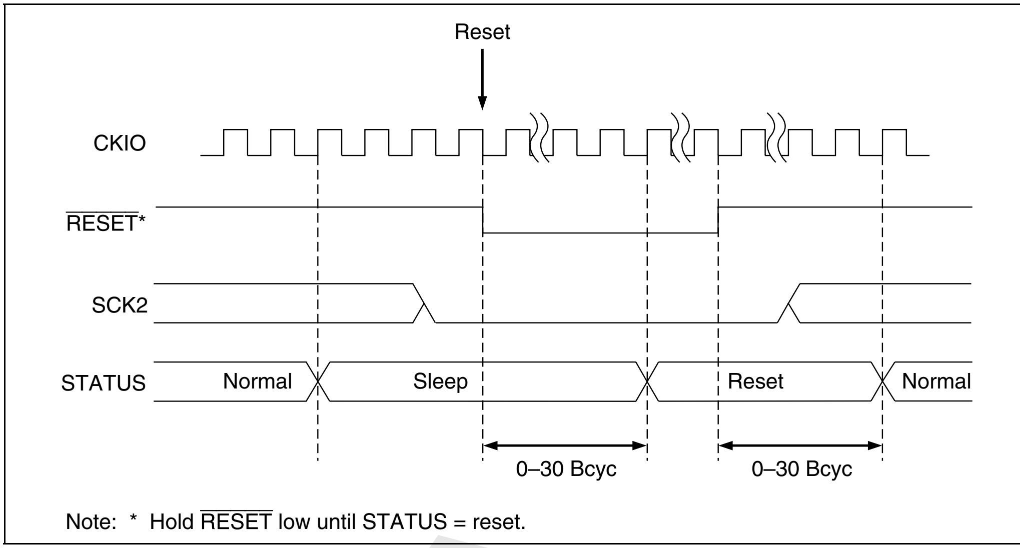


Figure 52: STATUS output in sleep manual reset sequence

10.5 Debug and power management

10.5.1 Debug enable/disable

Following POWERON reset, all debug logic is either disabled (lowest-possible power state) or enabled. The lowest possible power state is with the clock to the debug module turned off and the WPC functions in the core disabled.

The STBCR.DEBUG bit provides this debug enable/disable function. It enables/disables the clock to the debug module and also provides the enable/disable signal to the WPC logic. The state of the DM_ENABLE multi-function pin is sampled at the end of the reset sequence and determines the state of the STBCR.DEBUG bit.

The DM_ENABLE multi-function pin (see *Volume 3 Debug, Chapter 3 External Debug Interfaces*) must be provided in all SH-5-based product chips, even those without full SHDebug-link functionality.

10.5.2 Debug module state with no tool connected

For some applications, useful target system debugging can be performed without the target needing to be connected via SHDebug-Link or JTAG to a tool, that is, a traditional “monitor ROM” style debugger built into the application. In such circumstances, all on-chip debug resources must be enabled. Communication with resident debug code may occur via standard board-level interfaces such as a serial port or a LAN port.

In such systems, the debug module enable/disable function is controlled by a board-level jumper which either pulls the DM_ENABLE sensing pin (DM_CLKIN) low (DM enabled) or lets this signal be pulled high by its internal pull-up (DM disabled).

10.5.3 Debug wakeup from standby state

Wakeup via SHDebug-link

When SH-5 is in standby state, there are no clock pulses occurring on DM_CLKOUT. However, the tool can still generate clock pulses on the DM_CLKIN pin using a tool-generated clock. This allows the debug tool to wakeup the system by sending a wakeup message to the SH-5.

This wakeup message consists of a normal DBUS message as defined in *Volume 3 Debug, Chapter 3 External Debug Interfaces*. The act of receiving the DBUS message will wake the system, and the DBUS request will be processed as normal (that is, a DBUS response or error will be generated).

Wakeup via JTAG

Since the tool is the source of TCK clock, the TAP controller continues to operate even when the chip is in standby state. A JTAG-connected tool can wake-up SH-5 from standby state by shifting a special **wakeup** instruction into the sequence in the debug DR register.

The value of this **wakeup** instruction is implementation specific, and is defined in *Volume 3 Debug, Chapter 4 Implementation specifics*.

Use of the **wakeup** instruction when the JTAG port is not the active debug interface causes undefined effects.

Reception of the **wakeup** instruction when the JTAG port was previously selected as the active debug interface asserts the wakeup signal to the PMU, and JTAG port remains as the active debug interface.

For a JTAG tool to determine when SH-5 has completed the transition from standby state to normal operation, the tool must be capable of monitoring the STATUS0 and STATUS1 status pins.

10.5.4 Debug wakeup from sleep states

In order that a tool can debug SH-5 when it is in the sleep state, the debug module must be enabled. This causes clock to the debug module to be turned on but clocks to the CPU are turned off. Because the debug module is enabled, clock pulses are present on the DM_CLKOUT pin and the debug module is able to process DBUS messages received from the tool.

The debug module sends all DBUS request messages received from the tool to the SuperHyway, irrespective of the destination. If the destination is a debug register within the debug module, then the SuperHyway bus request is received by the debug module's SuperHyway target interface and forwarded to the appropriate register within the debug module. Because the SuperHyway is alive in sleep state, the tool can access all debug registers in the debug module and also access memory mapped registers in all other enabled modules.

If the DBUS message sent by the tool accesses a WPC register, the SuperHyway request initiated by the debug module accesses the SuperHyway target port of the CPU core. In sleep state, clocks to the core are turned off even though SuperHyway bus clocks are enabled. The bus request to the WPC register cannot be processed by the SuperHyway target port of the CPU core and the tool will receive a SuperHyway error response. The tool can then inspect the state of PMU registers to determine whether the CPU is powered down and can then send a debug interrupt if it wishes to wake up the CPU.

10.6 Reset controller

SH-5 supports four types of reset.

- CPU reset causes the CPU to disable the MMU and to start fetching instructions from RESVEC/DBRVEC.
- POWERON[1] reset initializes all register state in the SH-5 core and on-chip peripherals needed for correct functioning of the part, including debug resources. External DRAM state may be lost.
- MANUAL reset initializes the same register state as POWERON reset in the SH-5 core and on-chip peripherals needed for correct functioning of the part. It does not reset the memory controller and the PLL and because of this, the contents of DRAM are preserved.
- DEBUG reset initializes the same register state as POWERON reset in the SH-5 core and on-chip peripherals needed for correct functioning of the part, but excludes debug architecturally-visible registers in the WPC, DM and Bus Analyzer.

  However, a DEBUG reset initializes all temporary storage used to hold watchpoint hit information and trace messages (the capture buffer and the DM FIFO within the debug module), and clears any pending debug interrupts. Thus, registers in the DM which reflect the internal DM state (for example, DM.TRCTL.DL_N_JTAG, FIFO registers) are updated. For further information, see *Volume 3 Debug, Chapter 3 External Debug Interfaces*).

A tool can perform either a CPU reset or DEBUG reset by writing to the WPC.CPU_CTRL_ACTION register. However, neither a POWERON reset nor a MANUAL reset can be initiated this way.

POWERON reset, DEBUG reset and MANUAL reset can also be performed using the DM_IN, NOT_RESETP and NOT_RESETM pins.

1. For historical reasons the term hard reset is sometimes used. In all cases this is the same as the POWERON RESET state but additionally may apply to the other reset types depending on the module.

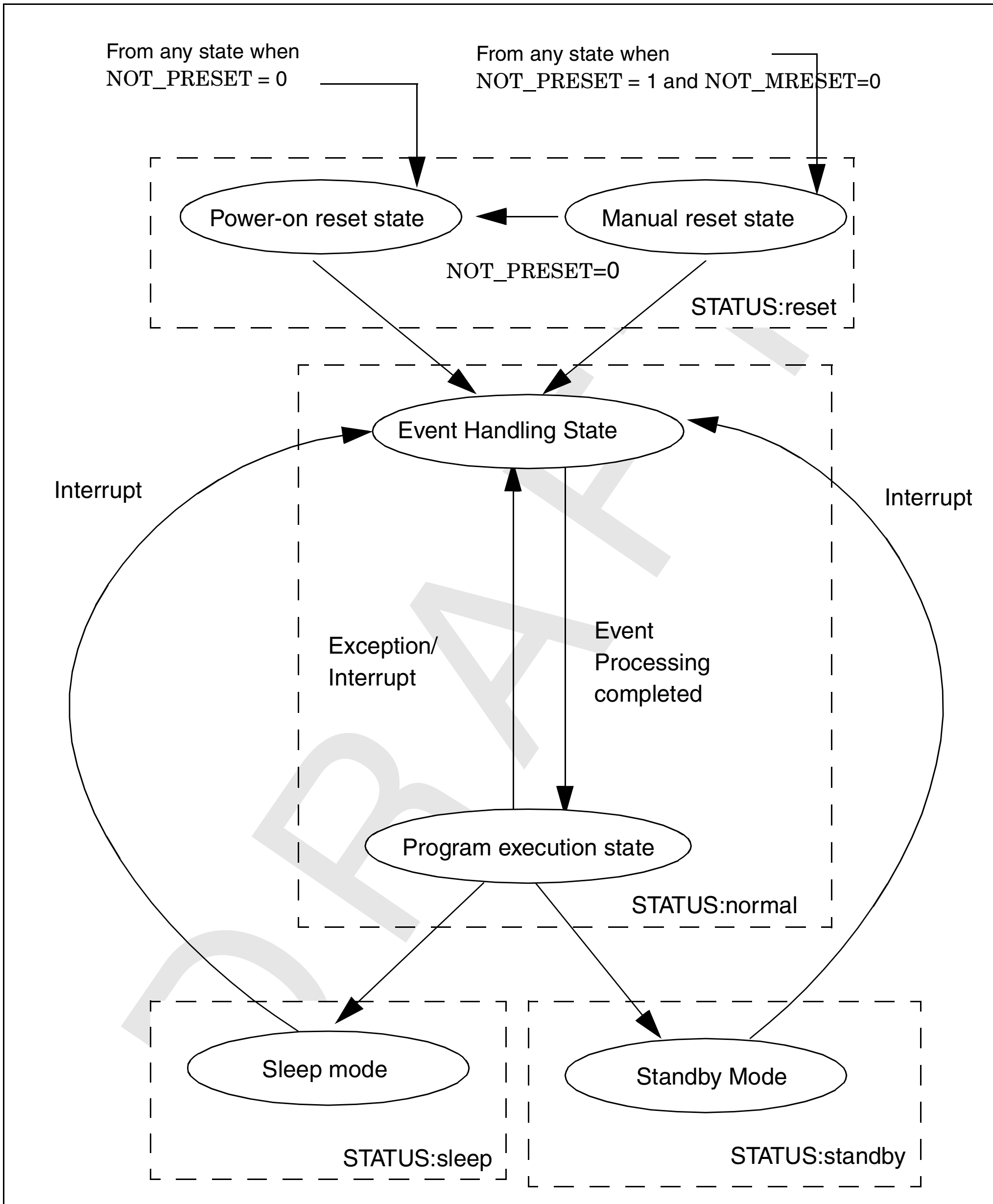


Figure 53: Processor state transitions

	CPU state	Memory state	Debug state	All other peripheral's state
Power-On Reset	Initialized	Initialized	Initialized	Initialized
Manual Reset	Initialized	Unchanged	Initialized	Initialized
CPU Reset	Initialized	Unchanged	Unchanged	Unchanged
Debug Reset	Initialized	Initialized	Unchanged	Initialized

Table 120: SH-5 reset types

10.6.1 Reset pins

SH-5 has two reset input pins NOT_RESETP and NOT_RESETM (the same as SH-3/4). Asserting NOT_RESETP produces a POWERON reset and asserting NOT_RESETM produces a MANUAL reset. A third type of reset, DEBUG reset, can also be initiated via either the NOT_RESETP pin or the NOT_RESETM pin when RESET_MODE is sampled low. Note that NOT_RESETP takes precedence over NOT_RESETM if both are asserted simultaneously

Operation	NOT_RESETP	NOT_RESETM	RESET_MODE
Debug reset	0	0	0
Power-on reset	0	0	1
Debug reset	0	1	0
Power-on reset	0	1	1
Debug reset	1	0	0
Manual reset	1	0	1
Normal operation	1	1	X

Table 121: Reset pin operation

A finished SH-5-based product will normally have a single reset button and the function of this button will be determined by the application, that is, it can be connected to either the NOT_RESETP pin or the NOT_RESETM pin. A typical development board may have a jumper which selects whether the reset button connects to either the NOT_RESETP pin or the NOT_RESETM pin. Either the same jumper or a second jumper connects the NOT_RESET signal on the SHDebug-Link header and the JTAG header to either the NOT_RESETP pin or the NOT_RESETM pin of SH-5.

RESET_MODE (DM_IN) is a normal signal in the SHDebug-Link interface. This same signal goes to the JTAG debug header (via an AC-decoupled series resistor) to allow the JTAG tool to force DEBUG reset whenever board-level reset button is pressed.

JTAG reset

TRST provides an asynchronous reset signal for the JTAG TAP controller finite state machine. This finite state machine is reset whenever TRST changes from a high-level to a low-level and this function is independent of the state of the CPU and other on-chip modules.

10.6.2 Reset function

POWERON reset, MANUAL reset and DEBUG reset are all performed using a signal which is fanned out to all flip-flops to achieve simultaneous reset. The number of clock cycles required to complete the reset function is implementation defined and is documented in the datasheet. $\overline{\text{RESET}}$ can be held low for much longer than the minimum value given in the datasheet but not for less. Note asserting reset for less than the minimum time will result with unpredictable behavior.

DEBUG reset is a variant of POWERON reset, the difference being that the state of architecturally-visible debug registers is not changed.

MANUAL reset is also a variant of POWERON reset, the difference being that none of the memory mapped registers in the memory controller are reset. This means that normal memory refresh functions continue during the reset operation ensuring that RAM contents are retained. This reset mode is particularly important for products in which an operating system file system is held in RAM and must be preserved during reset operations.

During the reset process, the reset mode (POWERON, MANUAL or DEBUG) is latched in a PMU register and is available for software to read. See *Section 10.6.4: Reset status on page 320*.

10.6.3 Reset functions available from debug tools

A full description of these features are available in the debug volume (see *Volume 3 Debug, Chapter 3 External Debug Interfaces*).

10.6.4 Reset status

The CPRC contains a register which allows software to determine the type of the most recent reset. A value is loaded into the register on completion of each reset process.

CPRC.RST				0x0038	
Field	**Bits**	**Size**	**Volatile?**	**Synopsis**	**Type**
RESET_TYPE	[1:0]	2	✓	Last type of reset	RO
	Operation		The value of this field indicates the type of the most recent reset 0b00: POWERON reset 0b01: MANUAL reset 0b10: DEBUG reset 0b11: Undefined		
	When read		Returns current value		
	When written		Ignored		
	HARD reset		Current value		
—	[31:2]	30	—	Reserved	RES
	Operation		Reserved		
	When read		Returns 0		
	When written		Ignored		
	HARD reset		0		

Table 122: CPRC.RST register

10.6.5 Register summary

All addresses are offset from the CPRC base address (CPRCBASE) whose value is given in *Section 5.2.2: Address map on page 95*.

	Register	Function	Address offset	Details
CLOCK controller (CPG)	CPRC.FRQ	Clock dividers specification	0x0000	*page 267*
	CPRC.PLL	Programmable PLL configuration	0x0008	*page 267*
WDT	CPRC.WTCNT	Watchdog counter	0x0010	*page 280*
	CPRC.WTCS	Watchdog control	0x0018	*page 282*
Power management unit	CPRC.MSTP	Module stop	0x0020	*page 290*
	CPRC.MSTPACK	Module stop ack	0x0028	*page 296*
	CPRC.STBCR	Power control	0x0030	*page 299*
Reset controller	CPRC.RST	Indicates reset cause	0x0038	*page 320*

Table 123: CPRC registers

Reset Behaviour

All CPRC registers are initialised by a power-on reset. On a manual reset CPRC register values retain their pre-manual reset value.

Incorrect register accesses

Registers may only be accessed in the size specified in the register definition. Writes to addresses which include all or part of a valid register but in the wrong access size will not update the contents of the register. Reads to addresses which include all or part of a valid register but in the wrong access size will not return the contents of the register.

An implementation may set the peripheral bridge error flag on receipt of an invalid access.

Index

D

E

F

H

I

J

L

M

N

O

P

R

T

U

V

W

XYZ