MSC711x Reference Manual

MSC7110, MSC7112, MSC7113, MSC7115, MSC7116, MSC7118, and MSC7119

MSC711xRM
Rev. 1, November 2006

freescale™ semiconductor

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

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

Document Order Number: MSC711xRM
Rev. 1
11/2006

Contents

15 Event Port

16 Debugging

17 Programmable Address Detection

18 Fast Ethernet Controller (FEC)

19 Time-Division Multiplexing (TDM) Interface

20 Host Interface (HDI16)

A System Usage and Tuning/ Programming Reference

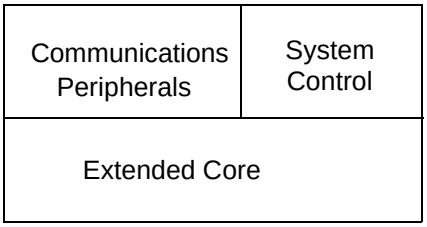
About This Book

The MSC711x family is a high-performance, cost-effective family of DSPs based on the StarCore™ SC1400 core, which offers system solutions, flexibility with peripherals and performance, and overall system cost savings. Devices in the MSC711x family target high-bandwidth highly computational DSP applications and are optimized for packet telephony applications, providing a competitive price per channel for voice over packet systems. MSC711x is logically partitioned into three distinct blocks: the extended core, the system control modules, and the communications peripherals.

Read Chapters 1–4 for an overview of the entire system.

Read Chapters 5–9 for information on data management and the memory system.

Read Chapters 16–22 for details on the communications peripherals.



Read Chapters 10–15 for details on configuration and reset, including the system control modules and functions.

Read Chapter 3 for an overview of the SC1400 extended core. Also, consult the *SC1000-Family Processor Core Reference Manual*.

Extended Core	System Control	Communications Peripherals
The extended core contains an SC1400 DSP core with internal memory for data and program storage. Memory includes up to 192 KB of zero wait state SRAM and 16 KB of instruction cache and 8 KB of boot ROM. Some devices in the MSC711x family also include 192 KB of shared memory (M2). Minimum code density is achieved using a 16-bit instruction set that the compiler or programmer groups into execution sets for high instruction parallelism.	Supports internal and external system-related functions. The system control modules include hardware such as a direct memory access (DMA) controller, clocks, and reset configuration registers. It also includes the crossbar switch which arbitrates between four master and six slave controllers.	Includes up to three TDM interfaces supporting 128 channels each, a UART, an I^2C interface, eight 16-bit timers, up to 37 programmable GPIO signals, an MII/RMII Ethernet interface, a DDR memory controller, and an external interrupt controller. The serial interfaces give additional functionality and flexibility.

Note: The amount of internal memory and the number and type of available external peripheral interfaces are device-dependent. Refer to the technical data sheet for your device to determine the memory and peripherals available. See also Chapter 1 of this manual, which summarizes the MSC711x devices and lists their differences.

Before Using This Manual—Important Note

This manual describes the structure and function of the MSC711x devices. The information in this manual is subject to change without notice, as described in the disclaimers on the title page of this manual. As with any technical documentation, it is your responsibility as the reader to ensure that you are using the most recent version of the documentation. For more information, contact your sales representative.

Before using this manual, determine whether it is the latest revision and whether there are errata or addenda. To locate any published errata or updates associated with this manual or this product, refer to the Freescale web site. The address for the web site is listed on the back cover of this manual.

Audience and Helpful Hints

This manual is intended for software and hardware developers and applications programmers who want to develop products with the MSC711x devices. It is assumed that you have a working knowledge of DSP technology and that you may be familiar with Freescale products based on the DSP56000, DSP56300, or SC140 DSP core. Familiarity with Freescale DSP products is not necessary.

For your convenience, the chapters of this manual are organized to make the information flow as predictably as possible. When feasible, the information in each chapter follows this general sequence:

- Features
- Architecture
- Operation/operating modes
- Application/Initialization
- Programming Examples
- Programming Model (registers)

In chapters that include a Programming Model section, this section is the last one in the chapter, or module subsection for those chapters that include multiple modules, and describes all registers for the module discussed. The Programming Model section begins with a bulleted overview of the registers that includes the page number where the description of each register begins.

Notational Conventions and Definitions

This manual uses the following notational conventions:

mnemonics	Instruction mnemonics appear in lowercase bold.

COMMAND names	Command names are set in small caps, as follows: GRACEFUL STOP TRANSMIT or ENTER HUNT MODE.
italics	Book titles in text are set in italics, as are cross-referenced section titles. Also, italics are used for emphasis and to highlight the main items in bulleted lists.
0x	Prefix to denote a hexadecimal number.
0b	Prefix to denote a binary number.
REG[FIELD]	Abbreviations or acronyms for registers or buffer descriptors appear in uppercase text. Specific bits, fields, or numeric ranges appear in brackets. For example, ICR[INIT] refers to the Force Initialization bit in the host Interface Control Register.
ACTIVE HIGH SIGNALS	Names of active high signals appear in sans serif capital letters, as follows: TT[04], TSIZ[0–3], and DP[0–7].
$\overline{\text{ACTIVE LOW SIGNALS}}$	Signal names of active low signals appear in sans serif capital letters with an overbar, as follows: $\overline{\text{DBG}}$, $\overline{\text{AACK}}$, and $\overline{\text{EXT\_BG[2]}}$.
x	A lowercase italicized x in a register or signal name indicates that there are multiple registers or signals with this name. For example, BRCG*x* refers to BRCG[1–8], and M*x*MR refers to the MAMR/MBMR/MCMR registers.

On the MSC711x devices, the SC1400 cores are 16-bit DSP processors. The following table shows the SC1400 assembly language data types. For details, see the *StarCore SC1000-Family Processor Core Reference Manual* (order number 10180).

SC1400 Core Assembly Data Types

Name	SC1400
Byte/Octet	8 bits
Half Word	8 bits
Word	16 bits
Long/Long Word/2 Words	32 bits
Quad Word/4 Words	64 bits

The following table lists the SC140 C language data types recognized by the StarCore C compiler. For details, see the *StarCore SC100 C Compiler User's Manual (MNSC100CC/D)*.

SC140 C Language Data Types and Sizes

Name	Size
char/unsigned char	8 bits
short/unsigned short	16 bits
int/unsigned int	16 bits
fractional short	16 bits
long/unsigned long	32 bits
fractional short	32 bits
pointer	32 bits

Conventions for Registers

The Programming Model section of each chapter includes a register bit table for each register in that module, as well as a table describing each bit in the register. The register bit table not only shows the names and positions of the bits/bit fields but also their reset value and their type (Read/Write). The register address is shown with the register name and mnemonic. Reserved bits/fields are indicated with a long dash (—). In the IEVENT register, notice that some of the bits are read/write (R/W), and others are read-only (R). Though none of the IEVENT bits are write-only (W), this is another option in the MSC711x registers. Notice also that in the MSC711x devices, the most significant bit (MSB) is 31, whereas on the MSC8101 device, the MSB is 0. That is, the MSC711x register bits have a big-endian order, whereas the MSC8101 register bits have a little-endian order.

EDIS Error Disabled Register 0x004

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	HBERR	BABR	BABT	GRA	TFINT	TXB	RFINT	RXB	MII	—	LC	CRL	TFU	—		
TYPE	R/W									R	R/W			R	R	R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Organization of this Manual

Following is a summary and brief description of the chapters in this manual:

- **Chapter 1**, *MSC711x Overview*. Features, descriptive overview of main modules, configurations, and application examples. A block diagram of each MSX711x device is included. A comparison table summarizes the differences between the devices in the MSC711x family.
- **Chapter 2**, *Signal Pins and Pinouts*. For each MSC711x device, identifies the external signals, lists signal groupings, including the number of signal connections in each group, and describes each signal within a functional group.
- **Chapter 3**, *SC1400 Core Overview*. Target markets, features, overview of development tools, descriptive overview of main modules.
- **Chapter 4**, *Extended Core*. Describes the structure of the extended core, which includes the SC1400 core and its internal memory (M1). Surveys the modules of the extended core, including the controller, extended core interface (ECI) system, the AMEC bus, ICache and instruction fetch unit, and programming model.
- **Chapter 5**, *Memory Map*. Defines the address space for the SC1400 core and for all MSC711x modules, including those not present on some MSC711x devices. Also considers different types of accesses, access restrictions, and big-endian operation.
- **Chapter 6**, *Crossbar Switch*. Describes the structure and function of the crossbar switch, which handles parallel transfers at the system level so that the various masters, including the DMA controller and the extended core interface, can get immediate service when they request resources.
- **Chapter 7**, *System Control*. Covers the system protection, software watchdog timer, and device configuration features of the system control unit.
- **Chapter 8**, *DMA Controller*. Describes the architecture, operation, and configuration of the DMA controller, which performs complex data transfers on 32 programmable channels with minimal intervention from a host processor.
- **Chapter 9**, *Memory Controller*. Describes the memory controller, which provides a glueless interface between the internal MSC711x bus and the external double data rate (DDR) SDRAM memory modules. The memory controller establishes an interface between MSC711x devices and external memories by translating internal bus accesses to appropriate address, data, and control signals for DDR SDRAMs.
- **Chapter 10**, *Memory Controller Interface*. Describes the how the memory controller interface works and how to program it. This interface increases the efficiency of accesses through the DDR memory controller to external DDR memory. It processes accesses from the crossbar switch to the DDR memory controller.
- **Chapter 11**, *Clocks and Power Management and Power Management*. Describes how to configure and use the MSC711x clock module and how to halt various MSC711x modules for low-power operation.

- **Chapter 12**, *Interrupt Processing*. Covers interrupt architecture, arbitration, vectors, and sources. The MSC711x interrupt system is optimized for real-time interrupts processing. Eight programmable priority levels and vectored interrupt servicing speeds up the processing.
- **Chapter 13**, *Reset*. Covers reset sources, causes, configurations, and timing.
- **Chapter 14**, *Boot Program*. Describes the bootloader program, which loads and executes source code that initializes an MSC711x device after it completes a reset and programs its registers for the required mode of operation. The chapter covers selection of bootloader modes and the normal sequence of events for bootloading a source program.
- **Chapter 15**, *Event Port*. Covers the architecture, functionality, and configuration of the event port, which works closely with the debug port (OCE10 emulator) and internal timers to manage internal and external MSC711x events.
- **Chapter 16**, *Debugging*. Describes the structure and functionality of the OCE10 on-chip emulator and the JTAG port to which it provides access.
- **Chapter 17**, *Programmable Address Detection*. Programmable address detection enables you to allocate regions of memory as data and program memory. There are programmable units for the extended core and for peripherals:
- **Chapter 18**, *Fast Ethernet Controller (FEC)*. Describes the architecture and FEC operation. The FEC supports 10/100 Mbps Ethernet as defined by **IEEE** 802.3. It has two MAC-PHY interfaces: the-media independent interface (MII) and the reduced MII (RMII) to provide MII functionality on a reduced pin count (10 instead of 18). The MSC711x FEC does *not* support the 7-Wire protocol.
- **Chapter 19**, *Time-Division Multiplexing (TDM) Interface*. The Time-Division Multiplexing Interface (TDM) is a full-duplex serial port by which DSP devices communicate with a variety of serial devices, including industry-standard framers, codecs, other DSPs, and microprocessors. The number of TDM modules present differs across MSC711x devices, but the functionality of the modules is identical.
- **Chapter 20**, *Host Interface (HDI16)*. Covers the architecture, functionality, and configuration of the HDI16 interface, which is a 16-bit wide, full-duplex, double-buffered parallel port that can directly connect to the data bus of a host processor. The HDI16 supports a variety of buses and gluelessly connects with a number of industry-standard microcomputers, microprocessors, and DSPs
- **Chapter 21**, *Timers Module*. The Quad Timer (TMR) module contains four identical counter/timer groups that serve as frequency dividers, clock generators, and event counters. There are two 16-bit counter/timer groups, timer module A and timer module B.
- **Chapter 22**, *I2C Software Module*. I^2C module is designed to be compatible with the standard Phillips I^2C bus protocol. This chapter details how the I^2C integrates into the MSC711x architecture. I^2C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This

bus is suitable for applications requiring occasional communications over a short distance between many devices.

- **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)*. The universal asynchronous receiver/transmitter (UART), also known as the serial communication interface (SCI), provides a full-duplex port for serial communications with other MSC711x devices, microprocessors, or DSPs.
- **Chapter 24**, *General-Purpose Input/Output (GPIO)*. Discusses the four GPIO ports: A, B, C, and D. Each signal in the I/O ports can be configured as a GPIO signal or as a dedicated peripheral interface signal. Port A is unique because a subset of its signals can generate interrupts to the interrupt controller.
- **Appendixes:**
  - **Appendix A,** *System Usage and Tuning/Programming Reference*.
  - **Appendix B**, *Boot Code*.

Other MSC711x Documentation

You can find the following documents on the web site listed on the back cover of this manual.

- *MSC7110 Technical Data sheet (MSC7110/D)*. Details the signals, AC/DC characteristics, clock configuration and signal characteristics, package and pinout, and electrical design considerations of the MSC7110 device.
- *MSC7112 Technical Data sheet (MSC7112/D)*. Details the signals, AC/DC characteristics, clock configuration and signal characteristics, package and pinout, and electrical design considerations of the MSC7112 device.
- *MSC7113 Technical Data sheet (MSC7113/D)*. Details the signals, AC/DC characteristics, clock configuration and signal characteristics, package and pinout, and electrical design considerations of the MSC7113 device.
- *MSC7115 Technical Data sheet (MSC7115/D)*. Details the signals, AC/DC characteristics, clock configuration and signal characteristics, package and pinout, and electrical design considerations of the MSC7115 device.
- *MSC7116 Technical Data sheet (MSC7116/D)*. Details the signals, AC/DC characteristics, clock configuration and signal characteristics, package and pinout, and electrical design considerations of the MSC7116 device.
- *Application Notes*. Cover various programming topics related to StarCore DSP core and the MSC711x devices.

Further Reading

You can find the following documents on the web site listed on the back cover of this manual:

- *SC1000-Family Processor Core Reference Manual* (document 10180). Covers the SC1400 core architecture, control registers, program control, and instruction set.
- *OCE10 On-Chip Emulator Reference Manual* (document 10055).
- *StarCore SC100 Application Binary Interface Reference Manual (MSC100ABI/D)*.

MSC711x Overview 1

The MSC711x family of highly integrated DSPs targets high-bandwidth computationally intensive DSP applications and is optimized for Enterprise class packet telephony applications. These processors deliver enhanced performance while maintaining low power dissipation and greatly reducing system cost.

At the heart of the system is the StarCore™ SC1400 core, providing the processing power for intensive numeric processing. The four ALUs in the SC1400 core work together to deliver 1200 million multiply and add commands per second (MMACS) performance with an internal 300 MHz clock at 1.2 V. An extended core services the SC1400 bandwidth requirements, with high speed zero wait state memory elements for both program and data accesses. In the extended core is a multi-ported 256 KB Level 1 internal memory (M1) for both high speed program and data storage. A 16 KB, 16-way instruction cache (ICache) provides an instruction stream to the core with no wait states on cache hits. The efficiency of the ICache is greatly enhanced by an intelligent fetch unit with advanced features for real-time processing. A 4-entry write buffer allows the SC1400 core to continue processing while the write buffer writes to locations outside the platform. A 192 KB Level 2 memory (M2) is also available on some MSC711x family devices for bursting to the ICache and for accesses from the extended core.

Each device in the MSC711x family is designed for optimal data flow to/from the SC1400 core. Wide full speed buses exactly match the busing requirements of the SC1400 core. Data is transferred to the DSP from either the external memory interface, the Ethernet controller on some devices, the host interface, or the TDM serial interfaces. The DMA controller transfers data through the bus switch from any of these ports to buffers in the internal memories. The SC1400 cores and other modules interconnect via a crossbar switch that manages rapid data transfer and storage between the MSC711x device, its internal components, and external devices. This multi-port crossbar switch allows multiple data transfers to occur in parallel outside the extended core. For example, the crossbar switch supports the following three operations in parallel: cache bursting from M2 memory, DMA data transfers from MSC711x memories, and atomic accesses by the SC1400 core to peripheral registers. The SC1400 core processes the data in the buffers and the result is transferred back to one of the ports.

The flexible 32-channel DMA controller transfers data to and from internal memory, external memory, peripherals, the host data interface, and the TDM interfaces. The DDR-RAM memory controller enables the SC1400 cores to access external memory devices with glueless accesses to DDR-RAM memory devices on the system bus.

The MSC711x architecture is optimized so that applications can efficiently use the available 1200 MMACS per second of the SC1400 core. For most applications:

- The data is accessed for a bounded number of times while the critical code runs in loops for many cycles. DSP applications have a high degree of code locality and a low degree of data locality.
- Different channels can share code but do not share data.
- A small portion of the code is run for most of the time (the “20–80” rule).

A typical application stores most of its program code in either M2 memory (if available) or in external memory (such as SDRAM memory). Since the DSP core typically spends most of the time running loops of selected routines, these time-critical routines can either be stored in the M1 memory or automatically fetched to the instruction cache. The high hit ratios achieved by the instruction cache prevents core stalls and thus boosts overall performance. During a miss, instructions are fetched from the M2 memory or from external memory through the crossbar switch. Since the miss ratio is very low, the probability of a collision with SC1400 accesses to the M2 memory or external memory is small. Therefore, the overall fetch latency is low.

1.1 Features

Table 1-1-1 lists the features that are common to all devices in the MSC711x family. **Table 1-1-2** lists the features that are specific to devices in the family.

Table 1-1. MSC711x Features

Feature	Description
StarCore™ SC1400 Core	• Up to 1200 million multiply-accumulates per second (MMACS) using an internal 300 MHz clock at 1.2 V. A multiply-accumulate operation includes a multiply-add instruction with the associated data move and pointer update. • 4 data ALUs. • 16 data registers, 40 bits each. • 27 address registers, 32 bits each. • Hardware support for fractional and integer data types. • Very rich 16-bit wide orthogonal instruction set. • Up to six instructions executed in a single clock cycle. • Variable-length execution set (VLES) that can be optimized for code density and performance. • **IEEE®** Std 1149.1™ JTAG port. • On-chip emulator (OCE10) module with real-time debugging capabilities: – 6 address breakpoint units. – 1 data breakpoint unit. – 8 KB trace buffer. – 62-bit counter. – on-chip emulator transmit and receive registers.

Table 1-1. MSC711x Features (Continued)

Feature	Description
Extended Core	The high performance extended core delivers up to 1200 MMACS using 4 ALUs running up to 300 MHz, including: • SC1400 core processor. • 256 KB multi-port SRAM (M1) accessed by the SC1400 core with no wait states. • 16 KB, 16-way instruction cache (ICache). • Programmable instruction fetch unit. • Write buffer (4-entry). • Extended core interface module.
Internal Memory	The large internal memory space totals up to 448 KB: • Up to 256 KB of M1 memory (device-specific). • 192 KB internal shared memory (M2), accessible from the SC1400 instruction fetch unit, extended core interface, and DMA controller via the crossbar switch (device-specific). • 16 KB ICache. • 8 KB boot ROM accessible from the SC1400 core.
External Memory Interface	• DDR memory controller: – Byte enables for up to 32-bit external data bus. – Glueless interface to 150 MHz 14-bit page mode DDR-RAM. – 14-bit external address bus supporting up to 1 GB. – 16- or 32-bit external data bus. • Memory controller interface supports: – Programmable buffer significantly improves efficiency through DDR memory controller. – Independent read buffers. – Programmable predictive read feature for each read buffer. – Write buffer.
Crossbar Switch	AHB-Lite crossbar switch, allowing up to four parallel data transfers between four master ports and six slave ports, where each port connects to an AHB-Lite bus: • Fixed or round robin priority independently programmable at each slave port. • Programmable bus parking at each slave port. • Low-power mode.
DMA Controller	Multi-channel DMA controller: • Up to 32 time-multiplexed channels. • Priority-based time-multiplexing between channels using 32 internal priority levels • Priorities can be fixed or round-robin. • Major-minor loop structure. • DONE or DACK protocol.
External Interfaces and Control Modules	External interfaces and control modules managed on the advanced peripheral bus (APB), including: • Time-division multiplexing (TDM) modules, each supporting up to 128 channels. • Software watchdog timer. • 16-bit host interface (HDI16). • System control. • RS-232 interface/universal asynchronous receiver/transmitter (UART). • General-purpose input/output (GPIO) signals. • Interrupt controller to handle external interrupt functions (input and output).
IPBus	Control modules on the IPBus include: • Programming model of the crossbar switch. • Programming model of the DMA controller. • Programming model of the DDR controller. • Programming model of the Ethernet MAC. • Clock synthesis module. • I^2C module. • System control unit. • Eight 16-bit timers.

Table 1-1. MSC711x Features (Continued)

Feature	Description
TDM Module	The MSC711x TDM module has the following features: • Totally independent receive and transmit, each having one data line, one clock line, and one frame sync line. • Frame sync line and/or clock line can be shared between receive and transmit. • Glueless interface to E1/T1 frames and MVIP, SCAS, and H.110 buses. • Hardware A-law/μ-law conversion • Up to 50 Mbps (50 MHz bit clock). • Maximum rate of the bit clock is 1/4 the core frequency. • Up to 128 channels. • Each channel can be programmed to be active or inactive. • 8- or 16-bit word widths. • The TDM Sync Signal (TDMxTFS/TDMxRFS) can be configured as either input or output. • The TDM clock signal (TDMxTCK/TDMxRCK) can be configured as either input or output. • Frame sync and data signals can be programmed to be sampled either on the rising edge or on the falling edge of the clock. • Frame sync can be programmed as active low or active high. • Selectable delay (0–3 bits) between the Frame Sync signal and the beginning of the frame. • MSB or LSB first support.
Host Interface (HDI16)	Enhanced 16-bit wide interface provides a glueless connection to industry-standard microcomputers, microprocessors, and DSPs. The HDI16 can also operate with an 8-bit host data bus, making it fully compatible with the DSP56300 HDI08 from the external host side.
Ethernet MAC Interface (Device-Specific)	• Designed to comply with **IEEE®** Std. 802.3™, 802.3u™, 802.3x™, and 802.3ac™. • Internal receive and transmit FIFOs and a FIFO controller. • Direct access to internal memories via its own DMA controller. • Support for 10/100 Mbps and 10 Mbps media independent interfaces (MIIs) and 10/100 Mbps reduced media independent interface (RMII). • Support for 10Mbps 7-Wire mode. • Full and half duplex operation. • Programmable maximum frame length. • Virtual local area network (VLAN) tag and priority support. • Retransmission of transmit FIFO following collision. • CRC generation and verification for inbound and outbound packets. • Address recognition including promiscuous, broadcast, individual address. hash/exact match, and multicast hash match. • Integrated FIFO controller and integrated DMA controller. • Ethernet statistics capturing.

Table 1-1. MSC711x Features (Continued)

Feature	Description
Timers	Two identical quad timer modules, each with four 16-bit counter groups, have the following features: • Timers clocked from: – Primary and secondary clock inputs. – External event counting. – Cascadable operation. • Multiple counting modes: – Basic counting. – Dual-edge counting. – Gated count. – Quadrature count. – Signed up/down count. – Triggered count. • Capture and compare capability. • Broadcast mode. • Maximum rate is 1/4 the core frequency. • Tightly coupled with the event port. • Selectable interrupts: – Overflow. – Edge. – Compare, compare 1, compare 2.
UART	• Two signals for transmit data and receive data. • No clock, asynchronous mode. • Full-duplex operation. • Standard mark/space non-return-to-zero (NRZ) format. • 13-bit baud rate selection. • Programmable 8-bit or 9-bit data format. • Separately enabled transmitter and receiver. • Programmable transmitter output polarity. • Two receiver wake-up methods: • Idle line wake-up. • Address mark wake-up. • Separate receiver and transmitter interrupt requests. • Eight flags, the first five can generate interrupt request: – Transmitter empty. – Transmission complete. – Receiver full. – Idle receiver input. – Receiver overrun. – Noise error. – Framing error. – Parity error. • Receiver framing error detection. • Hardware parity checking. • 1/16 bit-time noise detection. • Maximum bit rate 5.0 Mbps. • Single-wire and loop operations.
I^2C Port	• 2-wire serial interface through GPIO. • Filtered inputs for noise suppression. • Compatibility with I^2C bus standard up to 100 kbps for standard mode and up to 400 kbps for Fast mode. • Bidirectional data transfer protocol. • Multiple-master operation that also allows any number of devices implementing the I^2C master software module to access the memory simultaneously at boot or any time. • Compatible with the I^2C-serial EEPROM access protocol, allowing memory access of up to one MB.

Table 1-1. MSC711x Features (Continued)

Feature	Description
General-Purpose I/O (GPIO) Port	Bidirectional signal lines that either serve the peripherals or act as programmable I/O ports. Each port can be programmed separately to serve up to two dedicated peripherals. Port A lines can also be programmed as interrupt request inputs.
Programmable Interrupt Controller (PIC)	Consolidates maskable interrupt and non-maskable interrupt sources.
System Control	• Software watchdog timer function. • Bus programmable time-out monitors on AHB-Lite slave buses. • Bus error detection and programmable time-out monitors on AHB-Lite master buses. • Address out-of-range and misaligned access detection on crossbar switch buses.
Internal PLL	Generates up to 300 MHz clock for the SC1400 core and up to 150 MHz for the crossbar switch, DMA channels, M2 memory, and other peripherals.
Clock Synthesis Module	• Pre-division on PLL input clock. • Independent clocking of the internal timers and DDR module. • Programmable operation in the SC1400 low power Stop mode. • Independent shutdown of different regions on the device.
Reduced Power Dissipation	• Very low power CMOS design. • Separate power supply for internal logic and I/O. • Low-power standby modes. • Optimized power management circuitry (instruction-dependent, peripheral-dependent, and mode-dependent).
fieldBIST™ Hardware Diagnostics	Detects and provides visibility into unlikely field failures for systems with high availability. The Freescale unique fieldBIST ensures that the device: • Has structural integrity. • Operates at the rated speed. • Is free from reliability defects. Diagnostics can report partial or complete device inoperability. fieldBIST resolution can pinpoint the following uniquely: • 6 memory blocks, including ROM. • 3 logic levels (top, extended core, and peripherals). • 1 PLL. Simple JTAG interface allows easy integration to system firmware.
Packaging	• 400-pin MAP-BGA. • 17 × 17 mm, 0.8 mm pitch. • Lead-bearing or Pb-free.
Event Port	• Collects important signals on the device: – EVNT pins – DMA request, start, and done signals. – Interrupt request signals.L • Signals are combined as programmed by the user to provide triggering to on-device units such as interrupts, breakpoints, DMA transfer requests, or wake-up from low-power stop mode. • Units can operate independently, can be sequenced, or can be enabled by an outside source. • Can be used independently or in conjunction with the OCE10 emulator debug port. • Output to EVNTx pins.

Table 1-1. MSC711x Features (Continued)

Feature	Description
Programmable Address Detection	• Four user-programmable SC1400 core address detection units (program and data accesses). • Four user-programmable DMA address detection units. • Four user-programmable Ethernet MAC address detection units. • Each detection unit supports: – Programmable range or value detection on the unit buses. – Optional generation of maskable/non-maskable interrupt on core detection units. – Optional generation of event trigger. – Status of detections captured in status register. • Programmable out-of-range detection, patching, or user-programmable error detection.
Software Support	Real-time operating systems (RTOS) that fully supports MSC711x device architecture (multi-core, memory hierarchy, ICache, timers, DMA, interrupts, peripherals). This operating system, called SmartDSP OS, is bundled with CodeWarrior™: • High-performance and deterministic, delivering predictive response time. • Optimized to provide low interrupt latency with high data throughput. • Preemptive and priority-based multitasking. • Fully interrupt/event driven. • Small memory footprint. • Comprehensive set of APIs. • Fully supports MSC711x DMA, interrupts, and timer schemes. Distributed system support, enables transparent inter-task communications: • Messaging mechanism between tasks using mailboxes and semaphores. • Networking support; data transfer between tasks running inside and outside the device using networking protocols. • Includes integrated device drivers for such peripherals as TDM, UART, Ethernet, and external buses.
	Additional features: • Incorporates task debugging utilities integrated with compilers and vendors. • Board support package (BSP) for MSC711xADS.
	CodeWarrior Integrated Development Environment (IDE): • C/C++ compiler with in-line assembly. Enables the developer to generate highly optimized DSP code. It translates code written in C/C++ into parallel fetch sets and maintains high code density. • Librarian. Enables the user to create libraries for modularity. • C libraries. A collection of C/C++ functions for the developer's use. • Linker. Highly efficient linker to produce executables from object code. • Debugger. Seamlessly integrated real-time, non-intrusive multi-mode debugger that enables debugging of highly optimized DSP algorithms. The developer can choose to debug in source code, assembly code, or mixed mode. • Profiler. An analysis tool using a patented Binary Code Instrumentation (BCI) technique that enables the developer to identify program design inefficiencies.
Boot	Booting from on-device peripherals: • Boot from HDI16 and I^2C. • Boot also from serial SPI Flash/EEPROM devices using software in the boot ROM to access SPI memory devices. • Different clocking options allow for boot operation with the PLL ON/OFF, as well as with different input frequency ranges.

Table 1-1. MSC711x Features (Continued)

Feature	Description
Application Development System (ADS) Board	• Host debug through single JTAG connector supports both processors. • Two kinds of ADS configurations: one with the MPC8272 device as the host CPU and one without a host CPU. • Big Flash memory for stand-alone applications. • Support for the following communications ports: – 10/100Base-T. – 155 Mbit ATM over optical. – T1/E1 TDM interface. – H.110. – Voice codec. – RS-232. – High-density (MICTOR) logic analyzer connectors to monitor MSC711x signals. – 6U CompactPCI form factor. • Emulates DSP farm by connecting to three other ADS boards.
Low-Cost General-Purpose EVM Board	• 32 MB of DDR SDRAM memory. • 16-bit audio codec (3.5 mm jacks). • 256 KB I^2C EEPROM. • TDM interface. • Fast Ethernet. • Host port interface. • JTAG interface. • RS-232 interface.

Table 1-1-2 shows the device-specific features of the MSC711x family. Notice that the devices differ by size of internal memory, number of TDM interfaces, and whether they have an Ethernet interface.

Table 1-2. MSC711x Device-Specific Feature Comparison

Feature	MSC7110	MSC7112	MSC7113	MSC7115	MSC7116
Size of M1 memory	64 KB	192 KB	192 KB	192 KB	192 KB
M2 memory and memory bridge	0	0	0	192 KB	192 KB
Number of TDM interfaces	1	2	2	3	2
Ethernet interface	0	0	RMII/MII	0	RMII/MII

1.2 MSC711x Block Diagrams

This section shows the block diagrams of the MSC711x devices, which are as follows:

- MSC7110, see **Figure 1-1** on page 9.
- MSC7112, see **Figure 1-2** on page 10
- MSC7113, see **Figure 1-3** on page 11
- MSC7115, see **Figure 1-4** on page 12
- MSC7115, see **Figure 1-5** on page 13

Note: The arrows on the buses describe the direction of the address flow.

Figure 1-1 shows the MSC7110 block diagram. Device-specific features of the MSC7110 device are its M1 memory size of 64 KB, and single TDM interface. Notice that the MSC7110 device has no M2 memory and no Ethernet interface.

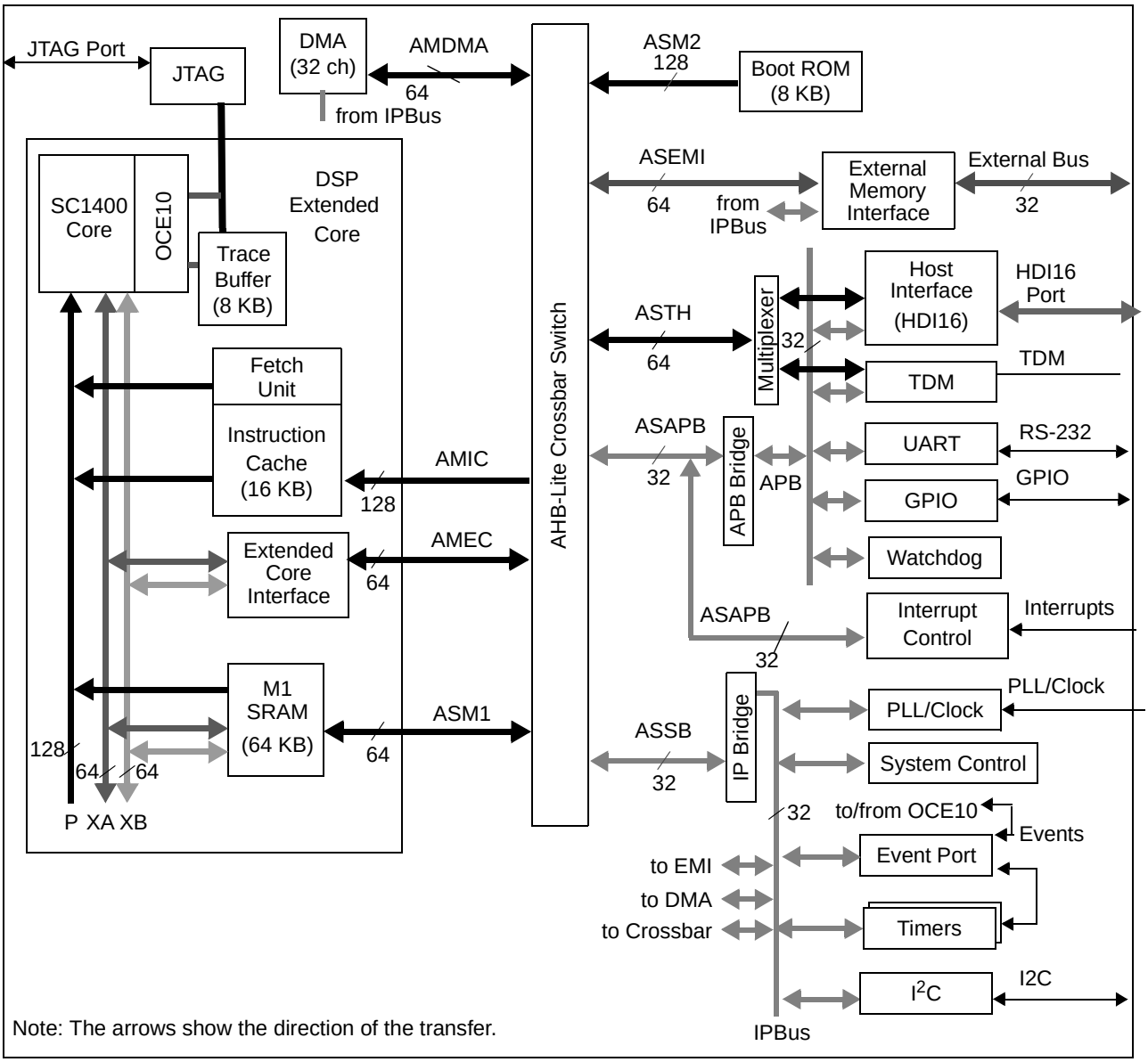


Figure 1-1. MSC7110 Block Diagram

Figure 1-2 shows the block diagram of the MSC7112 device. Device-specific features of the MSC7112 device are its two TDM interfaces. Notice that the MSC7112 device has no M2 memory and no Ethernet interface.

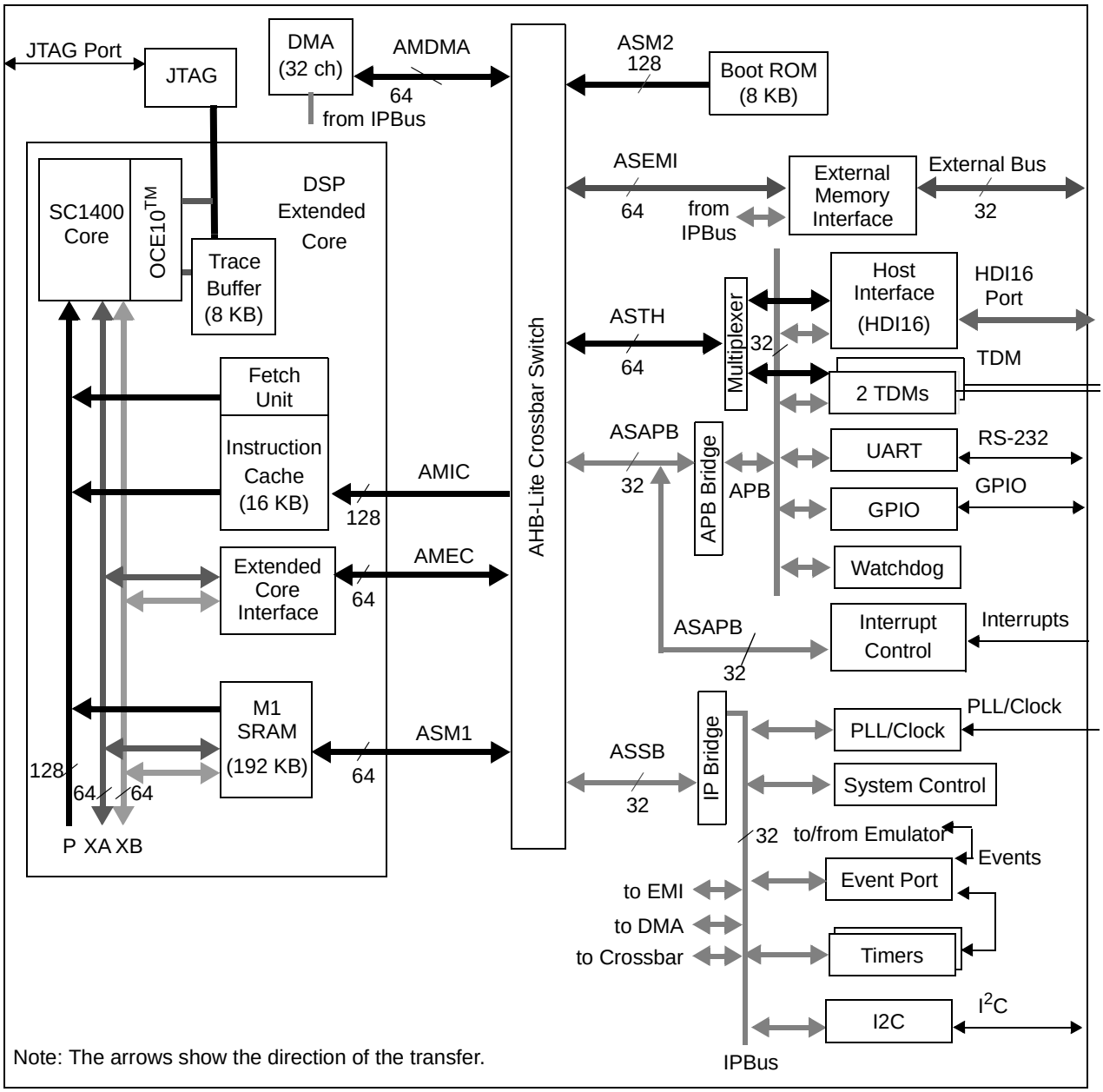


Figure 1-2. MSC7112 Block Diagram

Figure 1-3 shows the block diagram of the MSC7113 device. Device-specific features of the MSC7113 device are its two TDM interfaces and Ethernet interface. Notice that the MSC7113 device has no M2 memory.

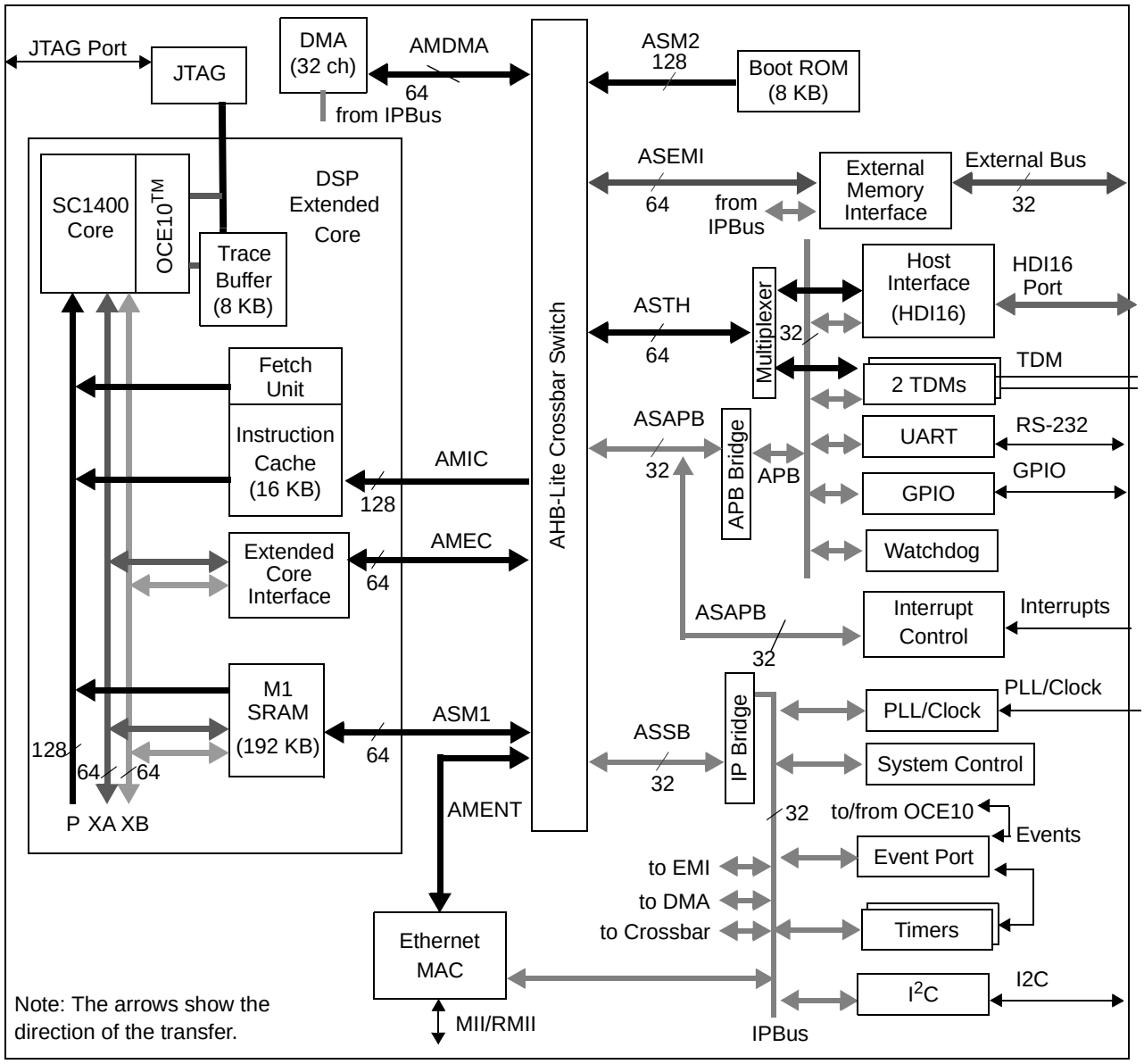


Figure 1-3. MSC7113 Block Diagram

Figure 1-4 shows the block diagram of the MSC7115 device. Device-specific features of the MSC7115 device are its three TDM interfaces and M2 memory. Notice that the MSC7115 device has no Ethernet interface.

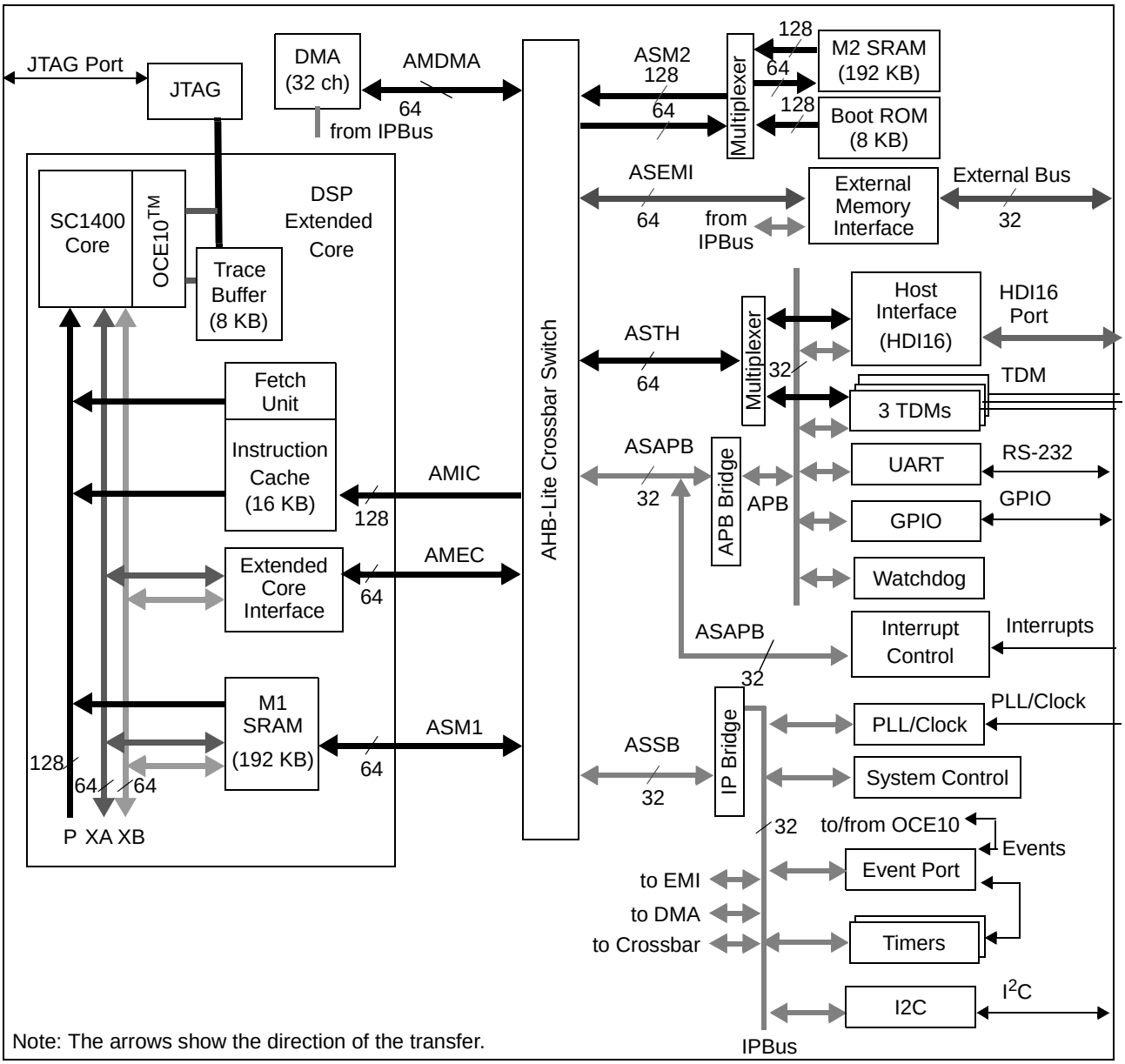


Figure 1-4. MSC7115 Block Diagram

Figure 1-5 shows the block diagram of the MSC7116 device. Device-specific features of the MSC7116 device are its two TDM interfaces, Ethernet interface, and M2 memory.

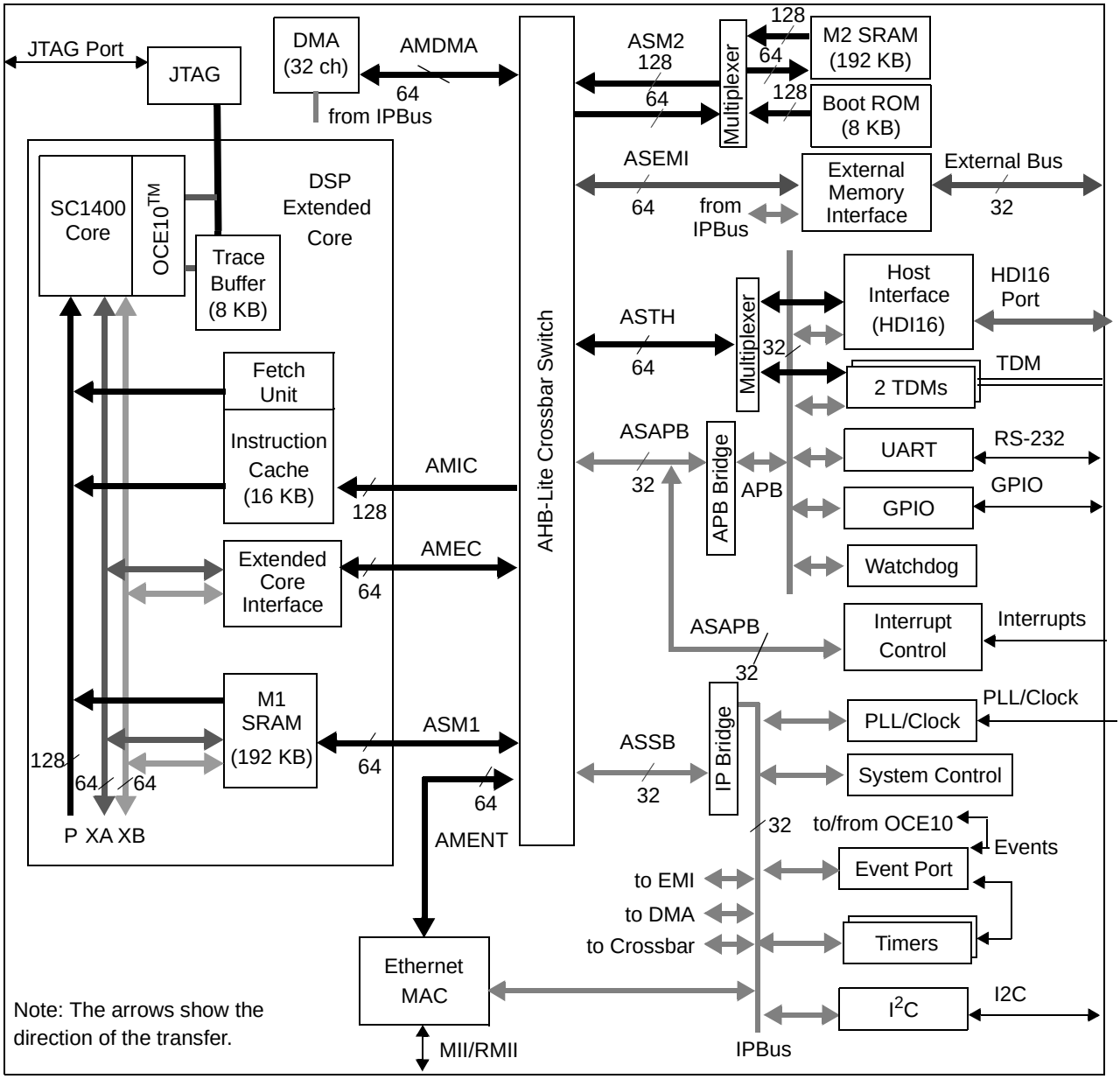


Figure 1-5. MSC7116 Block Diagram

Figure 1-6 shows the block diagram of the MSC7118 device. Device-specific features of the MSC7118 device are its three TDM interfaces and M2 memory.

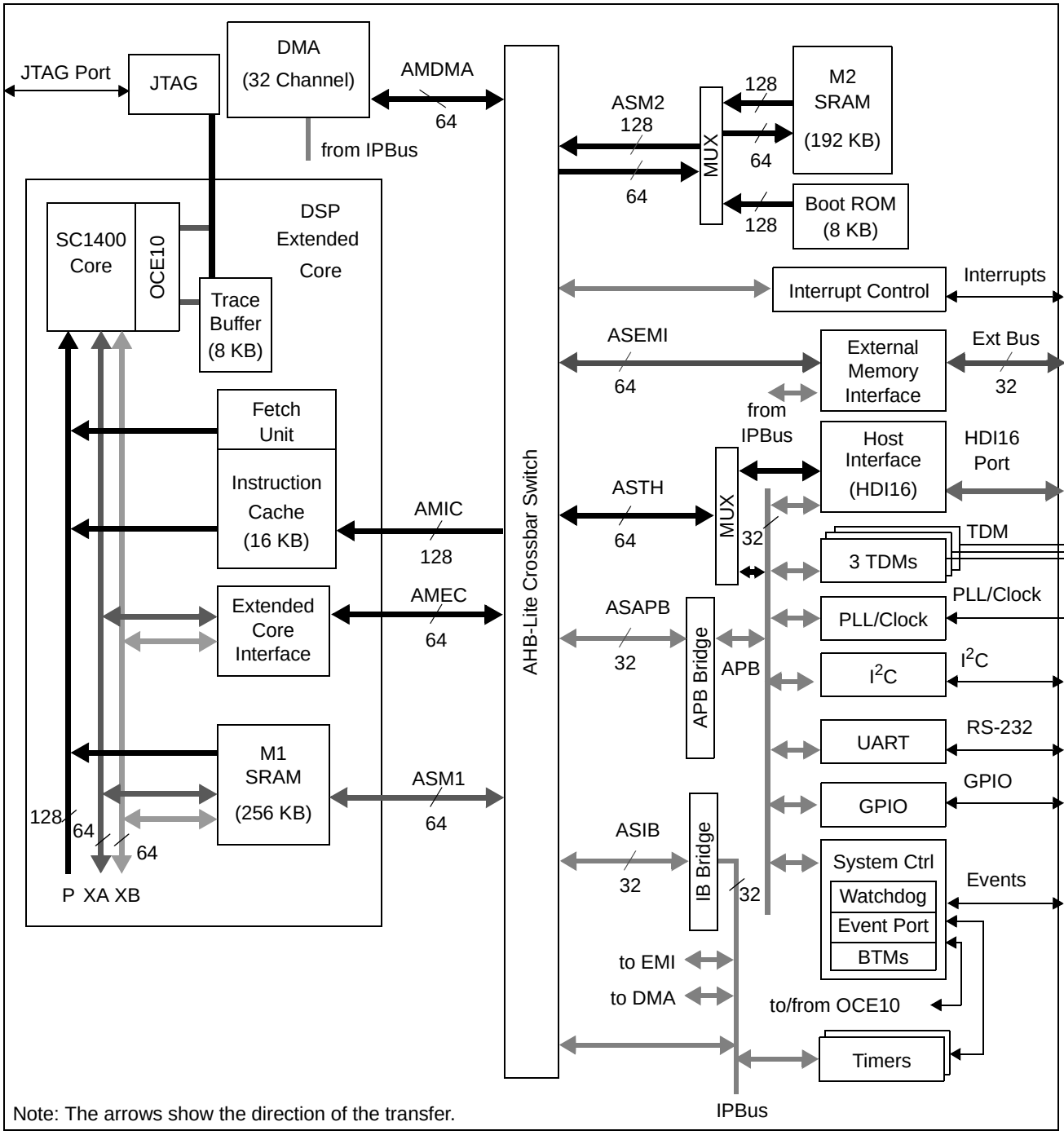


Figure 1-6. MSC7118 Block Diagram

Figure 1-7 shows the block diagram of the MSC7119 device. Device-specific features of the MSC7119 device are its two TDM interfaces, Ethernet interface, and M2 memory.

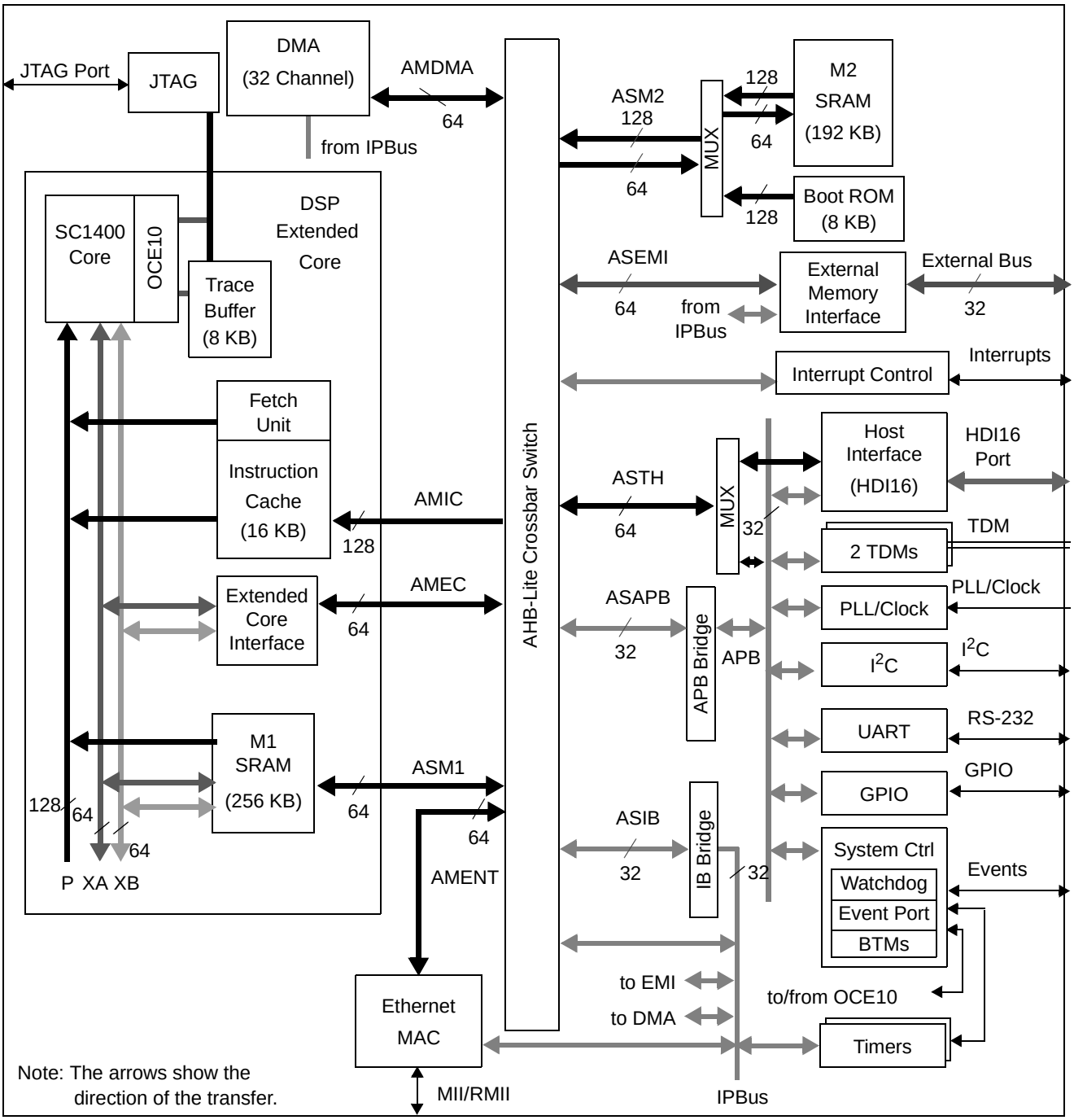


Figure 1-7. MSC7119 Block Diagram

1.3 Bus Architecture

The SC1400 cores and other MSC711x modules interconnect via a crossbar switch designed for optimal data flow when data is transferred and stored between an MSC711x device, its internal components, and external devices. The buses are grouped into the following categories:

- SC1400 core buses
- Crossbar master port buses
- Crossbar slave port buses
- Peripheral buses
- External buses

1.3.1 SC1400 Core Buses

The SC1400 core can access the M1 memory, ICache, and write buffer with zero wait states via its internal 128-bit instruction bus and two 64-bit data buses. The SC1400 buses include:

- *PAB.* A 32-bit program address bus that allows the SC1400 core to specify program addresses in the local unified memory (M1).
- *PDB.* A 128-bit program data bus that transfers program data from the ICache or M1 memory.
- *XABA and XABB.* Two 32-bit address buses to specify data locations in M1 memory for the two DSP data streams required for MAC operations.
- *XDBA and XDBB.* Two 64-bit data buses to transfer data values to and from M1 memory.

1.3.2 Crossbar Master Port Buses

Following are the masters to the crossbar switch (see **Figure 1-5**):

- *AHB master instruction cache (AMIC).* A 128-bit wide, read-only, single-master bus connecting the instruction fetch unit to the crossbar switch. The instruction fetch unit is the master and the crossbar switch is a slave. When the instruction cache misses, the fetch unit issues an address outside the platform to M2 memory or external memory to begin a cache burst. Accesses are pipelined.
- *AHB master extended core (AMEC).* A 64-bit wide, single-master bus connecting the extended core interface to the crossbar switch. The bus switch and write buffer within the extended core interface are multiplexed to form a single master, and the crossbar switch is a slave. When the SC1400 core accesses locations outside the extended core, this unit controls the access. Accesses are pipelined.
- *AHB master DMA (AMDMA).* A 64-bit wide, single-master bus connecting the DMA controller to the crossbar switch. The DMA controller is the master and the crossbar switch is a slave. When the DMA controller initiates transfers within the DSP device, this bus is used to transfer data. Accesses are pipelined.

- *AHB master Ethernet MAC (AMENT)*. A 32-bit wide, single-master bus connecting the Ethernet DMA to the crossbar switch. The Ethernet controller is the master and the crossbar switch is a slave. When the Ethernet DMA controller initiates transfers within the DSP device, this bus is used to transfer data. Accesses are pipelined. All accesses are a maximum of 32-bits.

Note: The master port buses run at the frequency of the AHB clock.

In addition to their connection to the APB, the HDI16 and TDM peripherals interface to the crossbar switch for high-speed data transfers. The register files of the DMA controller and crossbar switch are accessed through the IPBus. The register files of the ICache and extended core interface are accessed through an internal bus within the extended core interface.

Note: The interrupt controller registers are located within the APB address space but are accessed directly off the ASAPB bus, not the APB bus.

1.3.3 Crossbar Slave Port Buses

Following are the slaves to the crossbar switch:

- *AHB slave to M1 (ASM1)*. A 64-bit wide, single-master bus connecting the crossbar switch to the extended core memory (M1). The crossbar switch is the master and the memory is the slave. The DMA channels can access M1 memory. Accesses are pipelined.
- *AHB slave to M2 (ASM2)*. A single-master bus that accommodates 128-bit reads and 64-bit writes, s connecting the crossbar switch to the internal memory (M2) and boot ROM. The crossbar switch is the master and the memories are the slaves. The SC1400 core can access M2 memory via normal accesses through the write buffer. The instruction fetch unit can burst from M2 or boot ROM into the ICache. In addition, the DMA controller accesses this memory for bursts between M2 and the M1 and external memory. Accesses are pipelined.
- *AHB slave to external memory interface (ASEMI)*. A 64-bit wide, single-master bus connecting the crossbar switch to the MSC711x external memory interface. The crossbar switch is the master and the interface is the slave. The switch accesses external devices through this interface. Accesses are pipelined.
- *AHB slave to TDM/HDI16 interfaces (ASTH)*. A 64-bit wide, single-master bus connecting the crossbar switch to the HDI16 and TDM peripherals. The crossbar switch is the master and the interface is the slave. This bus provides a high speed port for accessing data coming through these peripherals. Accesses are pipelined.
- *AHB slave to APB (ASAPB)*. A 32-bit wide, single-master bus connecting the crossbar switch to the interface bridge for the APB bus. The crossbar switch is the master and the bridge is the slave. The switch accesses peripherals on the APB bus through this bridge. Each APB access requires two clocks. Accesses are typically pipelined.

- *AHB slave to IPBus (ASSB)*. A 32-bit wide, single-master bus connecting the crossbar switch to the interface bridge for the IPBus. The crossbar switch is the master and the bridge is the slave. The switch accesses peripherals on the IPBus through this bridge. Accesses are typically pipelined.

Note: Interrupt controller registers are located directly on the ASAPB bus and not on the APB bus. Accesses to these registers do not pass through the bridge. The slave port buses run at the frequency of the AHB clock.

1.3.4 Peripheral Buses

- *APB*. A 32-bit wide bus controlled by the master that connects the crossbar switch to device peripherals. This bus accesses the control and status registers of the HDI16, TDM, UART, PLL, and GPIO signals. Accesses issued by the SC1400 core pass through the extended core interface, the crossbar switch, the APB interface bridge, and finally reach the peripheral registers.
- *IPBus*. A 32-bit wide bus controlled by the master that connects the crossbar switch to device peripherals. This bus accesses the control and the status registers of the DMA controller, crossbar switch, system control, and timer module. Accesses issued by the SC1400 core pass through the extended core interface, the crossbar switch, the IPBus interface bridge, and finally reach the peripheral registers.

Note: The peripheral buses run at the frequency of the APB clock and IPBus clock, respectively. APB accesses require two clocks per transfer.

1.3.5 External Buses

The external bus interface provides access to external DDR-RAM drivers. It is clocked at up to 100 MHz and supports 14-bit addressing, a 16-bit or 32-bit data bus, and bursting operations. The data bus can be accessed in 8-bit, 16-bit, and 32-bit data widths. The address and data buses support synchronous, one-level pipelined transactions.

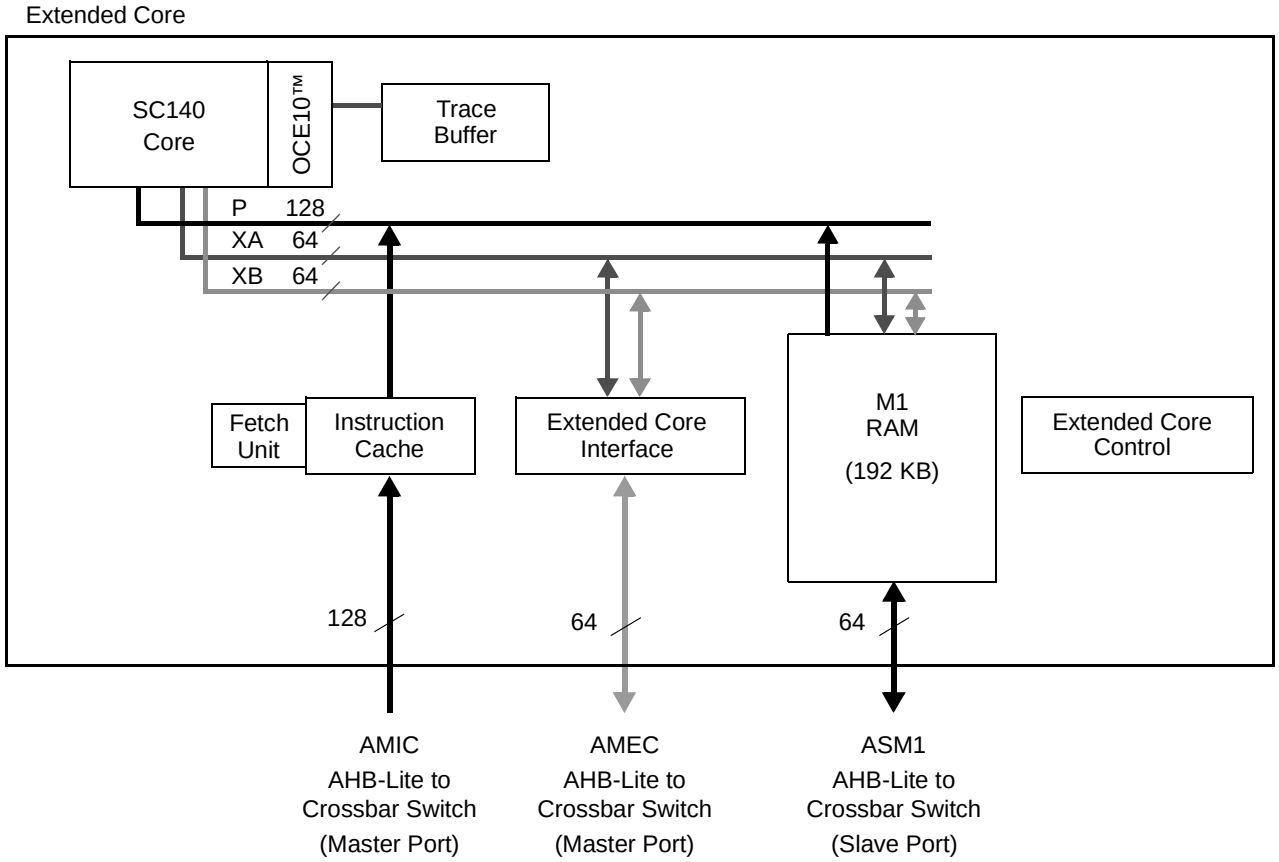
- *External data bus*. A 32-bit wide bus controlled by the external memory interface that connects the crossbar switch to the external 32-bit system bus. This bus runs at 150 MHz, where data is transferred at double rate for DDR-SDRAM (two values per clock).
  - 14-bit address bus.
  - 16-bit or 32-bit data.
  - Two, four or eight-beat burst transfers.
  - Support for external DDR-SDRAM.
- *Host data bus*. A 16-bit interface to connect to the data bus of an external host processor.
  - 8 or 16-bit data.
  - 4-bit address for selecting HDI16 registers from the external host.

1.4 Extended Core

The extended core, shown in **Figure 1-8**, contains the following:

- SC1400 core
- M1 memory
- ICache
- Instruction fetch unit
- Extended core interface (including a bus switch and write buffer)
- Extended core control unit

The SC1400 core accesses locations outside the extended core through the extended core interface.



Notes: 1. The arrows show the data transfer direction.
2. The extended core interface includes a bus switch and write buffer.

Figure 1-8. Extended Core System

1.4.1 SC1400 Core

The SC1400 core is a flexible, programmable DSP core that handles compute-intensive communications applications, providing high performance, low power, and code density. It

efficiently deploys a novel variable-length execution set (VLES) execution model, attaining maximum parallelism by allowing multiple address generation and data arithmetic logic units to execute multiple operations in a single clock cycle. The SC1400 core contains four ALU units, each with a 16-bit × 16-bit MAC that results in a 40-bit wide and 40-bit parallel barrel shifter. Each ALU performs one MAC operation per clock cycle. An address generation unit includes two address arithmetic units and one bit mask unit. There are also 16 address registers, eight of which can serve as base address registers.

The main reason for the high code density of the SC1400 is that its instructions are 16 bits wide. During each clock cycle, the SC1400 core reads eight instruction words, referred to as a *fetch set*. The SC1400 core identifies which instructions can be performed in parallel and runs them on the ALUs and address generation units. In one clock cycle, up to six instructions can execute (four ALU operations, and two address generation operations). In the rich instruction set, special attention is given to control code, making the SC1400 core ideal for applications embedding DSP and communications. Arithmetic operations are performed using both fractional and integer data types, enabling the user to choose a style of code development or use coding techniques derived from an application-specific standard. The SC1400 programming model is highly orthogonal, and both data and instructions reside in one unified memory. The SC1400 compiler translates code written in C/C++ into parallel fetch sets and maintains high code density and/or high performance by taking advantage of the high code orthogonality and unified memory architecture. For details, refer to the *SC1000-Family Processor Core Reference Manual* (order number 10108).

1.4.2 M1 Memory

The 256 KB M1 memory space is a full speed, zero wait state memory supporting parallel accesses from the SC1400 core. Up to three accesses can be performed concurrently on every SC1400 core clock cycle, one 128-bit instruction fetch set and two 64-bit data words. In addition, the DMA controller can simultaneously access a 64-bit word from M1 memory through the crossbar switch.

To optimize the memory size, the M1 memory is subdivided into different groups, each with a size of 64 KB. Each group has four ports and is implemented as a single-access memory. This subdivision allows four accesses to be performed, in parallel, to different groups. Parallel accesses can also occur when the two SC1400 data accesses occur to the same group. When a collision occurs due to two or more accesses to the same memory group, the SC1400 core stalls for one or more core clock cycles. Intelligent memory allocation significantly decreases collisions between an SC1400 core bus and the DMA bus. For example, two accesses cannot collide if they belong to different memory groups, which is usually the case since program code is stored in a different group than the data. The DMA controller stores the “next” buffers in yet a different group. Even in the same group, placing two data elements on a different module prevents a collision between two SC1400 core buses.

The overall memory size available for one SC1400 core in internal memory and the partition between the memories is a trade-off between chip size and the memory requirements imposed by the bandwidth of the SC1400 core. Typically, the M1 memory contains channel data and may also contain a few critical routines and M2 (if available) contains program code that is automatically burst into the cache when needed.

1.4.3 Instruction Cache

The ICache is highly optimized for real-time DSP applications and minimizes miss ratios, latencies, bus bandwidth requirements, and silicon area. The 16 KB ICache is 16-way set associative. Each of the 16 ways contains four 256-byte lines and is divided into 16 fetch sets, each with an associated valid bit. The 2-bit set field of the address selects one of four sets. The line with a tag that matches the tag field of the address is the selected line. A cache miss occurs when the fetch set accessed by the SC1400 core does not reside in the ICache, and a cache burst is initiated to bring in the requested data. The following cache bursting parameters can be tuned for your application:

- *Burst Size*. 1 or 4 fetch sets (16 or 64 bytes)
- *Primary set size*. 1, 2, or 4 fetch sets (16 or 64 bytes)
- *Prefetch to end of line*. Enable/Disable

When a cache miss occurs, the new data is fetched in bursts using one of these burst sizes. There is also an option to fetch to the end of the line, which takes advantage of the spatial locality of the code.

Note: The ICache does not burst in an entire line of the cache but instead bursts in only a portion of the line, 1/16th or 1/4th of a cache line. The ICache is optimized for real-time performance and its line size, 256 bytes, is larger than traditional implementations. This cache has a valid bit for each group of 16 bytes instead of one valid bit for the entire line, as in a typical cache.

When there is a need to fetch new data to the cache and the cache is full, one line of the cache is removed using the least recently used (LRU) algorithm. The cache can be programmed so that only part of it is thrashed. For example, if task A needs to be preempted in favor of task B, the instructions of task A are thrashed from the instruction cache while task B runs. When task B finishes and task A takes over, task A may not find its most recently used instructions in the cache. To prevent such a situation, the DSP software can exclude the ways of task A from the part of the cache that can be thrashed. Another method of guaranteeing that the critical routines are always available for a task is to store them in the SC1400 core M1 private memory. All ICache entries are flushed by issuing a cache flush command from the SC1400 core, which is useful, for example, when new code is written to lines in the M2 memory that are already cached. The ICache has run-time debug support. A counter in the emulator (OCE10) and debug module is incremented for cache hits and misses. When the SC1400 core is in Debug mode, its fetch unit is in Debug mode and all the cache arrays can be read.

Since different channels do not typically share data, data is located in M1 memory. The architecture is flexible enough to store data in M2 memory as well, if M2 memory is available on the device. In fact, the DMA controller can perform data overlays between the M2 and the M1 memories. For example, while the data for channel N is processed, the DMA controller can bring in the data needed for channel N+1.

The SC1400 core accesses M2, external memory, and internal peripherals through the extended core interface, which routes its request through the crossbar switch. Cache line fills are performed through the instruction fetch unit, which also routes its request through the crossbar switch. This separation ensures that the latencies for SC1400 core accesses to the M2 memory remain as low as possible. Write accesses to resources with high latencies are typically routed through the write buffer. The write buffer can store the write access, release the SC1400 core, and complete the write operation it at a later time.

1.4.4 Instruction Fetch Unit

Program fetches to locations outside the M1 memory occur through the instruction fetch unit (IFU). The fetch unit (FU) is triggered by a cache “miss” access. It brings the data into the SC1400 core and continues to update the cache until the end of the burst or until the end of the cache line. For a burst to the end of a line, if a new miss occurs, the line fill is discontinued and the new miss is serviced. This improves the overall performance of the cache in the system. The FU initiates cache update requests for data of consecutive addresses after every miss. The block and burst sizes are configurable. The ability to continue bursts to the end of a cache line is also selectable.

1.4.5 Extended Core Interface

The extended core interface connects the extended core through the crossbar switch with the rest of the MSC711x device. The module handles the SC1400 accesses, bringing the data on the AHB-Lite bus. **Figure 1-9** shows the components of the extended core interface: a bus switch to handle data read and write operations, a write buffer to handle data write operations, a multiplexor for routing accesses from the write buffer and bus switch to resources outside the extended core, and internal busing for accessing the ICache and extended core interface registers. The write buffer and bus switch are multiplexed onto a single AHB-Lite bus. The extended core interface is controlled by the extended core control unit.

The extended core interface is the mechanism by which the SC1400 core communicates with modules outside the extended core. It handles the switching between the three core buses, P, XA, and XB, the write buffer, and the AMEC (an AHB-Lite bus) that goes to the crossbar switch. It operates at the same frequency as the SC1400 core. The memory controller within the extended core interface handles memory contentions to the M1 memory. It snoops the activity on the buses connected to the internal memory (M1) and, if necessary, freezes the SC1400 core and address bus activity. It creates the atomic instruction acknowledge to the SC1400 core during the reservation process.

The bus switch handles data accesses to locations outside the extended core: all read accesses and write accesses when the write buffer is disabled. When the SC1400 core writes to locations outside the extended core, stalls may occur while the core waits for the access to complete. To prevent such stalls, all external accesses are first written to the write buffer. The write buffer releases the SC1400 core and then completes the access when its destination becomes available. Located on the SC1400 core buses, the write buffer is a zero wait state client with a 4-entry FIFO that automatically handles data coherency problems. For example, if the write buffer contains data to be written to address A, and a read access occurs before the buffer completes the write access, the contents of the write buffer are written to the destination before the read can be executed. Not all writes beyond the M1 memory are routed through the write buffers. Write accesses do not use the write buffer in the following cases:

- The address of the destination belongs to a bank that is defined as immediate.
- It is an atomic operation essentially writing to a semaphore
- The write buffer is disabled.

The write buffer counts the number of clocks that elapse between the time data is written to the write buffer and the time it is emptied. When the counter exceeds a pre-programmed value, the contents of the write buffer are flushed to minimize the time for write accesses.

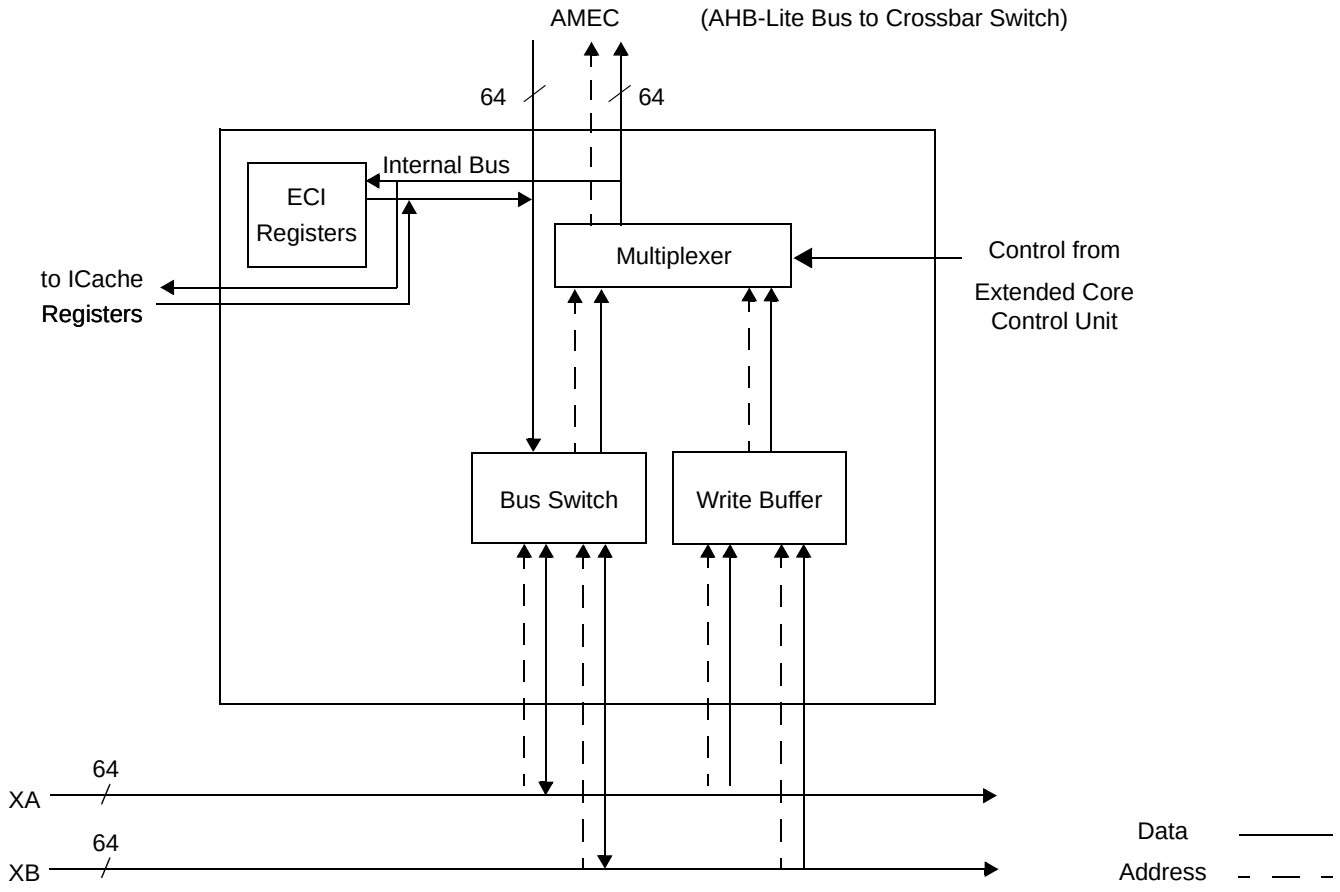


Figure 1-9. Extended Core Interface Block Diagram

Note: For details, see **Chapter 4**, *Extended Core*.

1.5 Direct Memory Access (DMA) Controller

The multi-channel DMA controller transfers data between device resources through the crossbar switch. Data transfers occur in two phases. First, data is read from the source to a DMA internal FIFO, and then it is written from that FIFO to its destination. Thirty-two internal DMA FIFOs support this mode. **Table 1-1-3** lists the possible DMA clients.

Table 1-3. DMA Clients

DMA Client	Direction	Size
M1 memory	Read/Write	8, 16, 32, or 64 bits
M2 memory, if available	Read/Write	8, 16, 32, or 64 bits
Boot ROM	Read Only	8, 16, 32, or 64 bits
External memory	Read/Write	8, 16, 32, or 64 bits
Host interface (DMA port)	Read/Write	8, 16, 32, or 64 bits
TDM interface (DMA port)	Read/Write	8, 16, 32, or 64 bits
APB peripherals: • Host interface • TDM interface • And so on	Read/Write	8, 16, or 32 bits
IPBus peripherals: • Crossbar registers • DMA registers	Read/Write	8, 16, or 32 bits

DMA requests are tied to up to 32 DMA channels that run concurrently. Each channel has programmable priority levels. The DMA controller supports a flexible buffer configuration, including simple buffers, cyclic buffers, single-address buffers (I/O device), incremental address buffers, chained buffers, complex buffers, and buffer alignment by hardware. DMA transfers to/from the Ethernet MAC are not performed by the DMA controller but are instead handled by a dedicated DMA unit within the Ethernet MAC unit.

Note: For details, see **Chapter 8**, *DMA Controller*.

1.6 Crossbar Switch

Communication between modules in the MSC711x architecture is greatly facilitated by the crossbar switch. All buses connecting to the switch are AHB-Lite 2.0.

The following modules are masters to the switch:

- Instruction fetch unit (128-bit read path)
- Extended core (64-bit read/write path)

- DMA controller (64-bit read/write path)
- Ethernet MAC (32-bit read/write path)

The following modules are slaves to the switch:

- M1 memory (64-bit read/write path)
- M2 memory (128-bit read/64-bit write path)
- External memory interface (64-bit read/write path)
- TDMs/HDI16 (64-bit read/write path)
- APB peripherals (32-bit read/write path)
- IPBus peripherals (32-bit read/write path)

Each slave port is individually programmable for fixed-priority or round-robin arbitration. Parking is also independently programmable at each slave port.

The crossbar switch makes it possible for data transfers at the system level to occur in parallel. The matrix provides powerful capabilities, supporting up to four parallel transfers, as follows.

- Parallel transfers when cache bursting from M2 memory:
  - Cache bursts in a portion of a cache line from M2 memory via the fetch unit.
  - DMA bursts in new data from external memory to the DMA controller.
  - Core accesses a peripheral via the extended core interface.
- Parallel transfers when cache bursting from external memory:
  - Cache bursts in a portion of a cache line from external memory via the fetch unit.
  - DMA bursts out new data from the DMA controller to M1 memory.
  - Write buffer writes to M2 memory via the extended core interface.

The combination of the DMA controller and crossbar switch ensures that the SC1400 core can focus on the intensive computational work while the DMA controller brings in new data buffers. Data can be prepared in the M1 (or M2) memory in a “next” buffer while the SC1400 core processes the current buffer. The SC1400 core can use the flexible DMA controller to transfer large blocks of data from the external memory to the internal memory and also between the internal memories.

Note: For details, see **Chapter 6**, *Crossbar Switch*.

1.7 System Control

The system control unit handles system start-up, initialization, monitoring, and protection. The system timers include a software watchdog timer and hardware bus monitors to ensure that accesses complete on important system buses. This unit also detects accesses to invalid portions of the memory map or write accesses to ROM blocks. The system control unit controls the event port, which takes input information from the EVNTx pins, system-level events such as DMA

transfers, and the OCE10 breakpoint logic. The signals can be combined as desired, such as ANDing, ORing, or sequencing, and the resulting action can be sent to the EVNTx pins, to the OCE10 breakpoint logic, or to the timer modules. The system control unit operates independently or in conjunction with the OCE10 debug port.

Note: For details, see **Chapter 7**, *System Control*.

1.8 Reset

Reset circuitry provides power-on reset, external hard and soft resets, software watchdog timer reset, and bus monitor reset.

Note: For details, see **Chapter 13**, *Reset*.

1.9 Boot ROM

The 8 KB boot ROM is used for booting the device out of reset. It allows for booting from different ports on the device. It provides execution packets on the 128-bit ASM2 bus. Immediately after reset, the core starts executing the code from the internal boot ROM. The value on the boot mode pins identifies the boot source and whether the PLL is enabled. Booting can occur from the following sources:

- HDI16.
- I^2C.
- SPI (using GPIO capability on the device)

Note: For details, see **Chapter 14**, *Boot Program*.

1.10 PLL and Clocks (PLL/Clock)

The clocking module clocks the entire device. An external clock is passed through a PLL providing both frequency division and multiplication. This clock is then used to generate internal system clocks. There is also a path for bypassing the PLL. A low frequency clock is generated directly from the input clock for clocking the timer modules and watchdog timer. A clock out signal can be enabled, if desired.

Note: For details, see **Chapter 11**, *Clocks and Power Management*.

1.11 Interrupt Scheme

A 120-channel interrupt controller collects interrupts from the device peripherals, external interrupt pins, or non-maskable interrupt sources. Interrupts are treated as level sources or captured as edge-triggered sources.

Note: For details, see **Chapter 12**, *Interrupt Processing*.

1.12 M2 Memory (Device-Specific)

The 192 KB M2 memory is used as a secondary memory within the MSC711x devices that contain it. It is tuned for fast cache bursting with a data bus of 128-bits, and a low latency path to the instruction cache, significantly reducing the penalty on cache misses. M2 memory can also be used for data storage. 128-bits of data can be read and 64-bits can be written to M2 at 150 MHz. M2 is single ported; that is, it supports only one access at a given time. The SC1400 core accesses this memory through the extended core interface and crossbar switch. The DMA controller also accesses M2 memory through the crossbar switch. Enabling the DMA controller to write program and data directly to M2 memory alleviates the load on the SC1400 core and keeps the focus on the intensive DSP processing. An SC1400 core access to the M2 memory requires a minimum of seven core wait states for the first access in a burst and then one data per clock (7-1-1-1). In a typical application that carefully considers memory allocation and uses the cache wisely, SC1400 core accesses to the M2 memory are greatly reduced.

1.13 Peripherals

The MSC711x peripherals include the TDM interface, HDI16 host interface, device-specific fast Ethernet controller, timers, UART, device-specific I^2C interface, and GPIO signals.

1.13.1 TDM Serial Interface

The TDM interface connects gluelessly to common telecommunication frame schemes, such as T1 and E1 lines. It can connect to multiple framers, such as MVIP, SCASA, and H.110 buses. The TDM contains up to three identical and independent engines, each configured in one of the following options:

- Two independent receive and transmit links.
  - The transmit has an input clock of up to 50 Mbps, output data and a frame sync configured as either input or output. Up to 128 transmit channels are supported.
  - The receive has an input clock of up to 50 Mbps, input data, and an input frame sync. Up to 256 receive channels are supported.
- One receive and one transmit link share the clock and the frame sync. The input clock runs at up to 50 Mbps, and the sync is configured as either input or output. There are up to 128 channels for the receive link and up to 128 channels for the transmit link.
- The receive and transmit links of both TDMs share one clock and one frame sync. The input clock runs at up to 50 Mbps, and the sync is configured as either input or output. There are up to 128 channels for the receive link and up to 128 channels for the transmit link.

Each channel can be 8 or 16 bits wide. When the slot size is 8 bits wide, selected channels can be defined as A-law/μ-law. These channels are converted to 13–14 bits, which are padded into 16 bits and stored as such in memory. Each receive channel and each transmit channel can be active or not. An active channel has a buffer that can be placed into the M1, M2, or external memory via DMA transfers through the crossbar switch. Transmit and receive operations each have their own eight-location FIFO for buffering data between the interface and the device. The SC1400 core empties the receive buffers while the TDM continues to fill the buffers until a second threshold line is reached and an interrupt is generated to the SC1400 core. The TDM supports an interrupt on frame start to the SC1400 core, which helps synchronize to the TDM system. For transmits, the SC1400 core fills the buffers of the desired TDM interface, and the TDM empties them.

Note: For details, see **Chapter 19**, *Time-Division Multiplexing (TDM) Interface*.

1.13.2 Host Interface (HDI16)

In addition to the system bus interface, MSC711x devices feature an enhanced 16-bit parallel host interface (HDI16) that supports a variety of standard buses and provides a glueless connection to industry-standard microcontrollers, microprocessors, and DSPs. The host interface can be used concurrently with the system interface. The combination of bus interfaces allows for greater system design flexibility. MSC711x devices can communicate via the 16-bit HDI16 interface while simultaneously connecting to external memory via a 32-bit system bus. The HDI16 can also operate with an 8-bit host data bus, making it fully compatible with the DSP56300 HDI08 from the external host side.

Note: For details, see **Chapter 20**, *Host Interface (HDI16)*.

1.13.3 Fast Ethernet Controller (Device-Specific)

The fast Ethernet controller (FEC) supports 10/100 Mbps Ethernet as defined by **IEEE** 802.3. The FEC supports two MAC-PHY interfaces, the media independent interface (MII) and reduced MII (RMII). The FEC architecture employs an Ethernet media access controller (MAC) to handle the MII interface, FIFO, and DMA. An MII bridge (MIIGSK) module supports the RMII interface. In addition, the FEC has a RISC microcontroller to manage DMA buffer descriptors, minimizing processor usage. A management information base (MIB) module is employed for tracking network activity on the MAC-PHY interface.

Note: The FEC is designated as a “device-specific” module, which means that some devices in the MSC711x family contain the FEC while others do not. To verify whether your MSC711x device has an Ethernet controller, study the comparisons in **Table 1-2**, *MSC711x Device-Specific Feature Comparison*, on page 1-8.

Note: For details, see **Chapter 18**, *Fast Ethernet Controller (FEC)*.

1.13.4 Timers

There are eight identical general-purpose 16-bit timers (or two 16-bit quad timers), and each can operate independently or as part of a programmable cascade of two or three timers. Each timer can be programmed as either one-shot or cyclic. The SC1400 cores can program the counters, read their updated values, and also be interrupted when the timers reach a predefined value. The timers are clocked by either the internal clock generator or one of two dedicated external signals or from the receive and transmit TDM clocks. When a timer reaches a predefined value, it either toggles or generates a pulse that can be directed to one of the two dedicated external signals or to other timers. In addition, it can generate an interrupt.

Note: For details, see **Chapter 21**, *Timers Module*.

1.13.5 Universal Asynchronous Receiver/Transmitter (UART)

The UART is used mainly for debugging or booting. It provides a full-duplex port for serial communications by transmit data (TXD) and receive data (RXD) lines. During reception, the UART generates an interrupt request when a new character is available to the UART data register. During transmission, the UART generates an interrupt request when its data register can be written with new character. When accepting an interrupt request, an SC1400 core or external host should read the UART status register to identify the interrupt source and service it accordingly.

Note: For details, see **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)*.

1.13.6 I^2C Interface

The I^2C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications requiring occasional communications over a short distance between many devices. The flexible I^2C allows additional devices to be connected to the bus for expansion and system development.

The interface operates up to 100 kbps with maximum bus loading and timing. The I^2C system is a true multiple-master bus including arbitration and collision detection that prevents data corruption if multiple devices attempt to control the bus simultaneously. This feature supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

Note: For details, see **Chapter 22**, *I2C Software Module*.

1.13.7 GPIO Signals

Up to 37 general-purpose I/O (GPIO) signals are available. Each connection in the I/O ports is configured either as a GPIO signal or as a dedicated peripheral interface signal. Each line is configured as an input or output (with a register for data output that is read or written at any

time). In this mode, the a signal drives a zero voltage but goes to tri-state (high impedance) when driving a high voltage. GPIO signals do not have internal pull-up resistors. Dedicated device peripheral functions are multiplexed onto the shared external connections. The functions are grouped to maximize connection use for the greatest number of MSC711x applications.

Note: For details, see **Chapter 24**, *General-Purpose Input/Output (GPIO)*.

1.13.8 Event Port

The event port takes as input information from the EVNTx pins, system events such as DMA transfers, and the emulator breakpoint logic. The signals can be combined as desired via ANDing, ORing, or sequencing, and the resulting action can be sent to the EVNTx pins, to the emulator breakpoint logic, to the timer modules, or to the interrupt controller.

The event port can be used independently or in conjunction with the MSC711x debug port (OCE10 emulator) and internal timers to manage internal and external MSC711x events. The event port interacts with the debug port breakpoint unit to perform a selectable event port action: drive an EVNTx pin, handle an event port interrupt request, pass the trigger directly to the timer module, enable a debug emulator detection module, or perform an emulator action.

Note: For details, see **Chapter 15**, *Event Port*.

Signal Pins and Pinouts 2

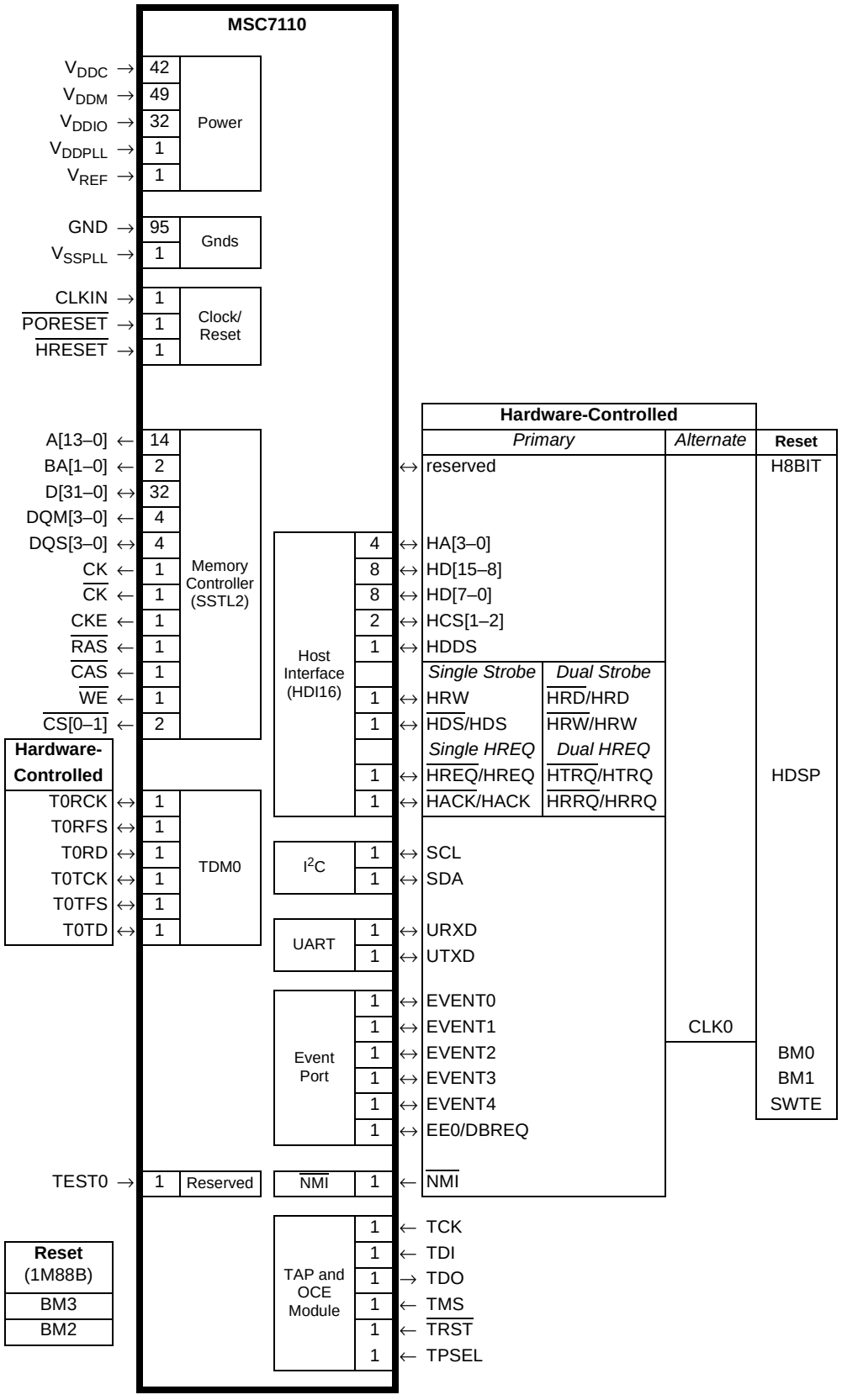
The MSC711x pins are organized into functional groups. **Table 2-1** lists the functional groups, states the number of pins in each group, and references the table that describes the multiplexed pins in each group. Most MSC71xx external peripherals are configured through ports A–D. The port configuration registers allow signal lines to be configured as software-controlled or hardware-controlled. If a signal is configured as software-controlled, it can be further defined as an input or an output. For port A, some signals configured as inputs may be configured as general-purpose signals or as maskable interrupt lines. If a signal is configured as hardware-controlled, it has a special function supporting one of the peripheral interfaces (for example, the HDI16). Some signals can also have an alternate hardware function (such as the CLKO signal). **Figure 2-1** through **Figure 2-8** show external signals for each device organized by hardware-controlled and software-controlled configuration combinations for signals enabled through ports A–D.

Table 2-1. MSC711x Functional Pin Groupings

Functional Group	Number of Signal Connections	Detailed Description
Power and Ground	221	**Table 2-2** on page 2-10
Clock and Reset	3	**Table 2-3** on page 2-10
Memory controller	64	**Table 2-4** on page 2-11
Signals configured through ports A–D: • TDM interfaces • Ethernet MAC • Host interface (HDI16) • I^2C interface • UART interface • Event port • GPIO ports (mask 1L44x or mask 1M88B) • $\overline{IRQ}$ and $\overline{NMI}$ lines	 6 per interface 18 27 2 2 5 42 or 46 28	 **Table 2-5** on page 2-12 **Table 2-6** on page 2-16 **Table 2-7** on page 2-20 **Table 2-8** on page 2-23 **Table 2-9** on page 2-23 **Table 2-10** on page 2-24 **Table 2-11** on page 2-25 **Table 2-12** on page 2-36
OCE10 on-chip emulator module and JTAG Test Access Port	7	**Table 2-13** on page 2-42
No connect (devices with Ethernet or devices without Ethernet)	37 or 40	Do not connect any line, component, or via to these pins.

Notes: **1.** One of the host port pins is multiplexed with one of the Ethernet MAC pins.
2. On MSC711x devices, the address and data bit ordering uses bit 0 as the least significant bit (LSB) which differs from that of the MSC8101, MSC8102, MSC8103, MSC8122, and MSC8126 devices that use bit 0 as the most significant bit (MSB).

Note: Although the package for this devices uses ball connections, the connections are sometimes conventionally referenced as pins.



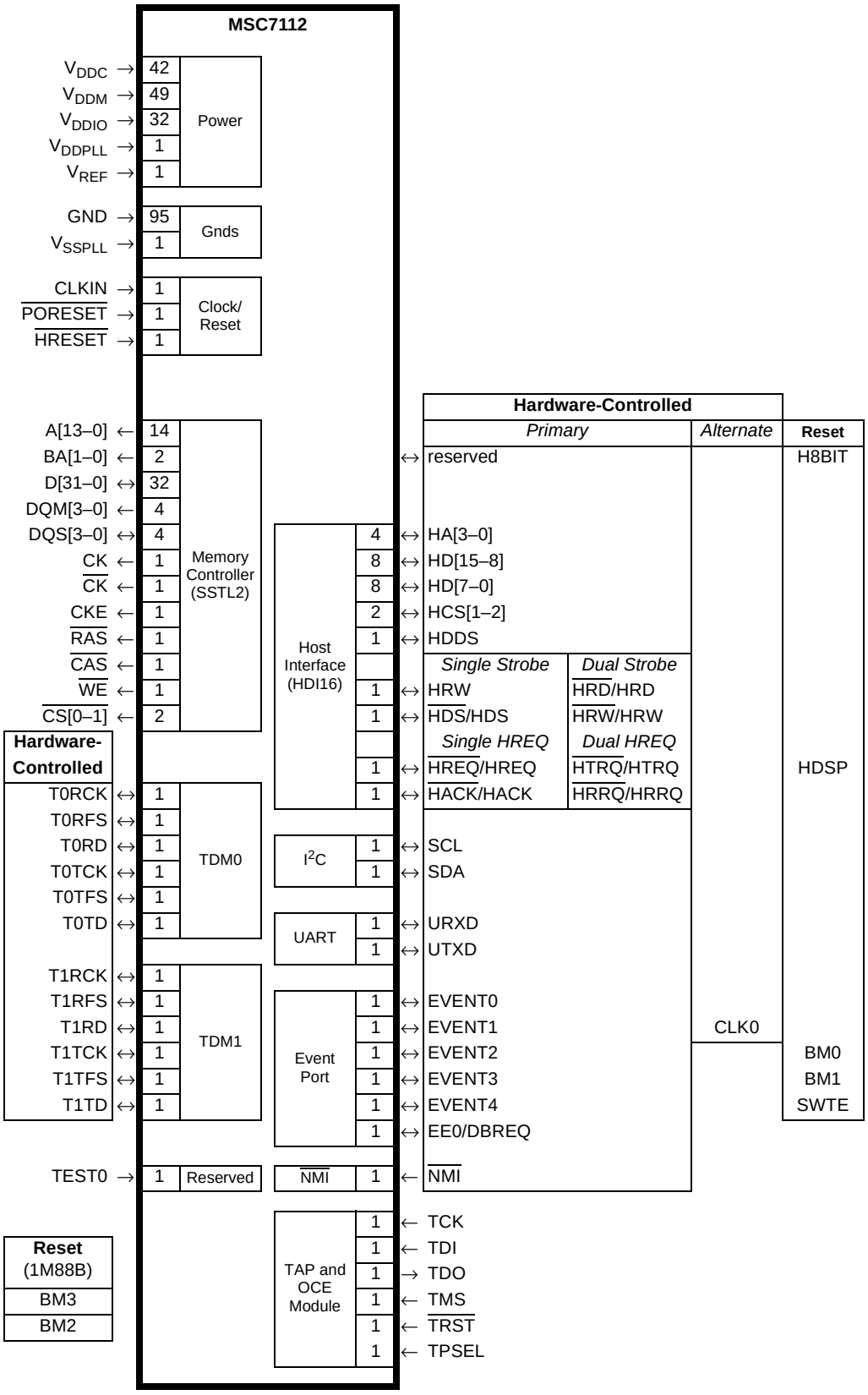
Note: For software-controlled functionality, see **Figure 2-2**. This figure does not include the NC pins.

Figure 2-1. MSC7110 External Signals (Hardware-Controlled Functions)

Port Signal	Software Controlled (GPxCTL[x] = 0)			Hardware Controlled (GPxCTL[x] = 1)						Reset Configuration (sampled at deassertion of PORESET only)
	GPI (Default) GPxDDR[x] = 0 GPA_INTEN[x] = 0	Interrupt GPxDDR[x] = 0 GPA_INTEN[x] = 1	GPO GPxDDR[x] = 1	Primary Function CHPCFG[PAS] and CHPCFG[PDS] = 0				Alternate Function CHPCFG[PAS] = 1 CHPCFG[PDS] = 1		
PA29	GPIA29	IRQ18	GPOA29	reserved				reserved		
PA28	GPIA28	IRQ17	GPOA28	reserved				reserved		
PA27	GPIA27	IRQ16	GPOA27	reserved				reserved		
PA26	GPIA26	IRQ26	GPOA26	reserved						
PA25	GPIA25	IRQ25	GPOA25	reserved						
PA24	GPIA24	IRQ24	GPOA24	reserved						
PA23	GPIA23	IRQ23	GPOA23	reserved						
PA22	GPIA22	IRQ22	GPOA22	reserved						
PA21	GPIA21	IRQ21	GPOA21	reserved						
PA20	GPIA20	IRQ20	GPOA20	reserved						
PA19	GPIA19	IRQ19	GPOA19	reserved						
PA18	reserved	NMI	reserved		Event				Clock	
PA17	GPIA17	IRQ13	GPOA17		EVNT1				CLKO	
PA16	GPIA16	IRQ12	GPOA16		EVNT4	I2C				SWTE
PA15	GPIA15	IRQ14	GPOA15			SCL				
PA14	GPIA14	IRQ15	GPOA14			SDA	UART			
PA13	GPIA13	IRQ2	GPOA13				URXD			
PA12	GPIA12	IRQ3	GPOA12			TDM0	UTXD			
PA11	GPIA11	IRQ4	GPOA11			T0RCK				
PA10	GPIA10	IRQ5	GPOA10			T0RFS				
PA9	GPIA9	reserved	GPOA9			T0RD				
PA8	GPIA8	IRQ6	GPOA8			T0TCK				
PA7	GPIA7	IRQ7	GPOA7			T0TFS				
PA6	GPIA6	reserved	GPOA6			T0TD				
PA5	GPIA5	IRQ0	GPOA5				reserved			
PA4	GPIA4	IRQ1	GPOA4				reserved			
PA3	GPIA3	IRQ8	GPOA3				reserved			
PA2	GPIA2	IRQ9	GPOA2				reserved			
PA1	GPIA1	IRQ10	GPOA1				reserved			
PA0	GPIA0	IRQ11	GPOA0				reserved			
							HDI16			
PB14	reserved		reserved				HDDS			
PB13	reserved		reserved				HDS or HWR			
PB12	reserved		reserved				HRW or HRD			
PB11	GPIB11[1]		GPOB11[1]				HCS2			
PB10	reserved		reserved				HCS1			
PB9	reserved		reserved				HACK or HRRQ			
PB8	reserved		reserved				HREQ or HTRQ			HDSP
PB7	reserved		reserved				HD7			
PB6	reserved		reserved				HD6			
PB5	reserved		reserved				HD5			
PB4	reserved		reserved				HD4			
PB3	reserved		reserved				HD3			
PB2	reserved		reserved				HD2			
PB1	reserved		reserved				HD1			
PB0	reserved		reserved				HD0			
PC15	GPIC15		GPOC15		EVNT3					BM1
PC14	GPIC14		GPOC14		EVNT2					BM0
PC13	reserved		reserved		EVNT0					
PC12	reserved		reserved		EE0/ DBREQ					
PC11	GPIC11[1]		GPOC11[1]				HA3			
PC10	reserved		reserved				HA2			
PC9	reserved		reserved				HA1			
PC8	reserved		reserved				HA0			
PC7	GPIC7		GPOC7				HD15			
PC6	GPIC6		GPOC6				HD14			
PC5	GPIC5		GPOC5				HD13			
PC4	GPIC4		GPOC4				HD12			
PC3	GPIC3		GPOC3				HD11			
PC2	GPIC2		GPOC2				HD10			
PC1	GPIC1		GPOC1				HD9			
PC0	GPIC0		GPOC0				HD8			
PD8	GPID8[1]		GPOD8[1]							BM3[2]
PD7	GPID7[1]		GPOD7[1]							BM2[2]
PD6	GPID6		GPOD6	reserved				reserved		
PD5	GPID5		GPOD5	reserved				reserved		
PD4	GPID4		GPOD4	reserved				reserved		
PD2	reserved		reserved	reserved						H8BIT

Notes: **1.** Mask set 1M88B. For mask set 1L44X, these signals are reserved.
1. Mask set 1M88B only. For mask set 1L44X, these signals are not implemented.

Figure 2-2. MSC7110 Port A–D Signal Configuration Diagram



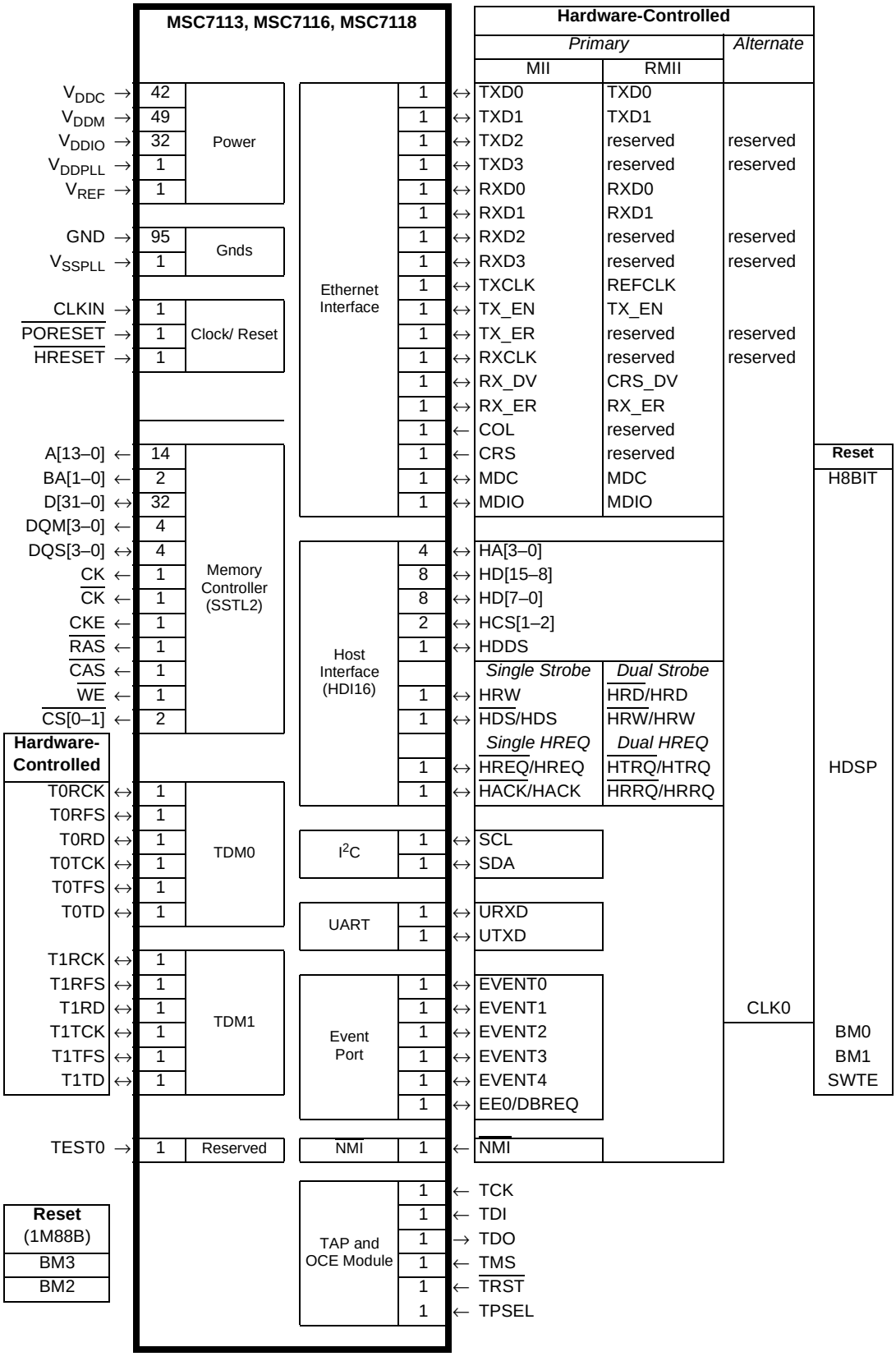
Note: For software-controlled functionality, see **Figure 2-4**. This figure does not include the NC pins.

Figure 2-3. MSC7112 External Signals (Hardware-Controlled)

Port Signal	Software Controlled (GPxCTL[x] = 0)			Hardware Controlled (GPxCTL[x] = 1)						Reset Configuration (sampled at deassertion of PORESET only)
	GPI (Default) GPxDDR[x] = 0 GPA_INTEN[x] = 0	Interrupt GPxDDR[x] = 0 GPA_INTEN[x] = 1	GPO GPxDDR[x] = 1	Primary Function CHPCFG[PAS] and CHPCFG[PDS] = 0				Alternate Function CHPCFG[PAS] = 1 CHPCFG[PDS] = 1		
PA29	GPIA29	IRQ18	GPOA29	reserved				reserved		
PA28	GPIA28	IRQ17	GPOA28	reserved				reserved		
PA27	GPIA27	IRQ16	GPOA27	reserved				reserved		
PA26	GPIA26	IRQ26	GPOA26	reserved						
PA25	GPIA25	IRQ25	GPOA25	reserved						
PA24	GPIA24	IRQ24	GPOA24	reserved						
PA23	GPIA23	IRQ23	GPOA23	reserved						
PA22	GPIA22	IRQ22	GPOA22	reserved						
PA21	GPIA21	IRQ21	GPOA21	reserved						
PA20	GPIA20	IRQ20	GPOA20	reserved						
PA19	GPIA19	IRQ19	GPOA19	reserved						
PA18	reserved	NMI	reserved		Event				Clock	
PA17	GPIA17	IRQ13	GPOA17		EVNT1				CLKO	
PA16	GPIA16	IRQ12	GPOA16		EVNT4	I^2C				SWTE
PA15	GPIA15	IRQ14	GPOA15			SCL				
PA14	GPIA14	IRQ15	GPOA14			SDA	UART			
PA13	GPIA13	IRQ2	GPOA13				URXD			
PA12	GPIA12	IRQ3	GPOA12			TDM0	UTXD			
PA11	GPIA11	IRQ4	GPOA11			T0RCK				
PA10	GPIA10	IRQ5	GPOA10			T0RFS				
PA9	GPIA9	reserved	GPOA9			T0RD				
PA8	GPIA8	IRQ6	GPOA8			T0TCK				
PA7	GPIA7	IRQ7	GPOA7			T0TFS				
PA6	GPIA6	reserved	GPOA6			T0TD	TDM1			
PA5	GPIA5	IRQ0	GPOA5				T1RCK			
PA4	GPIA4	IRQ1	GPOA4				T1RFS			
PA3	GPIA3	IRQ8	GPOA3				T1RD			
PA2	GPIA2	IRQ9	GPOA2				T1TCK			
PA1	GPIA1	IRQ10	GPOA1				T1TFS			
PA0	GPIA0	IRQ11	GPOA0				T1TD			
							HDI16			
PB14	reserved		reserved				HDDS			
PB13	reserved		reserved				HDS or HWR			
PB12	reserved		reserved				HRW or HRD			
PB11	GPIB11[1]		GPOB11[1]				HCS2			
PB10	reserved		reserved				HCS1			
PB9	reserved		reserved				HACK or HRRQ			
PB8	reserved		reserved				HREQ or HTRQ			HDSP
PB7	reserved		reserved				HD7			
PB6	reserved		reserved				HD6			
PB5	reserved		reserved				HD5			
PB4	reserved		reserved				HD4			
PB3	reserved		reserved				HD3			
PB2	reserved		reserved				HD2			
PB1	reserved		reserved				HD1			
PB0	reserved		reserved				HD0			
PC15	GPIC15		GPOC15		EVNT3					BM1
PC14	GPIC14		GPOC14		EVNT2					BM0
PC13	reserved		reserved		EVNT0 EE0/ DBREQ					
PC12	reserved		reserved							
PC11	GPIC11[1]		GPOC11[1]				HA3			
PC10	reserved		reserved				HA2			
PC9	reserved		reserved				HA1			
PC8	reserved		reserved				HA0			
PC7	GPIC7		GPOC7				HD15			
PC6	GPIC6		GPOC6				HD14			
PC5	GPIC5		GPOC5				HD13			
PC4	GPIC4		GPOC4				HD12			
PC3	GPIC3		GPOC3				HD11			
PC2	GPIC2		GPOC2				HD10			
PC1	GPIC1		GPOC1				HD9			
PC0	GPIC0		GPOC0				HD8			
PD8	GPID8[1]		GPOD8[1]							BM3[2]
PD7	GPID7[1]		GPOD7[1]							BM2[2]
PD6	GPID6		GPOD6	reserved				reserved		
PD5	GPID5		GPOD5	reserved				reserved		
PD4	GPID4		GPOD4	reserved				reserved		
PD2	reserved		reserved	reserved						H8BIT

Notes: **1.** Mask set 1M88B. For mask set 1L44X, these signals are reserved.
1. Mask set 1M88B only. For mask set 1L44X, these signals are not implemented.

Figure 2-4. MSC7112 Port A–D Signal Configuration Diagram

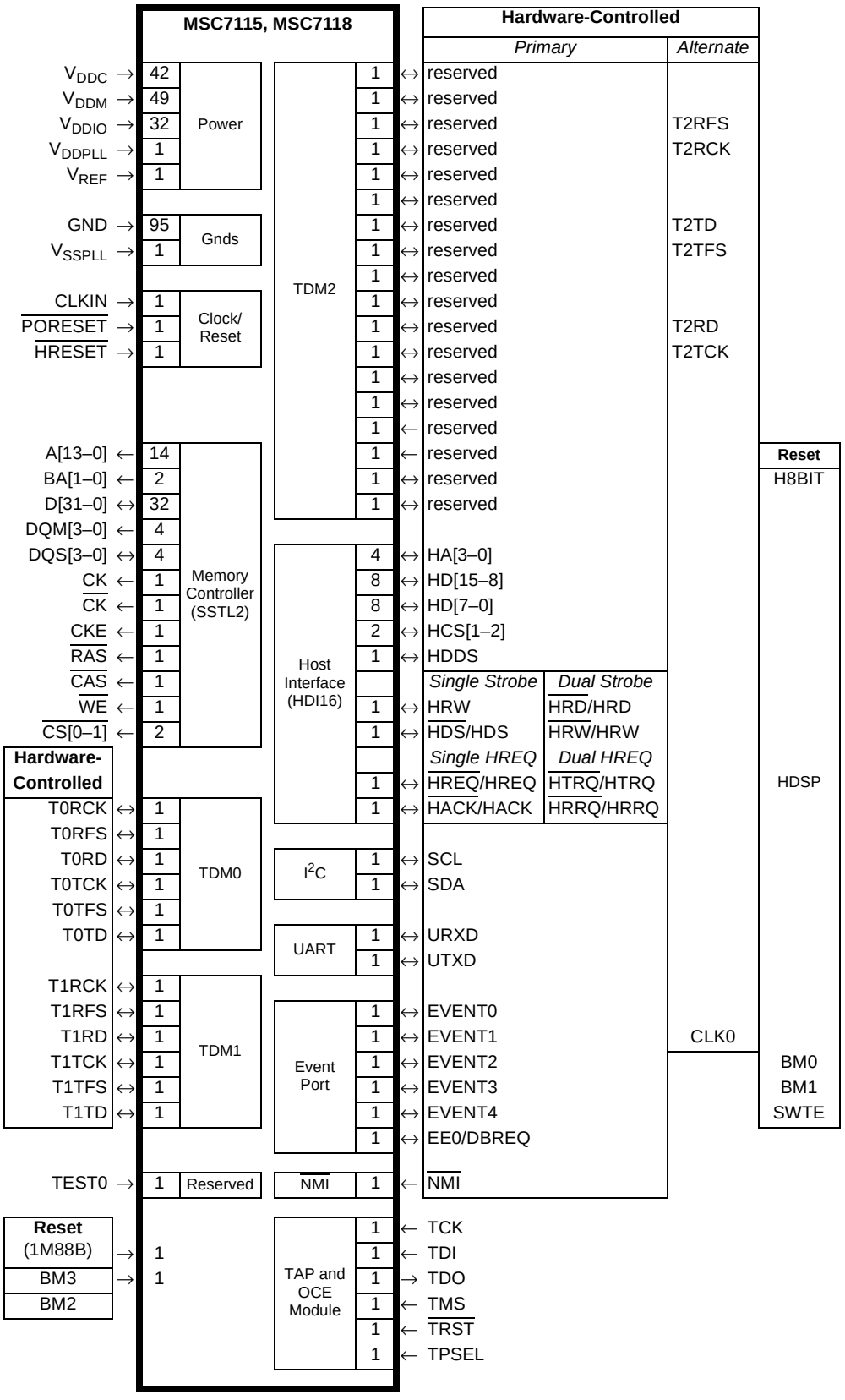


Note: For software-controlled functionality, see **Figure 2-6**. This figure does not include the NC pins.

Figure 2-5. MSC7113, MSC7116, and MSC7119 External Signals (Hardware-Controlled)

Port Signal	Software Controlled (GPxCTL[x] = 0)			Hardware Controlled (GPxCTL[x] = 1)						Reset Configuration (sampled at deassertion of PORESET only)
	GPI (Default) GPxDDR[x] = 0 GPA_INTEN[x] = 0	Interrupt GPxDDR[x] = 0 GPA_INTEN[x] = 1	GPO GPxDDR[x] = 1	Primary Function CHPCFG[PAS] and CHPCFG[PDS] = 0				Alternate Function CHPCFG[PAS] = 1 CHPCFG[PDS] = 1		
				Ethernet (MII or RMII)						
PA29	GPIA29	IRQ18	GPOA29	RXD3 or reserved				reserved		
PA28	GPIA28	IRQ17	GPOA28	TX_ER or reserved				reserved		
PA27	GPIA27	IRQ16	GPOA27	TXD3 or reserved				reserved		
PA26	GPIA26	IRQ26	GPOA26	RX_ER						
PA25	GPIA25	IRQ25	GPOA25	RX_DV or CRS_DV						
PA24	GPIA24	IRQ24	GPOA24	TX_EN						
PA23	GPIA23	IRQ23	GPOA23	TXCLK or REFCLK						
PA22	GPIA22	IRQ22	GPOA22	RXD0						
PA21	GPIA21	IRQ21	GPOA21	RXD1						
PA20	GPIA20	IRQ20	GPOA20	TXD0						
PA19	GPIA19	IRQ19	GPOA19	TXD1						
PA18	reserved	NMI	reserved		Event				Clock	
PA17	GPIA17	IRQ13	GPOA17		EVNT1				CLKO	
PA16	GPIA16	IRQ12	GPOA16		EVNT4	I^2C				SWTE
PA15	GPIA15	IRQ14	GPOA15			SCL				
PA14	GPIA14	IRQ15	GPOA14			SDA	UART			
PA13	GPIA13	IRQ2	GPOA13				URXD			
PA12	GPIA12	IRQ3	GPOA12			TDM0	UTXD			
PA11	GPIA11	IRQ4	GPOA11			T0RCK				
PA10	GPIA10	IRQ5	GPOA10			T0RFS				
PA9	GPIA9	reserved	GPOA9			T0RD				
PA8	GPIA8	IRQ6	GPOA8			T0TCK				
PA7	GPIA7	IRQ7	GPOA7			T0TFS				
PA6	GPIA6	reserved	GPOA6			T0TD	TDM1			
PA5	GPIA5	IRQ0	GPOA5				T1RCK			
PA4	GPIA4	IRQ1	GPOA4				T1RFS			
PA3	GPIA3	IRQ8	GPOA3				T1RD			
PA2	GPIA2	IRQ9	GPOA2				T1TCK			
PA1	GPIA1	IRQ10	GPOA1				T1TFS			
PA0	GPIA0	IRQ11	GPOA0				T1TD			
							HDI16			
PB14	reserved		reserved				HDDS			
PB13	reserved		reserved				HDS or HWR			
PB12	reserved		reserved				HRW or HRD			
PB11	GPIB11		GPOB11				HCS2			
PB10	reserved		reserved				HCS1			
PB9	reserved		reserved				HACK or HRRQ			
PB8	reserved		reserved				HREQ or HTRQ			HDSP
PB7	reserved		reserved				HD7			
PB6	reserved		reserved				HD6			
PB5	reserved		reserved				HD5			
PB4	reserved		reserved				HD4			
PB3	reserved		reserved				HD3			
PB2	reserved		reserved				HD2			
PB1	reserved		reserved				HD1			
PB0	reserved		reserved				HD0			
PC15	GPIC15		GPOC15		EVNT3					BM1
PC14	GPIC14		GPOC14		EVNT2					BM0
PC13	reserved		reserved		EVNT0					
PC12	reserved		reserved		EE0/ DBREQ					
PC11	GPIC11		GPOC11				HA3			
PC10	reserved		reserved				HA2			
PC9	reserved		reserved				HA1			
PC8	reserved		reserved				HA0			
PC7	GPIC7		GPOC7				HD15			
PC6	GPIC6		GPOC6				HD14			
PC5	GPIC5		GPOC5				HD13			
PC4	GPIC4		GPOC4				HD12			
PC3	GPIC3		GPOC3				HD11			
PC2	GPIC2		GPOC2				HD10			
PC1	GPIC1		GPOC1				HD9			
PC0	GPIC0		GPOC0				HD8			
PD8	GPID8		GPOD8	reserved				reserved		BM3
PD7	GPID7		GPOD7	reserved				reserved		BM2
PD6	GPID6		GPOD6	RXD2 or reserved				reserved		
PD5	GPID5		GPOD5	RXCLK or reserved				reserved		
PD4	GPID4		GPOD4	TXD2 or reserved				reserved		
PD3	reserved		reserved	MDIO						
PD2	reserved		reserved	MDC						H8BIT
PD1	reserved		reserved	CRS or reserved						
PD0	reserved		reserved	COL or reserved						

Figure 2-6. MSC7113, MSC7116, and MSC7119 Port A–D Signal Configuration Diagram



Note: For software-controlled functionality, see **Figure 2-8**. This figure does not include the NC pins.

Figure 2-7. MSC7115 and MSC7118 External Signals (Hardware-Controlled)

Port Signal	Software Controlled (GPxCTL[x] = 0)			Hardware Controlled (GPxCTL[x] = 1)						Reset Configuration (sampled at deassertion of PORESET only)
	GPI (Default) GPxDDR[x] = 0 GPA_INTEN[x] = 0	Interrupt GPxDDR[x] = 0 GPA_INTEN[x] = 1	GPO GPxDDR[x] = 1	Primary Function CHPCFG[PAS] and CHPCFG[PDS] = 0				Alternate Function CHPCFG[PAS] = 1 CHPCFG[PDS] = 1		
								TDM2		
PA29	GPIA29	IRQ18	GPOA29	reserved				T2TFS		
PA28	GPIA28	IRQ17	GPOA28	reserved				T2RD		
PA27	GPIA27	IRQ16	GPOA27	reserved				T2RCK		
PA26	GPIA26	IRQ26	GPOA26	reserved						
PA25	GPIA25	IRQ25	GPOA25	reserved						
PA24	GPIA24	IRQ24	GPOA24	reserved						
PA23	GPIA23	IRQ23	GPOA23	reserved						
PA22	GPIA22	IRQ22	GPOA22	reserved						
PA21	GPIA21	IRQ21	GPOA21	reserved						
PA20	GPIA20	IRQ20	GPOA20	reserved						
PA19	GPIA19	IRQ19	GPOA19	reserved						
PA18	reserved	NMI	reserved		Event				Clock	
PA17	GPIA17	IRQ13	GPOA17		EVNT1				CLKO	
PA16	GPIA16	IRQ12	GPOA16		EVNT4	I2C				SWTE
PA15	GPIA15	IRQ14	GPOA15			SCL				
PA14	GPIA14	IRQ15	GPOA14			SDA	UART			
PA13	GPIA13	IRQ2	GPOA13				URXD			
PA12	GPIA12	IRQ3	GPOA12			TDM0	UTXD			
PA11	GPIA11	IRQ4	GPOA11			T0RCK				
PA10	GPIA10	IRQ5	GPOA10			T0RFS				
PA9	GPIA9	reserved	GPOA9			T0RD				
PA8	GPIA8	IRQ6	GPOA8			T0TCK				
PA7	GPIA7	IRQ7	GPOA7			T0TFS				
PA6	GPIA6	reserved	GPOA6			T0TD	TDM1			
PA5	GPIA5	IRQ0	GPOA5				T1RCK			
PA4	GPIA4	IRQ1	GPOA4				T1RFS			
PA3	GPIA3	IRQ8	GPOA3				T1RD			
PA2	GPIA2	IRQ9	GPOA2				T1TCK			
PA1	GPIA1	IRQ10	GPOA1				T1TFS			
PA0	GPIA0	IRQ11	GPOA0				T1TD			
							HDI16			
PB14	reserved		reserved				HDDS			
PB13	reserved		reserved				HDS or HWR			
PB12	reserved		reserved				HRW or HRD			
PB11	GPIB11		GPOB11				HCS2			
PB10	reserved		reserved				HCS1			
PB9	reserved		reserved				HACK or HRRQ			
PB8	reserved		reserved				HREQ or HTRQ			HDSP
PB7	reserved		reserved				HD7			
PB6	reserved		reserved				HD6			
PB5	reserved		reserved				HD5			
PB4	reserved		reserved				HD4			
PB3	reserved		reserved				HD3			
PB2	reserved		reserved				HD2			
PB1	reserved		reserved				HD1			
PB0	reserved		reserved				HD0			
PC15	GPIC15		GPOC15		EVNT3					BM1
PC14	GPIC14		GPOC14		EVNT2					BM0
PC13	reserved		reserved		EVNT0					
PC12	reserved		reserved		EE0/ DBREQ					
PC11	GPIC11		GPOC11				HA3			
PC10	reserved		reserved				HA2			
PC9	reserved		reserved				HA1			
PC8	reserved		reserved				HA0			
PC7	GPIC7		GPOC7				HD15			
PC6	GPIC6		GPOC6				HD14			
PC5	GPIC5		GPOC5				HD13			
PC4	GPIC4		GPOC4				HD12			
PC3	GPIC3		GPOC3				HD11			
PC2	GPIC2		GPOC2				HD10			
PC1	GPIC1		GPOC1				HD9			
PC0	GPIC0		GPOC0				HD8			
PD8	GPID8		GPOD8							BM3
PD7	GPID7		GPOD7	reserved						BM2
PD6	GPID6		GPOD6	reserved				T2TD		
PD5	GPID5		GPOD5	reserved				T2TCK		
PD4	GPID4		GPOD4	reserved				T2RFS		
PD2	reserved		reserved	reserved						H8BIT

Figure 2-8. MSC7115 and MSC7118 Port A–D Signal Configuration Diagram

2.1 Power and Ground

Table 2-2. Power and Ground Inputs

Signal Name	Number of Pins	Description
V_{DDC}	42	**Internal Logic Power** A dedicated well-regulated power source for the device core. Provide an extremely low impedance path to the V_{DDC} power rail.
V_{DDM}	49	**SSTL IO Driver Power** A dedicated power source for the DDR DRAM interface buffers. Provide adequate external decoupling capacitors.
V_{DDIO}	32	**Input/Output Power** The power source for the I/O buffers. Provide adequate external decoupling capacitors.
V_{DDPLL}	1	**System PLL Power** A dedicated well-regulated power for the system Phase Lock Loop (PLL). Provide an extremely low impedance path to the V_{DDPLL} power rail.
V_{REF}	1	**SSTL Reference Power** A reference power level for the SSTL2 memory interface.
GND	95	**System Ground** An isolated common ground for the internal processing logic, I/O buffers, and the DDR DRAM interface buffers. Provide adequate external decoupling capacitors.
V_{SSPLL}	1	**System PLL Ground** An isolated ground for the system PLL. Provide an extremely low-impedance path to this ground.

2.2 Clocks and Resets

Table 2-3. Clock and Reset Pin Definitions

Pin	Number of pins	Data Flow	Description
CLKIN	1	Input	Clock In Primary clock input to the MSC711x PLL. This signal provides the main clock source for the device.
$\overline{\text{PORESET}}$	1	Input	Power-On Reset When asserted, this signal causes the MSC711x to enter the power-on reset state.
$\overline{\text{HRESET}}$	1	Input/Output	Hard Reset When asserted, this open-drain signal causes the MSC711x to enter the hard reset state. An external pull-up resistor must be used on this pin.

2.3 Memory System Interface (DDR Controller)

Table 2-4. Memory System Interface (DDR Controller) Signals

Pin	Number of Pins	Data Flow	Description
A[13–0]	14	Output	**Address Bus** Address lines connected to memory devices and controlled by the MSC711x memory controller. A0 is the LSB of the address bus. See **Table 2-5**.
BA[1–0]	2	Output	**Bank Address** Selects the SDRAM bank. BA0 is the LSB. See **Table 2-5**.
CS[1–0]	2	Output	**Chip Selects** Enables specific memory devices or peripherals connected to the bus.
D[31–0]	32	Input/Output	**Data Bus** In write transactions, the MSC711x drives the valid data on this bus. In read transactions, the memory device drives the valid data on this bus. In 16-pin mode, pins D[15–0] are used. D0 is the LSB of the data bus.
DQM[3–0]	4	Output	**SDRAM DQM** From the SDRAM memory controller. These pins select specific byte lanes of SDRAM devices. In 16-pin mode, pins DQM[1–0] are used.
DQS[3–0]	4	Input/Output	**SDRAM DQS** From the SDRAM memory controller. These pins are the data capture strobe of the byte lanes of SDRAM devices. In 16-pin mode, pins DQS[1–0] are used.
CK	1	Output	**Clock Out** Output clock running at half the frequency of the DDR clock.
$\overline{\text{CK}}$	1	Output	**Clock Out inverted** An inverted version of the CK pin.
CKE	1	Output	**Clock Enable** The enable of the system bus clock for the DDR SDRAM.
$\overline{\text{RAS}}$	1	Output	**SDRAM $\overline{\text{RAS}}$** From the SDRAM memory controller. Should connect to SDRAM $\overline{\text{RAS}}$ input.
$\overline{\text{CAS}}$	1	Output	**SDRAM $\overline{\text{CAS}}$** From the SDRAM memory controller. Should connect to SDRAM $\overline{\text{CAS}}$ input.
$\overline{\text{WE}}$	1	Output	**SDRAM Write Enable** From the SDRAM memory controller. Should connect to SDRAM $\overline{\text{WE}}$ input.

2.4 TDM Interfaces

Table 2-5. TDM Interface Signals

Pin	Data Flow	Description
GPIA11	Input	**General-Purpose Input A11 (default)** When configured through port A bit 11, performs as a general-purpose input.
$\overline{\text{IRQ4}}$	Input	**Interrupt Request 4** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA11	Output	**General-Purpose Output A11** When configured through port A bit 11, performs as a general-purpose output.
T0RCK	Input/Output	**TDM0 Receive Clock** The receive clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA10	Input	**General-Purpose Input A10 (default)** When configured through port A bit 10, performs as a general-purpose input.
$\overline{\text{IRQ5}}$	Input	**Interrupt Request 5** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA10	Output	**General-Purpose Output A10** When configured through port A bit 10, performs as a general-purpose output.
T0RFS	Input/Output	**TDM0 Receive Frame Sync** The receive frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA9	Input	**General-Purpose Input A9 (default)** When configured through port A bit 9, performs as a general-purpose input.
GPOA9	Output	**General-Purpose Output A9** When configured through port A bit 9, performs as a general-purpose output.
T0RD	Input/Output	**TDM0 Receive Data** The receive data for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA8	Input	**General-Purpose Input A8 (default)** When configured through port A bit 8, performs as a general-purpose input.
$\overline{\text{IRQ6}}$	Input	**Interrupt Request 6** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA8	Output	**General-Purpose Output A8** When configured through port A bit 8, performs as a general-purpose output.
T0TCK	Input/Output	**TDM0 Transmit Clock** The transmit clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-5. TDM Interface Signals (Continued)

Pin	Data Flow	Description
GPIA7	Input	**General-Purpose Input A7 (default)** When configured through port A bit 7, performs as a general-purpose input.
$\overline{\text{IRQ7}}$	Input	**Interrupt Request 7** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA7	Output	**General-Purpose Output A7** When configured through port A bit 7, performs as a general-purpose output.
T0TFS	Input/Output	**TDM0 Transmit Frame Sync** The transmit frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA6	Input	**General-Purpose Input A9 (default)** When configured through port A bit 6, performs as a general-purpose input.
GPOA6	Output	**General-Purpose Output A6** When configured through port A bit 6, performs as a general-purpose output.
T0TD	Input/Output	**TDM0 Transmit Data** The transmit data for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA5	Input	**General-Purpose Input A5 (default)** When configured through port A bit 5, performs as a general-purpose input.
$\overline{\text{IRQ0}}$	Input	**Interrupt Request 0** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA5	Output	**General-Purpose Output A5** When configured through port A bit 5, performs as a general-purpose output.
T1RCK	Input/Output	**TDM1 Receive Clock** The receive clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA4	Input	**General-Purpose Input A4 (default)** When configured through port A bit 4, performs as a general-purpose input.
$\overline{\text{IRQ1}}$	Input	**Interrupt Request 1** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA4	Output	**General-Purpose Output A4** When configured through port A bit 4, performs as a general-purpose output.
T1RFS	Input/Output	**TDM1 Receive Frame Sync** The receive frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-5. TDM Interface Signals (Continued)

Pin	Data Flow	Description
GPIA3	Input	**General-Purpose Input A3 (default)** When configured through port A bit 3, performs as a general-purpose input.
$\overline{\text{IRQ8}}$	Input	**Interrupt Request 8** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA3	Output	**General-Purpose Output A3** When configured through port A bit 3, performs as a general-purpose output.
T1RD	Input/Output	**TDM1 Receive Data** The receive data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA2	Input	**General-Purpose Input A2 (default)** When configured through port A bit 2, performs as a general-purpose input.
$\overline{\text{IRQ9}}$	Input	**Interrupt Request 9** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA2	Output	**General-Purpose Output A2** When configured through port A bit 2, performs as a general-purpose output.
T1TCK	Input/Output	**TDM1 Transmit Clock** The transmit clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA1	Input	**General-Purpose Input A1 (default)** When configured through port A bit 1, performs as a general-purpose input.
$\overline{\text{IRQ10}}$	Input	**Interrupt Request 10** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA1	Output	**General-Purpose Output A1** When configured through port A bit 1, performs as a general-purpose output.
T1TFS	Input/Output	**TDM1 Transmit Frame Sync** The transmit frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA0	Input	**General-Purpose Input A9 (default)** When configured through port A bit 6, performs as a general-purpose input.
$\overline{\text{IRQ11}}$	Input	**Interrupt Request 11** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA0	Output	**General-Purpose Output A0** When configured through port A bit 0, performs as a general-purpose output.
T1TD	Input/Output	**TDM1 Transmit Data** The transmit data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-5. TDM Interface Signals (Continued)

Pin	Data Flow	Description
GPIA27	Input	**General-Purpose Input A27 (default)** When configured through port A bit 27, performs as a general-purpose input.
$\overline{\text{IRQ16}}$	Input	**Interrupt Request 16** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA27	Output	**General-Purpose Output A27** When configured through port A bit 27, performs as a general-purpose output.
T2RCK	Input/Output	**TDM2 Receive Clock** The receive clock for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPID4	Input	**General-Purpose Input D4 (default)** When configured through port D bit 4, performs as a general-purpose input.
GPOD4	Output	**General-Purpose Output D4** When configured through port D bit 4, performs as a general-purpose output.
T2RFS	Input/Output	**TDM2 Receive Frame Sync** The receive frame sync for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA28	Input	**General-Purpose Input A28 (default)** When configured through port A bit 28, performs as a general-purpose input.
$\overline{\text{IRQ17}}$	Input	**Interrupt Request 17** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA28	Output	**General-Purpose Output A28** When configured through port A bit 28, performs as a general-purpose output.
T2RD	Input/Output	**TDM2 Receive Data** The receive data for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPID5	Input	**General-Purpose Input D5 (default)** When configured through port D bit 5, performs as a general-purpose input.
GPOD5	Output	**General-Purpose Output D5** When configured through port D bit 5, performs as a general-purpose output.
T2TCK	Input/Output	**TDM2 Transmit Clock** The transmit clock for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA29	Input	**General-Purpose Input A29 (default)** When configured through port A bit 29, performs as a general-purpose input.
$\overline{\text{IRQ18}}$	Input	**Interrupt Request 18** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA29	Input	**General-Purpose Output A29** When configured through port A bit 29, performs as a general-purpose output.
T2TFS	Input/Output	**TDM2 Transmit Frame Sync** The transmit frame sync for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-5. TDM Interface Signals (Continued)

Pin	Data Flow	Description
GPID6	Input	**General-Purpose Input D6 (default)** When configured through port D bit 6, performs as a general-purpose input.
GPOD6 ‘	Output	**General-Purpose Output D6** When configured through port D bit 6, performs as a general-purpose output.
T2TD	Input/Output	**TDM2 Transmit Data** The transmit data for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
Note: The availability of a specific TDM interface is device dependent: **1.** MSC7110 supports TDM0. **2.** MSC7112 supports TDM0 and TDM1. **3.** MSC7113 supports TDM0 and TDM1. **4.** MSC7115 supports TDM0, TDM1, and TDM2. **5.** MSC7116 supports TDM0 and TDM1. **6.** MSC7118 supports TDM0, TDM1, and TDM2. **7.** MCS7119 supports TDM0 and TDM1.		

2.5 Ethernet MAC Interface Port

The MSC7113, MSC7116, and MSC7119 devices support an FEC.

Table 2-6. Ethernet MAC Interface Signals

Pin	Data Flow	Description
GPIA19	Input	**General-Purpose Input A19 (default)** When configured through port A bit 19, performs as a general-purpose input.
$\overline{\text{IRQ19}}$	Input	**Interrupt Request 19** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA19	Output	**General-Purpose Output A19** When configured through port A bit 19, performs as a general-purpose output.
TXD1	Output	**Transmit Data 1** MII and RMII transmit data bit 1. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA20	Input	**General-Purpose Input A20 (default)** When configured through port A bit 20, performs as a general-purpose input.
$\overline{\text{IRQ20}}$	Input	**Interrupt Request 20** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA20	Output	**General-Purpose Output A20** When configured through port A bit 20, performs as a general-purpose output.
TXD0	Output	**Transmit Data 0** MII and RMII transmit data bit 0. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-6. Ethernet MAC Interface Signals (Continued)

Pin	Data Flow	Description
GPIA21	Input	**General-Purpose Input A21 (default)** When configured through port A bit 21, performs as a general-purpose input.
$\overline{\text{IRQ21}}$	Input	**Interrupt Request 21** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA21	Output	**General-Purpose Output A21** When configured through port A bit 21, performs as a general-purpose output.
RXD1	Input	**Receive Data 1** MII and RMII receive data bit 1. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA22	Input	**General-Purpose Input A22 (default)** When configured through port A bit 22, performs as a general-purpose input.
$\overline{\text{IRQ22}}$	Input	**Interrupt Request 22** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA22	Output	**General-Purpose Output A22** When configured through port A bit 22, performs as a general-purpose output.
RXD0	Input	**Receive Data 0** MII and RMII receive data bit 0. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA23	Input	**General-Purpose Input A23 (default)** When configured through port A bit 23, performs as a general-purpose input.
$\overline{\text{IRQ23}}$	Input	**Interrupt Request 23** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA23	Output	**General-Purpose Output A23** When configured through port A bit 23, performs as a general-purpose output.
TXCLK	Input	**Transmit Clock** MII transmit clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
REFCLK	Input	**Reference Clock** RMII reference clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA24	Input	**General-Purpose Input A24 (default)** When configured through port A bit 24, performs as a general-purpose input.
$\overline{\text{IRQ24}}$	Input	**Interrupt Request 24** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA24	Output	**General-Purpose Output A24** When configured through port A bit 24, performs as a general-purpose output.
TX_EN	Output	**Transmit Data Valid** MII and RMII transmit data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-6. Ethernet MAC Interface Signals (Continued)

Pin	Data Flow	Description
GPIA25	Input	**General-Purpose Input A25 (default)** When configured through port A bit 25, performs as a general-purpose input.
$\overline{\text{IRQ25}}$	Input	**Interrupt Request 25** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA25	Output	**General-Purpose Output A25** When configured through port A bit 25, performs as a general-purpose output.
RX_DV	Input	**Receive Data Valid** MII receive data valid. See the *MC711x Reference Manual* for operation details.
CRS_DV	Input	**Carrier Sense/Receive Data Valid** RMII carrier sense/receive data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA26	Input	**General-Purpose Input A26 (default)** When configured through port A bit 26, performs as a general-purpose input.
$\overline{\text{IRQ26}}$	Input	**Interrupt Request 26** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA26	Output	**General-Purpose Output A26** When configured through port A bit 26, performs as a general-purpose output.
RX_ER	Input	**Receive Error** MII and RMII receive error. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
Reserved	Input	**Reserved D0 (default)** When configured through port D bit 0, a reserved signal.
COL	Input	**Collision** MII collision. In RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
Reserved	Input	**Reserved D1 (default)** When configured through port D bit 1, a reserved signal.
CRS	Input	**Carrier Sense** MII carrier sense. In RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
Reserved	Input	**Reserved D2 (default)** When configured through port D bit 2, a reserved signal.
MDC	Output	**Management Clock** MII and RMII management clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
H8BIT	Input	**Host 8/16 Bit Mode** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. If the line is pulled up at reset, the HDI16 operates in 8-bit mode when enabled. If the line is pulled down at reset, the HDI16 operates in 16-bit mode.

Table 2-6. Ethernet MAC Interface Signals (Continued)

Pin	Data Flow	Description
Reserved	Input	**Reserved D3 (default)** When configured through port D bit 3, a reserved signal.
MDIO	Input/Output	**Management Data** MII and RMII management data. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA27	Input	**General-Purpose Input A27 (default)** When configured through port A bit 27, performs as a general-purpose input.
$\overline{\text{IRQ16}}$	Input	**Interrupt Request 16** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA27	Output	**General-Purpose Output A27** When configured through port A bit 27, performs as a general-purpose output.
TXD3	Output	**Transmit Data 3** MII transmit data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPID4	Input	**General-Purpose Input D4 (default)** When configured through port D bit 4, performs as a general-purpose input.
GPOD4	Output	**General-Purpose Output D4** When configured through port D bit 4, performs as a general-purpose output.
TXD2	Output	**Transmit Data 2** MII transmit data bit 2. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA28	Input	**General-Purpose Input A28 (default)** When configured through port A bit 28, performs as a general-purpose input.
$\overline{\text{IRQ17}}$	Input	**Interrupt Request 17** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA28	Output	**General-Purpose Output A28** When configured through port A bit 28, performs as a general-purpose output.
TX_ER	Output	**Transmit Error** MII transmit error. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPID5	Input	**General-Purpose Input D5 (default)** When configured through port D bit 5, performs as a general-purpose input.
GPOD5	Output	**General-Purpose Output D5** When configured through port D bit 5, performs as a general-purpose output.
RXCLK	Output	**Receive Clock** MII receive clock. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-6. Ethernet MAC Interface Signals (Continued)

Pin	Data Flow	Description
GPIA29	Input	**General-Purpose Input A29 (default)** When configured through port A bit 29, performs as a general-purpose input.
$\overline{\text{IRQ18}}$	Input	**Interrupt Request 18** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA29	Input	**General-Purpose Output A29** When configured through port A bit 29, performs as a general-purpose output.
RXD3	Input	**Receive Data 3** MII receive data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPID6	Input	**General-Purpose Input D6 (default)** When configured through port D bit 6, performs as a general-purpose input.
GPOD6	Output	**General-Purpose Output D6** When configured through port D bit 6, performs as a general-purpose output.
RXD2	Input	**Receive Data 2** MII receive data bit 2. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

2.6 Host Interface Port

Table 2-7. Host Interface Signals

Pin	Data Flow	Description
GPIC11	Input	**General-Purpose Input C11 (default)** When configured through port C bit 11, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOC11	Output	**General-Purpose Output C11** When configured through port C bit 11, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
HA3	Input	**Host Address 3** Host address line 3. Tie this signal to ground.
HA[2–0]	Input	**Host Bus Address** The address lines used to address internal host registers. HA0 is the LSB of the host address bus.
GPIC[0–7]	Input	**General-Purpose Inputs C7–C0 (default)** When configured through port C bits 7–0, perform as a general-purpose inputs.
GPOC[0–7]	Output	**General-Purpose Outputs C7–C0** When configured through port C bits 7–0, perform as a general-purpose outputs.
HD[8–15]	Input/Output	**Host Data Bus (Upper Half)** The host data bus is used to access the internal host registers. See ***Chapter 20***, *Host Interface (HDI16)* for operation details.

Table 2-7. Host Interface Signals (Continued)

Pin	Data Flow	Description
Reserved	Input	**Reserved B7–B0 (default)** When configured through port B bits 7–0, reserved signals.
HD[7–0]	Input/Output	**Host Data Bus (Lower Half)** The host data bus is used to access the internal host registers. See the *MC711x Reference Manual* for operation details.
Reserved	Input	**Reserved B10 (default)** When configured through port B bit 10, reserved signal.
$\overline{\text{HCS1}}$/HCS1	Input	**Host Chip Select 1** When the HDI16 interface is enabled, this is one of the two chip-select pins. The polarity of this pin is programmable. The HDI16 chip select is a logical OR of HCS1 and HCS2 with appropriate polarity.
GPIB11	Input	**General-Purpose Input B11 (default)** When configured through port B bit 11, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOB11	Output	**General-Purpose Output B11** When configured through port B bit 11, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
$\overline{\text{HCS2}}$/HCS2	Input	**Host Chip Select 2** When the HDI16 interface is enabled, this is one of the two chip-select pins. The polarity of this pin is programmable. The HDI16 chip select is a logical OR of HCS1 and HCS2 with appropriate polarity. See ***Chapter 20***, *Host Interface (HDI16)* for operation details.
Reserved	Input	**Reserved B12 (default)** When configured through port B bit 12, reserved signal.
HRW	Input	**Host Read Write** When HDI16 is configured to work in single strobe mode, this is the Read/Write input (HRW).
$\overline{\text{HRD}}$/HRD	Input	**Host Read Data Strobe** When HDI16 is programmed to interface a double data strobe host bus, this pin is the Read Data Strobe input (HRD). The polarity of the data strobe is programmable.
Reserved	Input	**Reserved B13 (default)** When configured through port B bit 13, reserved signal.
$\overline{\text{HDS}}$/HDS	Input	**Host Data Strobe** When the HDI16 is programmed to interface a single data strobe host bus, this pin is the Data Strobe input (HDS). The polarity of the data strobe is programmable.
$\overline{\text{HWR}}$/HWR	Input	**Host Write Data Strobe** When the HDI16 is programmed to interface a double data strobe host bus, this pin is the Write Data Strobe input (HWR). The polarity of the data strobe is programmable.
Reserved	Input	**Reserved B14 (default)** When configured through port B bit 14, reserved signal.
HDDS	Input	**Host Dual Data Strobe** When the HDI16 is enabled, this pin indicates whether to use Single or Dual Data Strobe mode.

Table 2-7. Host Interface Signals (Continued)

Pin	Data Flow	Description
Reserved	Input	**Reserved B8 (default)** When configured through port B bit 8, reserved signal.
$\overline{\text{HREQ}}$/HREQ	Output	**Host Request** When the HDI16 is programmed to interface a single host request host bus, this pin is the Host Request output (HREQ). The polarity of the host request is programmable. The host request may be programmed as a driven or open-drain output. When configured for open drain, an external pull-up must be used on this pin.
$\overline{\text{HTRQ}}$/HTRQ	Output	**Host Transmit Request** When the HDI16 is programmed to interface a double host request host bus, this pin is the Transmit Host Request output (HTRQ). The polarity of the host request is programmable. The host request may be programmed as a driven or open-drain output. When configured for open drain, an external pull-up must be used on this pin.
HDSP	Input	**Host Data Strobe Polarity** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. This pin defines the polarity of host port read-write strobes.
Reserved	Input	**Reserved B9 (default)** When configured through port B bit 9, reserved signal.
$\overline{\text{HACK}}$/HACK	Input	**Host Acknowledge** When the HDI16 is programmed to interface a single host request host bus, this pin is the Host Acknowledge input (HACK). The polarity of the host acknowledge is programmable.
$\overline{\text{HRRQ}}$/HRRQ	Output	**Host Receive Request** When the HDI16 is programmed to interface a double host request host bus, this pin is the Receive Host Request output (HRRQ). The polarity of the host request is programmable. The host request may be programmed as a driven or open-drain output.
Reserved	Input	**Reserved D2 (default)** When configured through port D bit 2, a reserved signal.
MDC	Output	**Management Clock** MII and RMII management clock. See the *MC711x Reference Manual* for operation details.
H8BIT	Input	**Host 8/16 Bit Mode** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. If the line is pulled up at reset, the HDI16 operates in 8-bit mode when enabled. If the line is pulled down at reset, the HDI16 operates in 16-bit mode.

2.7 I²C Port

Table 2-8. I²C Interface Signals

Pin	Data Flow	Description
GPIA15	Input	**General-Purpose Input A15 (default)** When configured through port A bit 15, performs as a general-purpose input.
$\overline{\text{IRQ14}}$	Input	**Interrupt Request 14** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA15	Output	**General-Purpose Output A15** When configured through port A bit 15, performs as a general-purpose output.
SCL	Input/Output	**I²C Clock** The I²C clock signal. For I²C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.
GPIA14	Input	**General-Purpose Input A14 (default)** When configured through port A bit 14, performs as a general-purpose input.
$\overline{\text{IRQ15}}$	Input	**Interrupt Request 15** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA14	Output	**General-Purpose Output A14** When configured through port A bit 14, performs as a general-purpose output.
SDA	Input/Output	**I²C Data** I²C data signal. When used for I²C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.

2.8 UART Port

Table 2-9. UART Interface Signals

Pin	Data Flow	Description
GPIA13	Input	**General-Purpose Input A13 (default)** When configured through port A bit 13, performs as a general-purpose input.
$\overline{\text{IRQ2}}$	Input	**Interrupt Request 2** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA13	Output	**General-Purpose Output A13** When configured through port A bit 13, performs as a general-purpose output.
URXD	Input	**UART Receive Data** UART receive data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.

Table 2-9. UART Interface Signals

Pin	Data Flow	Description
GPIA12	Input	**General-Purpose Input A12 (default)** When configured through port A bit 12, performs as a general-purpose input.
$\overline{\text{IRQ3}}$	Input	**Interrupt Request 3** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA12	Output	**General-Purpose Output A12** When configured through port A bit 12, performs as a general-purpose output.
UTXD	Output	**UART Transmit Data** UART transmit data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.

2.9 Event Port

Table 2-10. Event Port Signals

Pin	Data Flow	Description
EVNT0	Input/Output	**Event 0** Provides input and output events to the system control unit event multiplexers.
GPIA17	Input	**General-Purpose Input A17 (default)** When configured through port A bit 17, performs as a general-purpose input.
$\overline{\text{IRQ13}}$	Input	**Interrupt Request 13** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA17	Output	**General-Purpose Output A17** When configured through port A bit 17, performs as a general-purpose output.
EVNT1	Input/ Output	**Event 1** Provides input and output events to the system control unit event multiplexers.
CLKO	Output	**CLKO** Output clock signal when the function is enabled.
GPIC14	Input	**General-Purpose Input C14 (default)** When configured through port C bit 14, performs as a general-purpose input.
GPOC14	Output	**General-Purpose Output C14** When configured through port C bit 14, performs as a general-purpose output.
EVNT2	Input/ Output	**Event 2** Provides input and output events to the system control unit event multiplexers.
BM0	Input	**Boot Mode 0** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. With BM1, the value of this signal defines the boot mode of the MSC71xx. See the *MC711x Reference Manual* for operation details.

Table 2-10. Event Port Signals (Continued)

Pin	Data Flow	Description
GPIC15	Input	**General-Purpose Input C15 (default)** When configured through port C bit 15, performs as a general-purpose input.
GPOC15	Output	**General-Purpose Output C15** When configured through port C bit 15, performs as a general-purpose output.
EVNT3	Input/ Output	**Event 3** Provides input and output events to the system control unit event multiplexers.
BM1	Input	**Boot Mode 1** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. With BM0, the value of this signal defines the boot mode of the MSC71xx. See the *MC711x Reference Manual* for operation details.
GPIA16	Input	**General-Purpose Input A16 (default)** When configured through port A bit 16, performs as a general-purpose input.
$\overline{\text{IRQ12}}$	Input	**Interrupt Request 12** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA16	Output	**General-Purpose Output A16** When configured through port A bit 16, performs as a general-purpose output.
EVNT4	Input/ Output	**Event 4** Provides input and output events for the system control unit event multiplexers. Can be used to indicate that the SC1400 core is in Debug mode.
SWTE	Input	**Software Watchdog Timer Disable** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. If the signal is sampled high, the watchdog timer is enabled. If it is sampled low, the watchdog timer is disabled.

2.10 GPIO Ports

See **Chapter 24**, *General-Purpose Input/Output (GPIO)* for details about programming these ports.

Table 2-11. GPIO Port Signals

Pin	Data Flow	Description
GPIA0	Input	**General-Purpose Input A9 (default)** When configured through port A bit 6, performs as a general-purpose input.
$\overline{\text{IRQ11}}$	Input	**Interrupt Request 11** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA0	Output	**General-Purpose Output A0** When configured through port A bit 0, performs as a general-purpose output.
T1TD	Input/Output	**TDM1 Transmit Data** The transmit data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA1	Input	**General-Purpose Input A1 (default)** When configured through port A bit 1, performs as a general-purpose input.
$\overline{\text{IRQ10}}$	Input	**Interrupt Request 10** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA1	Output	**General-Purpose Output A1** When configured through port A bit 1, performs as a general-purpose output.
T1TFS	Input/Output	**TDM1 Transmit Frame Sync** The transmit frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA2	Input	**General-Purpose Input A2 (default)** When configured through port A bit 2, performs as a general-purpose input.
$\overline{\text{IRQ9}}$	Input	**Interrupt Request 9** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA2	Output	**General-Purpose Output A2** When configured through port A bit 2, performs as a general-purpose output.
T1TCK	Input/Output	**TDM1 Transmit Clock** The transmit clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA3	Input	**General-Purpose Input A3 (default)** When configured through port A bit 3, performs as a general-purpose input.
$\overline{\text{IRQ8}}$	Input	**Interrupt Request 8** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA3	Output	**General-Purpose Output A3** When configured through port A bit 3, performs as a general-purpose output.
T1RD	Input/Output	**TDM1 Receive Data** The receive data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA4	Input	**General-Purpose Input A4 (default)** When configured through port A bit 4, performs as a general-purpose input.
$\overline{\text{IRQ1}}$	Input	**Interrupt Request 1** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA4	Output	**General-Purpose Output A4** When configured through port A bit 4, performs as a general-purpose output.
T1RFS	Input/Output	**TDM1 Receive Frame Sync** The receive frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA5	Input	**General-Purpose Input A5 (default)** When configured through port A bit 5, performs as a general-purpose input.
$\overline{\text{IRQ0}}$	Input	**Interrupt Request 0** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA5	Output	**General-Purpose Output A5** When configured through port A bit 5, performs as a general-purpose output.
T1RCK	Input/Output	**TDM1 Receive Clock** The receive clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA6	Input	**General-Purpose Input A9 (default)** When configured through port A bit 6, performs as a general-purpose input.
GPOA6	Output	**General-Purpose Output A6** When configured through port A bit 6, performs as a general-purpose output.
T0TD	Input/Output	**TDM0 Transmit Data** The transmit data for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA7	Input	**General-Purpose Input A7 (default)** When configured through port A bit 7, performs as a general-purpose input.
$\overline{\text{IRQ7}}$	Input	**Interrupt Request 7** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA7	Output	**General-Purpose Output A7** When configured through port A bit 7, performs as a general-purpose output.
T0TFS	Input/Output	**TDM0 Transmit Frame Sync** The transmit frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA8	Input	**General-Purpose Input A8 (default)** When configured through port A bit 8, performs as a general-purpose input.
$\overline{\text{IRQ6}}$	Input	**Interrupt Request 6** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA8	Output	**General-Purpose Output A8** When configured through port A bit 8, performs as a general-purpose output.
T0TCK	Input/Output	**TDM0 Transmit Clock** The transmit clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA9	Input	**General-Purpose Input A9 (default)** When configured through port A bit 9, performs as a general-purpose input.
GPOA9	Output	**General-Purpose Output A9** When configured through port A bit 9, performs as a general-purpose output.
T0RD	Input/Output	**TDM0 Receive Data** The receive data for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA10	Input	**General-Purpose Input A10 (default)** When configured through port A bit 10, performs as a general-purpose input.
$\overline{\text{IRQ5}}$	Input	**Interrupt Request 5** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA10	Output	**General-Purpose Output A10** When configured through port A bit 10, performs as a general-purpose output.
T0RFS	Input/Output	**TDM0 Receive Frame Sync** The receive frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA11	Input	**General-Purpose Input A11 (default)** When configured through port A bit 11, performs as a general-purpose input.
$\overline{\text{IRQ4}}$	Input	**Interrupt Request 4** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA11	Output	**General-Purpose Output A11** When configured through port A bit 11, performs as a general-purpose output.
T0RCK	Input/Output	**TDM0 Receive Clock** The receive clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA12	Input	**General-Purpose Input A12 (default)** When configured through port A bit 12, performs as a general-purpose input.
$\overline{\text{IRQ3}}$	Input	**Interrupt Request 3** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA12	Output	**General-Purpose Output A12** When configured through port A bit 12, performs as a general-purpose output.
UTXD	Output	**UART Transmit Data** UART transmit data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.
GPIA13	Input	**General-Purpose Input A13 (default)** When configured through port A bit 13, performs as a general-purpose input.
$\overline{\text{IRQ2}}$	Input	**Interrupt Request 2** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA13	Output	**General-Purpose Output A13** When configured through port A bit 13, performs as a general-purpose output.
URXD	Input	**UART Receive Data** UART receive data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA14	Input	**General-Purpose Input A14 (default)** When configured through port A bit 14, performs as a general-purpose input.
$\overline{\text{IRQ15}}$	Input	**Interrupt Request 15** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA14	Output	**General-Purpose Output A14** When configured through port A bit 14, performs as a general-purpose output.
SDA	Input/Output	**I^{2}C Data** I^{2}C data signal. When used for I^{2}C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.
GPIA15	Input	**General-Purpose Input A15 (default)** When configured through port A bit 15, performs as a general-purpose input.
$\overline{\text{IRQ14}}$	Input	**Interrupt Request 14** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA15	Output	**General-Purpose Output A15** When configured through port A bit 15, performs as a general-purpose output.
SCL	Input/Output	**I^{2}C Clock** The I^{2}C clock signal. For I^{2}C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.
GPIA16	Input	**General-Purpose Input A16 (default)** When configured through port A bit 16, performs as a general-purpose input.
$\overline{\text{IRQ12}}$	Input	**Interrupt Request 12** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA16	Output	**General-Purpose Output A16** When configured through port A bit 16, performs as a general-purpose output.
EVNT4	Input/ Output	**Event 4** Provides input and output events for the system control unit event multiplexers. Can be used to indicate that the SC1400 core is in Debug mode.
SWTE	Input	**Software Watchdog Timer Disable** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. If the signal is sampled high, the watchdog timer is enabled. If it is sampled low, the watchdog timer is disabled.
GPIA17	Input	**General-Purpose Input A17 (default)** When configured through port A bit 17, performs as a general-purpose input.
$\overline{\text{IRQ13}}$	Input	**Interrupt Request 13** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA17	Output	**General-Purpose Output A17** When configured through port A bit 17, performs as a general-purpose output.
EVNT1	Input/ Output	**Event 1** Provides input and output events to the system control unit event multiplexers.
CLKO	Output	**CLKO** Output clock signal when the function is enabled.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA19	Input	**General-Purpose Input A19 (default)** When configured through port A bit 19, performs as a general-purpose input.
$\overline{\text{IRQ19}}$	Input	**Interrupt Request 19** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA19	Output	**General-Purpose Output A19** When configured through port A bit 19, performs as a general-purpose output.
TXD1	Output	**Transmit Data 1 (MSC7113, MSC7116, and MSC7119)** MII and RMII transmit data bit 1. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA20	Input	**General-Purpose Input A20 (default)** When configured through port A bit 20, performs as a general-purpose input.
$\overline{\text{IRQ20}}$	Input	**Interrupt Request 20** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA20	Output	**General-Purpose Output A20** When configured through port A bit 20, performs as a general-purpose output.
TXD0	Output	**Transmit Data 0 (MSC7113, MSC7116, and MSC7119)** MII and RMII transmit data bit 0. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA21	Input	**General-Purpose Input A21 (default)** When configured through port A bit 21, performs as a general-purpose input.
$\overline{\text{IRQ21}}$	Input	**Interrupt Request 21** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA21	Output	**General-Purpose Output A21** When configured through port A bit 21, performs as a general-purpose output.
RXD1	Input	**Receive Data 1 (MSC7113, MSC7116, and MSC7119)** MII and RMII receive data bit 1. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA22	Input	**General-Purpose Input A22 (default)** When configured through port A bit 22, performs as a general-purpose input.
$\overline{\text{IRQ22}}$	Input	**Interrupt Request 22** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA22	Output	**General-Purpose Output A22** When configured through port A bit 22, performs as a general-purpose output.
RXD0	Input	**Receive Data 0 (MSC7113, MSC7116, and MSC7119)** MII and RMII receive data bit 0. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA23	Input	**General-Purpose Input A23 (default)** When configured through port A bit 23, performs as a general-purpose input.
$\overline{\text{IRQ23}}$	Input	**Interrupt Request 23** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA23	Output	**General-Purpose Output A23** When configured through port A bit 23, performs as a general-purpose output.
TXCLK	Input	**Transmit Clock (MSC7113, MSC7116, and MSC7119)** MII transmit clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
REFCLK	Input	**Reference Clock (MSC7113, MSC7116, and MSC7119)** RMII reference clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA24	Input	**General-Purpose Input A24 (default)** When configured through port A bit 24, performs as a general-purpose input.
$\overline{\text{IRQ24}}$	Input	**Interrupt Request 24** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA24	Output	**General-Purpose Output A24** When configured through port A bit 24, performs as a general-purpose output.
TX_EN	Output	**Transmit Data Valid (MSC7113, MSC7116, and MSC7119)** MII and RMII transmit data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA25	Input	**General-Purpose Input A25 (default)** When configured through port A bit 25, performs as a general-purpose input.
$\overline{\text{IRQ25}}$	Input	**Interrupt Request 25** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA25	Output	**General-Purpose Output A25** When configured through port A bit 25, performs as a general-purpose output.
RX_DV	Input	**Receive Data Valid (MSC7113, MSC7116, and MSC7119)** MII receive data valid. See the *MC711x Reference Manual* for operation details.
CRS_DV	Input	**Carrier Sense/Receive Data Valid (MSC7113, MSC7116, and MSC7119)** RMII carrier sense/receive data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA26	Input	**General-Purpose Input A26 (default)** When configured through port A bit 26, performs as a general-purpose input.
$\overline{\text{IRQ26}}$	Input	**Interrupt Request 26** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA26	Output	**General-Purpose Output A26** When configured through port A bit 26, performs as a general-purpose output.
RX_ER	Input	**Receive Error (MSC7113, MSC7116, and MSC7119)** MII and RMII receive error. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIA27	Input	**General-Purpose Input A27 (default)** When configured through port A bit 27, performs as a general-purpose input.
$\overline{\text{IRQ16}}$	Input	**Interrupt Request 16** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA27	Output	**General-Purpose Output A27** When configured through port A bit 27, performs as a general-purpose output.
TXD3	Output	**Transmit Data 3 (MSC7113, MSC7116, and MSC7119)** MII transmit data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2RCK	Input/Output	**TDM2 Receive Clock (MSC7115 and MSC7118)** The receive clock for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA28	Input	**General-Purpose Input A28 (default)** When configured through port A bit 28, performs as a general-purpose input.
$\overline{\text{IRQ17}}$	Input	**Interrupt Request 17** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA28	Output	**General-Purpose Output A28** When configured through port A bit 28, performs as a general-purpose output.
TX_ER	Output	**Transmit Error (MSC7113, MSC7116, and MSC7119)** MII transmit error. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2RD	Input/Output	**TDM2 Receive Data (MSC7115 and MSC7118)** The receive data for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA29	Input	**General-Purpose Input A29 (default)** When configured through port A bit 29, performs as a general-purpose input.
$\overline{\text{IRQ18}}$	Input	**Interrupt Request 18** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA29	Input	**General-Purpose Output A29** When configured through port A bit 29, performs as a general-purpose output.
RXD3	Input	**Receive Data 3 (MSC7113, MSC7116, and MSC7119)** MII receive data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2TFS	Input/Output	**TDM2 Transmit Frame Sync (MSC7115 and MSC7118)** The transmit frame sync for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIB11	Input	**General-Purpose Input B11 (default)** When configured through port B bit 11, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOB11	Output	**General-Purpose Output B11** When configured through port B bit 11, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
$\overline{\text{HCS2}}$/HCS2	Input	**Host Chip Select 2** When the HDI16 interface is enabled, this is one of the two chip-select pins. The polarity of this pin is programmable. The HDI16 chip select is a logical OR of HCS1 and HCS2 with appropriate polarity. See ***Chapter 20***, *Host Interface (HDI16)* for operation details.
GPIC[0–7]	Input	**General-Purpose Inputs C7–C0 (default)** When configured through port C bits 7–0, perform as a general-purpose inputs.
GPOC[0–7]	Output	**General-Purpose Outputs C7–C0** When configured through port C bits 7–0, perform as a general-purpose outputs.
HD[8–15]	Input/Output	**Host Data Bus (Upper Half)** The host data bus is used to access the internal host registers. See ***Chapter 20***, *Host Interface (HDI16)* for operation details.
GPIC11	Input	**General-Purpose Input C11 (default)** When configured through port C bit 11, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOC11	Output	**General-Purpose Output C11** When configured through port C bit 11, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
HA3	Input	**Host Address 3** Host address line 3. Tie this signal to ground.
GPIC14	Input	**General-Purpose Input C14 (default)** When configured through port C bit 14, performs as a general-purpose input.
GPOC14	Output	**General-Purpose Output C14** When configured through port C bit 14, performs as a general-purpose output.
EVNT2	Input/ Output	**Event 2** Provides input and output events to the system control unit event multiplexers.
BM0	Input	**Boot Mode 0** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. With BM1, the value of this signal defines the boot mode of the MSC71xx. See the *MC711x Reference Manual* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPIC15	Input	**General-Purpose Input C15 (default)** When configured through port C bit 15, performs as a general-purpose input.
GPOC15	Output	**General-Purpose Output C15** When configured through port C bit 15, performs as a general-purpose output.
EVNT3	Input/ Output	**Event 3** Provides input and output events to the system control unit event multiplexers.
BM1	Input	**Boot Mode 1** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. With BM0, the value of this signal defines the boot mode of the MSC71xx. See the *MC711x Reference Manual* for operation details.
GPID4	Input	**General-Purpose Input D4 (default)** When configured through port D bit 4, performs as a general-purpose input.
GPOD4	Output	**General-Purpose Output D4** When configured through port D bit 4, performs as a general-purpose output.
TXD2	Output	**Transmit Data 2 (MSC7113, MSC7116, and MSC7119)** MII transmit data bit 2. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2RFS	Input/Output	**TDM2 Receive Frame Sync (MSC7115 and MSC7118)** The receive frame sync for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPID5	Input	**General-Purpose Input D5 (default)** When configured through port D bit 5, performs as a general-purpose input.
GPOD5	Output	**General-Purpose Output D5** When configured through port D bit 5, performs as a general-purpose output.
RXCLK	Output	**Receive Clock (MSC7113, MSC7116, and MSC7119)** MII receive clock. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2TCK	Input/Output	**TDM2 Transmit Clock (MSC7115 and MSC7118)** The transmit clock for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPID6	Input	**General-Purpose Input D6 (default)** When configured through port D bit 6, performs as a general-purpose input.
GPOD6	Output	**General-Purpose Output D6** When configured through port D bit 6, performs as a general-purpose output.
RXD2 ‘	Input	**Receive Data 2 (MSC7113, MSC7116, and MSC7119)** MII receive data bit 2. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2TD	Input/Output	**TDM2 Transmit Data (MSC7115 and MSC7118)** The transmit data for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-11. GPIO Port Signals (Continued)

Pin	Data Flow	Description
GPID7	Input	**General-Purpose Input D7 (default)** When configured through port D bit 7, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOD7	Output	**General-Purpose Output B11** When configured through port D bit 7, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
BM2	Input	**Boot Mode 2** For the 1M88B mask set only, this pin is sampled at the deassertion of $\overline{\text{PORESET}}$. Along with BM[0–1] and BM3, the value of this signal defines the boot mode for the device. For designs developed using the 1L44X mask set, this signal can be left unconnected.
GPID8	Input	**Reserved D8 or General-Purpose Input D8 (default)** When configured through port D bit 8, a reserved signal (mask set 1L44X) or a general-purpose input (mask set 1M88B).
GPOD8	Output	**Reserved D8 or General-Purpose Output D8** When configured through port D bit 8, a reserved signal (mask set 1L44X) or a general-purpose output (mask set 1M88B).
BM3	Input	**Boot Mode 3** For the 1M88B mask set only, this pin is sampled at the deassertion of $\overline{\text{PORESET}}$. Along with BM[0–2] the value of this signal defines the boot mode for the device. For designs developed using the 1L44X mask set, this signal can be left unconnected.

2.11 Interrupts

Table 2-12. Interrupt Signals

Pin	Data Flow	Description
$\overline{\text{NMI}}$	Input	**Non-Maskable Interrupt** External device may assert this line to generate a non-maskable interrupt to the MSC71xx device.
GPIA5	Input	**General-Purpose Input A5 (default)** When configured through port A bit 5, performs as a general-purpose input.
$\overline{\text{IRQ0}}$	Input	**Interrupt Request 0** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA5	Output	**General-Purpose Output A5** When configured through port A bit 5, performs as a general-purpose output.
T1RCK	Input/Output	**TDM1 Receive Clock** The receive clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA4	Input	**General-Purpose Input A4 (default)** When configured through port A bit 4, performs as a general-purpose input.
$\overline{\text{IRQ1}}$	Input	**Interrupt Request 1** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA4	Output	**General-Purpose Output A4** When configured through port A bit 4, performs as a general-purpose output.
T1RFS	Input/Output	**TDM1 Receive Frame Sync** The receive frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA13	Input	**General-Purpose Input A13 (default)** When configured through port A bit 13, performs as a general-purpose input.
$\overline{\text{IRQ2}}$	Input	**Interrupt Request 2** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA13	Output	**General-Purpose Output A13** When configured through port A bit 13, performs as a general-purpose output.
URXD	Input	**UART Receive Data** UART receive data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.
GPIA12	Input	**General-Purpose Input A12 (default)** When configured through port A bit 12, performs as a general-purpose input.
$\overline{\text{IRQ3}}$	Input	**Interrupt Request 3** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA12	Output	**General-Purpose Output A12** When configured through port A bit 12, performs as a general-purpose output.
UTXD	Output	**UART Transmit Data** UART transmit data line. See **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)* for details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA11	Input	**General-Purpose Input A11 (default)** When configured through port A bit 11, performs as a general-purpose input.
$\overline{\text{IRQ4}}$	Input	**Interrupt Request 4** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA11	Output	**General-Purpose Output A11** When configured through port A bit 11, performs as a general-purpose output.
T0RCK	Input/Output	**TDM0 Receive Clock** The receive clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA10	Input	**General-Purpose Input A10 (default)** When configured through port A bit 10, performs as a general-purpose input.
$\overline{\text{IRQ5}}$	Input	**Interrupt Request 5** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA10	Output	**General-Purpose Output A10** When configured through port A bit 10, performs as a general-purpose output.
T0RFS	Input/Output	**TDM0 Receive Frame Sync** The receive frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA8	Input	**General-Purpose Input A8 (default)** When configured through port A bit 8, performs as a general-purpose input.
$\overline{\text{IRQ6}}$	Input	**Interrupt Request 6** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA8	Output	**General-Purpose Output A8** When configured through port A bit 8, performs as a general-purpose output.
T0TCK	Input/Output	**TDM0 Transmit Clock** The transmit clock for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA7	Input	**General-Purpose Input A7 (default)** When configured through port A bit 7, performs as a general-purpose input.
$\overline{\text{IRQ7}}$	Input	**Interrupt Request 7** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA7	Output	**General-Purpose Output A7** When configured through port A bit 7, performs as a general-purpose output.
T0TFS	Input/Output	**TDM0 Transmit Frame Sync** The transmit frame sync for TDM0. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA3	Input	**General-Purpose Input A3 (default)** When configured through port A bit 3, performs as a general-purpose input.
$\overline{\text{IRQ8}}$	Input	**Interrupt Request 8** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA3	Output	**General-Purpose Output A3** When configured through port A bit 3, performs as a general-purpose output.
T1RD	Input/Output	**TDM1 Receive Data** The receive data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA2	Input	**General-Purpose Input A2 (default)** When configured through port A bit 2, performs as a general-purpose input.
$\overline{\text{IRQ9}}$	Input	**Interrupt Request 9** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA2	Output	**General-Purpose Output A2** When configured through port A bit 2, performs as a general-purpose output.
T1TCK	Input/Output	**TDM1 Transmit Clock** The transmit clock for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA1	Input	**General-Purpose Input A1 (default)** When configured through port A bit 1, performs as a general-purpose input.
$\overline{\text{IRQ10}}$	Input	**Interrupt Request 10** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA1	Output	**General-Purpose Output A1** When configured through port A bit 1, performs as a general-purpose output.
T1TFS	Input/Output	**TDM1 Transmit Frame Sync** The transmit frame sync for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA0	Input	**General-Purpose Input A9 (default)** When configured through port A bit 6, performs as a general-purpose input.
$\overline{\text{IRQ11}}$	Input	**Interrupt Request 11** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA0	Output	**General-Purpose Output A0** When configured through port A bit 0, performs as a general-purpose output.
T1TD	Input/Output	**TDM1 Transmit Data** The transmit data for TDM1. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA16	Input	**General-Purpose Input A16 (default)** When configured through port A bit 16, performs as a general-purpose input.
$\overline{\text{IRQ12}}$	Input	**Interrupt Request 12** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA16	Output	**General-Purpose Output A16** When configured through port A bit 16, performs as a general-purpose output.
EVNT4	Input/ Output	**Event 4** Provides input and output events for the system control unit event multiplexers. Can be used to indicate that the SC1400 core is in Debug mode.
SWTE	Input	**Software Watchdog Timer Disable** This pin is sampled at the deassertion of $\overline{\text{PORESET}}$. If the signal is sampled high, the watchdog timer is enabled. If it is sampled low, the watchdog timer is disabled.
GPIA17	Input	**General-Purpose Input A17 (default)** When configured through port A bit 17, performs as a general-purpose input.
$\overline{\text{IRQ13}}$	Input	**Interrupt Request 13** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA17	Output	**General-Purpose Output A17** When configured through port A bit 17, performs as a general-purpose output.
EVNT1	Input/ Output	**Event 1** Provides input and output events to the system control unit event multiplexers.
CLKO	Output	**CLKO** Output clock signal when the function is enabled.
GPIA15	Input	**General-Purpose Input A15 (default)** When configured through port A bit 15, performs as a general-purpose input.
$\overline{\text{IRQ14}}$	Input	**Interrupt Request 14** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA15	Output	**General-Purpose Output A15** When configured through port A bit 15, performs as a general-purpose output.
SCL	Input/Output	**I^2C Clock** The I^2C clock signal. For I^2C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.
GPIA14	Input	**General-Purpose Input A14 (default)** When configured through port A bit 14, performs as a general-purpose input.
$\overline{\text{IRQ15}}$	Input	**Interrupt Request 15** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA14	Output	**General-Purpose Output A14** When configured through port A bit 14, performs as a general-purpose output.
SDA	Input/Output	**I^2C Data** I^2C data signal. When used for I^2C, use an external pull-up on this pin. See ***Chapter 22***, *I2C Software Module* for operation details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA27	Input	**General-Purpose Input A27 (default)** When configured through port A bit 27, performs as a general-purpose input.
$\overline{\text{IRQ16}}$	Input	**Interrupt Request 16** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA27	Output	**General-Purpose Output A27** When configured through port A bit 27, performs as a general-purpose output.
TXD3	Output	**Transmit Data 3** MII transmit data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2RCK	Input/Output	**TDM2 Receive Clock** The receive clock for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA28	Input	**General-Purpose Input A28 (default)** When configured through port A bit 28, performs as a general-purpose input.
$\overline{\text{IRQ17}}$	Input	**Interrupt Request 17** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA28	Output	**General-Purpose Output A28** When configured through port A bit 28, performs as a general-purpose output.
TX_ER	Output	**Transmit Error** MII transmit error. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2RD	Input/Output	**TDM2 Receive Data** The receive data for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.
GPIA29	Input	**General-Purpose Input A29 (default)** When configured through port A bit 29, performs as a general-purpose input.
$\overline{\text{IRQ18}}$	Input	**Interrupt Request 18** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA29	Input	**General-Purpose Output A29** When configured through port A bit 29, performs as a general-purpose output.
RXD3	Input	**Receive Data 3** MII receive data bit 3. For RMII mode, this signal is reserved. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
T2TFS	Input/Output	**TDM2 Transmit Frame Sync** The transmit frame sync for TDM2. See ***Chapter 19***, *Time-Division Multiplexing (TDM) Interface* for operation details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA21	Input	**General-Purpose Input A21 (default)** When configured through port A bit 21, performs as a general-purpose input.
$\overline{\text{IRQ21}}$	Input	**Interrupt Request 21** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA21	Output	**General-Purpose Output A21** When configured through port A bit 21, performs as a general-purpose output.
RXD1	Input	**Receive Data 1** MII and RMII receive data bit 1. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA22	Input	**General-Purpose Input A22 (default)** When configured through port A bit 22, performs as a general-purpose input.
$\overline{\text{IRQ22}}$	Input	**Interrupt Request 22** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA22	Output	**General-Purpose Output A22** When configured through port A bit 22, performs as a general-purpose output.
RXD0	Input	**Receive Data 0** MII and RMII receive data bit 0. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA23	Input	**General-Purpose Input A23 (default)** When configured through port A bit 23, performs as a general-purpose input.
$\overline{\text{IRQ23}}$	Input	**Interrupt Request 23** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA23	Output	**General-Purpose Output A23** When configured through port A bit 23, performs as a general-purpose output.
TXCLK	Input	**Transmit Clock** MII transmit clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
REFCLK	Input	**Reference Clock** RMII reference clock. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA24	Input	**General-Purpose Input A24 (default)** When configured through port A bit 24, performs as a general-purpose input.
$\overline{\text{IRQ24}}$	Input	**Interrupt Request 24** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA24	Output	**General-Purpose Output A24** When configured through port A bit 24, performs as a general-purpose output.
TX_EN	Output	**Transmit Data Valid** MII and RMII transmit data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

Table 2-12. Interrupt Signals (Continued)

Pin	Data Flow	Description
GPIA25	Input	**General-Purpose Input A25 (default)** When configured through port A bit 25, performs as a general-purpose input.
$\overline{\text{IRQ25}}$	Input	**Interrupt Request 25** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA25	Output	**General-Purpose Output A25** When configured through port A bit 25, performs as a general-purpose output.
RX_DV	Input	**Receive Data Valid** MII receive data valid. See the *MC711x Reference Manual* for operation details.
CRS_DV	Input	**Carrier Sense/Receive Data Valid** RMII carrier sense/receive data valid. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.
GPIA26	Input	**General-Purpose Input A26 (default)** When configured through port A bit 26, performs as a general-purpose input.
$\overline{\text{IRQ26}}$	Input	**Interrupt Request 26** One of the 27 maskable interrupts that can be configured for the MSC71xx device.
GPOA26	Output	**General-Purpose Output A26** When configured through port A bit 26, performs as a general-purpose output.
RX_ER	Input	**Receive Error** MII and RMII receive error. See ***Chapter 18***, *Fast Ethernet Controller (FEC)* for operation details.

2.12 JTAG/OCE10 Enhanced On-Chip Emulator Port

Table 2-13. JTAG/OCE10 Port

Pin	Data Flow	Description
TCK	Input	**Test Clock** (JTAG) Clock input for the MSC711x JTAG/COP controller to synchronize the test logic.
TDI	Input	**Test Data In** (JTAG) A test data input (with an internal pull-up resistor) that is sampled on the rising edge of TCK.
TDO	Output	**Test Data Out** (JTAG) A data output that can be three-stated and actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.
TMS	Input	**Test Mode Select** (JTAG) A test mode select input (with an internal pull-up resistor) that is sampled on the rising edge of TCK to sequence the TAP controllers state machine.
$\overline{\text{TRST}}$	Input	**Test Reset** (JTAG) The reset input to the MSC711x JTAG/COP controller (with an internal pull-up resistor).

Table 2-13. JTAG/OCE10 Port (Continued)

<table>
<tr><th>Pin</th><th>Data Flow</th><th>Description</th></tr>
<tr><td>DBREQ</td><td>Input</td><td>Debug Request
Puts the MSC711x device into Debug mode (EE0DEF=11 in SC1400 EE_CTRL register).</td></tr>
<tr><td>EE0</td><td>Input/Output</td><td>EOnCE Event Bit 0
Debug port EE0 functionality. EDCA0 indicates detection when EE0DEF = 00. Generates an OCE10 emulator event or enables EDCA0 when EE0DEF = 10 in the SC1400 EE_CTRL register.

Note: The selection between DBREQ and EE0 occurs in the emulator EE_CTRL[EE0DEF]:

<table>
<tr><td>00</td><td>EE0 pin is configured as an output.</td></tr>
<tr><td>01</td><td>Reserved.</td></tr>
<tr><td>10</td><td>EE0 pin is configured as an input to enable EDCA0 or generate an emulator event.</td></tr>
<tr><td>11</td><td>EE0 pin is configured as a debug request input (also enables EDCA0 or generates an emulator event).</td></tr>
</table>
</td></tr>
<tr><td>TPSEL</td><td>Input</td><td>Tap Select
When this signal is deasserted, the boundary scan TAP controller is selected, allowing for boundary scan. When this signal is asserted, the Debug TAP controller is selected, allowing access to the emulator.</td></tr>
</table>

2.13 Boot Behavior of Pins

When the device exits f reset, the boot program configures different pins for the boot operation. When booting completes, these pins may be in a different state than their values immediately out of reset. **Table 2-14** shows how the pins are used during an HDI boot.

Table 2-14. Pin Behavior During HDI Boot

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
Boot Mode Pins					
BM3/GPIO	BM3	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM2/GPIO	BM2	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM1/GPIO/EVNT3	BM1	GPIO-I	Remains as is	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM0/GPIO/EVNT2	BM0	GPIO-I	Remains as is	GPIO/EVNT2	Sampled when $\overline{\text{PORESET}}$ is deasserted.
HDSP/HTRQ/ HREQ	HDSP	HTRQ/ HREQ	Remains as is	HTRQ/HREQ	Uses HTRQ functionality during boot if configured for this in ICR[HDRQ]. Otherwise uses HREQ.

Table 2-14. Pin Behavior During HDI Boot (Continued)

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
MDC/H8BIT	H8BIT	(input)	Remains as is	MDC	Sampled when $\overline{\text{PORESET}}$ is deasserted. Configured as input during reset.
Peripheral Port Pins					
HA3/GPIO	—	HA3	Remains as is	GPIO	HA3 functionality is unused. When the device exits reset, this pin is not configured for GPIO. If GPIO is desired, it must explicitly be configured.
HA[2–0]	—	HA[2–0]	Remains as is	HA[2:0]	Configured by the boot program in the GPIO.
HD[15–0]	—	HD[15–0]	Remains as is	HD[15:0]	Configured by the boot program in the GPIO.
GPIO/HCS2	—	HCS2	Remains as is	GPIO/HCS2	DEVCFG[HCOV] is cleared.
HCS1	—	HCS1	Remains as is	HCS1	—
HRD/HRW	—	HRD/HRW	Remains as is	HRD/HRW	Operation set by HDDS and HDSP.
HWR/HDS	—	HWR/HDS	Remains as is	HWR/HDS	Operation set by HDDS and HDSP.
HDDS	—	HDDS	Remains as is	HDDS	—
HRRQ/HACK	—	HRRQ	Remains as is	HRRQ/HACK	Uses HRRQ functionality during boot if configured for this in ICR[HDRQ]. HACK is unused during boot.
HDSP/HTRQ/ HREQ	HDSP	HTRQ/ HREQ	Remains as is	HTRQ/HREQ	Uses HTRQ functionality during boot if configured for this in ICR[HDRQ]. Otherwise, it uses HREQ.

Note: Errors are reported through the HCR[HF7] bit. No pins are used.

Table 2-15 shows how pins are used during an SPI boot using the main pins.

Table 2-15. Pin Behavior During an SPI Boot from the Main Pins

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
Boot Mode Pins					
BM3/GPIO	BM3	GPIO-O	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then it is used as the SPI MOSI pin.
BM2/GPIO	BM2	GPIO-O	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then it is used as the SPI SPICLK pin.
BM1/GPIO/EVNT3	BM1	GPIO-I	Remains as is	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM0/GPIO/EVNT2	BM0	GPIO-I	Remains as is	GPIO/EVNT2	Sampled when $\overline{\text{PORESET}}$ is deasserted.
Peripheral Port Pins					
HA3/GPIO	—	GPIO-O	Remains as is	GPIO	Used as the SPI SEL pin.

Table 2-15. Pin Behavior During an SPI Boot from the Main Pins (Continued)

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
HCS2/GPIO	—	GPIO-I	Remains as is	HCS2/GPIO	Used as the SPI MISO pin.
BM3/GPIO	BM3	GPIO-O	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then used as the SPI MOSI pin.
BM2/GPIO	BM2	GPIO-O	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then used as the SPI SPICLK pin.
Error Reporting Pin					
BM1/GPIO/EVNT3	BM1	GPIO-I or GPIO-O	GPIO-I	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then it optionally signals a failed SPI boot. Configures as a GP output only if an error is detected.
Note: The boot program also sets the override capability in DEVCFG[HCOV] so that the HCS2 pin can be used as GPIO.					

Table 2-16 shows how pins are used during an SPI boot using the alternate pins.

Table 2-16. Pin Behavior During An SPI Boot from the Alternate Pins

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
Boot Mode Pins					
BM3/GPIO	BM3	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM2/GPIO	BM2	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM1/GPIO/EVNT3	BM1	GPIO-I	Remains as is	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM0/GPIO/EVNT2	BM0	GPIO-I	Remains as is	GPIO/EVNT2	Sampled when $\overline{\text{PORESET}}$ is deasserted.
Peripheral Port Pins					
$\overline{\text{IRQ14}}$/GPIO/SCL	—	GPIO-I	Remains as is	$\overline{\text{IRQ14}}$/GPIO/SCL	The SPI MISO pin.
$\overline{\text{IRQ15}}$/GPIO/SDA	—	GPIO-O	Remains as is	$\overline{\text{IRQ15}}$/GPIO/SDA	The SPI SEL pin.
$\overline{\text{IRQ2}}$/GPIO/URXD	—	GPIO-O	Remains as is	$\overline{\text{IRQ2}}$/GPIO/URXD	The SPI SPICLK pin.
$\overline{\text{IRQ3}}$/GPIO/UTXD	—	GPIO-O	Remains as is	$\overline{\text{IRQ3}}$/GPIO/UTXD	The SPI MOSI pin.
Error Reporting Pin					
BM1/GPIO/EVNT3	BM1	GPIO-I or GPIO-O	GPIO-I	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted. Then it optionally signals a failed SPI boot. Configures as a GP output only if an error is detected.

Table 2-17 shows how pins are used during an I2C boot:

Table 2-17. Pin Behavior During I^2C Boot

Pin Name	POR Deassert	During Booting	Upon Exit from Boot	Normal Operation	Comments
Boot Mode Pins					
BM3/GPIO	BM3	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM2/GPIO	BM2	GPIO-I	Remains as is	GPIO	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM1/GPIO/EVNT3	BM1	GPIO-I	Remains as is	GPIO/EVNT3	Sampled when $\overline{\text{PORESET}}$ is deasserted.
BM0/GPIO/EVNT2	BM0	GPIO-I	Remains as is	GPIO/EVNT2	Sampled when $\overline{\text{PORESET}}$ is deasserted.
Peripheral Port Pins					
$\overline{\text{IRQ14}}$/GPIO/SCL	—	SCL	Remains as is	$\overline{\text{IRQ14}}$/GPIO/SCL	For I^2C transfers.
$\overline{\text{IRQ15}}$/GPIO/SDA	—	SDA	Remains as is	$\overline{\text{IRQ15}}$/GPIO/SDA	For I^2C transfers.
Note: No pins are used for error reporting.					

2.14 Schmidt Triggering

The input pins in **Table 2-18** contain schmidt triggering circuitry.

Table 2-18. Pins with Schmidt Triggers on Inputs

Module	Pin
Reset	$\overline{\text{HRESET}}$
Interrupt	$\overline{\text{NMI}}$
TDMs	T0RCK T0TCK T1RCK T1TCK T2RCK T2TCK
Ethernet MAC	TXD3 RXCLK TXCLK
HDI	HRW HDS HACK
I^2C	SCL SDA
UART	URXD

2.15 Connectivity Guidelines

Unused output pins can be disconnected, and unused input pins should be connected to their non-active value, except for the following:

- An unused GPIO pin can be disconnected. After boot, it should be programmed as an output pin.
- Examine each pin description to locate open-drain pins requiring external pull-up resistors (for example, the $\overline{\text{HRESET}}$ pin).
- Examine each pin description to locate pins that contain on-chip pull-up resistors (for example, the TMS pin).
- The SWTE pin is used to configure the device and is sampled when the $\overline{\text{PORESET}}$ signal is deasserted. Therefore, it should be tied to V_{SS} or V_{DD} either directly or through a pull-down or pull-up resistor until the $\overline{\text{PORESET}}$ signal is deasserted. It can be floating afterwards.
- The BM pins are used to configure the device and are sampled when the $\overline{\text{PORESET}}$ signal is deasserted. Therefore, they should be tied to GND or V_{CC} either directly or through a pull-down or a pull-up resistor.
- When they are used, the $\overline{\text{IRQx}}$ pins must be pulled up if they are not full drive.
- Pins labelled NO CONNECT must not be connected.
- The DBREQ pin should be tied to its deasserted value when it is not used.

SC1400 Core Overview 3

The SC1400 DSP core features an innovative architecture that addresses the key market needs of next-generation DSP applications, mainly in the field of wireline and wireless infrastructure and subscriber communication. This flexible DSP core targets compute-intensive applications, providing high performance, low power, efficient compilation, and high code density. The SC1400 core efficiently deploys a novel variable-length execution set (VLES) execution model, maximizing parallelism by allowing multiple address generation and data arithmetic logic units to execute multiple operations in a single clock cycle. This section provides an overview of the key features and main modules of the SC1400 core, as well as the programming model and instruction set list.

Note: The information in this chapter is based on Revision 2 of the *SC1000-Family Processor Core Reference Manual*. To get the updates in later revisions of this manual, visit the Freescale Web site shown on the back cover of this manual.

The 16-bit SC1400 core packs four data arithmetic-logic execution units (ALUs), each consisting of a multiply-accumulate unit (MAC), a logic unit, and a bit field unit (BFU), which also serves as a barrel shifter. In addition to the four data execution units, the core contains two address arithmetic units (AAUs), one bit manipulation unit (BMU) and one branch unit. Overall, the SC1400 can issue and execute up to six instructions per clock—for example, four independent arithmetic instructions and two pointer-related instructions (such as moves or other operations on addresses).

At a clock speed of 200 MHz (which is the speed of the MSC7110, MSC7112, MSC7113, MSC7115, and MSC7116 devices), the SC1400 core can therefore execute 800 true DSP MIPS—800 million multiply-accumulate operations per second (MMACS), concurrent with associated data movement functions and pointer updates. One MMACS is the equivalent of several RISC MIPS, the performance measure used by other DSPs. For purposes of comparison, the SC1400 can be said to perform 2000 RISC MIPS. At a clock speed of 300 MHz (which is the speed of the MSC7118 and MSC7119 devices), the SC1400 core performs 3000 RISC MIPS or 1200 MMACS.

The SC1400 core can sustain this high performance over time because of the flexibility of its data execution units and ability to transfer up to 128 data bits per cycle. The four data execution units can operate simultaneously in any combination. For example, the SC1400 core can execute four multiply-accumulate operations in a single clock cycle, or one MAC, two arithmetic/logical operations and one bit field operation. All four data ALUs are identical, permitting great

flexibility in assigning and executing instructions and increasing the likelihood that four execution units can be kept busy on any given cycle. Therefore, programs can take better advantage of the SC1400 parallel architecture.

3.1 MSC711x Architecture

This section discusses the main functional modules of the SC1400 core. **Figure 3-1** shows a block diagram of the SC1400 core as used by the devices in the MSC711x family.

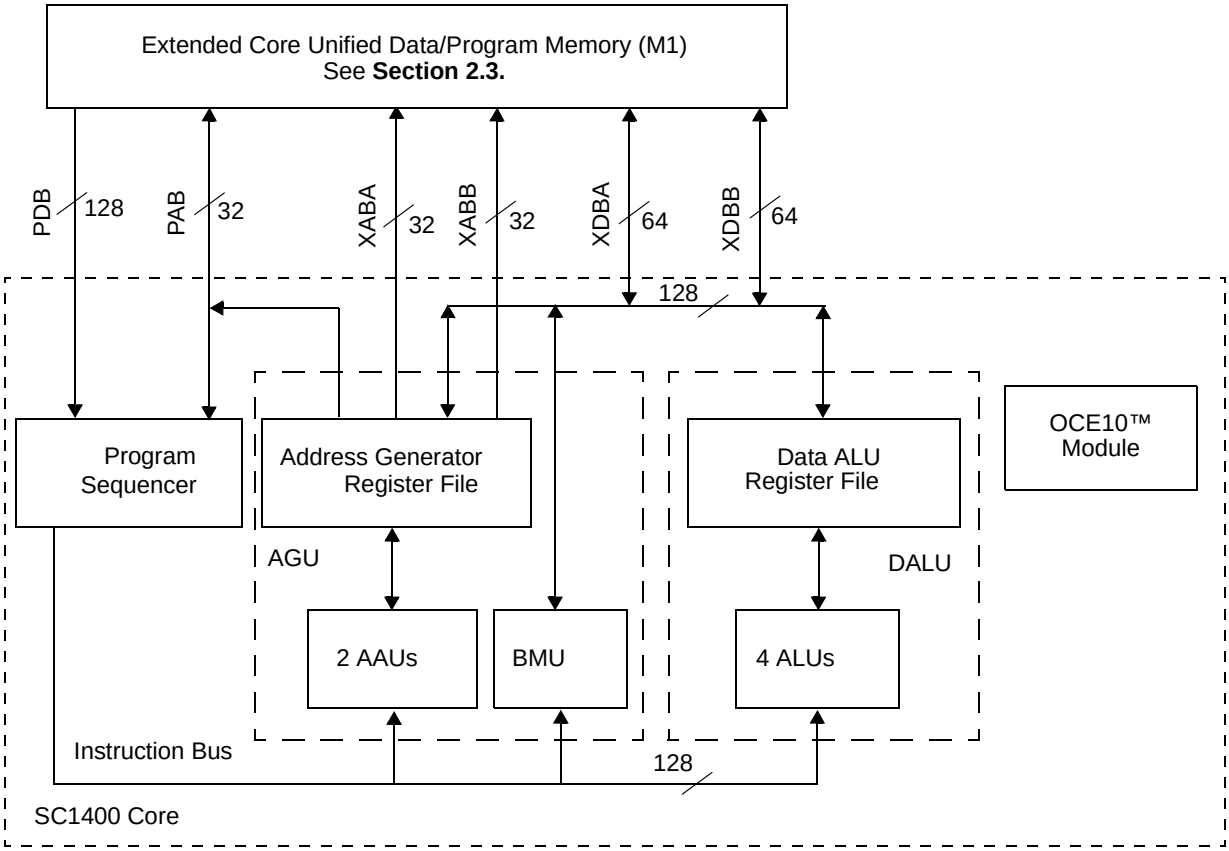


Figure 3-1. Block Diagram of the SC1400 Core

3.1.1 Address Generation Unit (AGU)

One of the execution units in the SC1400 core, the AGU calculates addresses using integer arithmetic to address data operands in memory, and it contains the registers to generate the addresses. It performs four types of arithmetic: linear, modulo, multiple wrap-around modulo, and reverse-carry. The AGU operates in parallel with other resources to minimize address generation overhead. The AGU also generates change-of-flow program addresses and updates the SP. **Section 3.2.1**, *AGU Programming Model*, on page 3-8 describes the AGU registers.

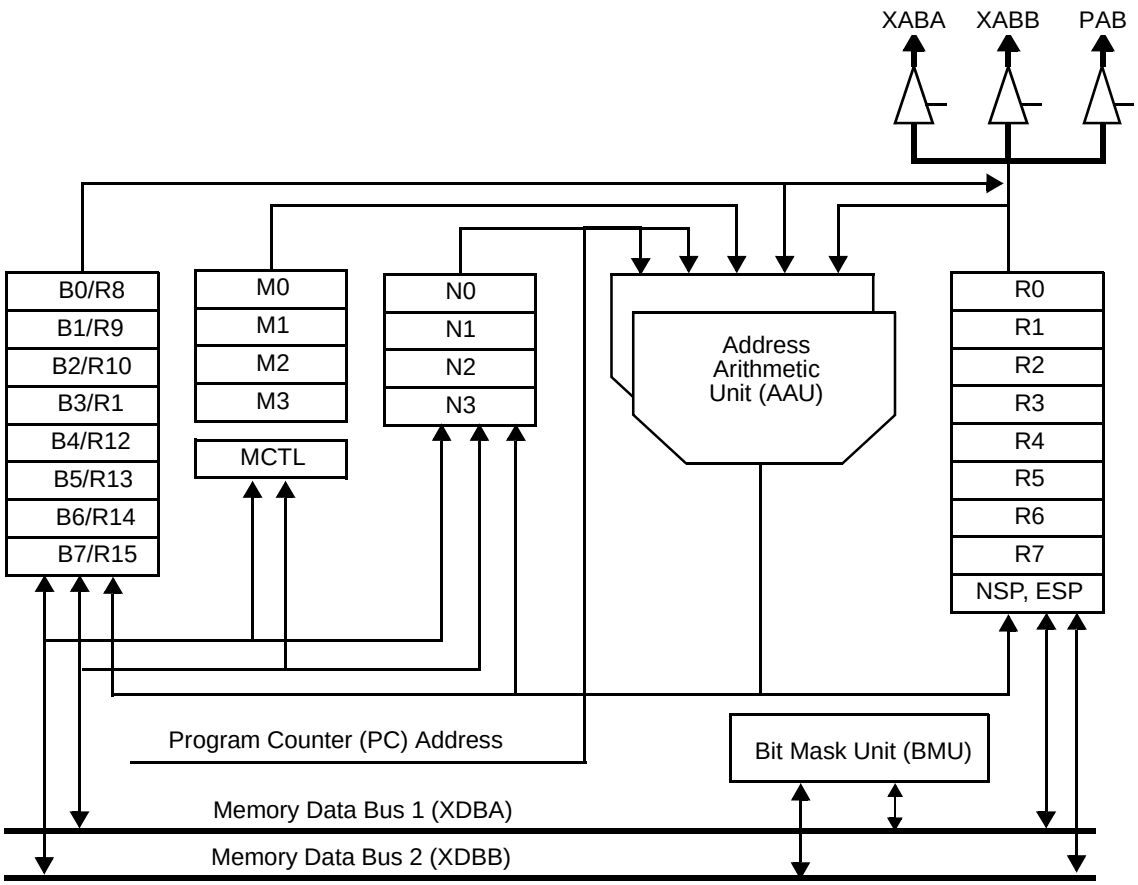


Figure 3-2. AGU Block Diagram

3.1.1.1 AAUs

The two AAUs are identical. Each contains a 32-bit full adder called an offset adder and a 32-bit full adder called a modulo adder. The offset adder performs the following operations:

- Add or subtract two AGU registers
- Add immediate value
- Increment or decrement an AGU register
- Add PC
- Add with reverse-carry

The offset adder also performs compare or test operations and arithmetic and logical shifts. The offset values added are preshifted by 1, 2, or 3, according to the access width. In reverse-carry mode, the carry propagates in the opposite direction. The modulo adder adds the summed result of the first full adder to a modulo value, M or minus M, where M is stored in the selected modifier register. In modulo mode, the modulo comparator tests whether the result is inside the buffer by comparing the results to the B register and chooses the correct result from between the offset adder and the modulo adder.

3.1.1.2 Stack Pointer Registers

To facilitate use of a software stack, two special registers with special addressing modes are assigned to the AGU: the Normal Mode Stack Pointer (NSP) and the Exception Mode Stack Pointer (ESP). Both the ESP and the NSP are 32-bit read/write address registers with predecrement and post-increment updates, as well as offset with immediate values to allow random access to the software stack. Stack instructions use the ESP when the MSC711x device is in the Exception mode of operation, which it enters when exceptions occur. The NSP is used in Normal mode, while there are no exceptions. The two stack pointers makes it easier to support multitasking systems and optimizes stack usage for these systems.

3.1.1.3 Bit Mask Unit (BMU)

The BMU performs bit mask operations, such as setting, clearing, changing, or testing a destination, according to an immediate mask operand. Data is loaded to the BMU over the data memory buses XDBA or XDBB. The result is written back over XDBA or XDBB to the destinations in the next cycle. All bit mask instructions typically execute in two cycles and work on 16-bit data. This data can be a memory location, or a portion (high or low) of a register.

The BMU supports a set of bit mask instructions that operate on:

- All AGU pointers (R[0–15])
- All data ALU registers (D[0–15])
- All control registers (EMR, VBA, PCTL0, PCTL1, SR, MCTL)
- Memory locations

Note: In the MSC711x devices, the PCTL1, PCTL0 registers are unused. To ensure compatibility with future devices, do not write to these registers.

Only a single bit mask instruction is allowed in any single execution set, since only one execution unit exists for these instructions. A subset of the bit mask instructions (BMTSET) allows support for hardware semaphores.

3.1.2 Data Arithmetic Logic Unit (Data ALU)

The data ALU performs arithmetic and logical operations on data operands in MSC711x devices. The data registers can be read or written to memory over the XDBA and the XDBB as 8-bit, 16-bit, or 32-bit operands. The 64-bit wide data buses, XDBA and XDBB, support the transfer of several operands on a single access. The source operands for the Data ALU, which can be 16, 32, or 40 bits, originate either from data registers or from immediate data. The results of all data ALU operations are stored in the data registers. All data ALU operations are performed in one clock cycle. Up to four parallel arithmetic operations can be performed in each cycle. The destination of every arithmetic operation can be used as a source operand for the operation immediately following, without any time penalty. The components of the data ALU are as follows:

- A bank of sixteen 40-bit registers.
- Four parallel ALUs, each containing a MAC unit and a BFU with a 40-bit barrel shifter.
- Eight data bus shifter/limiter circuits to allow saturation of a 64-bit transfer over each XDBA and XDBB bus in a single cycle.

All MAC units and BFUs can access all data ALU registers. Each register is partitioned into three portions: two 16-bit registers (low and high portion of the register) and one 8-bit register (extension portion). The low and high parts of each register serve either as inputs for the arithmetic operations or as part of the 40-bit registers as output for the operation result. The two 64-bit wide data buses that connect between the data ALU register file and the memory enable a very high data bandwidth between memory and registers. Load and store instructions use the maximum width of the bus according to the application requirement because there are different versions of the instructions for different bandwidths:

- `MOVE.B` loads or stores bytes (8-bit)
- `MOVE.W` or `MOVE.F` loads or stores integer or fractional words (16-bit)
- `MOVE.2W`, `MOVE.2F`, or `MOVE.L` loads or stores double-integers, double-fractions, and long words, respectively (32-bit)
- `MOVE.4W` or `MOVE.4F` loads or stores quad-integers and quad-fractions respectively (64-bit)
- `MOVE.2L` loads or stores double-long words (64-bit)

With the ability to execute any two `MOVE` instructions in parallel every clock cycle, a maximum data throughput of 4.8 GBps (at 300 MHz) can be achieved between the memory and the register file. **Figure 3-3** shows the architecture of the data ALU. For information on the data registers, refer to **Section 3.2.2**, *Data Arithmetic Logic Programming Model*, on page 3-9.

3.1.2.1 Data Registers

The Data ALU registers are read or written over the data buses (XDBA and XDBB). The source operands for data ALU arithmetic instructions always originate from the data ALU registers. All data ALU operations are performed in one clock cycle so that a new instruction can be initiated in every clock, yielding a rate of up to four data ALU instructions per clock cycle. The destination of every arithmetic operation can be used as a source operand for the operation immediately following, without any time penalty.

3.1.2.2 Multiply-Accumulate (MAC) Unit

The MAC unit is the main arithmetic processing unit of each SC1400 core and performs all the calculations on data operands. The MAC unit outputs one 40-bit result in the form of [Extension:Most Significant Portion:Least Significant Portion] (EXT:MSP:LSP). The multiplier executes 16-bit × 16-bit fractional or integer multiplication between two’s complement signed, unsigned, or mixed operands. The 32-bit product is right-justified and added to the 40-bit contents of one of the sixteen data registers.

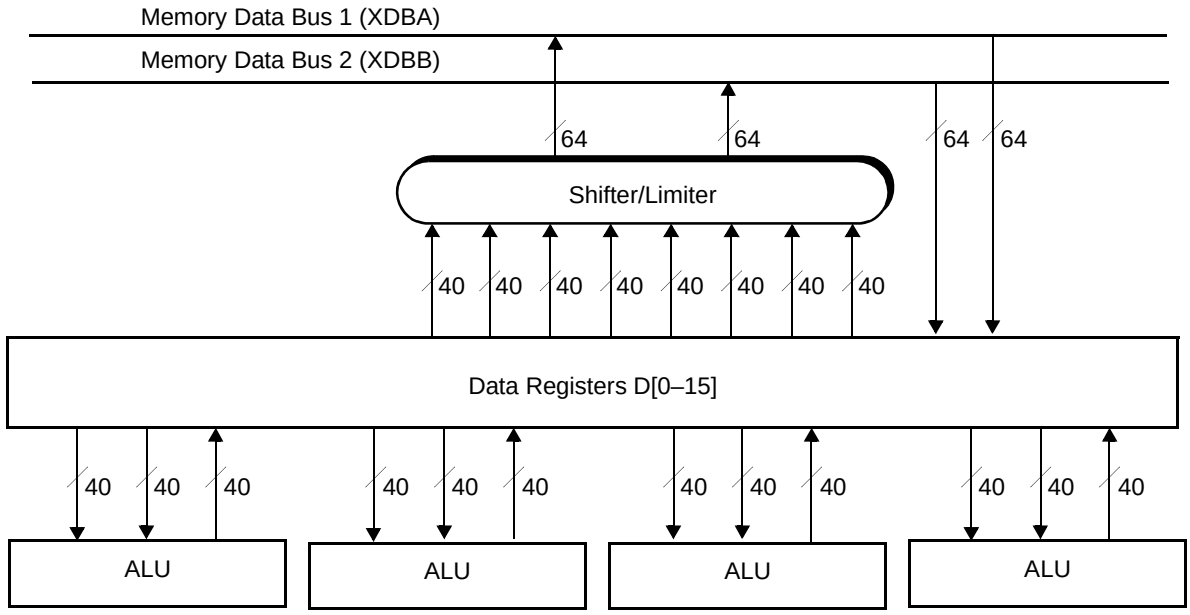


Figure 3-3. Data ALU Architecture

3.1.2.3 Bit-Field Unit (BFU)

The BFU contains a 40-bit parallel bidirectional shifter with a 40-bit input and a 40-bit output, mask generation unit, and logic unit. The BFU is used in the following operations:

- Multi-bit left/right shift (arithmetic or logical)
- One-bit rotate (right or left)
- Bit-field insert and extract
- Count leading bits
- Logical operations
- Sign or zero extension operations

3.1.3 Program Sequencer Unit (PSEQ)

The PSEQ fetches and dispatches instructions and controls hardware loops and exception processing. The PSEQ implements three out of the five stages of the pipeline and controls the different processing states of the SC1400 core. It consists of three hardware blocks:

- *Program address generator (PAG)*. Generates the program counter (PC) for instruction fetch operations and controls the hardware loop functionality.
- *Program dispatch unit (PDU)*. Detects the execution set out of the fetch set and dispatches the instructions of the execution set to their appropriate execution units.
- *Program control unit (PCU)*. Controls overall program pipeline behavior.

Section 3.2.3, *Program Control Unit (PCU) Programming Model*, on page 3-10 discusses the registers that implement the PSEQ functions.

3.1.4 On-Chip Emulator

The emulator allows nonintrusive interaction with an MSC711x device and its peripherals so that you can examine registers, memory, or on-chip peripherals, define various breakpoints, and read the trace-FIFO. These interactions facilitate hardware and software development on an MSC711x processor. The emulator interfaces with the debugging system through on-chip JTAG TAP controller signals. For details, refer to the *OCE10 On-Chip Emulator Reference Manual*.

3.2 Programming Model

The three main units of the SC1400 DSP core programming model are the data AGU, data ALU, and PSEQ (see **Figure 3-4**).

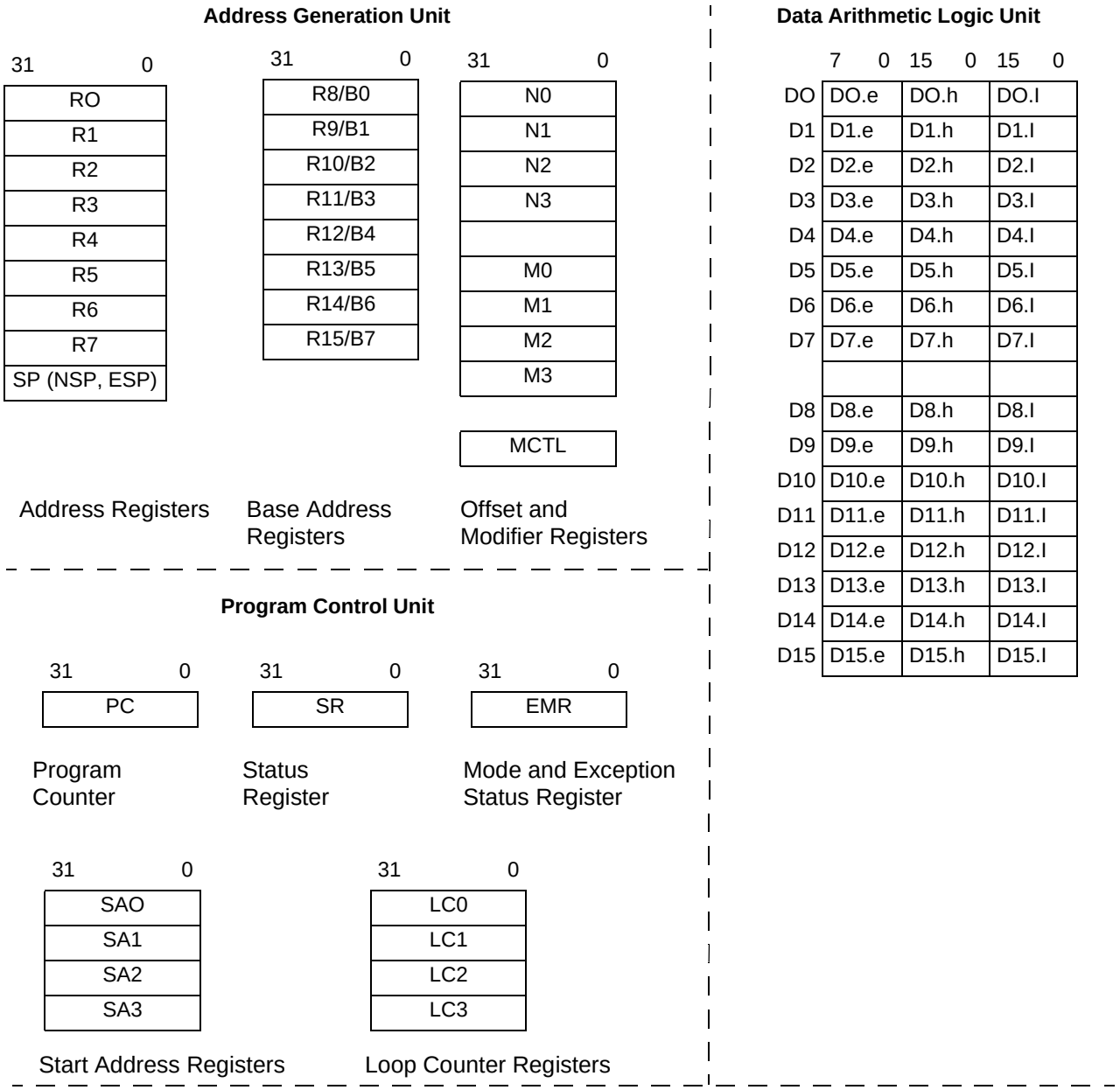


Figure 3-4. SC1400 Programming Model

3.2.1 AGU Programming Model

The address registers can be programmed for linear, modulo (regular or multiple wrap-around), and bit-reverse addressing. Address registers are automatically updated when address register indirect addressing is used.

- *Address Registers (R[0–15])*. The sixteen 32-bit read/write address registers R[0–15] contain addresses or general-purpose data. The 32-bit address in a selected address register is used in calculating the effective address of an operand. The contents of an address register point directly to data or are used as an index. R[0–15] are composed of two separate banks, a lower bank (R[0–7]) and an upper bank (R[8–15]). The lower bank registers can be used for linear, modulo, or bit reverse addressing. The upper bank registers can be use in linear addressing modes only if the lower banks are not using modulo addressing mode. In modulo addressing mode, each lower bank register Rn is assigned a corresponding base address register Bx. Registers B[0–7] and R[8–15] are mapped to the same physical address space, respectively. Therefore, for example, R8 is available only if R0 is not being used in modulo addressing, since this requires the base address register B0. For details, see **Section 3.2.2**, *Data Arithmetic Logic Programming Model*, on page 3-9.

  If an address register is updated, one of the modifier control registers (MCTL) specifies the type of update arithmetic. Offset registers (Ni) are used for post-incrementing and indexing by offset. Either of the two AAUs can modify the address register.
- *Stack Pointer Registers (NSP, ESP)*. There are two stack pointer registers: the Normal Stack Pointer (NSP) and the Exception Stack Pointer (ESP). These 32-bit registers are used implicitly in all PUSH and POP instructions. Only one stack pointer is active at a time, according to the mode:
  - In Normal mode, the NSP is used.
  - In Exception mode, the ESP is used.

  The Status Register EXP bit determines the active mode. The active stack pointer (SP) is used explicitly for memory references in the address register indirect modes. The stack pointers point to the next unoccupied location in the stacks. They are post-incremented on all the implicit PUSH operations and pre-decremented on all the implicit POP operations.

Note: The programmer must explicitly initialize both stack pointer registers after reset.

- *Shadow Stack Pointer Registers*. Both stack pointers have shadow registers that contain a decremented value of the stack pointers. When the shadow register is not valid, the POP instruction executes in two cycles. The first cycle decrements the stack pointer. When the shadow register is valid, the POP instruction executes in only one cycle. When an SP is written (by TFRA), its shadow register automatically becomes invalid. When a PUSH/POP instruction executes, the shadow register of the active SP becomes valid. As a result, during consecutive POPs, even in the worst case, only the first POP requires an additional cycle.

- *Offset Registers (N[0–3])*. The 32-bit read/write offset registers N[0–3] contain offset values to increment or decrement address registers in address register update calculations. These registers are also used for 32-bit general-purpose storage. For example, the contents of an offset register specify the offset into a table or the base of the table for indexed addressing. An offset register can be used to step through a table at a specified rate—such as five locations per step for waveform generation. Each address register can be used with each offset register. For example, R0 can be used with N0, N1, N2, or N3 for offset address calculation.
- *Base Address Registers (B[0–7])*. The 32-bit read/write base address registers B[0–7] are used in modulo calculations. Each B register is associated with an R register (B0 with R0, and so on). When the modulo addressing mode is activated, the B register contains the lower boundary value of the modulo buffer. The upper boundary of the modulo buffer is calculated by B+M-1, where M is the modifier register associated with the register used. When not used for modulo accessing, these registers can function as alternative address registers (R[8–15]). Both Rx and B_{x-8} share the same physical register. For example, if R0 is not programmed for modulo addressing, the base address register B0 can serve as an additional address register R8.
- *Modifier Registers (M[0–3])*. The 32-bit read/write modifier registers M[0–3] contain the value of the modulus modifier. These registers are also used for general-purpose storage. The address arithmetic unit (AAU) supports linear, modulo, multiple wrap-around modulo, and reverse-carry arithmetic types for most address register indirect addressing modes. When the modulo arithmetic is activated, the contents of Mj specify the modulus. Each address register can be used with each modifier register, as programmed in the MCTL register.
- *Modifier Control Register (MCTL)*. The 32-bit read/write register to program the address mode (AM) for each of the eight address registers (R[0–7]). The addressing mode of the upper address register file (R[8–15]) cannot be programmed and functions in linear mode only.

3.2.2 Data Arithmetic Logic Programming Model

The data ALU programming model is shown in **Figure 3-4**. Register D0 refers to the entire 40-bit register, whereas D0.e, D0.h, D0.l refer to the extension, most significant, and least significant portions of the D0 register, respectively. The D[0–15] data registers, referred to as D*x*, give maximum flexibility, since they are used as source operands, destination storage, or accumulators. The registers serve as input buffer registers between the XDBA or XDBB and the ALUs. They are used as data ALU source operands, allowing new operands to be loaded for the next instruction while the current arithmetic instruction processes the register contents.

Each data register Dx has an additional associated flag bit, the limit tag bit Lx, to signify whether an overflow can occur when Dx is read over XDBA and XDBB. The limit tag bit Lx is coupled with the extension portion Dx.e, to form a 9-bit operand. The limit tag bit Lx is updated when a

result is written from the ALU to the Dx register. The data registers are accessed with three types of data width:

- A long-word access, writing or reading 32-bit operands
- A word access, writing or reading 16-bit operands
- A byte access, writing or reading 8-bit operands

The transfer of the Dx register to memory over XDBA and XDBB is protected against overflow by substituting a limiting constant for the data being transferred. The contents of Dx are not affected if limiting occurs. Only the value transferred over XDBA or XDBB is limited. This process is commonly referred to as transfer saturation, and it should not be confused with the arithmetic saturation mode. The overflow protection is performed after the contents of the register are shifted according to the scaling mode. Shifting and limiting are performed only when a fractional operand is specified as the source for a data move over XDBA or XDBB. When an integer operand is specified as the source for a data move, shifting and limiting are not performed. Automatic sign extension or zero extension of the data values into the 40-bit registers is provided when an operand is transferred from memory to a data register. Sign extension can occur when the Dx register is loaded from memory through XDBA and/or XDBB. If a fractional word operand is to be written to a data register, the MSP portion of the register is written with the word operand, the LSP portion is zero-extended, and the EXT portion is sign-extended from MSP. When an integer operand is to be written to a data register, the LSP portion of the register is written with the word operand, and the MSP portion and EXT are either zero-extended or sign-extended from the LSP. Long-word operands are written into the MSP:LSP portions of the register, and the EXT portion is sign-extended. When a byte operand is to be written to a data register, the register's first eight bit portion of the LSP (Dx.1[7–0]) is written with the byte operand, and the remaining bits are either zero-extended or sign-extended from the LSP lower byte.

3.2.3 Program Control Unit (PCU) Programming Model

Part of the PSEQ, the PCU controls the overall pipeline behavior of the program flow. The PCU registers are as follows:

- Program Counter Register (PC)
- Status Register (SR)
- Four Start Address Registers (SA[0–3])
- Four Loop Counter Registers (LC[0–3])
- Exception and Mode Register (EMR). The EMR reflects and controls exception situations in the core. It contains bits that reflect memory configuration, servicing of a non-maskable interrupt, and the following exception conditions: data ALU overflow, illegal execution set, and illegal instruction flow. The EMR[BEM] bit is set to 1 out of reset to configure the device for big-endian operation. The value of the BEM bit is read-only; it cannot be changed after reset. The EMR[GP] bits are loaded with a value of 0x0 out of reset.

3.3 Instruction Set Overview

The SC1400 instruction set is divided into the following functional groups:

- AGU arithmetic
- Data ALU arithmetic
- Move
- Stack support
- Bit mask
- Change-of-flow
- Program control

The following tables list the SC1400 instructions alphabetically within the appropriate functional group.

Table 3-1. DALU Logical Instructions

Instruction	Description
AND	Logical AND
ASLL	Multi-bit arithmetic shift left
ASLW	Word arithmetic shift left (16-bit shift)
ASRR	Multi-bit arithmetic shift right
ASRW	Word arithmetic shift right (16-bit shift)
CLB	Count leading bits
EOR	Logical exclusive OR
EXTRACT	Extract signed bit field
EXTRACTU	Extract unsigned bit field
INSERT	Insert bit field
LSLL	Multi-bit logical shift left
LSR	Logical shift left by one bit
LSRR	Multi-bit logical shift right
LSRW	Word logical shift right (16-bit shift)
NOT	Logical complement
OR	Logical inclusive OR
ROL	Rotate one bit left through the carry bit
ROR	Rotate one bit right through the carry bit
SXT.B	Sign extend byte
SXT.L	Sign extend long
SXT.W	Sign extend word
ZXT.B	Zero extend byte

Table 3-1. DALU Logical Instructions (Continued)

Instruction	Description
ZXT.L	Zero extend long
ZXT.W	Zero extend word

Table 3-2. DALU Arithmetic Instructions

Instruction	Description
ABS	Absolute value
ADC	Add long with carry
ADD	Add
ADD2	Add two 16-bit values
ADDNC.W	Add without changing the carry bit in the status register
ADR	Add and round
ASL	Arithmetic shift left by one bit
ASR	Arithmetic shift right by one bit
CLR	Clear
CMPEQ	Compare data registers for equal
CMPEQ.W	Compare immediate value to data register for equal
CMPGT	Compare data registers for greater than
CMPGT.W	Compare data register to immediate for greater than
CMPHI	Compare for higher (unsigned)
DECEQ	Decrement a data register and set T if zero
DECGE	Decrement a data register and set T if greater than or equal to zero
DIV	Divide iteration
DMACSS	Multiply signed by signed and accumulate with data register right shifted by word size
DMACSU	Multiply signed by unsigned and accumulate with data register right shifted by word size
IADD	Integer addition - no saturation
IMAC	Signed integer multiply-accumulate
IMACLHUU	Integer multiply-accumulate unsigned times unsigned; first source from lower portion second from upper
IMACUS	Integer multiply-accumulate unsigned times signed
IMPY	Signed integer multiply
IMPYHLUU	Integer multiply unsigned times unsigned; first source from upper portion second from lower
IMPYSU	Integer multiply signed times unsigned
IMPYUU	Integer multiply unsigned times unsigned
INC	Increment a data register (as integer data)
INC.F	Increment a data register (as fractional data)
MAC	Signed fractional multiply-accumulate

Table 3-2. DALU Arithmetic Instructions (Continued)

Instruction	Description
MACR	Signed fractional multiply-accumulate and round
MACSU	Signed/unsigned fractional multiply-accumulate
MACUS	Unsigned/signed fractional multiply-accumulate
MACUU	Unsigned/unsigned fractional multiply-accumulate
MAX	Transfer maximum signed value
MAX2	Transfer two 16-bit maximum signed value
MAX2VIT	Specialized MAX2 version for Viterbi kernel
MAXM	Transfer maximum magnitude value
MIN	Transfer minimum signed value
MPY	Signed fractional multiply
MPYR	Signed fractional multiply and round
MPYSU	Signed/unsigned fractional multiply
MPYUS	Unsigned/signed fractional multiply
MPYUU	Unsigned/unsigned fractional multiply
NEG	Negate
RND	Round
SAT.F	Saturate value in data register to fit in top 16 bits
SAT.L	Saturate value in data register to fit in 32 bits
SBC	Subtract long with carry
SBR	Subtract and round
SUB	Subtract
SUB2	Subtract two 16-bit values
SUBL	Shift left and subtract
SUBNC.W	Subtract without changing the carry bit in the status register
TFR	Transfer data register to a data register
TFRF	Conditional data register transfer if the T bit is clear
TFRT	Conditional data register transfer if the T bit is set
TSTEQ	Test for equal to zero
TSTGE	Test for greater than or equal to zero
TSTGT	Test for greater than zero

Table 3-3. AGU Arithmetic Instructions

Instruction	Description
ADDA	AGU add
ADDL1A	AGU add with 1-bit left shift of source operand
ADDL2A	AGU add with 2-bit left shift of source operand

Table 3-3. AGU Arithmetic Instructions (Continued)

Instruction	Description
ASL2A	AGU arithmetic shift left by 2 bits (32-bit)
ASLA	AGU arithmetic shift left (32-bit)
ASRA	AGU arithmetic shift right (32-bit)
CMPEQA	AGU compare for equal
CMPGTA	AGU compare for greater than
CMPHIA	AGU compare for higher (unsigned)
DECA	AGU decrement register
DECEQA	AGU decrement and set T if zero
DECGEA	AGU decrement and set T if equal or greater than zero
INCA	AGU increment register
LSRA	AGU logical shift right (32-bit)
SUBA	AGU subtract
SXTA.B	AGU sign extend byte
SXTA.W	AGU sign extend word
TFRA	AGU register transfer
TSTEQA.L	AGU test for equal on all 32 bits
TSTEQA.W	AGU test for equal on lower 16 bits
TSTGEA.L	AGU test for greater than or equal
TSTGTA	AGU test for greater than
ZXTA.B	AGU zero extend byte
ZXTA.W	AGU zero extend word

Table 3-4. Move Instructions

Instruction	Description
MOVE.2F	Move two fractional words from memory to a register pair
MOVE.2L	Move two longs to/from a register pair
MOVE.2W	Move two integer words to/from a register pair
MOVE.4F	Move four fractional words from memory to a register quadrant
MOVE.4W	Move four integer words to/from a register quadrant
MOVE.B	Move byte (sign-extended for memory reads)
MOVE.F	Move fractional word to and from memory
MOVE.L	Move long (sign extended for memory or register reads)
MOVE.W	Move integer word (sign extended for memory reads)
MOVEF	Move address register to address register, depending on T bit of SR
MOVES.F	Move fractional word to memory with saturation enabled
MOVES.L	Move long to memory with saturation enabled
MOVES.2F	Move two fractional words to memory with saturation enabled

Table 3-4. Move Instructions (Continued)

Instruction	Description
MOVES.4F	Move four fractional words to memory with saturation enabled
MOVET	Move address register to address register, depending on T bit of SR
MOVEU.B	Move unsigned byte from memory
MOVEU.L	Move unsigned long from memory
MOVEU.W	Move unsigned integer word from memory
VSL.2F	Viterbi shift left: specialized move to support Viterbi kernel
VSL.2W	Viterbi shift left: specialized move to support Viterbi kernel
VSL.4F	Viterbi shift left: specialized move to support Viterbi kernel
VSL.4W	Viterbi shift left: specialized move to support Viterbi kernel

Table 3-5. Stack Support Instructions

Instruction	Description
POP	Pop a register from the software stack
POPN	Pop a register from the software stack using the normal stack pointer
PUSH	Push a register into the software stack
PUSHN	Push a register into the software stack using the normal stack pointer
TFRA	OSP Move the “other” stack pointer to/from a register, inversely defined by the exception mode

Table 3-6. Bit Mask Instructions

Instruction	Description
AND	Logical AND on a 16-bit operand
AND.W	Logical AND on a 16-bit immediate value
BMCHG	Bit-mask change for a 16-bit operand
BMCHG.W	Bit-mask change for a 16-bit operand in memory
BMCLR	Bit-mask clear for a 16-bit operand
BMCLR.W	Bit-mask clear for a 16-bit operand in memory
BMSET	Bit-mask set for a 16-bit operand
BMSET.W	Bit-mask set for a 16-bit operand in memory
BMTSET	Bit mask test and set for a 16-bit operand
BMTSET.W	Bit mask test and set for a 16-bit operand in memory
BMTSTC	Bit-mask test if clear for a 16-bit operand
BMTSTC.W	Bit-mask test if clear for a 16-bit operand in memory
BMTSTS	Bit-mask test if set for a 16-bit operand
BMTSTS.W	Bit-mask test if set for a 16-bit operand in memory

Table 3-6. Bit Mask Instructions (Continued)

Instruction	Description
EOR	Logical Exclusive OR on a 16-bit operand
EOR.W	Logical Exclusive OR on a 16-bit operand in memory
NOT	Binary inversion of a 16-bit operand
NOT.W	Binary inversion of a 16-bit operand in memory
OR	Logical OR on a 16-bit operand
OR.W	Logical OR on a 16-bit operand in memory

Table 3-7. Change-of-Flow Instructions

Instruction	Description
BF	Branch if false
BFD	Branch if false (delayed)
BRA	Branch
BRAD	Branch (delayed)
BREAK	Terminate the loop and branch to an address
BSR	Branch to subroutine
BSRD	Branch to subroutine (delayed)
BT	Branch if true
BTD	Branch if true (delayed)
CONT	Jump to the start of the loop to start the next iteration
CONTD	Jump to the start of the loop to start the next iteration (delayed)
DOENn	Do enable - set the “nth” loop counter and enable the loop as a long loop
DOENSHn	Do enable short - set the “nth” loop counter and enable the loop as a short loop
DOSETUPn	Setup the “nth” hardware loop start address
JF	Jump if false
JFD	Jump if false (delayed)
JMP	Jump
JMPD	Jump (delayed)
JSR	Jump to subroutine
JSRD	Jump to subroutine (delayed)
JT	Jump if true
JTD	Jump if true (delayed)
RTE	Return from exception
RTED	Return from exception (delayed)
RTS	Return from subroutine
RTSD	Return from subroutine (delayed)

Table 3-7. Change-of-Flow Instructions (Continued)

Instruction	Description
RTSTK	Force restore PC from the stack, updating SP
RTSTKD	Force restore PC from the stack, updating SP (delayed)
SKIPLS	Test the active LC and skip the loop if LCn is equal or smaller than zero

Table 3-8. Program Control Instructions

Instruction	Description
DEBUG	Enter debug mode
DEBUGEV	Signal debug event
DI	Disable interrupts (sets the DI bit in the status register)
EI	Enable interrupts (clears the DI bit in the status register)
IFA	Execute current execution set or subset unconditionally
IFF	Execute current execution set or subset if the T bit is clear
IFT	Execute current execution set or subset if the T bit is set
ILLEGAL	Trigger an illegal instruction exception
LPMARKA	End-of-loop mark
LPMARKB	End-of-loop mark
MARK	Push the PC into the trace buffer
NOP	No operation, not dispatched to an execution unit
STOP	Stop processing (lowest power stand-by)
TRAP	Execute a software exception
WAIT	Wait for interrupt (low power stand-by)

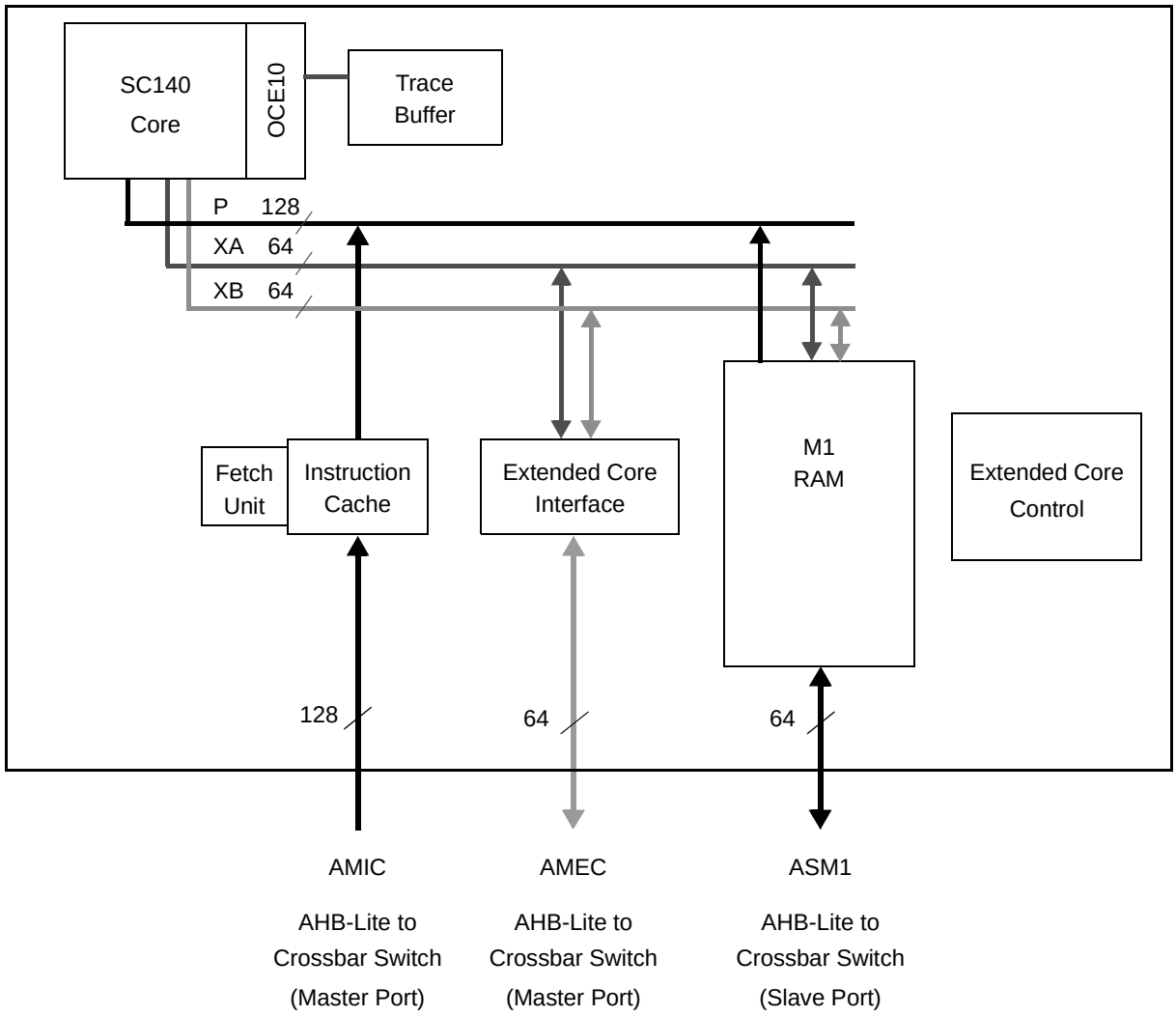
3.4 Programming Considerations

Use the last 64 bytes of M2 memory for data only. Because of system pipelining, code fetches from this area by the SC1400 core can result in an attempt to access the reserved areas beyond the end of the M2 memory. Such fetches may cause the system to stop operation. To prevent this occurrence, do not store instruction code in the 64 bytes of M2 memory. For example, for MSC711x devices with 192 KB of M2 memory, the memory range 0x0102FFC0–0x0102FFFF should be reserved for data only.

Extended Core

4

On the MSC711x devices, each SC1400 core is surrounded by an extended core system that enhances its power and provides a relatively simple interface to the SC1400 core. **Figure 4-1** shows the components of the extended core system, which are the subject of this chapter.



Notes: 1. The arrows show the data transfer direction.
2. The extended core interface includes a bus switch and write buffer.

Figure 4-1. MSC711x Extended Core

4.1 SC1400 DSP Core

The SC1400 core is the CPU powering the MSC711x devices, providing high computational bandwidth. It can execute up to six instructions per clock, issuing to six independent functional units: 4 data-arithmetic-logic execution units (ALUs) and 2 address arithmetic units (AAUs). At a clock speed of 300 MHz, the SC1400 can therefore execute 1200 true DSP MIPS—1200 million multiply-accumulate operations per second (MMACS), together with associated data movement functions and pointer updates.

Note: One MMACS is the equivalent of several RISC MIPS, which is the performance measure used by some other DSPs. For purposes of comparison, the SC1400 core can be said to perform 3000 RISC MIPS—ten RISC operations per cycle at 300 MHz.

The peak bandwidth requirements of the SC1400 core are as follows:

- *Instruction Stream*. Up to 6 instructions (128-bits) can execute every core clock. Each program access from the core is *exactly* 128-bits. The extended core matches the instruction stream needs of the SC1400 core. Its extended core memory (M1) and the instruction cache (for cache hits) provide 128-bits every core cycle via the 128-bit PDB bus.
- *Data Stream*. The SC1400 core generates two 64-bit data accesses every core clock. Each port can generate 8, 16, 32, or 64-bit accesses. The extended core meets this need with the extended core memory (M1) that has two ports for data, each providing up to 64-bits every core cycle via the 64-bit XDBA or 64-bit XDBB buses.

The extended core memory (**Section 4.2**, *Extended Core Memory (M1)*) is multi-ported to allow for parallel accesses from the core. The extended core interface (**Section 4.4**, *Extended Core Interface (ECI) System*) allows the SC1400 core to access memory blocks, peripherals, and external memory outside the extended core. For details on the SC1400 core, see the *SC1000-Family Processor Core Reference Manual.* To get updates or later revisions of this manual, visit the web site listed on the back cover of this manual.

4.2 Extended Core Memory (M1)

M1 memory is a unified 256 KB SRAM memory for program and data within the extended core system. It is a zero wait state memory that operates at core frequency and is four ported to support parallel accesses. The M1 memory is divided into four memory groups of 64 KB each. Each group is accessed through four ports and contains eight modules, as shown in **Figure 4-2**.

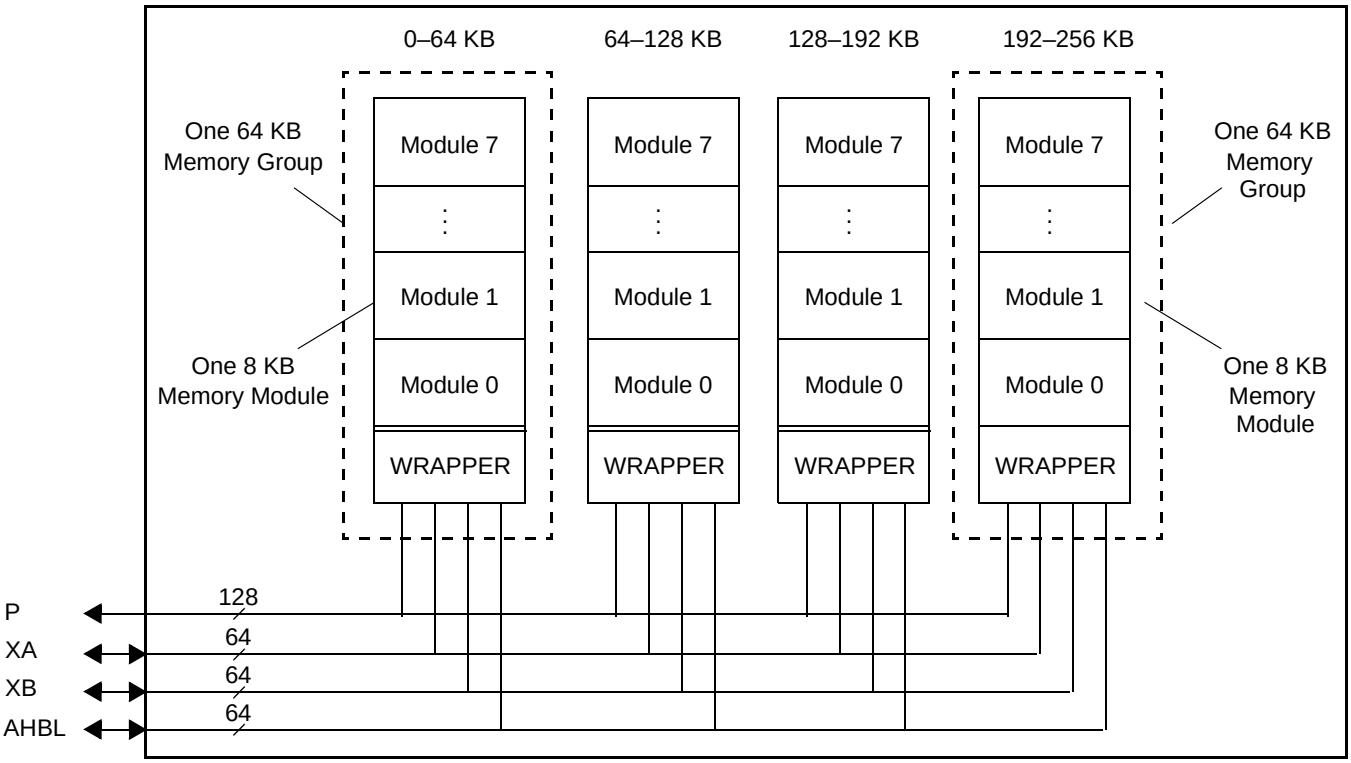


Figure 4-2. M1 Memory Organization (256 KB)

Each memory group contains a wrapper with eight modules. The number of modules determines the amount of eight interleaving (see **Section 4.2.1**, *Interleaving Within a Memory Group*, on page 4-4). Each 64 KB module is organized as an array of 2048 32-bit rows. **Figure 4-3** shows the structure of a memory group. Four ports (XA, XB, P, AHBL) connect to each wrapper. Each memory group has four ports, three that are accessed from the SC1400 core buses (P-bus and the two data buses Xa and Xb) and one that is accessed from the ASM1 (AHB-Lite) bus for DMA and Ethernet MAC transfers (see **Table 4-1**).

Table 4-1. M1 Memory Ports

Memory Port	Connects to	Usage	Accesses	Description
P	PAB/PDB	Program fetches	Read: 128 bits only	The SC140 core uses this port to fetch the next instruction execution set from M1 memory.
XA	XABA/XDBA	First data access from the SC1400 core	Read/Write: 8, 16, 32, or 64-bits	The SC140 core uses this port in data accesses to M1; first of two parallel buses.
XA	XABB/XDBB	Second data access from the SC1400 core	Read/Write: 8, 16, 32, or 64-bits	The SC140 core uses this port in data accesses to M1; second of two parallel buses.
AHBL	ASM1 (AHB-Lite Bus)	Transfers through the crossbar	Read/Write: 8, 16, 32, or 64-bits	The crossbar switch uses this port in DMA and Ethernet MAC transfers to M1 memory.

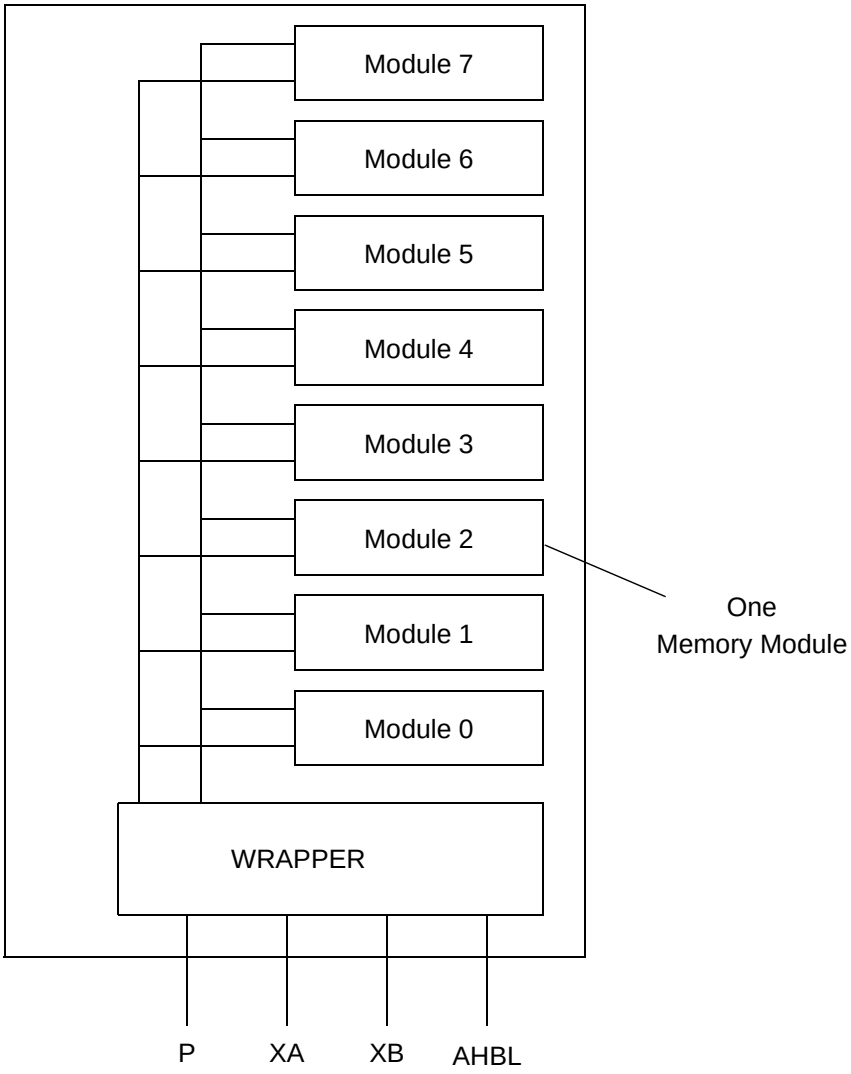


Figure 4-3. One Memory Group

4.2.1 Interleaving Within a Memory Group

Interleaving of the modules within a memory group minimizes SC1400 data access contentions. **Figure 4-4** shows how the M1 memory interprets the different fields of the SC1400 address (program or data):

- *Upper bits*. Select the M1 memory.
- *Group*. Selects one of three memory groups within M1 memory.
- *Row*. Selects one of the rows within a memory module.
- *Module*. Selects one of eight memory modules within a group.
- *Offset*. Selects the desired address within a row.

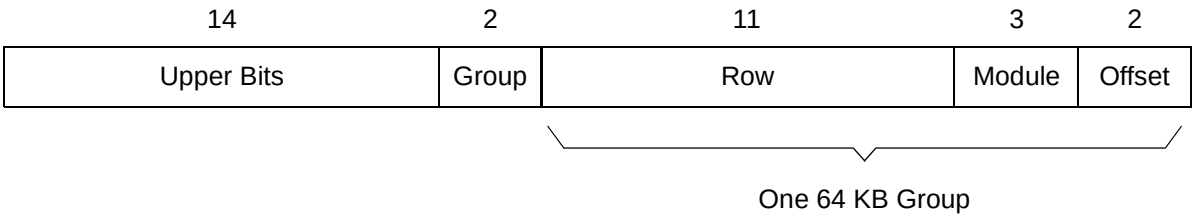


Figure 4-4. Fields of an M1 Memory Address, 64 KB Memory Groups

Interleaving is achieved by placing the module bits below the row bits (a row is a portion of a module). The eight modules of a memory group are interleaved so that the next row of an address is always at the next module. Interleaving increases the probability of performing two simultaneous data accesses to a memory group. When the accesses occur to different memory modules, there is no memory contention and the SC1400 core does not stall. The rules for determining contention are discussed in **Section 4.3.1**, *Memory Contention,* on page 4-6. **Table 4-2** demonstrates 8-way address interleaving for the 64 KB memory groups.

Table 4-2. Interleaving of Addresses, 64 KB Memory Group

	Module 0 Addresses (16 LSBs)	**Module 1 Addresses (16 LSBs)**	**Module 2 Addresses (16 LSBs)**	**...**	**Module 7 Addresses (16 LSBs)**
Row 0	0x0000– 0x0003	0x0004– 0x0007	0x0008– 0x000B	...	0x001C–0x001F
Row 1	0x0020–0x0023	0x0024–0x0027	0x0028–0x002B	...	0x003C–0x003F
...	...	...	...	...	...
Row 2048–1	0xFFE0–0xFFE3	0xFFE4–0xFFE7	0xFFE8–0xFFEB	...	0xFFFC–0xFFFF

4.3 Extended Core Controller

The extended core control unit provides control for the extended core, as follows:

- Arbitrates between the write buffer and bus switch when internal device resources are accessed through the extended core.
- Controls priority of SC1400 accesses through the extended core interface to resources outside the extended core.
- Detects and handles internal memory contentions.
  - — Freezes the address bus.
  - — Freezes the SC1400 core.
- Controls priority of accesses to M1 memory.
- Handles bus access exceptions, including misaligned transfers.
- Handles atomic instruction acknowledge to the SC1400 core for accesses through the ECI.

4.3.1 Memory Contention

There are four ports to M1 memory (**Figure 4-2**) because a MSC711x device produces up to four simultaneous requests to M1 memory: one program, two data, and one AHB access. If a memory group cannot service all the accesses requested, this condition is called a contention.

4.3.1.1 Detecting Contentions

Memory contention is automatically detected and handled by the extended core control unit. Contention can occur when more than one access goes to a half memory group, which consists of modules 0–3 of a memory group or modules 4–7. Contention occurs if any of the following conditions are true:

- All three of the following accesses occur to the same half of a memory group:
  - Program access.
  - DMA access
  - SC1400 data access (XA and/or XB)
- Any two of the following accesses occur to the same half of a memory group:
  - Program access.
  - DMA access
  - SC1400 data access (XA and/or XB)
- There are two SC1400 data read accesses to the same half of a memory group and:
  - Both read accesses are to the same memory module.
  - The read accesses are to different rows within this module.

Each SC1400 write access is captured in a dedicated late write buffer that greatly reduces the probability of contention. In each half-memory group, there are two late write buffers, one on the XA bus and one on the XB bus.

Following are examples of parallel accesses where memory contention occurs:

- A program access and a SC1400 data read access to the same half-memory group.
- A DMA access and a SC1400 data read access to the same half-memory group.
- Two SC1400 data read accesses to different rows within the same memory module.

Following are examples of parallel accesses with no memory contention:

- A program access and an SC1400 data access each to a different half-memory group.
- A program access and an SC1400 data access each to a different memory group.
- A DMA access and a SC1400 data access each to a different half-memory group.
- A DMA access and a SC1400 data access each to a different memory group.
- One program access, one DMA access, and two SC1400 data accesses, each to a different half-memory group.

- Two SC1400 data accesses to different modules in the same half-memory group.
- Two SC1400 data accesses to the same row of the same memory module.

Table 4-3 summarizes the possible simultaneous accesses and shows the number of stall cycles inserted by the SC1400 core to resolve the contention. Except for dual SC1400 data accesses, each memory module serves only one access in a cycle.

Table 4-3. Contention Summary Within One Half-Memory Group

Number of Simultaneous Accesses	Program Access (P)	DMA Access (ASM1)	Core Data Read Access (XA)	Core Data Read Access (XB)	Stall Cycles Inserted by SC1400 Core
0	—	—	—	—	None
1	yes	—	—	—	None
1	—	yes	—	—	None
1	—	—	yes	—	None
1	—	—	—	yes	None
2	—	—	yes	yes	None or 1*
2	yes	yes	—	—	1
2	yes	—	yes	—	1
2	yes	—	—	yes	1
2	—	yes	yes	—	1
2	—	yes	—	yes	1
3	yes	—	yes	yes	1 or 2*
3	—	yes	yes	yes	1 or 2*
3	yes	yes	yes	—	2
3	yes	yes	—	yes	2
4	yes	yes	yes	yes	2 or 3*

Note: The larger number of stall cycles is inserted if the XA and XB accesses are to the same memory module within a group but to different rows within this module. Otherwise, the smaller value is used.

4.3.1.2 Access Priority During Memory Contention

When contention to a memory group occurs, the memory controller prioritizes the order in which the accesses are serviced. The priority of accesses between the buses is programmable with the ASM1 bus at the highest or lowest priority:

Table 4-4. M1 Memory Access Priority

ASM1 Port at Highest Priority (GPSCTL[ASM1P] = 0)	ASM1 Port at Lowest Priority (GPSCTL[ASM1P] = 1)
ASM1 Bus — Highest Priority	Program Fetch — Highest Priority
Program Fetch	XA Read
XA Read	XB Read
XB Read	XA Write
XA Write	XB Write
XB Write — Lowest Priority	ASM1 Bus — Lowest Priority

Placing the DMA controller at lowest priority reduces the contention encountered by the SC1400 core, which is important in MIPs intensive applications. Placing the DMA controller at highest priority permits higher-bandwidth transfers to the M1 memory for applications with high DMA bandwidth to this memory.

When the ASM1 port is programmed with the lowest priority, the DMA controller may be delayed in gaining access to the M1 memory if the SC1400 core is busy processing an algorithm.

4.3.1.3 Allocating M1 Memory to Avoid Contention

The best solution is to reduce memory contention between the ASM1 port and the SC1400 core by placing data for each in different memory groups. Although each M1 memory group has only two ports, intelligent allocation of memory yields a high degree of parallelism in accesses to M1. This section gives practical guidelines for efficient M1 memory usage. Following are strategies for reducing memory contentions:

- Place data and DMA buffers in different memory groups.
- Keep any program code located in M1 memory together in its own memory group.
- Put data to be accessed in parallel into different memory groups. If this is not feasible, place data buffers at an offset from each other so different modules are accessed. A suitable offset, for example, is $N \times 32 + 16$ bytes, where N is an integer less than 1024.

4.3.2 Errors, Exceptions, and Events

The extended core controller detects contentions and errors on the internal core buses and outputs exception signals to the interrupt controller. See **Chapter 11**, *Interrupt Processing*.

4.3.2.1 Errors

Errors generate interrupts using NMI requests to the interrupt controller, as follows:

- *Misaligned program*. SC1400 instructions are 16 bits (two bytes) and must be aligned. If the address on the program bus is not 16-bit aligned, a misaligned program error occurs and an NMI request, MISAL_P, is generated.
- *SC1400 core program address out-of-range*. When the SC1400 core generates a program address within the extended core address space that is not a valid address, a program address out-of-range error occurs and an NMI request, AORP_E, is generated.
- *SC1400 core data address out-of-range*. When the SC1400 core generates a data address on either XA or XB that lies within the extended core's address space that is not a valid address, a data address out-of-range error occurs and an NMI request, AORX_E, is generated (see **Section 5.5**, *Access Restrictions*, on page 5-37.

No exception is detected when an atomic access by the SC1400 core to a location in M1 memory is broken by a higher-priority access from the DMA controller. An application must architect its software so that DMA transfers can never occur to the same location in M1 memory where an atomic instruction may be modifying a location. In actual practice, this is typically the case. There is no issue when an atomic access is separated by a DMA access to a different location in M1 memory.

4.3.2.2 Exceptions

Exceptions assert the interrupt request lines of the interrupt controller and can be masked. The contention exceptions are mainly used for debug and profiling and can be masked otherwise. The exceptions generate the following interrupts:

- *Dual data write*. When accesses on the XA and XB data buses simultaneously write to the same location or portion of a location, an exception occurs and an interrupt request, EC_DUALWR, is generated.
- *Misaligned data*. When an address on either of the data buses (Xa or Xb) is misaligned with its data size, a misaligned data exception occurs and an interrupt request, MISAL_D, is generated.

4.3.2.3 Events

The extended core interface generates the following signals for system-level debugging:

- *M1 contention*. When there is a contention between any of the buses accessing M1 memory, this condition is detected and sent as an input to the event port. The signal asserts one AHB clock for each detection of two contention cycles within the SC1400 core. For example, for eight M1 memory contentions in a given interval of time (not necessarily sequential), this signal asserts for four AHB clocks. Similarly, for seven M1 memory

contentions (the SC1400 core contention signal asserts for seven core clock cycles), this signal asserts for 3 or 4 AHB clock cycles, depending on the previous state. This allows the timers connected to the event port more accurately to count the number of cycles with contention asserted (number of occurrences divided by 2).

- *Instruction cache misses*. When there is a miss on the instruction cache, this information is sent to the event port. Two different types of misses are detected: any instruction cache miss and misses only to external memory. The signal asserts one AHB clock for each detection of two cycles where the SC1400 core is stalled by the miss. For example, if the SC1400 core stalls for eight core clock cycles because of instruction cache misses in a given interval of time (not necessarily sequential), this signal asserts for four AHB clocks. For a stall of seven core clock cycles, this signal asserts for a total of 3 or 4 AHB clock cycles, depending on the previous state. This allows the timers connected to the event port more to count the number of stall cycles more accurately due to cache misses (number of core clocks divided by 2).
- *Interrupt service (INTSV)*. When the SC1400 core is about to service an interrupt, a pulse is generated and sent to notify the event port. This signal asserts when the PAB is driven with the address of the interrupt vector and remains asserted a total of 5 core clock cycles.

4.4 Extended Core Interface (ECI) System

The ECI enables the SC1400 core to communicate efficiently with resources outside the extended core. The module handles the SC1400 core access requests to outside resources. **Figure 4-5** shows the ECI structure. The ECI handles the switching between the core buses and the AHB-Lite for Extended Core Master (AMEC) bus, which goes to the crossbar switch. The ECI transfers these accesses to the AMEC. The ECI operates at the same frequency as the SC1400 core.

Note: Program accesses to non-cacheable regions of memory are handled through the instruction fetch unit.

Different attributes of the extended core can be tuned for an application. These attributes are programmable through the ECI and instruction cache registers, which are described in **Section 4.8**, *Extended Core Programming Model*, on page 4-40.

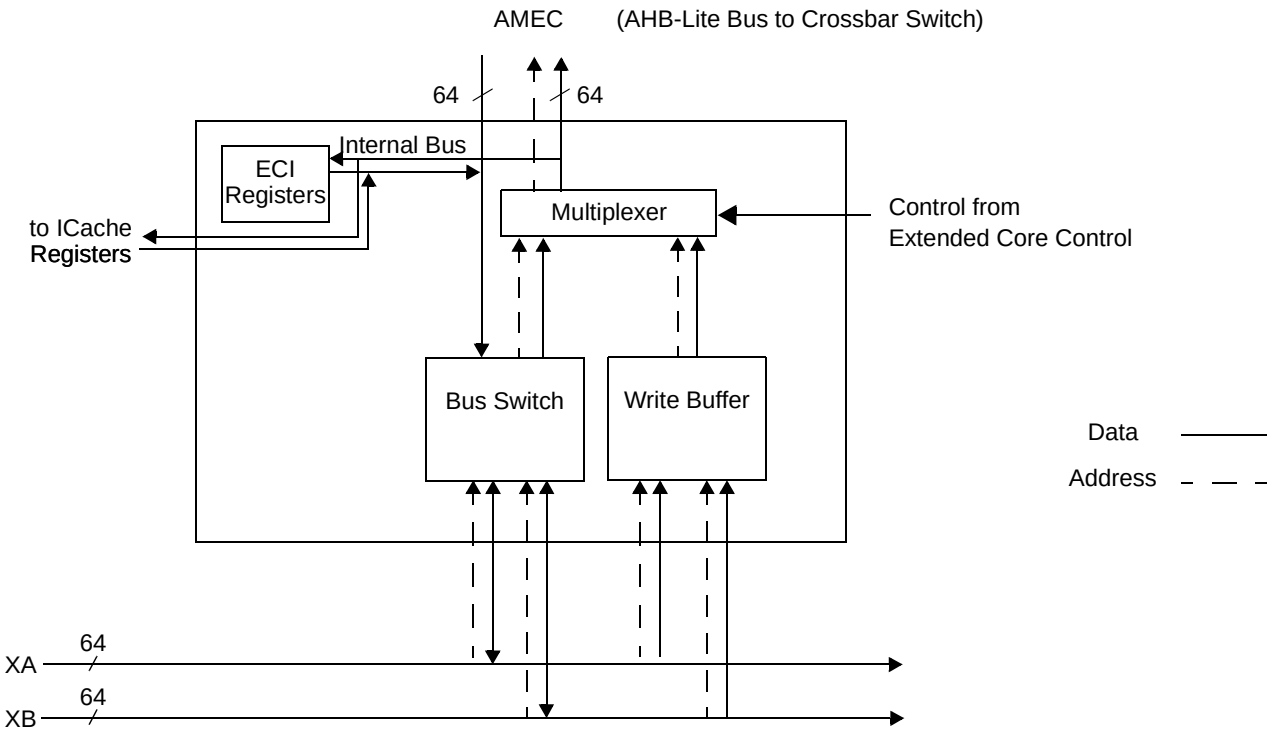


Figure 4-5. Extended Core Interface Block Diagram

4.4.1 AMEC Bus

The AMEC bus connects the extended core to the MSC711x system through a master port on the crossbar switch. All SC1400 core data accesses outside the extended core are transferred on the AMEC bus. It has a relatively simple protocol. The features of the AMEC bus are as follows:

- Operates at half the SC1400 core frequency.
- 32-bit address.
- 128-bit program reads to the SC1400 core.
- 64-bit data reads to the SC1400 core.
- 64-bit data writes from the SC1400 core.
- Pipeline between the address and data phases.

The ECI prioritizes simultaneous access requests for the extended core interface in the following order (descending order):

- XA-bus read — Highest Priority
- XB-bus read
- XA-bus write immediate or immediate with no freeze
- XB-bus write immediate or immediate with no freeze
- XA-bus write
- XB-bus write — Lowest Priority

The ECI services the addresses according to their priority. However, a write buffer flush gets the highest priority within accesses of the same core cycle.

Note: When the SC1400 core writes two values at the same time to the ECI and the write buffer is enabled, an XA-bus write access (not immediate) may occur before an XB-bus write immediate or immediate with no freeze access. This is an exception to the usual priorities.

4.4.2 Bus Switch and Write Buffer

The bus switch handles the following accesses to resources outside the extended core:

- All data read operations.
- Write operations when the write buffer is disabled.
- Atomic (read-modify-write) operations.

4.4.2.1 Write Buffer

The write buffer has a four-entry buffer that enables the SC1400 core to write out to the external memory with no freeze. A write access to resources outside the extended core goes to the buffer while the SC1400 core continues execution. The write buffer operates in one of the following modes:

- *Normal Operation*. Writes from the SC1400 core are stored in the write buffer FIFO and written out to the crossbar switch.
- *Immediate accesses*. If a write access is for an immediate memory area, as programmed in the IMM bits in the WBDAR registers, the access bypasses all other in-buffer commands. The write buffer halts the SC1400 core in an immediate access.
- *Immediate access with no freeze*. This access is handled the same way as an immediate but with no freeze to the SC1400 core.
- *Disabled*. Write buffer is not in use.

These modes apply only for writes to resources outside the extended core. Different regions of memory are configured for one of these modes using the Data Area Registers (**page 4-42**). The regions can be configured for either normal, immediate access, or immediate access with no freeze mode operation. For details on programming the Data Area Registers, see **Table 4-8**, *Programming the Write Buffer Data Area Base and Size*, on page 4-36.

Note: The address range that contains the register files for the ECI and ICache is always defined as a write immediate. This definition ensures in-order execution. For the exact range of addresses, see **Chapter 5**, *Memory Map*.

The write buffer does not handle accesses when there are atomic operations or if the WBOFF flag is asserted (see **page 4-41**). The write buffer is disabled using the Write Buffer Control Register WBOFF bit (**page 4-41**).

Writes using the write buffer (to a region of memory where WBDARn[IMM] == 00) do not automatically ensure that read and write accesses are executed in the correct order of appearance. Using any of the following ensures that the order of accesses is preserved:

- Write accesses when the write buffer is disabled.
- Execution of an atomic Read-Modify-Write instruction. For details, see **Section 4.4.3**, *Atomic Accesses (Read-Modify-Write)*, on page 4-14.
- Write accesses to regions of memory programmed for immediate access or immediate access with no freeze

Thus, if it is necessary that read and write accesses be ordered correctly to a particular range or set of ranges in memory, the WBDAR for these ranges should be programmed with one of the Write-Immediate modes; that is, Write Immediate (freezes core) or Write Immediate with No Freeze.

During normal operation and immediate access with no freeze, the write buffer transfers its contents to the destination without further SC1400 core intervention. Exact timing of the transfer depends on the traffic on the AHB-Lite buses. In the following cases, the write buffer stalls the SC1400 core to protect data from running over:

- *Write buffer is full*. The write buffer is already full, and another write access is issued.
- *Immediate access*. A write to an area of memory programmed for immediate accesses executes before all other writes in the write buffer queue and in order with the read access (the read access in the next cycle executes after the write immediate).
- *Flush of write buffer content*. The write buffer elevates the ECI priority. A flush writes all contents of the write buffer to the AMEC bus through the ECI. A flush of write buffer content is initiated in four cases:
  - *Read from an address within the write buffer*. To keep the logical constancy of commands, the write operation should execute before the read from the same address. When a read from an address held in the write buffer is detected, the write buffer flushes all its contents and executes the read.
  - *Flush command in software*. To activate a software flush, one should issue a read from a pre-defined address. This address is set in hardware and is not programmable (see WB Software Flush Register on **page 4-41**).
  - *Watchdog flush*. If the write buffer attempts to transfer an access but does not get acknowledgment, a flush is initiated to give it the highest priority on the AMEC bus. The watchdog expiration time is programmed in the WB Control Register (see **page 4-41**).
  - The write buffer is turned off with the WBOFF bit while the write buffer is not empty.

4.4.3 Atomic Accesses (Read-Modify-Write)

The SC1400 BMTSET.W instruction is useful for test-and-set operations on critical sections of code. This instruction issues a read-modify-write operation to ensure that the bit is set and tested in one operation. This atomic instruction ensures that the sequence is not broken.

The BMTSET.W instruction tests the destination, sets the true (T) bit if each bit with a value of 1 in the mask also has a value of 1 in the destination, and it sets every bit in the destination (register or memory) that has a value of 1 in the mask. This operation involves both a read access while an atomic signal is sent on the bus and a write access. Atomic operation applies only to the SC1400 core and ECI. The other masters in the device do not require and do not support atomic accesses.

The BMTSET.W instruction determines whether a resource is idle and available for access or is currently in use by another master or process. One bit in a predetermined memory location is allocated to specify whether a resource is free or busy.

- When the value of this bit is 0, the desired resource is free for access by any master or process.
- When the value of this bit is 1, the resource is busy and not available to another master or process.

When the BMTSET.W instruction finishes, the bit in the predetermined memory location is set, regardless of whether the program has obtained access to the resource. The program now examines the SC1400 SR[T] flag:

- If T = 0, the resource is free and the SC1400 core now owns the resource and can access it.
- If T = 1, the resource is occupied by another process or master and is not available for access by the current program.

When a program has gained access to the resource, it uses the resource as required and then clears this bit in memory.

4.4.3.1 Coherency at the System Level, Against Interrupts

The SC1400 BMTSET.W atomic instruction ensures that no interrupt can occur between read and write accesses. All atomic operations to any address in the system are adequately protected against interrupts.

4.4.3.2 Coherency at the System Level, Accesses Issued from the ECI

Any atomic access of the SC1400 core to locations outside the extended core first passes through the crossbar switch and continues to a destination on one of the crossbar slave ports. Atomic accesses are protected through the crossbar. After the read access is issued through the crossbar to a particular slave port, no other master can access the same slave port until the atomic write access completes. Thus, all atomic operations on locations outside the extended core are

adequately protected, including all accesses to M2 memory, DDR memory, APB peripherals, IPBus peripherals, or peripherals accessed on the ASTH bus.

Note: If an atomic access is performed on an address within the write buffer, the read access first flushes the write buffer and then performs the read access. This ensures coherency and protects all atomic accesses to locations within the write buffer. The write cycle of the atomic instruction is always performed as a write-immediate.

4.4.3.3 Coherency at the System Level, Accesses to M1 Memory

Accesses to locations in the M1 memory can occur from either the SC1400 core or any master performing an access through the crossbar to the ASM1 bus. Examples of the later include DMA transfers from the MSC711x DMA controller or accesses by the Ethernet MAC. There are two techniques for protecting atomic accesses to M1 memory:

- Program all SC1400 data accesses to M1 memory to a higher priority than accesses from ASM1. That is, program the GPSCTL[ASM1P] bit so that ASM1 accesses have lowest priority.
- Design your system to guarantee that no write occurs from any internal DMA controller to a location in M1 memory that can be simultaneously accessed by the SC1400 core. The application software must be structured so that DMA transfers can never occur to the same location (or region) in memory where an atomic instruction may be modifying a location, as if often the case.

Note: No exception is flagged when the read and write accesses of an atomic access by the SC1400 core to an M1 memory location are broken by a higher-priority access from the ASM1 bus.

4.5 Instruction Cache (ICache)

Accessing instruction code from memory areas with high access latencies (for example, external memories, internal memories connected to slow buses, and so on) imposes timing penalties on the SC1400 core, degrading system performance. The ICache improves system execution time by dynamically mapping relevant memory areas to a fast 16 KB memory. ICaches allow system designers to place a large amount of code into slower memories, yet achieve high performance as if the code were stored in a fast internal zero-wait-state memory. The MSC711x ICache has the following features:

- 16 KB of memory
- 16-way associativity (that is, 16-way set associative)
- Programmable burst of 1, 2, or 4 fetch-sets upon miss
- Prefetch support
- Replacement based on least-recently used (LRU) algorithm

- Locking data in the cache through flexible LRU boundaries, multi-task support
- Real-time debugging support with misses and hits counted through the OCE10 on-chip emulator
- Non-real-time debugging; enable to read full cache state, clear line command for breakpoint insertion
- Programmable cacheable regions

The key attributes of the ICache are as follows:

- *Line*. The smallest division of cache memory for which there is a distinct tag. The line size is an integer number of the processor's fetch sets. A fetch set (entry) is the number of bytes requested by the core in a single request. A line must contain consecutive program code.
- *Set (Index)*. The address space of the SC1400 core is logically divided into a large number of lines, with the size of each line determined by the size of a line in the cache. Each line within the cacheable region of the SC1400 address space maps to a unique set within the cache using a replacement algorithm discussed in **Section 4.5.2**, *MSC711x Set Associative Mapping*, on page 4-19. Each line is marked by a unique number called an index, determined by a field within the address issued by the SC1400 core (**Figure 4-6**).
- *Way*. When the program access space of an SC1400 program is mapped to a particular set, its line can be mapped to any of 16 different places (ways) in the ICache memory array. The number of ways indicates the degree of associativity of a cache.
- Memory address partitioning (see **Figure 4-6**):
  - The tag field partitions the external memory into 64 KB segments.
  - The SET field partitions each tag-defined area into 16 KB segments.
  - The ENTRY bits partition each line into 16 fetch sets.
- *TAG*. Holds the upper 22 address bits for the corresponding line in the cache and is compared with the current access address.
- *Valid Bit*. Each 128-bit entry in the cache line has an associated bit indicating whether the entry contains valid cache data for the current tag. This bit is called a valid bit.
- *Replacement Algorithm*. An algorithm that determines which line to replace when a miss occurs. The ICache uses the Least Recently Used (LRU) algorithm; that is, for the selected set, the way that is marked as least-recently-used (LRU = 0) is replaced on a miss. Cache locking modifies the operation of the replacement algorithm.
- *Fetch block*. Number of bytes fetched on every cache miss. A fetch block can be smaller than a line and is associated with the burst size.

4.5.1 Set Associative Address Mapping

The ICache memory optimizes access to its instruction storage area using a specialized indexing system called set associative mapping to separate the fetch address from the SC1400 core into different fields:

- *TAG field*. Holds the upper 22 address bits for the corresponding line in the cache and is compared to the current access address.
- *SET field*. 2 bits
- *ENTRY field*. 4 bits
- *LSB field*. Lowest 4 bits

When the SC1400 core requests instruction code, the first 22 bits of the requested address (A[31–10]) identify a region in external cacheable memory and form the TAG value for that region. The next two bits of the address (A[9–8]) identify a set (index) within the tag-defined memory region and form the SET field. Each set is further divided into 16 entries (fetch sets) using the next four bits of the address (A[7–4]). Each entry (fetch set) is 16 bytes long and contains its own valid bit that is initialized as 0 and changed to 1 when that block is written into cache memory. Bits A[3-0] always have a value of 0 because the SC1400 core always exactly 128-bits on program fetches. **Figure 4-6** shows the different fields.

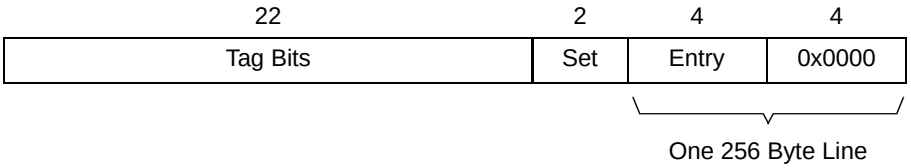


Figure 4-6. SC1400 Program Address Fields for ICache

Within each of the four sets of the cache, there are 16 different ways (locations) where a cache line can be stored. In contrast, a direct-mapped cache has only one location to place a cache line, a 2-way set associative cache has only two different locations, and so on. The MSC711x 16-way cache greatly reduces the chances of a conflict. **Figure 4-7** shows the ICache structure.

After the SET field maps each program address from the SC1400 core to one of four sets, any of the 16 ways of the set can be used. This results in a total of 64 cache lines (4 sets × 16 ways = 64), where a line represents a fixed amount of consecutively located code. Each cache line has its own tag. **Figure 4-8** shows the structure of one cache line, which is equivalent to one of the ways in any of the sets shown in **Figure 4-7**. Each entry contains 128-bits, the size of one SC1400 fetch set. Each cache line is assigned a least-recently-used (LRU) value that specifies its LRU status level.

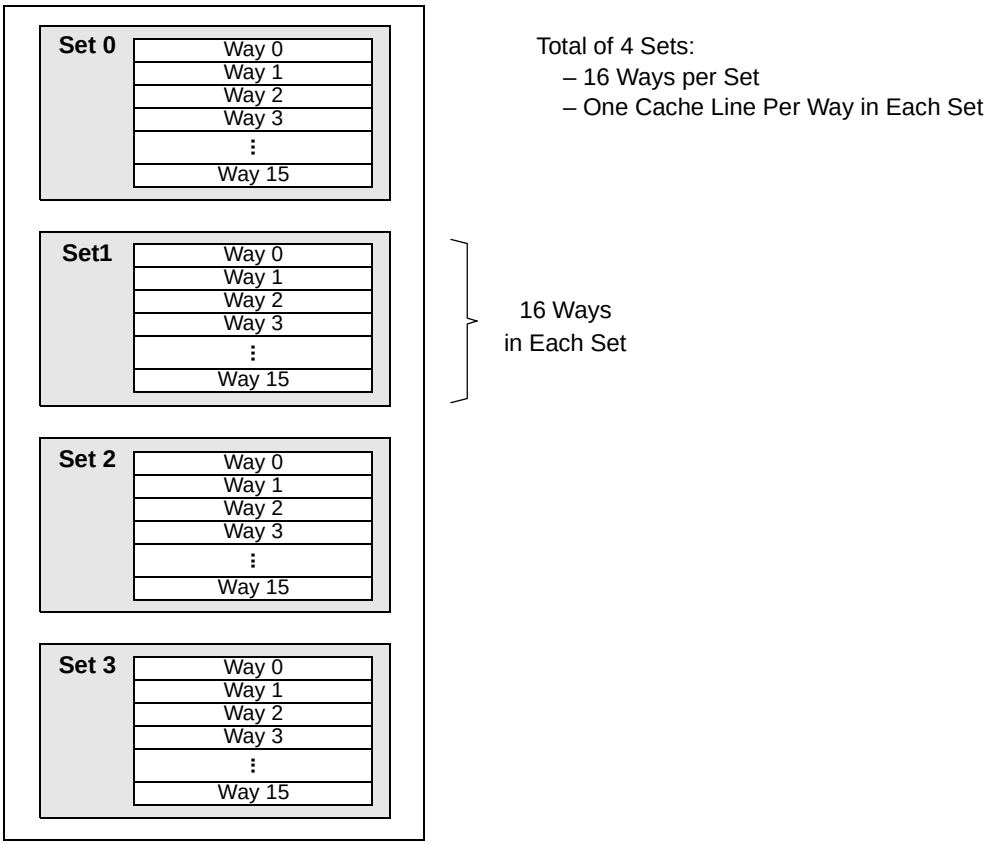


Figure 4-7. Structure of the 16 KB ICache

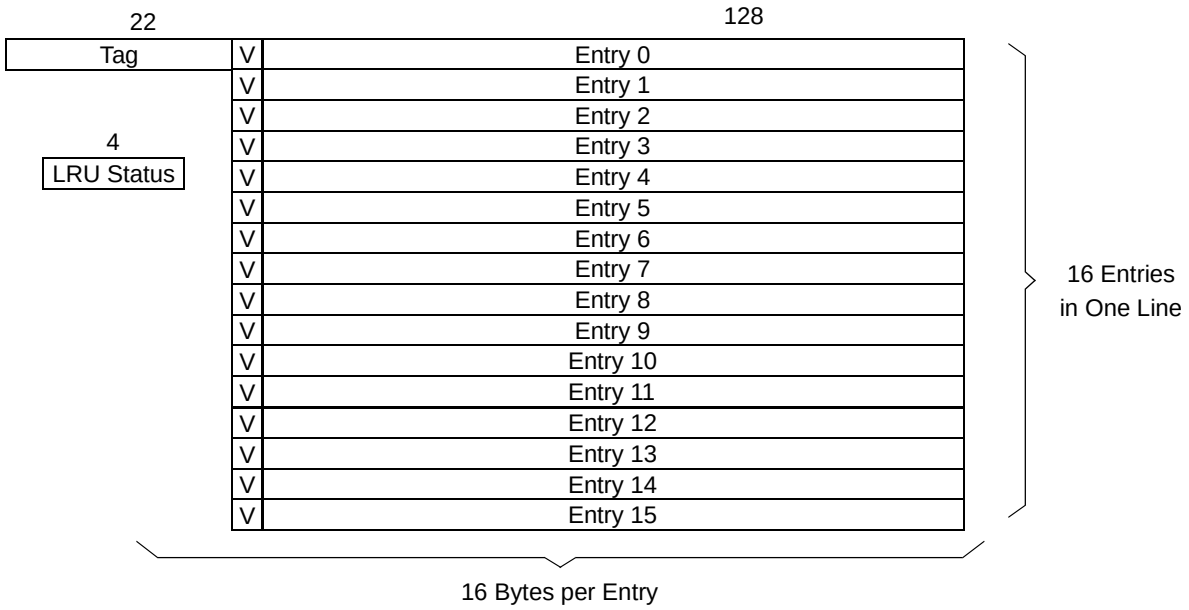


Figure 4-8. Structure of One of the 64 Cache Lines in the 16 KB ICache

4.5.2 MSC711x Set Associative Mapping

The algorithm used by MSC711x for mapping addresses in the cacheable memory region to lines in the cache is set associative mapping, which proceeds as follows:

1. Each SC1400 program fetch address is mapped into one of four sets within the instruction cache using the SET field (**Figure 4-6**).
2. Within this set, the address is placed into any of the 16 ways in the selected set. The way selected is determined by the LRU algorithm:
   - If cache locking is not in use, the least recently used way is selected.
   - If part of the cache is locked, the LRU algorithm is used for the remaining ways of the cache that are not locked.
   - If the entire cache is locked, the desired address cannot map to a location in the cache.
3. The LRU status of all lines in the set is updated.

Cache locking and the LRU algorithm are discussed in **Section 4.5.4**. Addresses to non-cacheable regions of the memory cannot be mapped into the ICache. These locations are accessed directly, bypassing the cache. See **Section 4.7.2** for information on configuring the cacheable regions of memory.

4.5.3 Cache Hits and Misses

A request for code already in the cache is termed a *cache hit*. When the required code is not present in the ICache memory array (termed a *cache miss*), the code is fetched from the slower memories to the SC1400 core and simultaneously loaded into the ICache memory array. The performance degradation resulting from the slower access is termed a *miss penalty*.

Note: The instruction cache is active only on accesses to cacheable regions of memory (**Chapter 5**, *Memory Map*), which are defined by the instruction region registers (**Section 4.7.2**, *Instruction Cacheable Area,* on page 4-37). This excludes all program accesses to the M1 memory.

4.5.3.1 Servicing a Miss

The ICache is located between the SC1400 program address bus (PAB) and the AMIC bus connected to the crossbar switch. The crossbar switch connects to the M2 or external memory. **Figure 4-9** shows the ICache in the extended SC1400 core system. When a cache miss occurs, the ICache initiates a cache fill via the instruction fetch unit (IFU). These accesses fetch a stream of instructions to take advantage of the benefits of burst transfers. Cache bursts occur from the M2 or external memory to the cache array. The SC1400 core stalls for the number of clock cycles required to fetch the required instruction into the ICache array.

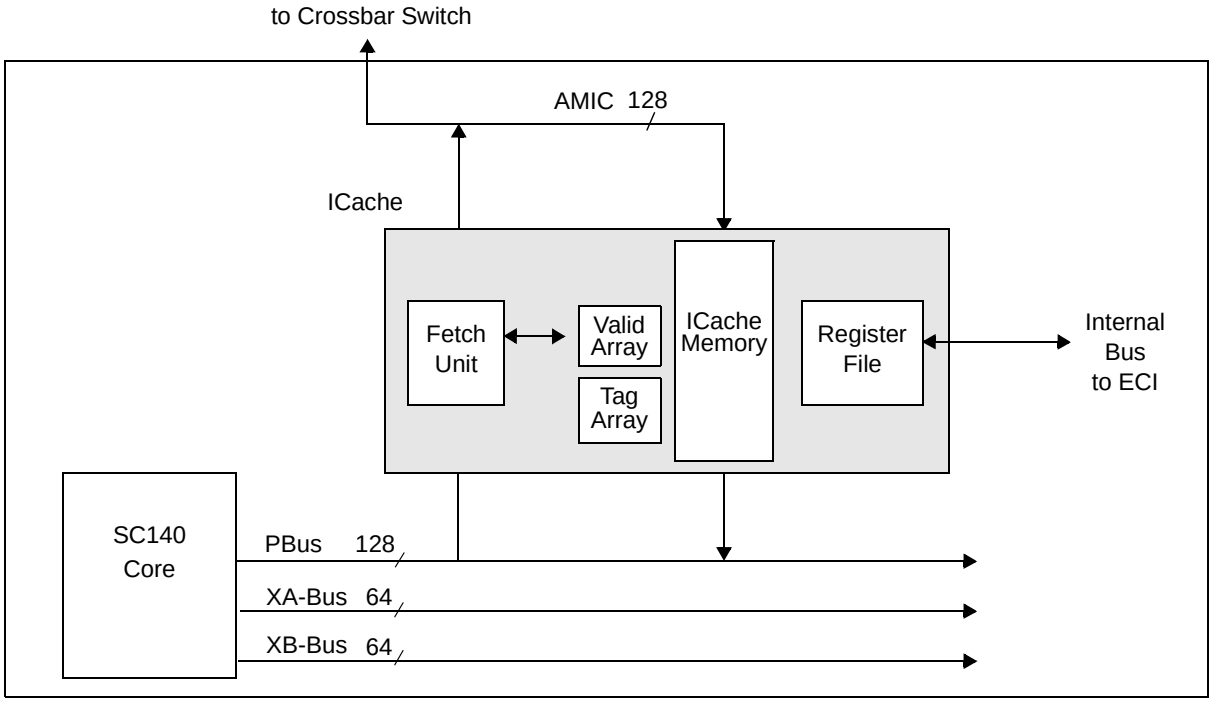


Figure 4-9. MSC711x ICache System

Cache updates are initiated by the instruction fetch unit (IFU). The amount of data transferred by the IFU is configured in the IRCR, which defines the number of fetch sets in each burst as well as the total number of fetch sets to transfer in a block. In addition to the basic fetch operation (called “block transfer”), there is an optional prefetch to the end of the cache line or until a new cache miss begins the next fetch sequence (called a “prefetch transfer”). For details on servicing cache misses, including bursting parameters, see **Section 4.6**, *Instruction Fetch Unit*, on page 4-31.

Note: Unlike a data cache, the instruction cache depends on the fact that code does not change during run time. If the code changes, the cache contents should be cleared (cache flush, no coherency support).

4.5.3.2 Loading the Cache Array on a Miss

The cache array is loaded differently depending on the state of the cache and type of cache miss:

- *Tag-match, valid-bit-miss.* In the selected set, if the upper 22 bits of the address matches a tag in any of the 16 ways but the VALID bit for that cache line entry is clear, the IFU fetches code and loads it into the cache line entry and also sends it to the SC1400 core for execution. The VALID bit for this entry is set accordingly.
- *Tag-miss with available cache entry.* If there is no tag match, but not all ways for the selected set are filled, the IFU fetches code and loads it into the cache line entry in the next available way within the set and also sends it to the SC1400 core for execution.
- *Tag-miss with no available cache entry.* If all sixteen ways with the selected set are filled and there is no tag match, code is transferred to the SC1400 core and into the set’s cache

line for which the LRU = 0 (lowest value). If part of the cache is locked, the value is loaded into the unlocked line in the set with the lowest LRU value.

For all instruction fetches, the number of fetch sets written to the cache depends on the burst and block settings configured in the IRCR. The VALID bit is set in the appropriate position for each fetch set retrieved. If prefetch is enabled, the remainder of the fetch units specified by the cache line are written into cache and the VALID bit is set. In addition, the LRU value for the line is set to 0xF and the LRU values for all other lines with the same INDEX number, except for the line for which LRU = 0, are decremented by 1.

4.5.3.3 Tuning the Cache to Improve Performance

Replacing the least-recently-used existing line with a new one is called thrashing. Frequent thrashing indicates cache ineffectiveness. Cache effectivity is based on locality. Programs have two locality attributes:

- *Temporal locality*. A parameter indicating the likelihood that the SC1400 core will often request a given address in memory. A high temporal locality can be caused by a large number of loops in the code.
- *Spatial locality*. A parameter indicating the likelihood that, given a core program request to a certain address in memory, the SC1400 core will also request the “close by” addresses. A high spatial locality means that the code has few change-of-flows.

Because the cacheable memory area and the burst and block sizes are configurable, you may have to determine the optimal sizes and placement of code within the memory for your application. Configuration allows a trade-off between efficient use of the burst capability of the system bus and burdening the M2 and/or external memory with transactions that the SC1400 core may not need. This trade-off is covered in detail in **Section 4.6.1**.

4.5.4 Cache Locking

The ICache can be unlocked, partially locked, or completely locked. The cache is locked by programming the lower and upper boundaries of the LRU via the ICache Control Register (ICCR). The LRU mechanism is then functional only within the programmed boundaries, while the lines outside the boundaries are considered frozen. The Lock Mode bit in the ICCR is used for locking the entire cache. It is recommended that the ICCR be written every time a task starts working. Each task should change the boundaries to enable all multi-task support. The OS can determine which tasks need fixed allocation and which tasks can work with flexible boundaries. The boundary resolution is the size of one LRU priority level for all indexes, that is, one way.

The ICache supports multi-tasking. Allowing partial locks of the ICache, a multi-tasking operating system can return an old task while the ICache still holds the task’s most recently used code. Multi-task support includes both single-stack and multi-stack prioritized and preemptive real-time operating system (RTOS) models. Upon activation, the single-stack OS model executes to completion, but the active task can be preempted by another task with a higher priority. In the

multiple-stack OS model, a task can be activated and preempted at any time. When multi-tasking is introduced, caches add non-determinism to the execution time of each task because of cache thrashing when a task switch occurs. **Figure 4-10** demonstrates the difference between flexible boundaries and fixed allocation strategies.

With flexible boundaries, task 1 can work with all the available cache space. When task 2 arrives, it should change only the upper boundary so the cache space of task 2 is smaller than the full cache space. When task 1 resumes operation, it should change the upper boundary back to the previous value so that the cache now consists of the least recently used parts of the task. This practice helps to ensure that if the task needs to use the code already in the cache, there is no penalty of a miss. If the upper boundary is not changed during the transition from task 1 to task 2, some task 1 instructions may be overwritten by task 2 instructions. When task 1 resumes execution, there may be a resulting cache miss and penalty. In the flexible boundary mechanism, each time a higher-priority task starts operation, a smaller cache space is available.

Note: In single-stack tasks, the nested task with the higher priority must end before the lower-priority task resumes operation.

During fixed allocation, the OS reserves a cache space for each task or group of tasks. When a new task resumes operation, it works only with the cache space allocated for it, meaning that each task should change the lower and upper boundaries of the cache space. In summary:

- *Flexible boundaries* are most suitable for the single-stack OS model.
- *Fixed allocation* is most suitable for the multi-stack OS model.
- *Full cache shared for all tasks* is the simplest technique and very powerful because of the high number of ways in the cache.

In fixed allocation, different parts of the cache can be reserved for different tasks, almost like having an independent cache for each group of tasks. For example, 10 ways (10 KB) of the cache can be exclusively reserved for interrupt handlers, and the remaining 6 ways (6 KB) of the cache can be exclusively reserved for control code. Many other scenarios are possible.

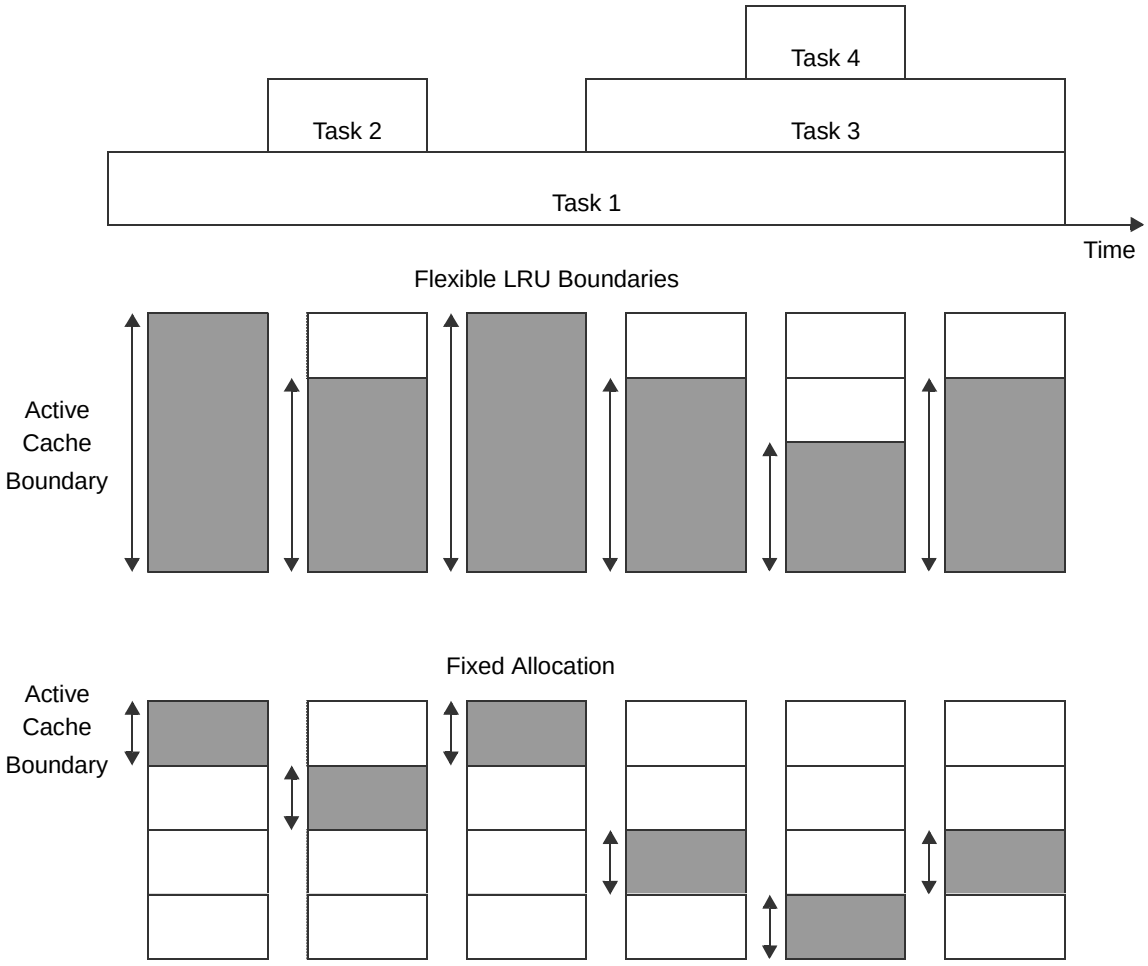


Figure 4-10. Cache Support in Run-Time Multi-tasking

4.5.5 Debugging Support

The ICache can operate in one of the following modes:

- Cache Disabled
- Normal mode
- Cache Debug mode

The ICache supports two types of debugging:

- Run-time debug (ICache in Normal mode)
- Debugging via ICache Debug mode.

4.5.5.1 Run-time Debugging

The event port allows run-time debugging of the ICache using the ICache_Miss and ICache_Miss_External signals to count stall cycles. For details on OCE10 configuration, see the SC1400 extended core header and the *OCE10 On-Chip Emulator Reference Manual.*

4.5.5.2 Cache Debug Mode Debugging

In Cache Debug mode, the normal usage of the ICache is not enabled so that the contents of the ICache array, tag array, valid bit array, and LRU registers can be read out and examined. This mode can give you a more in-depth view of ICache usage and bottlenecks (compared with the hit/miss count) when code performance is optimized on the device. This information can also help in devising LRU boundary allocation schemes for multi-task support.

Cache Debug mode can be useful when the device is placed into Device Debug mode (**Section 16.1**, *Debugging Modes*, on page 16-1) to examine the state of the ICache in a debug session. It can also be used in diagnostic routines for dumping the contents of the cache when the device is not in SC1400 Debug mode.

Note: The Cache Debug mode *operates independently of the SC1400 Debug mode*, so the ICache can be in Cache Debug mode when the SC1400 core is not in Debug mode.

4.5.5.2.1 Entering Cache Debug Mode

There are two different ways to use Cache Debug mode:

- From the debugger when the SC1400 is halted in SC1400 Debug mode
- From a program executing on the SC1400 core. SC1400. All instructions executed within this mode are located in M1 memory or in a non-cacheable region of M2 or DDR memory.

Before the contents of the cache can be accessed, the ICache must first be placed into Cache Debug mode. When there is a read access to addresses assigned for the Cache Debug mode and the ICache is not in Cache Debug mode, the access is discarded. The device is placed into Cache Debug Mode by setting ICCR[DM] and enabling the cache (ICCR[ON] = 1).

Note: If a miss occurs while the cache is bringing in data to service the miss (normal or prefetch accesses), a write to the ICCR register stalls until all accesses from the instruction fetch unit have completed.

Entering Cache Debug mode immediately stops the ICache update mechanism (load of new data by the instruction fetch unit), regardless of the fetch unit status. In Cache Debug mode, the ICache does not issue any hits (as it does in Lock mode) or perform thrashes. Cache Debug mode is only for viewing the ICache status and breakpoint support.

4.5.5.2.2 ICache Structure

The ICache is structured as follows:

- Four sets
- 16 cache lines in each set, one corresponding to each of the 16 ways (for a total of 64 cache lines)
- Sixteen 128-bit entries in each cache line (for a total of 64 cache lines × 16 entries/Line = 1024 16-byte entries = 16 KB)

For a better understanding of the procedures for reading out the ICache contents, review the following figures:

- **Figure 4-7**, *Structure of the 16 KB ICache*, on page 4-18
- **Figure 4-8**, *Structure of One of the 64 Cache Lines in the 16 KB ICache*, on page 4-18.

All entries in the cache array are directly accessible from a 16 KB portion of the extended core memory map beginning at the address ICARRAY_BASE (see **Table 5-2** on page 5-5). Both read and write accesses are supported.

When the ICache array is accessed in Cache Debug mode, only 64-bit read and write accesses are permitted. The ICache array contents are accessed using data addresses of the form shown in **Figure 4-11**, where the 4-bit entry field is split into two different locations in the address:

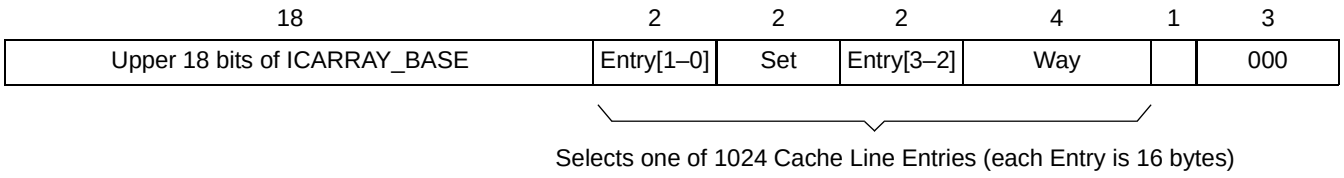


Figure 4-11. SC1400 Address for Data Accesses to Entries in the ICache

4.5.5.3 Techniques for Accessing the Tag, Valid Bit, and LRU Arrays

ICache resources such as the tag array, valid bit array, and LRU machine, have a memory-mapped status register that holds 16 bits of the contents of the resource to which it belongs. To read the ICache status, the SC1400 core reads from a specific memory address. The contents of the register are sent to the SC1400 core, and that register is automatically loaded from the next address of its resource for the next core read.

In addition, the ICCMR provides a special initialization command that simultaneously initializes all status registers by reading the first data from each resource into the status register of that resource.

Each state register is 16 bits long. The resources larger than 16 bits (the tag is 22 bits and the LRU register is 64 bits) are read in more than one read:

- Two reads per tag, first the LSBs and then the MSBs (zero padded)
- Four reads per LRU of a particular index, again LSB to MSB in sequential order.

4.5.5.3.1 Reading the Contents of the Tag Array

There is a total of 64 tags in the tag array, one corresponding to each cache line. These 64 tags are not directly readable through the memory map but are instead accessed through a special-purpose 16-bit ICache register, TASR. An internal 7-bit pointer is used for reading out the tag array in Cache Debug mode, where:

- Bits 6–3 of the pointer select the way.
- Bits 2–1 of the pointer select the set.
- Bit 0 of the pointer selects either the lowest 16 bits of the tag or the upper 10 bits of tag.

This pointer is not directly accessible through the programming model but can be reset to zero. **Table 4-5** shows how the entire tag array is read.

Table 4-5. Procedure for Reading Tag Array

SC1400 Core Action	Way Accessed	Set Accessed	Bits of Tag Accessed	Comments
Write 0x8 to ICCMR[15–12]	—	—	—	Initialize internal pointers so that the first TASR access is to Way 0, Set 0. Must already be in Cache Debug mode.
NOP	—	—	—	One cycle delay required after pointers are reset.
Read All Tags in Way 0				
Read 16 bits from TASR	0	0	[15–0]	Reads tag 1 of 64 tags (lowest 16-bits of 22-bit tag).
Read 16 bits from TASR	0	0	[21–16]	Reads tag 1 of 64 tags (uppermost 10-bits of 22-bit tag, zero extended to 16-bits).
Read 16 bits from TASR	0	1	[15–0]	Reads tag 2
Read 16 bits from TASR	0	1	[21–16]	Reads tag 2 ...
Read 16 bits from TASR	0	2	[15–0]	Reads tag 3...
Read 16 bits from TASR	0	2	[21–16]	Reads tag 3...
Read 16 bits from TASR	0	3	[15–0]	Reads tag 4...
Read 16 bits from TASR	0	3	[21–16]	Reads tag 4...
Read All Tags in Way 1				
Read 16 bits from TASR	1	0	[15–0]	Reads tag 5...
Read 16 bits from TASR	1	0	[21–16]	Reads tag 5...
Read 16 bits from TASR	1	1	[15–0]	Reads tag 6...
Read 16 bits from TASR	1	1	[21–16]	Reads tag 6...
Read 16 bits from TASR	1	2	[15–0]	Reads tag 7...

Table 4-5. Procedure for Reading Tag Array (Continued)

SC1400 Core Action	Way Accessed	Set Accessed	Bits of Tag Accessed	Comments
Read 16 bits from TASR	1	2	[21–16]	Reads tag 7...
Read 16 bits from TASR	1	3	[15–0]	Reads tag 8...
Read 16 bits from TASR	1	3	[21–16]	Reads tag 8...
Read All Tags in Way 2				
(Continue as for Way 1...)				
(Continue reading tags in similar manner for Ways 3, 4, 5, ..., 14)				
Read All Tags in Way 15				
Read 16 bits from TASR	15	0	[15–0]	Reads tag 61...
Read 16 bits from TASR	15	0	[21–16]	Reads tag 61...
Read 16 bits from TASR	15	1	[15–0]	Reads tag 62...
Read 16 bits from TASR	15	1	[21–16]	Reads tag 62...
Read 16 bits from TASR	15	2	[15–0]	Reads tag 63...
Read 16 bits from TASR	15	2	[21–16]	Reads tag 63...
Read 16 bits from TASR	15	3	[15–0]	Reads tag 64...
Read 16 bits from TASR	15	3	[21–16]	Reads tag 64...

4.5.5.3.2 Reading the Contents of the Valid Bit Array

There is a total of 64 cache lines, each with 16 valid bits (one per entry). The valid bits are not directly readable through the memory map but are instead accessed through a special-purpose 16-bit ICache register, VBASR. For each way within a set, there is a set of 16 valid bits that are accessible to the SC1400 core via one move.w instruction that reads a 16-bit value. An internal 6-bit pointer is used to read out the valid bit array in Cache Debug mode:

- Bits [5–4] of the pointer select the set.
- Bits [3–0] of the pointer select the entry.

This pointer is not directly accessible through the programming model but can be reset to zero. **Table 4-6** shows how the entire valid bit array can be read.

Table 4-6. Procedure for Reading Valid Bit Array

SC1400 Core Action	Set Accessed	Entry Accessed	Comments
Write 0x8 to ICCMR[15–12]	—	—	Initialize internal pointers so that the first VBASR access is for Set 0, Entry 0. Must already be in Cache Debug mode.
NOP	—	—	One cycle delay required after pointers are reset.

Table 4-6. Procedure for Reading Valid Bit Array (Continued)

SC1400 Core Action	Set Accessed	Entry Accessed	Comments
Read Valid Bit Vectors in Set 0			
Read 16 bits from VBASR	0	0	Reads the first 16-bit valid bit vector (vector 1) for Entry 0 in Set 0, where: • MSB corresponds to Way 15 • LSB corresponds to Way 0
Read 16 bits from VBASR	0	1	Reads the second 16-bit valid bit vector (vector 2) for Entry 1 in Set 0, where: • MSB corresponds to Way 15 • LSB corresponds to Way 0
Read 16 bits from VBASR	0	2	Reads vector 3...
Read 16 bits from VBASR	0	3	Reads vector 4...
Read 16 bits from VBASR	0	4	Reads vector 5...
Read 16 bits from VBASR	0	5	Reads vector 6...
Read 16 bits from VBASR	0	6	Reads vector 7...
Read 16 bits from VBASR	0	7	Reads 8th vector 8...
Read 16 bits from VBASR	0	8	Reads vector 9...
Read 16 bits from VBASR	0	9	Reads vector 10...
Read 16 bits from VBASR	0	10	Reads vector 11...
Read 16 bits from VBASR	0	11	Reads vector 12...
Read 16 bits from VBASR	0	12	Reads vector 13...
Read 16 bits from VBASR	0	13	Reads vector 14...
Read 16 bits from VBASR	0	14	Reads vector 15...
Read 16 bits from VBASR	0	15	Reads vector 16...
Read All Tags in Set 1			
(continue as before ...)			
Read All Tags in Set 2			
(continue as before ...)			
Read Valid Bit Vectors in Set 3			
Read 16 bits from VBASR	3	0	Reads vector 49...
Read 16 bits from VBASR	3	1	Reads vector 50...
Read 16 bits from VBASR	3	2	Reads vector 51...
Read 16 bits from VBASR	3	3	Reads vector 52...
Read 16 bits from VBASR	3	4	Reads vector 53...
Read 16 bits from VBASR	3	5	Reads vector 54...
Read 16 bits from VBASR	3	6	Reads vector 55...
Read 16 bits from VBASR	3	7	Reads vector 56...

Table 4-6. Procedure for Reading Valid Bit Array (Continued)

SC1400 Core Action	Set Accessed	Entry Accessed	Comments
Read 16 bits from VBASR	3	8	Reads vector 57...
Read 16 bits from VBASR	3	9	Reads vector 58...
Read 16 bits from VBASR	3	10	Reads vector 59...
Read 16 bits from VBASR	3	11	Reads vector 60...
Read 16 bits from VBASR	3	12	Reads vector 61...
Read 16 bits from VBASR	3	13	Reads vector 62...
Read 16 bits from VBASR	3	14	Reads vector 63...
Read 16 bits from VBASR	3	15	Reads vector 64...

4.5.5.3.3 Reading the LRU Registers

The LRU status is not directly readable through the memory map but is instead accessed through a special-purpose 16-bit ICache register, LRUSR. An internal 4-bit pointer is used to read out the LRU status array in Cache Debug mode:

- Bits [3–2] of the pointer select the set.
- Bits [1–0] of the pointer select a set of four ways.

This pointer is not directly accessible through the programming model but can be reset to zero. **Table 4-7** shows how the entire LRU status array can be read.

Table 4-7. Procedure for Reading LRU Status Array

SC1400 Core Action	Set Accessed	Ways Accessed	Comments
Write 0x8 to ICCMR[15–12]	—	—	Initialize internal pointers so that the first LRUSR access is to Set 0, Ways 0–3. Must already be in Cache Debug mode.
NOP	—	—	One cycle delay required after resetting pointers.
Read All LRU Status for Set 0			
Read 16 bits from LRUSR	0	3-0	Reads LRU status for four cache lines: • [3–0] for Way 0 • [7–4] for Way 1 • [11–8] for Way 2 • [15–12] for Way 3
Read 16 bits from LRUSR	0	7-4	Reads LRU status for four cache lines: • [3–0] for Way 4 • [7–4] for Way 5 • [11–8] for Way 6 • [15–12] for Way 7

Table 4-7. Procedure for Reading LRU Status Array (Continued)

SC1400 Core Action	Set Accessed	Ways Accessed	Comments
Read 16 bits from LRUSR	0	11-8	Reads LRU status for four cache lines: • [3–0] for Way 8 • [7–4] for Way 9 • [11–8] for Way 10 • [15–12] for Way 11
Read 16 bits from LRUSR	0	15-12	Reads LRU status for four cache lines: • [3–0] for Way 12 • [7–4] for Way 13 • [11–8] for Way 14 • [15–12] for Way 15
Read All Tags in LRU Status for Set 1			
(continue as before ...)			
Read All Tags in LRU Status for Set 2			
(continue as before ...)			
Read All LRU Status for Set 3			
Read 16 bits from LRUSR	3	3-0	Reads LRU status for four cache lines: • [3–0] for Way 0 • [7–4] for Way 1 • [11–8] for Way 2 • [15–12] for Way 3
Read 16 bits from LRUSR	3	7-4	Reads LRU status for four cache lines: • [3–0] for Way 4 • [7–4] for Way 5 • [11–8] for Way 6 • [15–12] for Way 7
Read 16 bits from LRUSR	3	11–8	Reads LRU status for four cache lines: • [3–0] for Way 8 • [7–4] for Way 9 • [11–8] for Way 10 • [15–12] for Way 11
Read 16 bits from LRUSR	3	15–12	Reads LRU status for four cache lines: • [3–0] for Way 12 • [7–4] for Way 13 • [11–8] for Way 14 • [15–12] for Way 15

4.5.5.4 Setting Breakpoints with the ICache

To enable breakpoint insertion in cached code, the ICache includes a clear line command that resets all valid bits for a specific line ({way,index}). This is not the same as a read line. The line to be cleared can be obtained via the status reading mechanism. For detailed encodings and addresses, see **Section 4.8.2**, *ICache Registers*, on page 4-45.

4.6 Instruction Fetch Unit

The IFU handles all accesses to external address space and conducts all ICache update activity. It accelerates SC1400 core performance by initiating cache updates to memory outside the extended core. The data updates consist of data needed with high probability in a sequential code (prefetch). IFU operation supports the following types of instruction accesses:

- *Cache fill*. Triggered by a cache miss. The IFU initiates an access on the AMIC to bring the data into both the SC1400 core and the ICache.
- *Prefetch*. Includes cache updates with data of sequential addresses following the miss. It is triggered by the fetch and occurs in bursts for efficient use of the external memory and associated interfaces. The burst size is programmable to 1, 2, or 4 fetch sets of 128 bits (see **Section 4.8**, *Extended Core Programming Model*, on page 4-40). The prefetch ends when it reaches the end of cache line or when a new “miss” access occurs. The prefetch phase greatly improves cache performance in a system running a program with sequential code because in the next access to the code area, the data is probably already in the cache. The prefetch occurs in two phases:
  - *Prefetch of a block*. The prefetch occurs in bursts on the AMIC bus following the first miss. A block is defined as a programmable number of fetch sets (128 bits) that the IFU brings without interference with the miss and with a high priority. It must be an integer multiple of the burst size. A block is the minimal unit of data in the first stage of fetch and prefetch.
  - *Prefetch of data to end of cache line*. A cache line is 256 bytes long. The prefetch initiates cache updates from the address following the end of phase 1 until the end of the cache line. Accesses occur in bursts. This phase of prefetching can be turned off to reduce IFU traffic on the crossbar switch, making way for other transfers.
- *Non-cacheable accesses*. Accesses to non-cacheable regions of memory are processed through the IFU, which initiates an access on the AMIC to bring the data to the SC1400 core.

4.6.1 Cache Bursting Parameters

The ICache is designed for maximum performance in real-time applications. It has many user-programmable features for tuning an application, as described in **Section 4.8**, *Extended Core Programming Model*, on page 4-40. Programmable ICache parameters include the following:

- *Burst size*. 1 or 4 entries, where each entry is 16 bytes (one fetch set)
- *Primary set size*. 1, 2, or 4 entries, where each entry is 16 bytes
- Optional prefetch to end of cache line

When a cache miss occurs, a cache fill begins. A portion of the cache line is loaded from M2 memory or external memory as a primary set of bursts composed of simple cache bursts. The IFU

priority is elevated during the primary set. The size of the primary set is always greater than or equal to the burst size. For burst sizes greater than one entry, the first burst accesses its data critical entry first, immediately bringing the instructions at the missed address to the SC1400 core. Then all other entries in the burst are loaded. When the primary set is loaded, the IFU fetches the remainder of the cache line (the prefetch set) if this option has been selected for this region.

4.6.1.1 Burst of 1, Primary Set Size of 1

Figure 4-12 shows an example with a burst size of 1 (that is, 1 × 128 bits), the primary set size is 1 (1 × 128 bits), and the prefetch capability is enabled. This is a non-wrapping case because the burst size is 1.

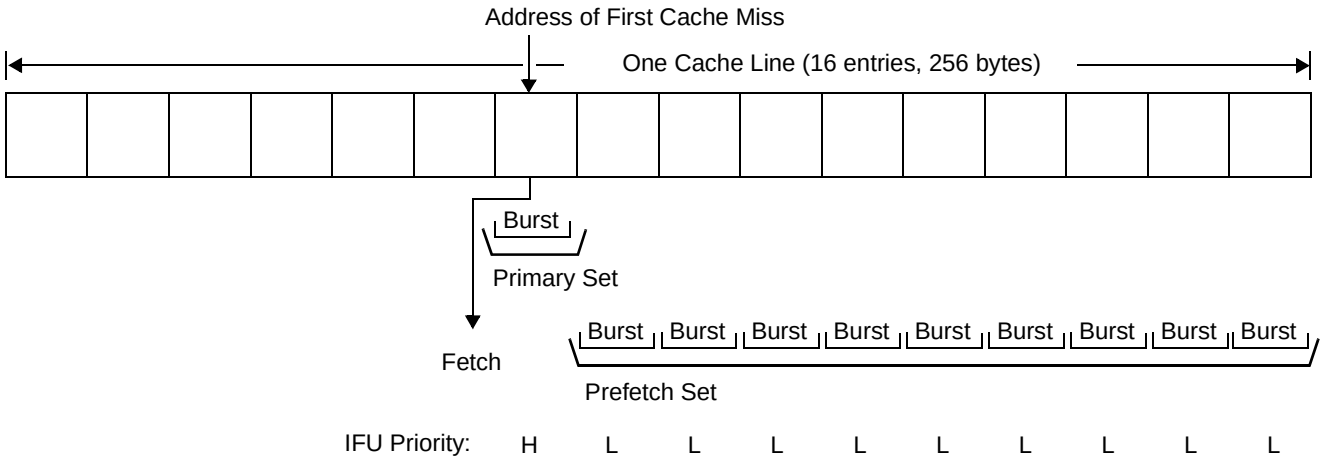


Figure 4-12. Responding to a Cache Miss, Burst = 1and Primary Set Size = 1

4.6.1.2 Burst of 1, Primary Set Size of 2

Figure 4-13 shows an example that demonstrates how the IFU first responds to a cache miss with a primary set of bursts of a predefined burst size, followed by an optional set of prefetch bursts. The burst size is 1 (that is, 1 × 128 bits), the primary set size is 2 (2 × 128 bits), and the prefetch capability is enabled using the same burst size. The IFU priority is elevated to high (H) in the Primary Set. The IFU returns to its normal priority (L) for the prefetch set. This is a non-wrapping case because its burst size is 1.

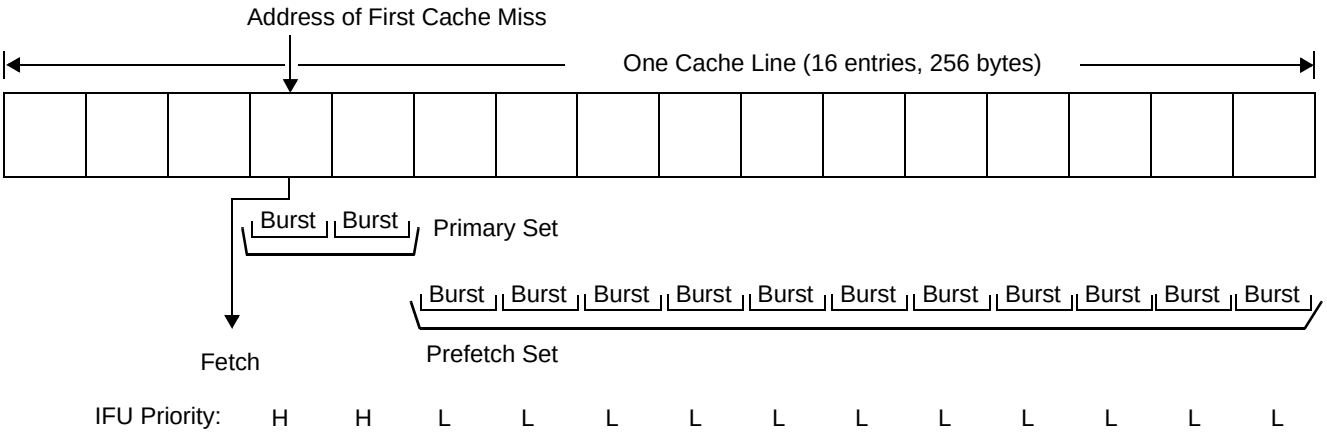


Figure 4-13. Responding to a Cache Miss, Burst = 1, Primary Set Size = 2

4.6.1.3 Burst of 1, Primary Set Size of 4

Figure 4-14 shows an example that demonstrates how the IFU first responds to a cache miss with a primary set of bursts with a predefined burst size, followed by an optional set of prefetch bursts. In this example, the burst size is 1 (that is, 1×128 bits), the primary set size is 4 (4×128 bits), and the prefetch capability is enabled using the same burst size. The IFU priority is elevated to high (H) in the primary set. The IFU returns to its normal priority (L) for the prefetch set. This is a non-wrapping case because its burst size is 1.

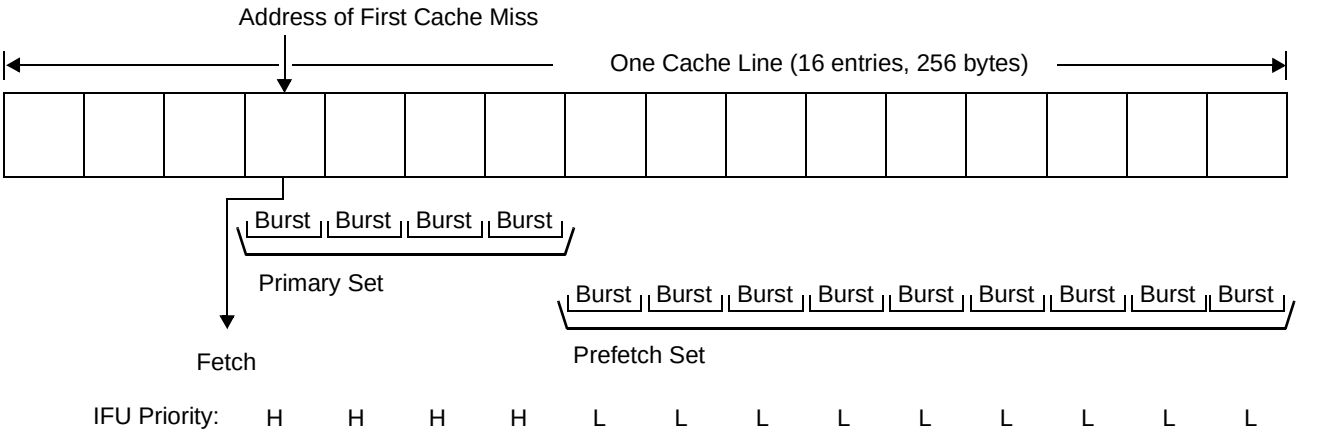


Figure 4-14. Responding to a Cache Miss, Burst = 1 and Primary Set Size = 4

4.6.1.4 Burst of 4, Primary Set Size of 4

Figure 4-15 shows an example with a burst size of 4 (4×128 bits), a primary set size of 4 (4×128 bits), and the prefetch capability enabled. For burst sizes greater than 1, data is accessed critical entry first. In this example there are 4 entries in the primary set. The first entry fetched is the entry where the cache miss occurred, 1. The primary set then fetches the next two sequential entries, 2 and 3. The last entry accessed in the primary set is the entry immediately preceding the entry containing the cache miss, 4. The prefetch set, 5 and beyond, are always accessed sequentially.

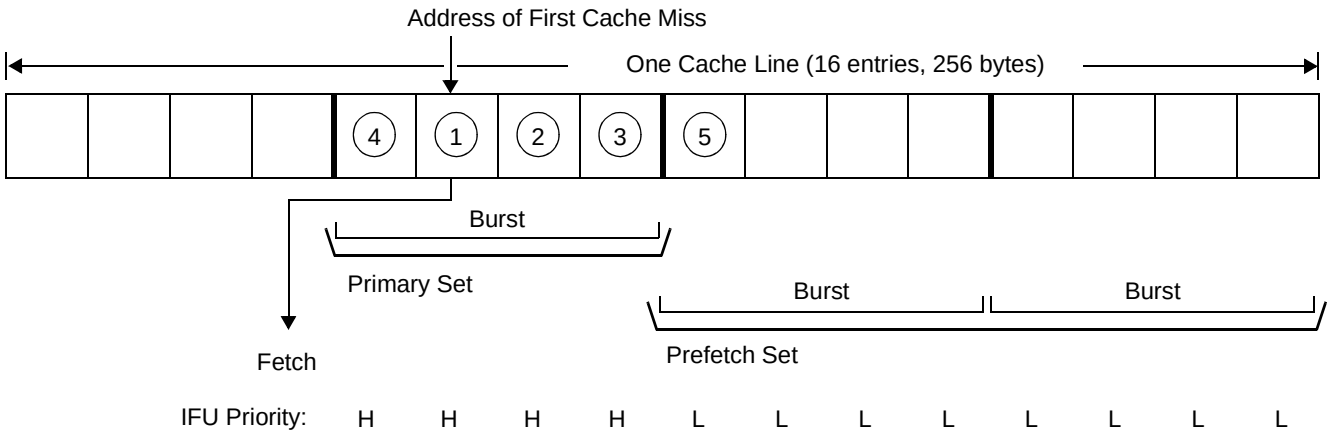


Figure 4-15. Responding to a Cache Miss, Burst = 4, Primary Set Size = 4

4.6.1.5 Trade-offs in Setting the Burst and Primary Set Sizes

Setting the burst size is a trade-off between using the bus efficiently in the primary set and occupying external memories and buses with transactions that the SC1400 core may not need. The primary set size reflects the importance given to the fetch process. The primary set is an inseparable unit of transfer. Each miss causes a fetch + prefetch of a block at a minimum, so setting a large block causes each miss to bring a lot of data into the cache. The SC1400 core may not necessarily need this data, possibly delaying other operations that occur during a block, such as a read. However, a cache miss can also bring data that the SC1400 core needs without delay by other operations.

For instruction regions that access external DDR memory, a burst with a primary burst size of 4 is recommended to overcome the latency associated with accessing an external DDR memory.

The bursts in the primary set remain grouped together if the priority of the IFU master is elevated while the bursts pass through the crossbar switch. If a second master with a higher priority than the IFU also elevates its priority, this second master can interrupt the bursts in the primary set on a burst boundary. However, the AULB bits in the crossbar switch Master General Purpose Control Register can ensure that the bursts in the primary set cannot be interrupted even by a master with a higher priority. The AULB bits are applicable only when the burst size is 1, and they are individually programmable for each master port in the crossbar switch.

4.6.2 Servicing a Second Cache Miss

When a cache miss occurs, the first entry fetched by the IFU contains the data requested by the SC1400 core so it can resume operation. As the SC1400 core executes instructions, the remaining entries of the block transfer are performed. If a region is programmed for prefetching to the end of the cache line, the prefetch bursts begin when the block transfer completes. When the prefetch bursts are loaded, the SC1400 core may request a new address that is not in the cache. In this case, a second cache miss occurs. Following are two cases in which the IFU responds to a cache miss:

- *Prefetch hit, continue current cache miss servicing*. The IFU identifies cache misses to an address in the IFU prefetch that is being loaded. This identification process is called a prefetch hit. It saves an extra access on the system buses. This feature is effective when sequential data is transferred from external memory.
- *Terminate prefetch and initiate new cache miss servicing*. The IFU can stop a prefetch access before it completes. If a prefetch is executing and a new prefetch access that is a cache miss is waiting on the SC1400 core buses, the IFU stops the prefetch operation in progress and initiates a new cache miss operation.

4.6.3 Transaction Priorities

The IFU transfers an attribute to the system to indicate whether a transaction is a primary burst or prefetch transaction. The crossbar switch uses this attribute to prioritize the access. During the accesses in the primary set, the priority is high. The priority is also high during prefetch accesses when there is a prefetch hit.

4.7 Configuring the Address Space Outside the Extended Core

Regions of memory outside the extended core can be configured for write buffer operation or a cacheable area.

4.7.1 Write Buffer Data Areas

Four different regions of memory outside the extended core can be programmed for write buffer operation. The four write buffer data areas are specified in the Write Buffer Data Area Registers 0–3 (WBDAR[0–3]). An area is identified by its base address and size, and any of the four data area registers can be disabled. Attributes for these areas indicate the type of write, as follows, and whether the write is global:

- Normal writes through the write buffer
- Immediate writes (bypassing write buffer) with no freeze
- Immediate writes with a freeze

The area base must always be a multiple of the area size, with the exception of base = 0, when the size can be any value. The data area registers are programmed by setting the base register and SZ256 bits. **Table 4-8** summarizes the base addresses and sizes that can be programmed for a data area. The original base column represents the 24 MSB of the base address needed for the area definition. The 8 LSBs are not used.

In the following example, a write buffer data area is created that starts at 1 MB with a size of 256 KB. Recall the basic condition that the base address (1 MB) must be an integer multiple of the size (256 KB). The steps in defining the example area are as follows:

1. According to the area size (256 KB), choose line 11 in **Table 4-8**.
2. Clear WBDARx[SZ256] as shown in line 11 in **Table 4-8**.
3. Write the upper 24-bits of the base address in binary. For a base address of 1 MB, the value is 0b00000000 00010000 00000000.
4. Verify that this base address matches the required format in line 11:
   - `00000000 00010000 00000000`: Upper 24-bits of the address
   - `xxxxxxxx xxxxxx00 00000000`: Required format for line 11 matches.
5. The 24-bit value placed into the WBDARx[BASE] field is the ORing of
   - `00000000 00010000 00000000`: Upper 24-bits of the address
   - `00000000 00000010 00000000`: Value from line 11 (*x* replaced with zeros).

   This procedure results in base = 0x001200 or in binary form 0b00000000 00010010 00000000.

Table 4-8. Programming the Write Buffer Data Area Base and Size

Case	Size of Region	Upper 24-Bits of Desired Base Address[1]	Value Placed into WBDARx[BASE] Field[2]	Placed into SZ256 Bit
1	256	xxxxxxxx xxxxxxxx xxxxxxxx	xxxxxxxx xxxxxxxx xxxxxxxx	1
2	512	xxxxxxxx xxxxxxxx xxxxxxx0	xxxxxxxx xxxxxxxx xxxxxxx1	0
3	1 KB	xxxxxxxx xxxxxxxx xxxxxx00	xxxxxxxx xxxxxxxx xxxxxx10	0
4	2 KB	xxxxxxxx xxxxxxxx xxxxx000	xxxxxxxx xxxxxxxx xxxxx100	0
5	4 KB	xxxxxxxx xxxxxxxx xxxx0000	xxxxxxxx xxxxxxxx xxxx1000	0
6	8 KB	xxxxxxxx xxxxxxxx xxx00000	xxxxxxxx xxxxxxxx xxx10000	0
7	16 KB	xxxxxxxx xxxxxxxx xx000000	xxxxxxxx xxxxxxxx xx100000	0
8	32 KB	xxxxxxxx xxxxxxxx x0000000	xxxxxxxx xxxxxxxx x1000000	0
9	64 KB	xxxxxxxx xxxxxxxx 00000000	xxxxxxxx xxxxxxxx 10000000	0
10	128 KB	xxxxxxxx xxxxxxx0 00000000	xxxxxxxx xxxxxxx1 00000000	0
11	256 KB	xxxxxxxx xxxxxx00 00000000	xxxxxxxx xxxxxx10 00000000	0
12	512 KB	xxxxxxxx xxxxx000 00000000	xxxxxxxx xxxxx100 00000000	0
13	1 MB	xxxxxxxx xxxx0000 00000000	xxxxxxxx xxxx1000 00000000	0
14	2 MB	xxxxxxxx xxx00000 00000000	xxxxxxxx xxx10000 00000000	0
15	4 MB	xxxxxxxx xx000000 00000000	xxxxxxxx xx100000 00000000	0
16	8 MB	xxxxxxxx x0000000 00000000	xxxxxxxx x1000000 00000000	0
17	16 MB	xxxxxxxx 00000000 00000000	xxxxxxxx 10000000 00000000	0
18	32 MB	xxxxxxx0 00000000 00000000	xxxxxxx1 00000000 00000000	0
19	64 MB	xxxxxx00 00000000 00000000	xxxxxx10 00000000 00000000	0
20	128 MB	xxxxx000 00000000 00000000	xxxxx100 00000000 00000000	0
21	256 MB	xxxx0000 00000000 00000000	xxxx1000 00000000 00000000	0

Table 4-8. Programming the Write Buffer Data Area Base and Size (Continued)

Case	Size of Region	Upper 24-Bits of Desired Base Address[1]	Value Placed into WBDARx[BASE] Field[2]	Placed into SZ256 Bit
22	512 MB	xxx00000 00000000 00000000	xxx10000 00000000 00000000	0
23	1 GB	xx000000 00000000 00000000	xx100000 00000000 00000000	0
24	2 GB	x0000000 00000000 00000000	x1000000 00000000 00000000	0
25	4 GB	00000000 00000000 00000000	10000000 00000000 00000000	0

Notes: 1. The base address must always be an integer multiple of (and must be greater than) the size—except when a base address of 0 is selected.

2. In this column, the value in the x bits defines the base address of the region, and the location of the right-most 1 specifies the size of the region.

4.7.2 Instruction Cacheable Area

Four different regions of M2 memory or external memory can be programmed as cacheable memory for instruction fetches (the default is non-cacheable). The cacheable regions are set up in the Instruction Cacheable Area registers 0–3 (IRCR[0–3] and IRBSR[0–3]), where an area is specified by its base address and size. Attributes for each area define the region and its burst characteristics. Cacheable regions must be located at an address higher than the larger of 16 MB or the external system base address. In a conflict, the cacheable area is always higher than 16 MB and the external system base address.

Note: The four user-configured regions must not overlap.

The cacheable area is determined by a base address and size. The base address must always be a multiple of the area size, with the exception of base = 0, when the size can be any value. The cacheable area is programmed by correctly loading the Instruction Cacheable Area Base Register (IRBSR[0-3]) and the 64KB bit in the Instruction Cacheable Area Configuration Register (IRCR[0-3]). These 17 bits specify both the base address and size of the region.

The upper 16-bits of the base address are loaded into the IRBSR register (the lowest 16-bits of the base address are not used). The LSBs of the base address are 0’s because the base address must be a multiple of the region’s size. The size of the region is determined by placing a value of 1 into the IRBSR register in the correct location specified for the desired size, as specified in **Table 4-10**. The steps for using this table are as follows:

1. Determine size needed for your cacheable region.
2. Consulting **Table 4-9**, determine the value of the IRCR[64KB] bit.

   Sizes must always be a power of two and the maximum size for a region is one GB, which is represented by a single value of 1 in the IRBSR, where the location of this 1 specifies the size. Use **Table 4-9** to determine the bit where a 1 is loaded into the IRBSR.

Base addresses must be an integer multiple of the size (that is, a power of 2). Therefore, a valid base address has zeros in its N + 16 LSBs (N is shown in **Table 4-9**).

If a size of 64 KB is desired, the 1 is loaded into the 64KB bit of IRCR[0–3].

Table 4-9. Programming the Base Address and Size of a Cacheable Region

Desired Size of Cacheable Region	N	Valid Base Addresses for Cacheable Regions (Upper 16 Bits)	Value in IRCR[64KB]	Place a 1 into IRBSR[N–1]	Upper (16–N) Bits of Base Address Loaded into IRBSR[15–N]
64 KB	0	BBBB BBBB BBBB BBBB	1	N/A	BBBB BBBB BBBB BBBB
128 KB	1	BBBB BBBB BBBB BBB0	0	IRBSR[0]	BBBB BBBB BBBB BBB1
256 KB	2	BBBB BBBB BBBB BB00	0	IRBSR[1]	BBBB BBBB BBBB BB10
512 KB	3	BBBB BBBB BBBB B000	0	IRBSR[2]	BBBB BBBB BBBB B100
1 MB	4	BBBB BBBB BBBB 0000	0	IRBSR[3]	BBBB BBBB BBBB 1000
2 MB	5	BBBB BBBB BBB0 0000	0	IRBSR[4]	BBBB BBBB BBB1 0000
4 MB	6	BBBB BBBB BB00 0000	0	IRBSR[5]	BBBB BBBB BB10 0000
8 MB	7	BBBB BBBB B000 0000	0	IRBSR[6]	BBBB BBBB B100 0000
16 MB	8	BBBB BBBB 0000 0000	0	IRBSR[7]	BBBB BBBB 1000 0000
32 MB	9	BBBB BBB0 0000 0000	0	IRBSR[8]	BBBB BBB1 0000 0000
64 MB	10	BBBB BB00 0000 0000	0	IRBSR[9]	BBBB BB10 0000 0000
128 MB	11	BBBB B000 0000 0000	0	IRBSR[10]	BBBB B100 0000 0000
256 MB	12	BBBB 0000 0000 0000	0	IRBSR[11]	BBBB 1000 0000 0000
512 MB	13	BBB0 0000 0000 0000	0	IRBSR[12]	BBB1 0000 0000 0000
1 GB	14	BB00 0000 0000 0000	0	IRBSR[13]	BB10 0000 0000 0000

Notes: 1. The base address must always be an integer multiple of (and must be greater than) the size, except when a base address of 0 is selected.

2. In this column, the value in the *x* bits defines the base address of the region and the location of the rightmost 1 specifies the size of the region.

In the following example, a cacheable region is created that starts at 32 MB with a size of 256 KB. Recall the basic condition that the base address (32 MB) must be an integer multiple of the size (256 KB). The steps in defining the example area are as follows:

1. Write the base address in 32-bit representation. The value of 32 MB is written as 0b00000010 00000000 00000000 00000000.
2. Based on the size (256 KB), choose Case 3 in **Table 4-10**. For this case, the lowest two bits are 10 and the value of the 64KB bit is 0.
3. Determine the base bits for the IRBSR. The two lowest bits are 10 (from step 2). The upper bits are determined according to the remaining bits of the base address [31–18], that is, 0b00000010 000000.

This procedure results in base = 0x0202, or in binary form, 0b00000010 00000010.

When an instruction region is disabled or its base or size is modified, portions of the address space that were previously cacheable may no longer be cacheable. This change occurs, for example, when the size of a region is decreased or the base of the region is moved. To ensure that these locations are removed from the cache, flush the cache immediately after changing the parameters. The procedure for changing a region's parameters after the cache is enabled is presented in **Section 4.8.2.5**, *ICache Registers,* on page 4-47.

4.7.3 Data Coherency

MSC711x is a single-core environment. Therefore, solving sharing issues between tasks that run on the system is fairly straightforward. The only exception is that a process can intervene by writing to the M1 memory through the DMA port, which may not be synchronized with the activity of the SC1400 core. This problem is limited in its effect because the M1 memory is not cacheable. This issue is well known in the DSP application field and is easily resolved.

However, a multi-core system in which MSC711x represents one core is more complicated. Such a system may have shared high-hierarchy memory, which requires data coherency support with the external system using software mechanisms, such as semaphores, events, and so on.

4.7.3.1 Global Memory Attributes

An external processor can manage a coherency protocol for accesses on the system bus by snooping the addresses of all accesses. However, this is a demanding requirement that is expensive to implement. Therefore, the assumption is that the system snoops addresses only on a subset of the memory space, which is defined as shared in the system memory map (for example, the area where the system global variables are stored). A data memory segment is tagged using the global attribute. When an external MSC71xx access to this segment is identified, one or both global attribute signals are asserted with it, thereby informing the external master to activate the coherency logic for this access. From the MSC711x perspective, these are simple access tags.

4.7.3.2 Semaphore Support

The SC1400 Bit Mask Test and Set instruction (BMTSET.W) supports the use of semaphores. This is a read-modify-write instruction that reads from an address, performs a test on a group of bits in the data, and if the these bits are clear, performs a write to that address with these bits set. If either the test or the write are not successful, it is indicated in an internal core status update (the T bit in the SR). The read and write cycles are issued as an atomic (non-interruptible) pair with an attribute signal. For details, see **Section 4.4.3**, *Atomic Accesses (Read-Modify-Write),* on page 4-14.

4.7.3.3 Program and Data Coherency

The SC1400 core has a unified memory map, so an address can be both a program and a data location. This is why the M1 memory, for example, has both program and data buses routed to it.

Most DSP applications use this feature to provide software flexibility, enabling memory usage between tasks and applications to be changed. However, a program rarely uses this feature to modify itself. That is a program rarely uses data writes to change the contents of the code to be used as program instructions by that same program.

The hardware does not ensure coherency between the instruction and data caches. The program and data fetch paths are completely diverged. If data is written to an address already fetched into the ICache, it is not reflected by the hardware. The software should flush the relevant area in the ICache so that it can be reloaded with the modified program. For completeness, the SC1400 core should perform a change-of-flow instruction to flush its internal fetch pipe and reload new fetch sets from the change-of-flow destination.

4.8 Extended Core Programming Model

The extended core can be configured to fit the needs of the system architecture and the application. The extended core registers are accessed from an internal bus. All registers are memory-mapped. The registers are accessed via data accesses from the SC1400 core that are the same size as the register. For example, if the register size is 16 bits, the command is MOVE.W. If the register size is 32 bits, the command is MOVE.L. If a write is initiated to a register with a logical influence, the new value is valid according to the unit that is affected by the register. Changes in registers that affect the write buffer are valid, as follows:

- *WBCR[9-0] Watch Dog*. If the write buffer is empty, the new value is ready on the next core cycle after the write to the register. If the write buffer is not empty, the new value is valid at the next restart of the watch dog, that is, at flush or when the write buffer ends all accesses.
- *WBCR[12] wb_off*. Valid at the next core cycle after the write to the register.
- *WBFR software flush*. Valid at the next core cycle after the read from the register.
- *WBDARx memory areas*. Valid at the next core cycle after the write to the register.

4.8.1 ECI Registers

Following is the list of ECI registers and the pages on which they are discussed. For the value of the base address of this register file, ECI_BASE, see **Table 5-1** on page 5-3:

- WB Control Register (WBCR), **page 4-41**.
- WB Software Flush Register (WBFR), **page 4-41**.
- WB Data Area Register (WBDAR[0–3], **page 4-42**.
- Version Register (VR), **page 4-44**.
- General-Purpose System Control Register (GPSCTL), **page 4-44**.
- General-Purpose Register 0, **page 4-44**.

WBCR WB Control Register ECI_BASE + 0x04

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	WBOFF	—	—	WD[9–0]									
TYPE	R/W															
RESET	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1

WBCR configures the write buffer attributes.

Table 4-10. WBCR Bit Descriptions

Name	Reset	Description	Settings
— 15–13	0	Reserved. Write to zero for future compatibility.	
WBOFF 12	1	**Enable WB** Enables/disables the write buffer. When the write buffer is disabled, all write accesses are performed through the BS.	0 Enable write buffer operation. 1 Disable write buffer operation.
— 11–10	0	Reserved. Write to zero for future compatibility.	
WD 9–0	0x3FF	**Watchdog Count** Value for the watch dog count, which is specified in the SC1400 core clocks.	Only counts between 0x80 to 0x3FF are permitted. Values smaller than 0x80 are not allowed.

WBFR WB Software Flush Register ECI_BASE + 0x00

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TYPE	R															
RESET																

A read access to this register causes a software flush of the write buffer.

WBDAR[0–3] WB Data Area Register 0–3 (WBDAR0) ECI_BASE + 0x40
(WBDAR1) ECI_BASE + 0x44
(WBDAR2) ECI_BASE + 0x48
(WBDAR3) ECI_BASE + 0x4C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BASE															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BASE								SZ256	IMM		—	—	EN	—	GBL
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

WBDAR configures the base address and size of a write buffer data area.

Table 4-11. WBDARx Bit Descriptions

Name	Reset	Description	Settings
BASE 31–8	0	**Base Address/Size** Specifies the upper 24-bits of the data area base address and size.	See **Section 4.7.1**, *Write Buffer Data Areas,* on page 4-35.
SZ256 7	0	**Size Indication** Indicates whether the size is 256 bytes or not.	0 Size is other than 256 bytes. 1 Size = 256 bytes.
IMM 6–5	00	**Immediate** Define immediate access to that area. This will force in-order execution of writes. The write can be with or without freeze to the core.	00 Regular write through write buffer. 01 Write immediate. 10 Write immediate with no freeze. 11 Reserved.
— 4–3	0	Reserved. Write to zero for future compatibility.	
EN 2	0	**Enable Operation** Enables/disables this area register operation.	0 Disable operation. 1 Enable operation.
— 1	0	Reserved. Write to zero for future compatibility.	
GBL 0	0	**Global** Determines whether a memory area is non-global or global. This bit is usually used for data cache coherency. **NOTE:** This bit is active on read or write accesses. It is not asserted if EN = 0 or if it does not match the region. It is not affected by the WBCR[WBOFF] bit.	0 Non-global. 1 Global.

GPSCTL General-Purpose System Control Register ECI_BASE + 0x30

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—									ASM1P	—	INDBG	—		XHACK	XHRQ
TYPE	R/W														R	R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPSCTL is for system-level control. This register can be accessed even when the AHB subsystem is shut down or when the AHB clock is turned off and the crossbar switch is powered down. This register is useful for shutting down and restarting the AHB subsystem as described in **Section 11.4.3**, *AHB Subsystem Low-Power Operation*. Under normal operation, the XHRQ bit is cleared, and this bit can be set for low-power operation.

Table 4-12. GPSCTL Bit Descriptions

Name	Reset	Description	Settings
— 15–7	0	Reserved. Write to zero for future compatibility.	
ASM1P 6	0	**ASM1 Priority** Specifies priority of ASM1 bus for M1 accesses.	0 ASM1 has highest priority. 1 ASM1 has lowest priority.
— 5	0	Reserved. Write to zero for future compatibility.	
INDBG 4	0	**In Device Debug Mode** The development tools set this bit when the device first enters Debug mode, which keeps the remainder of the device in Debug mode until the debugging session ends. It ensures that the following modules remain in Device Debug mode: • Software watchdog timer • DMA controller • Interrupt controller The development tools clear INDBG when the device exits a debugging session. This bit can be modified only when the SC1400 is in a Debug processing state. You should not modify this bit. It is reserved for use by the development tools.	0 Device is not necessarily in Device Debug mode. 1 Device remains in Device Debug mode as long as this bit is set.
— **3–2**	0	Reserved. Write to zero for future compatibility.	
XHACK 1	0	**Crossbar Halt Acknowledge** Status bit indicating whether the module is currently halted. Writing to this bit has no effect.	0 Module is not currently halted. 1 Module is currently halted.
XHRQ 0	0	**Crossbar Halt Request** Requests a crossbar halt, which uses the halt capability of the crossbar switch. This bit is automatically reset if a system-level non-maskable interrupt request occurs.	0 No request for halt. 1 Request halt from module.

VR Version Register ECI_BASE + 0x10

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PRVER								ECVER							
TYPE	R															
RESET	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0

VR specifies the versions associated with the MSC711x device.

Table 4-13. VR Bit Descriptions

Name	Reset	Description
PRVER 15–8	0x01	**Process Version** Contains a different number for each process version.
ECVER 7–0	0x42	**Extended Core Version** Contains a different number for each extended core version.

GPR0 General-Purpose Register 0 ECI_BASE + 0x20

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GPRx															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GPRx															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPR0 provides general-purpose I/O signals out of the extended core. These bits are reserved for use by the factory. User applications must not write to these bits.

Table 4-14. GPRx Bit Descriptions

Name	Reset	Description
GPR0 31–0	0	**General-Purpose Bits** Output from the extended core.

4.8.2 ICache Registers

ICache programming refers to all memory accesses that can occur to the memory-mapped registers of the ICache. This section summarizes the different accesses, their functionality in the ICache, and restrictions. The cache is programmed and read through the extended core internal bus from the ECI. It can be accessed as a zero-wait-state slave and always has an *immediate* attribute, thus preventing ICache commands and mode changes from being randomly delayed by the write buffer and taking effect at unexpected times. Through the programming interface, you can set cache modes, send commands to the ICache, and read ICache registers.

4.8.2.1 Commands

All instructions are implemented by writes to a memory-mapped command register, the ICache Command Register (ICCMR). There are two types of ICache commands: run-time commands and Cache Debug Mode commands. The run-time commands are as follows:

- *Flush cache*. Reset all valid bit array and tag array.
- *Flush cache between boundaries*. Clear all valid bits and tags in the ways that are currently inside the LRU boundaries, partial flush.

The Cache Debug mode commands (performed only in Cache Debug mode) are as follows:

- *Clear line*. Clear all valid bits for a one cache line (see **Figure 4-8**) (line = {way[3–0],index[1–0]}, unlike lines for reads). Useful for breakpoint insertion.
- *Initialize status registers*. Perform an initial load to the different cache status registers.

4.8.2.2 Reads

You can read the ICache state and mode information in the following four ICache registers:

- Read the tag array state (Cache Debug mode only): Tag Array Status Register, **page 4-52**.
- Read the LRU State (Cache Debug Mode only): LRU Status Register (LRUSR), **page 4-51**.
- Read the valid bit array state (Cache Debug mode only): Valid Bit Array Status Register (VBASR), **page 4-52**.
- Read the cache control register (cache mode and LRU boundaries): ICache Control Register (ICCR), **page 4-49**.

4.8.2.3 Instruction Regions

The instruction region registers define the base address and size of one of four instruction regions, as well as configuring the attributes for each region. You can access the information for one instruction region in the IRBSRx and IRCRx registers using two 16-bit accesses to access each register individually or with a single 32-bit access to access both registers simultaneously.

When any instruction region parameter changes (for example, prefetch enable, region base address or size, burst size, primary burst size, and so on), the following procedure is required:

1. Disable the region via the IRCRx[EN] bit.
2. Execute 32 NOP instructions.
3. Modify the desired parameter(s) in the IRBSRx or IRCRx register(s).
4. Re-enable the region via the IRCRx[EN] bit.

The code to perform this sequence must not be located in the region where the parameter(s) are to be modified. If parameters are modified after the ICache is enabled, it is recommended that this code be located in M1 memory.

4.8.2.4 ICache Programming Restrictions

Following are the restrictions/issues on ICache programming:

- The steps described in **Section 4.8.2.3** are one of the ICache programming restrictions. These steps are required when any instruction region parameter changes.
- Changes in registers that affect the IFU (IRCR[0–3], IRBSR[0–3]) are valid at the next fetch miss after the write to the register. Changing the registers during prefetch does not affect the prefetch. For example, turning the prefetch off (see the IRCR) does not stop the current prefetch but disables prefetching after the next fetch miss.
- Data newly written to a control register can be read only in the second execution set following the write. There should be at least one execution set between the read and the write of the register so that the new data can be observed.
- Before you enable a disabled cache in any way (either by setting the on bit, resetting the lock or Cache Debug mode bits, or returning the lower LRU boundary to be less or equal to the upper boundary) the code must be preceded and followed by two no operation (NOP) execution sets, as illustrated in the following code example.

```
move.l #$0000f001,d1
nop
nop
move.w d1,($<ICCR_ADDRESS>)
nop
nop
```

- When a run-time command runs parallel with a control register write and if a flush between boundaries runs parallel with a boundary change, the new boundaries are used. However, if any flush command runs parallel with a cache disable (cache off, Cache Debug mode, and so on), the flush is performed.
- Cache run-time commands cause SC1400 core stall penalties.
- Cache run-time commands are performed in lock mode.

- If a flush command is paralleled with a flush between boundaries command, the full cache flush is performed, yet the timing penalty is of the flush between boundaries (the longer penalty of the two).
- Debug commands and read state registers are served in Cache Debug mode only or else they are discarded (an exception flag is raised). One execution must be set at least between turning on the Cache Debug mode bit and the first debug command/debug read. Similarly, there must be at least one execution set between the last debug command and an exit from Cache Debug mode.
- No Cache Debug mode command can be paralleled with an ICCR write, causing the ICache to exit Cache Debug mode.
- Debug commands must be at least one execution set apart from any run-time command.
- The first read status command must be at least one execution set apart from an initialize debug command.

4.8.2.5 ICache Registers

Following is the list of ICache registers and the pages on which they are described. For the value of the base address for this register file, IC_BASE, see **Table 5-1** on page 5-3:

- Instruction Region Base/Size Register (IRBSR[0–3]), **page 4-47**.
- Instruction Region Configuration Register (IRCR[0–3]), **page 4-48**.
- ICache Control Register (ICCR), **page 4-49**.
- ICache Command Register (ICCMR), **page 4-50**.
- LRU Status Register (LRUSR), **page 4-51**.
- Tag Array Status Register (TASR), **page 4-52**.
- Valid Bit Array Status Register (VBASR), **page 4-52**.

IRBSR[0–3] Instruction Region Base/Size Register (IRBSR0) IC_BASE + 0x82
(IRBSR1) IC_BASE + 0x86
(IRBSR2) IC_BASE + 0x8A
(IRBSR3) IC_BASE + 0x8E

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RBAS															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

IRBSR configures the base address and size of a cacheable region. Addresses between 0x00000000–0x00FFFFFF are always defined as not cacheable. When the size of a region is 64 KB, set the IRCR 64KB bit. Writes to this register when an IFU access is in progress freeze the SC1400 core until all accesses for the current cache miss are completed.

Table 4-15. IRBSR Bit Descriptions

Name	Reset	Description
RBAS 15–0	0x0080	**Region Base Address and Size** Specifies the base address and size for the area defining the cacheable area (see **Section 4.7.2**, *Instruction Cacheable Area,* on page 4-37).

IRCR[0–3] Instruction Region Configuration Register (IRCR0) IC_BASE + 0x80
(IRCR1) IC_BASE + 0x84
(IRCR2) IC_BASE + 0x88
(IRCR3) IC_BASE + 0x8C

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	64KB	—	—	—	—	EN	—	—	—	—	—	PFE	—	SIZE		
TYPE	R/W															
RESET	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0

IRCR configures the attributes for a region of cacheable memory. Writes to this register when an IFU access is in progress freeze the SC1400 core until all accesses for the current cache miss are completed.

Table 4-16. IRCR Bit Descriptions

Name	Reset	Description	Settings
64KB 15	0	**Size Indication** Sets the size to the 64 KB minimum or sets it to a different size.	0 Size is other than 64 KB. 1 Size is 64 KB.
— 14–11	0	Reserved. Write to zero for future compatibility.	
EN 10	1	**Enable Area Operation** Enables/disables caching for the region.	0 Caching disabled. 1 Caching enabled.
— 9–5	0	Reserved. Write to zero for future compatibility.	
PFE 4	1	**Prefetch Enable** Enables/disables Prefetch mode.	0 Prefetch mode enabled. 1 Prefetch mode disabled.
— 3	0	Reserved. Write to zero for future compatibility.	

Table 4-16. IRCR Bit Descriptions (Continued)

Name	Reset	Description	Settings
SIZE 2–0	000	**Burst and Primary Set Size** Sets the burst size as well as the size of the number of bursts in the primary set.	

SIZE[2–0]	Primary Set Size	Burst Size
000	1	1
001	2	1
010	4	1
101	4	4

ICCR ICache Control Register IC_BASE + 0x00

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UB				LB				—	—	—	—	DM	—	LM	ON
TYPE	R/W															
RESET	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1

ICCR configures the ICache modes and specifies upper and lower boundaries for locking and flushing. The ICache operates in the following modes:

- *On/Off*. When the ICache is turned off, all caching-related machines and the command and status mechanisms are OFF (clocks turned OFF). Only the control register periphery remains ON.
- *Cache Debug mode*. Enables the ICache non-real-time debug commands. All ICache updates are disabled except the flush commands.
- *Lock mode*. Locks data in the ICache with no updates permitted (thrashing/new valid bit setting). Hits are served in Lock mode, so all tag match LRU updates can take effect. All commands work in Lock mode, including flushes. The cache also enters Lock mode if the upper boundary has a value that is less than the value of the lower boundary. That is, when there is an attempt to read the register, the Lock mode bit is set. The ICache does not enter Lock mode if it is OFF or set to Cache Debug mode (mode bit is read as 0).

Table 4-17. ICCR Bit Descriptions

Name	Reset	Description	Value
UB 15–12	0xF	**Upper Boundary Value** Selects the upper boundary (way number) for LRU consideration.	

Table 4-17. ICCR Bit Descriptions (Continued)

Name	Reset	Description	Value
LB 11–8	0	**Lower Boundary Value** Selects the lower boundary (way number) for LRU consideration. If LB > UB, the cache is locked. Values outside the range LB to UB are considered frozen.	
— 7–4	0	Reserved. Write to zero for future compatibility.	
DM 3	0	**Cache Debug Mode** Specifies whether the ICache is in Normal or Debug mode.This bit reads as a 1 only when both the DM and ON bits are programmed to a value of 1. Cache Debug mode enables the cache non-real-time debug commands. All ICache updates are disabled in Cache Debug mode (except the flush commands). The ICache does not enter Cache Debug mode if it is set to off (mode bit is read as 0).	0 Cache in Normal mode. 1 Cache in Debug mode.
— 2	0	Reserved. Write to zero for future compatibility.	
LM 1	0	**Cache Lock Mode** Specifies whether the ICache is locked. This bit reads as a 1 only when the ON bit is programmed to a 1 and the cache is fully locked[1] and the DM is programmed with a value of 0. Lock mode locks data in the ICache, with no updates permitted (thrashing/new valid bit setting). Hits are served in lock mode so all tag match LRU updates can take effect. All commands work in Lock mode, including flushes. The cache also enters lock mode if the upper boundary is set to be less than the lower boundary. When there is an attempt to read the register, the lock mode bit is on. The ICache does not enter Lock mode if it is off or set to Cache Debug mode (mode bit is read as 0).	0 Cache not locked. 1 Cache locked.
ON 0	1	**On/Off Bit** Enables/disables the ICache.	0 Cache disabled. 1 Cache enabled.
Note: The cache is fully locked when all ways are locked. The ICache is fully locked either by programming DM bit with a value of 1 or programming the LB bits with a value larger than that of the UB bits.			

ICCMR ICache Command Register IC_BASE + 0x04

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	C[3–0]				—	—	—	—	—	—	DA[5–0]					
TYPE	W															
RESET																

ICCMR flushes portions of the cache or the entire cache and clears a line of the cache. Writes to this register when an IFU access is in progress freeze the SC1400 core until all accesses for the current cache miss are completed. This ensures that the flush completes before the SC1400 core resumes execution with the instructions that follow.

Table 4-18. ICCMR Bit Descriptions

Name	Reset	Description
C 15–12		**Commands Bits:** 0000: Flush cache 0001: Flush cache between boundaries 1000: Initialize state registers 1001: Clear line (Line to clear in the DA bits) Other Combinations reserved
— 11–6		Reserved. Write to zero for future compatibility.
DA 5–0		**Destination Address Field** Defines a line to clear (00000 = Way 0, Index 0 to 11111 = Way 15, Index 3)

LRUSR LRU Status Register IC_BASE + 0x10

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LS[15–0]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LRUSR reads the LRU status of the ICache.

Table 4-19. LRUSR Bit Descriptions

Name	Reset	Description
LS 15–0	0x0000	**LRU Status Register Contents** An LRU status bit for each line that shares an index number. A register value is stored for each index. The individual values are accessed via a sequential read. The first read by the SC1400 core returns the value for Index = 0x0. A second read returns the value for Index = 0x1. A third read returns the value for Index = 0x2. A fourth read returns the value for Index = 0x3. For each bit, a 0 indicates that the line is not the LRU for the specified index and a 1 indicates that the line is the LRU for that index.

TASR Tag Array Status Register IC_BASE + 0x14

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TS[15–0]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TASR reads the tag array status of the ICache.

Table 4-20. TASR Bit Descriptions

Name	Reset	Description
TS 15–0	0x0000	**Tag State Register** A TAG status bit for each line that shares an index number. A register value is stored for each index. The individual values are accessed via a sequential read. The first read by the SC1400 core returns the value for Index = 0x0. A second read returns the value for Index = 0x1. A third read returns the value for Index = 0x2. A fourth read returns the value for Index = 0x3. For each bit, a 0 indicates that the TAG is not in use. A 1 indicates that a TAG value exists for the cache line.

VBASR Valid Bit Array Status Register IC_BASE + 0x18

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	VS[15–0]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VBASR reads the VALID bit array status of the ICache.

Table 4-21. VBASR Bit Descriptions

Name	Reset	Description
VS 15–0	0x0000	**Valid Bit Array Line Content** The array line content for each line and bit position. The individual values are accessed via a sequential read. The first read by the SC1400 core returns the value for Index = 0x0, position 0. A second read returns the value for Index = 0x0, position 1. A third read returns the value for Index = 0x0, position 2, and so forth for each index up to position 15, and then for each index and position up to Index = 0x3, position 15. For each bit, a 0 indicates that memory location is not cached. A 1 indicates that the memory location is cached.

Memory Map 5

The MSC711x device uses a unified system memory and device map. Internal module locations are fixed. and most registers and memory blocks have a single address region within the map. The exception is the M1 memory and the HDI16 module, which have two addressable regions. The MSC711x devices have several different bus controllers within the system. Along with the SC1400 core, these devices address specified regions within the memory map, shown in **Figure 5-1**.

The address space for the M1 memory depends on which master is accessing it. When the SC1400 core access M1 memory, the address space begins at 0x00000000. When AHB masters access M1 memory, the address space begins at 0x01800000. Address regions within the memory map are categorized as follows:

- *SC1400 core internal address space.* The SC1400 core accesses M1 memory and the OCE10 registers using the core P, XA, and XB buses.
- The extended core address space on the master port, both within and outside the system is as follows:
  - *Extended core internal address space*. The SC1400 core accesses the registers of the extended core interface (ECI), instruction cache (ICache), and interrupt controller through an internal bus that taps off the AMEC bus.
  - *Extended core external address space*. The SC1400 core accesses external resources through the AMEC bus, which passes through the crossbar switch and provides access to all system resources.
- The address space of master controller ports is organized as follows:
  - *AMIC address space*. The ICache fetch unit services cache misses and accesses to non-cacheable regions of memory by accesses to M2 memory, external memory, and the boot ROM through the crossbar switch via the AMIC bus.
  - *AMDMA address space*. The DMA controller accesses external resources through the AMDMA bus, which passes through the crossbar switch and provides access to all system resources. The DMA controller can also access internal M1 memory at addresses available to the DMA controller beginning at 0x01800000.
  - *AMENT address space*. The dedicated DMA located within the Ethernet MAC can access resources outside the platform through the AMENT bus which passes through the crossbar switch and provides access to all system resources. It can also access the M1 memory at addresses available to the crossbar ports beginning at 0x01800000.

- The address space of the slave ports is organized as follows:
  - *ASM1 address space*. Accesses M1 memory through the crossbar switch via the ASM1 bus. Although this memory is located within the extended core, it is accessed by the crossbar switch in a portion of the address map located outside the extended core's address space.
  - *ASM2 address space*. Accesses M2 memory and the Boot ROM through the crossbar switch via the ASM2 bus.
  - *ASEMI address space*. Accesses external memory through the crossbar switch via the ASEMI bus.
  - *ASTH address space*. Accesses the high speed ports of the TDM and HDI16 peripherals through the crossbar switch via the ASTH bus.
  - *ASAPB address space*. Accesses peripherals located on the APB through the crossbar switch via the ASAPB bus.
  - *ASSB address space*. Accesses peripherals located on the IPBus through the crossbar switch via the ASSB bus.

Note: Although they are located in the APB address space, the interrupt control and interrupt priority registers are accessed through the ASAPB bus directly. They are not accessed through the APB.

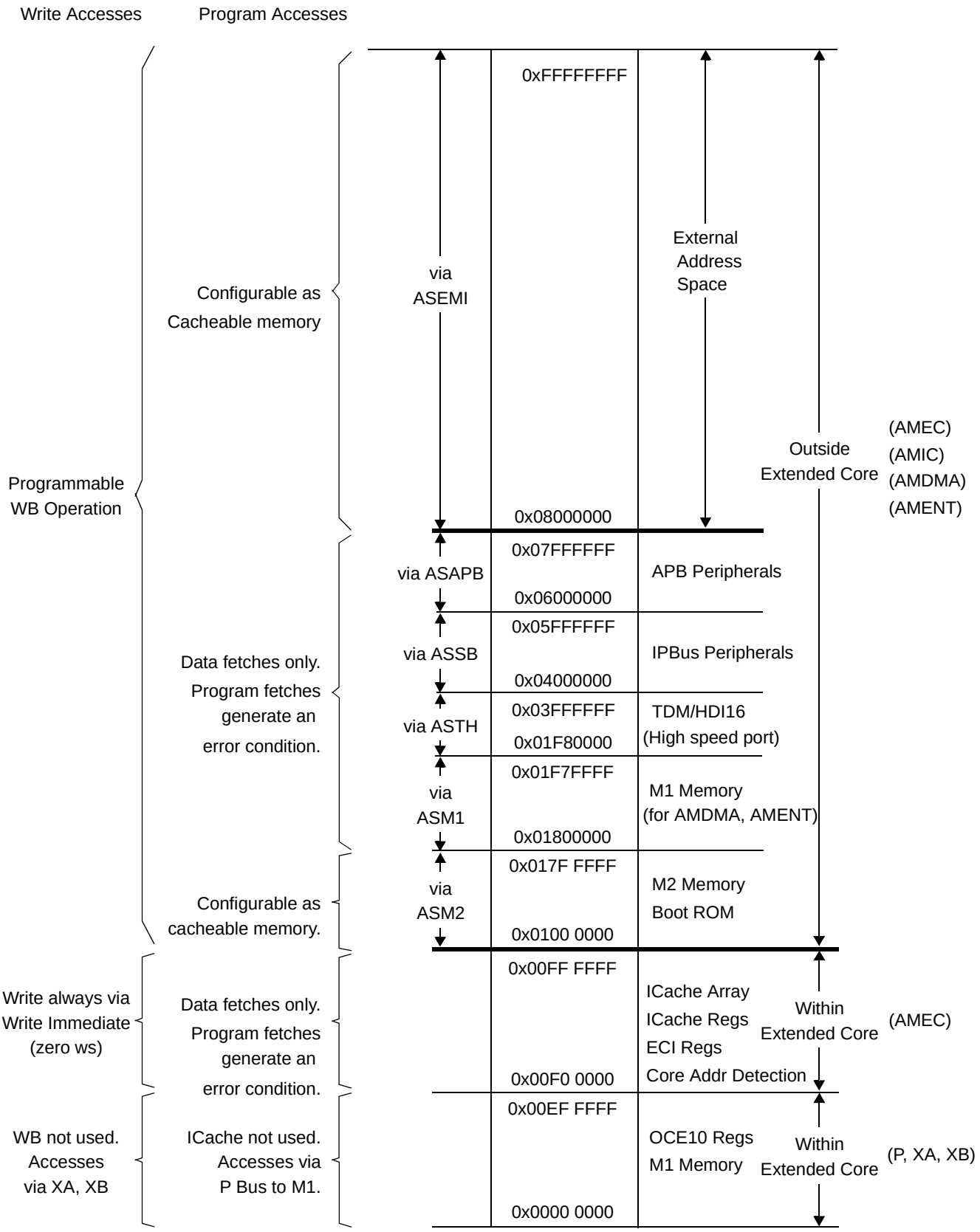


Figure 5-1. MSC711x Memory Map

5.1 Register Base Addresses

The MSC711x module registers and memory blocks are all defined with respect to the base addresses listed in **Table 5-1**. There is no ENETAHB_BASE because the Ethernet data (stored in internal FIFOs) is accessed directly by the internal DMA controller and transferred out to the system.

Table 5-1. Summary — Base Addresses for MSC711x Register Files

Location in Memory Map	Base Name	Description	Address
Extended Core	EONCE_BASE	EOnCE Registers	0x00EF F000
	ECI_BASE	Extended Core Interface Registers	0x00F0 0000
	IC_BASE	Instruction Cache Registers	0x00F0 0100
	CAD_BASE	Core Address Detection Registers	0x00F0 0300
	ICARRAY_BASE	Instruction Cache Array Contents	0x00F2 0000
IPBus Peripherals	—	(Reserved)	0x0400 0000
	TMRA_BASE	Timer A Registers (1st Quad Timer Module)	0x0400 1000
	TMRB_BASE	Timer B Registers (2nd Quad Timer Module)	0x0400 2000
	XBAR_BASE	Crossbar Switch Registers	0x0400 3000
	DMA_BASE	DMA Registers	0x0400 4000
	—	(Additional space reserved for DMA Registers)	0x0400 5000
	ENET_BASE	Ethernet MAC Registers	0x0400 6000
	—	(Reserved)	0x0400 7000
	DDR_BASE	DDR Memory Controller Registers	0x0400 8000
	I2C_BASE	I^2C Registers	0x0400 9000
	BTM_BASE	Bus Timeout Monitor Registers	0x0400 A000
	EV_BASE	Event Port Registers	0x0400 B000
	CLK_BASE	Clock Control and Reset Registers	0x0400 C000
	PAD_BASE	Peripheral Address Detection Registers	0x0400 D000
	MCIF_BASE	Memory Controller Interface Registers	0x0400 E000

Table 5-1. Summary — Base Addresses for MSC711x Register Files

Location in Memory Map	Base Name	Description	Address
APB Peripherals	—	(Reserved)	0x0600 0000
	SWT_BASE	Software Watchdog Timer Registers	0x0600 1000
	—	(Reserved)	0x0600 2000
	—	(Reserved)	0x0600 3000
	TDM0_BASE	TDM0 Registers — Accessed via APB	0x0600 4000
	TDM1_BASE	TDM1 Registers — Accessed via APB	0x0600 5000
	TDM2_BASE	TDM2 Registers — Accessed via APB (for devices which have this TDM)	0x0600 6000
	HDI16_BASE	HDI16 Registers — Accessed via APB	0x0600 7000
	UART_BASE	UART Registers	0x0600 8000
	GPIO_BASE	GPIO Registers	0x0600 9000
	ICTL_BASE	Interrupt Controller Registers	0x0600 A000
	IPL_BASE	Interrupt Priority Level Registers	0x0600 B000
Peripherals accessible on the ASTH (AHB-Lite) Bus	TDM0AHB_BASE	TDM0 Registers — Accessed via ASTH bus	0x01F8 4000
	TDM1AHB_BASE	TDM1 Registers — Accessed via ASTH bus	0x01F8 5000
	TDM2AHB_BASE	TDM2 Registers — Accessed via ASTH bus (for devices which have this TDM)	0x01F8 6000
	HDI16AHB_BASE	HDI16 Registers — Accessed via ASTH bus	0x01F8 7000

5.2 Memory-Mapped Registers

Table 5-3 summarizes the MSC711x memory map. Refer to the individual device technical data sheet or product brief to determine the peripherals and memory supported by a specific device. If an individual device does not include a specified peripheral or memory, the respective memory/device address is reserved in that device.

Table 5-2. MSC711x Detailed Memory Map

Address	Register/Memory Region	Acronym	Size in Bytes
0x00000000–0x0000FFFF	M1 Memory Lowest 64 KB for core access	M1MEML	64K
0x00010000–0x0002FFFF	M1 Memory Highest 128 KB for core access	M1MEMH	128K
0x00030000–0x00EFFDFF	Reserved		
0x00EFF000	Emulator Status Register	ESR	4
0x00EFF004	Emulator Monitor and Control Register	EMCR	4
0x00EFF008	Emulator Receive Register (LSBs)	ERCV	4
0x00EFF00C	Emulator Receive Register (MSBs)	ERCV	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x00EFF010	Emulator Transmit Register (LSBs)	ETRSMTL	4
0x00EFF014	Emulator Transmit Register (MSBs)	ETRSMTH	4
0x00EFF018	EE Signals Control Register	EE_CTRL	2
0x00EFF01C	Exception PC Register	PC_EXCP	4
0x00EFF020	PC of Next Execution Set	PC_NEXT	4
0x00EFF024	PC of Last Execution Set	PC_LAST	4
0x00EFF028	PC Breakpoint Detection Register	PC_DETECT	4
0x00EFF02C–0x00EFF039	Reserved		
0x00EFF040	EDCA0 Control Register	EDCA0_CTRL	2
0x00EFF044	EDCA1 Control Register	EDCA1_CTRL	2
0x00EFF048	EDCA2 Control Register	EDCA2_CTRL	2
0x00EFF04C	EDCA3 Control Register	EDCA3_CTRL	2
0x00EFF050	EDCA4 Control Register	EDCA4_CTRL	2
0x00EFF054	EDCA5 Control Register	EDCA5_CTRL	2
0x00EFF058–0x00EFF05F	Reserved		
0x00EFF060	EDCA0 Reference Value A	EDCA0_REFA	4
0x00EFF064	EDCA1 Reference Value A	EDCA1_REFA	4
0x00EFF068	EDCA2 Reference Value A	EDCA2_REFA	4
0x00EFF06C	EDCA3 Reference Value A	EDCA3_REFA	4
0x00EFF070	EDCA4 Reference Value A	EDCA4_REFA	4
0x00EFF074	EDCA5 Reference Value A	EDCA5_REFA	4
0x00EFF078–0x00EFF07F	Reserved		
0x00EFF080	EDCA0 Reference Value B	EDCA0_REFB	4
0x00EFF084	EDCA1 Reference Value B	EDCA1_REFB	4
0x00EFF088	EDCA2 Reference Value B	EDCA2_REFB	4
0x00EFF08C	EDCA3 Reference Value B	EDCA3_REFB	4
0x00EFF090	EDCA4 Reference Value B	EDCA4_REFB	4
0x00EFF094	EDCA5 Reference Value B	EDCA5_REFB	4
0x00EFF098–0x00EFF0BF	Reserved		
0x00EFF0C0	EDCA0 Mask Value	EDCA0_MASK	4
0x00EFF0C4	EDCA1 Mask Value	EDCA1_MASK	4
0x00EFF0C8	EDCA2 Mask Value	EDCA2_MASK	4
0x00EFF0CC	EDCA3 Mask Value	EDCA3_MASK	4
0x00EFF0D0	EDCA4 Mask Value	EDCA4_MASK	4
0x00EFF0D4	EDCA5 Mask Value	EDCA5_MASK	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x00EFF0D8–0x00EFF0DF	Reserved		
0x00EFF0E0	Event Detection Channel n Register	EDCD_CTRL	2
0x00EFF0E4	EDCD Reference Value Register	EDCD_REF	4
0x00EFF0E8	EDCD Mask Value Register	EDCD_MASK	4
0x00EFF0EC–0x00EFF0FF	Reserved		
0x00EFF100	Event Counter Control Register	ECNT_CTRL	2
0x00EFF104	Event Counter Value	ECNT_VAL	4
0x00EFF108	Event Counter Extension Value	ECNT_EXT	4
0x00EFF10C–0x00EFF11F	Reserved		
0x00EFF120	Event Selector Control Register	ESEL_CTRL	1
0x00EFF124	Event Selector Mask Debug Mode	ESEL_DM	2
0x00EFF128	Event Selector Mask Debug Exception	ESEL_DI	2
0x00EFF12C–0x00EFF12F	Reserved		
0x00EFF130	Event Selector Mask Enable Trace Buffer	ESEL_ETB	2
0x00EFF134	Event Selector Mask Disable Trace Buffer	ESEL_DTB	2
0x00EFF138–0x00EFF13F	Reserved		
0x00EFF140	Trace Buffer Control Register	TB_CTRL	1
0x00EFF144	Trace Buffer Read Pointer	TB_RD	2
0x00EFF148	Trace Buffer Write Pointer	TB_WR	2
0x00EFF14C	Trace Buffer	TB_BUFF	4
0x00EFF150–0x00EFF1F7	Reserved		
0x00EFF1F8	Core Command Register	CORE_CMD	6
0x00EFF1FE–0x00EFFFFF	Reserved		
0x00F00000	WB Software Flush Register	WBFR	2
0x00F00002–0x00F00003	Reserved		
0x00F00004	WB Control Register	WBCR	2
0x00F00006–0x00F0000F	Reserved		
0x00F00010	Version Register	VR	2
0x00F00012–0x00F0001F	Reserved		
0x00F00020	General-Purpose Register 0	GPR0	4
0x00F00024–0x00F0002F	Reserved		
0x00F00030	General-Purpose System Control Register	GPSCTL	2
0x00F00032–0x00F0003F	Reserved		
0x00F00040	WB Data Area 0 Register	WBDAR0	4
0x00F00044	WB Data Area 1 Register	WBDAR1	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x00F00048	WB Data Area 2 Register	WBDAR2	4
0x00F0004C	WB Data Area 3 Register	WBDAR3	4
0x00F00050–0x00F000FF	Reserved		
0x00F00100	ICache Control Register	ICCR	2
0x00F00102–0x00F00103	Reserved		
0x00F00104	ICache Command Register	ICCMR	2
0x00F00106–0x00F0010F	Reserved		
0x00F00110	LRU Status Register	LRUSR	2
0x00F00112–0x00F00113	Reserved		
0x00F00114	Tag Array Status Register	TASR	2
0x00F00116–0x00F00117	Reserved		
0x00F00118	Valid Bit Array Status Register	VBASR	2
0x00F0011A–0x00F0017F	Reserved		
0x00F00180	Instruction Region 0 Configuration Register	IRCR0	2
0x00F00182	Instruction Region 0 Base/Size Register	IRBSR0	2
0x00F00184	Instruction Region 1 Configuration Register	IRCR1	2
0x00F00186	Instruction Region 1 Base/Size Register	IRBSR1	2
0x00F00188	Instruction Region 2 Configuration Register	IRCR2	2
0x00F0018A	Instruction Region 2 Base/Size Register	IRBSR2	2
0x00F0018C	Instruction Region 3 Configuration Register	IRCR3	2
0x00F0018E	Instruction Region 3 Base/Size Register	IRBSR3	2
0x00F00190–0x00F002FF	Reserved		
0x00F00300	Extended Core Address Detection Control Register 0	CADCTL0	4
0x00F00304	Extended Core Address Detection Control Register 1	CADCTL1	4
0x00F00308	Reserved		
0x00F0030C	Extended Core Address Detection Status Register	CADSR	4
0x00F00310	Reserved		
0x00F00314	Extended Core Address Detection PAB Lower bound Register 0	CADLWRP0	4
0x00F00318	Extended Core Address Detection PAB Lower bound Register 1	CADLWRP1	4
0x00F0031C	Extended Core Address Detection PAB Lower bound Register 2	CADLWRP2	4
0x00F00320	Extended Core Address Detection PAB Lower bound Register 3	CADLWRP3	4
0x00F00324–0x00F0032B	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x00F0032C	Extended Core Address Detection PAB Upper bound Register 0	CADUPRP0	4
0x00F00330	Extended Core Address Detection PAB Upper bound Register 1	CADUPRP1	4
0x00F00334	Extended Core Address Detection PAB Upper bound Register 2	CADUPRP2	4
0x00F00338	Extended Core Address Detection PAB Upper bound Register 3	CADUPRP3	4
0x00F0033C–0x00F00343	Reserved		4
0x00F00344	Extended Core Address Detection XAB Lower Bound Register 0	CADLWRX0	4
0x00F00348	Extended Core Address Detection XAB Lower Bound Register 1	CADLWRX1	4
0x00F0034C	Extended Core Address Detection XAB Lower Bound Register 2	CADLWRX2	4
0x00F00350	Extended Core Address Detection XAB Lower Bound Register 3	CADLWRX3	4
0x00F00354–0x00F0035B	Reserved		
0x00F0035C	Extended Core Address Detection XAB Upper Bound Register 0	CADUPRX0	4
0x00F00360	Extended Core Address Detection XAB Upper Bound Register 1	CADUPRX1	4
0x00F00364	Extended Core Address Detection XAB Upper Bound Register 2	CADUPRX2	4
0x00F00368	Extended Core Address Detection XAB Upper Bound Register 3	CADUPRX3	4
0x00F0036C–0x00F00373	Reserved		
0x00F00374	Extended Core Address Detection Capture Program Address Register	CADCPTP	4
0x00F00378	Extended Core Address Detection Capture XA Address Register	CADCPTXA	4
0x00F0037C	Extended Core Address Detection Capture XB Address Register	CADCPTXB	4
0x00F00380–0x00F1FFFF	Reserved		
0x00F20000–0x00F23FFF	ICache Array Contents	IARRAY	16 K
0x00F24000–0x00FFFFFF	Reserved		
0x01000000–0x0102FFFF	M2 Memory	M2MEM	192 K
0x01030000–0x013FFFFF	Reserved		
0x01400000–0x01401FFF	Boot ROM (VBA points to 01401000–0xinitial ISR address)	BOOTROM	4 K
0x01402000–0x017FFFFF	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x01800000–0x0180FFFF	M1 Memory Lowest 64 KB from outside core	M1MEML	64 K
0x01810000–0x0182FFFF	M1 Memory Upper 128 KB from outside core	M1MEMH	128 K
0x1830000–0x01F83FFF	Reserved		
0x01F84000	TDM0 Receive Data Register	TDM0RDR	8
0x01F84008	TDM0 Transmit Data Register	TDM0TDR	8
0x01F84010–0x01F84FFF	Reserved		
0x01F85000	TDM1 Receive Data Register	TDM1RDR	8
0x01F85008	TDM1 Transmit Data Register	TDM1TDR	8
01F85010–0x01F85FFF	Reserved		
0x01F86000	TDM2 Receive Data Register	TDM2RDR	8
0x01F86008	TDM2 Transmit Data Register	TDM2TDR	8
0x01F86010–0x04000FFF	Reserved		
0x04001000	Timer Channel 0 Compare Register 1	TMR0CMP1	2
0x04001002–0x04001003	Reserved		
0x04001004	Timer Channel 0 Compare Register 2	TMR0CMP2	2
0x04001006–0x04001007	Reserved		
0x04001008	Timer Channel 0 Capture Register	TMR0CAP	2
0x04001002–0x04001003	Reserved		
0x0400100C	Timer Channel 0 Load Register	TMR0LOAD	2
0x04001002–0x04001003	Reserved		
0x04001010	Timer Channel 0 Hold Register	TMR0HOLD	2
0x04001002–0x04001003	Reserved		
0x04001014	Timer Channel 0 Counter Register	TMR0CNTR	2
0x04001002–0x04001003	Reserved		
0x04001018	Timer Channel 0 Control Register	TMR0CTL	2
0x04001002–0x04001003	Reserved		
0x0400101C	Timer Channel 0 Status and Control Register	TMR0SCTL	2
0x04001002–0x04001003	Reserved		
0x04001020	Timer Channel 0 Compare Load Register 1	TMR0CMPLD1	2
0x04001002–0x04001003	Reserved		
0x04001024	Timer Channel 0 Compare Load Register 2	TMR0CMPLD2	2
0x04001002–0x04001003	Reserved		
0x04001028	Timer Channel 0 Comparator Status and Control Register	TMR0COMSC	2
0x0400102A–0x0400103F	Reserved		
0x04001040	Timer Channel 1 Compare Register 1	TMR1CMP1	2

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04001002–0x04001003	Reserved		
0x04001044	Timer Channel 1 Compare Register 2	TMR1CMP2	2
0x04001002–0x04001003	Reserved		
0x04001048	Timer Channel 1 Capture Register	TMR1CAP	2
0x04001002–0x04001003	Reserved		
0x0400104C	Timer Channel 1 Load Register	TMR1LOAD	2
0x04001002–0x04001003	Reserved		
0x04001050	Timer Channel 1 Hold Register	TMR1HOLD	2
0x04001002–0x04001003	Reserved		
0x04001054	Timer Channel 1 Counter Register	TMR1CNTR	2
0x04001002–0x04001003	Reserved		
0x04001058	Timer Channel 1 Control Register	TMR1CTL	2
0x04001002–0x04001003	Reserved		
0x0400105C	Timer Channel 1 Status and Control Register	TMR1SCTL	2
0x04001002–0x04001003	Reserved		
0x04001060	Timer Channel 1 Compare 1 Load Register	TMR1CMPLD1	2
0x04001002–0x04001003	Reserved		
0x04001064	Timer Channel 1 Compare 2 Load Register	TMR1CMPLD2	2
0x04001002–0x04001003	Reserved		
0x04001068	Timer Channel 1 Comparator Status and Control Register	TMR1COMSC	2
0x0400106A–0x0400107F	Reserved		
0x04001080	Timer Channel 2 Compare Register 1	TMR2CMP1	2
0x04001002–0x04001003	Reserved		
0x04001084	Timer Channel 2 Compare Register 2	TMR2CMP2	2
0x04001002–0x04001003	Reserved		
0x04001088	Timer Channel 2 Capture Register	TMR2CAP	2
0x04001002–0x04001003	Reserved		
0x0400108C	Timer Channel 2 Load Register	TMR2LOAD	2
0x04001002–0x04001003	Reserved		
0x04001090	Timer Channel 2 Hold Register	TMR2HOLD	2
0x04001002–0x04001003	Reserved		
0x04001094	Timer Channel 2 Counter Register	TMR2CNTR	2
0x04001002–0x04001003	Reserved		
0x04001098	Timer Channel 2 Control Register	TMR2CTL	2
0x04001002–0x04001003	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0400109C	Timer Channel 2 Status and Control Register	TMR2SCTL	2
0x04001002–0x04001003	Reserved		
0x040010A0	Timer Channel 2 Compare 1 Load Register	TMR2CMPLD1	2
0x04001002–0x04001003	Reserved		
0x040010A4	Timer Channel 2 Compare 2 Load Register	TMR2CMPLD2	2
0x04001002–0x04001003	Reserved		
0x040010A8	Timer Channel 2 Comparator Status and Control Register	TMR2COMSC	2
0x040010AA–0x040010BF	Reserved		
0x040010C0	Timer Channel 3 Compare Register 1	TMR3CMP1	2
0x04001002–0x04001003	Reserved		
0x040010C4	Timer Channel 3 Compare Register 2	TMR3CMP2	2
0x04001002–0x04001003	Reserved		
0x040010C8	Timer Channel 3 Capture Register	TMR3CAP	2
0x04001002–0x04001003	Reserved		
0x040010CC	Timer Channel 3 Load Register	TMR3LOAD	2
0x04001002–0x04001003	Reserved		
0x040010D0	Timer Channel 3 Hold Register	TMR3HOLD	2
0x04001002–0x04001003	Reserved		
0x040010D4	Timer Channel 3 Counter Register	TMR3CNTR	2
0x04001002–0x04001003	Reserved		
0x040010D8	Timer Channel 3 Control Register	TMR3CTL	2
0x04001002–0x04001003	Reserved		
0x040010DC	Timer Channel 3 Status and Control Register	TMR3SCTL	2
0x04001002–0x04001003	Reserved		
0x040010E0	Timer Channel 3 Compare 1 Load Register	TMR3CMPLD1	2
0x04001002–0x04001003	Reserved		
0x040010E4	Timer Channel 3 Compare 2 Load Register	TMR3CMPLD2	2
0x04001002–0x04001003	Reserved		
0x040010E8	Timer Channel 3 Comparator Status and Control Register	TMR3COMSC	2
0x040010EA–0x040010FF	Reserved		
0x04002000	Timer Channel 4 Compare Register 1	TMR4CMP1	2
0x04001002–0x04001003	Reserved		
0x04002004	Timer Channel 4 Compare Register 2	TMR4CMP2	2
0x04001002–0x04001003	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04002008	Timer Channel 4 Capture Register	TMR4CAP	2
0x04001002–0x04001003	Reserved		
0x0400200C	Timer Channel 4 Load Register	TMR4LOAD	2
0x04001002–0x04001003	Reserved		
0x04002010	Timer Channel 4 Hold Register	TMR4HOLD	2
0x04001002–0x04001003	Reserved		
0x04002014	Timer Channel 4 Counter Register	TMR4CNTR	2
0x04001002–0x04001003	Reserved		
0x04002018	Timer Channel 4 Control Register	TMR4CTL	2
0x04001002–0x04001003	Reserved		
0x0400201C	Timer Channel 4 Status and Control Register	TMR4SCTL	2
0x04001002–0x04001003	Reserved		
0x04002020	Timer Channel 4 Compare 1 Load Register	TMR4CMPLD1	2
0x04001002–0x04001003	Reserved		
0x04002024	Timer Channel 4 Compare 2 Load Register	TMR4CMPLD2	2
0x04001002–0x04001003	Reserved		
0x04002028	Timer Channel 4 Comparator Status and Control Register	TMR4COMSC	2
0x0400202A–0x0400203F	Reserved		
0x04002040	Timer Channel 5 Compare Register 1	TMR5CMP1	2
0x04001002–0x04001003	Reserved		
0x04002044	Timer Channel 5 Compare Register 2	TMR5CMP2	2
0x04001002–0x04001003	Reserved		
0x04002048	Timer Channel 5 Capture Register	TMR5CAP	2
0x04001002–0x04001003	Reserved		
0x0400204C	Timer Channel 5 Load Register	TMR5LOAD	2
0x04001002–0x04001003	Reserved		
0x04002050	Timer Channel 5 Hold Register	TMR5HOLD	2
0x04001002–0x04001003	Reserved		
0x04002054	Timer Channel 5 Counter Register	TMR5CNTR	2
0x04001002–0x04001003	Reserved		
0x04002058	Timer Channel 5 Control Register	TMR5CTL	2
0x04001002–0x04001003	Reserved		
0x0400205C	Timer Channel 5 Status and Control Register	TMR5SCTL	2
0x04001002–0x04001003	Reserved		
0x04002060	Timer Channel 5 Compare 1 Load Register	TMR5CMPLD1	2

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04001002–0x04001003	Reserved		
0x04002064	Timer Channel 5 Compare 2 Load Register	TMR5CMPLD2	2
0x04001002–0x04001003	Reserved		
0x04002068	Timer Channel 5 Comparator Status and Control Register	TMR5COMSC	2
0x0400206A–0x0400207F	Reserved		
0x04002080	Timer Channel 6 Compare Register 1	TMR6CMP1	2
0x04001002–0x04001003	Reserved		
0x04002084	Timer Channel 6 Compare Register 2	TMR6CMP2	2
0x04001002–0x04001003	Reserved		
0x04002088	Timer Channel 6 Capture Register	TMR6CAP	2
0x04001002–0x04001003	Reserved		
0x0400208C	Timer Channel 6 Load Register	TMR6LOAD	2
0x04001002–0x04001003	Reserved		
0x04002090	Timer Channel 6 Hold Register	TMR6HOLD	2
0x04001002–0x04001003	Reserved		
0x04002094	Timer Channel 6 Counter Register	TMR6_CNTR	2
0x04001002–0x04001003	Reserved		
0x04002098	Timer Channel 6 Control Register	TMR6CTL	2
0x04001002–0x04001003	Reserved		
0x0400209C	Timer Channel 6 Status and Control Register	TMR6SCTL	2
0x04001002–0x04001003	Reserved		
0x040020A0	Timer Channel 6 Compare 1 Load Register	TMR6CMPLD1	2
0x04001002–0x04001003	Reserved		
0x040020A4	Timer Channel 6 Compare 2 Load Register	TMR6CMPLD2	2
0x04001002–0x04001003	Reserved		
0x040020A8	Timer Channel 6 Comparator Status and Control Register	TMR6COMSC	2
0x040020AA–0x040020BF	Reserved		
0x040020C0	Timer Channel 7 Compare Register 1	TMR7CMP1	2
0x04001002–0x04001003	Reserved		
0x040020C4	Timer Channel 7 Compare Register 2	TMR7CMP2	2
0x04001002–0x04001003	Reserved		
0x040020C8	Timer Channel 7 Capture Register	TMR7CAP	2
0x04001002–0x04001003	Reserved		
0x040020CC	Timer Channel 7 Load Register	TMR7LOAD	2

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04001002–0x04001003	Reserved		
0x040020D0	Timer Channel 7 Hold Register	TMR7HOLD	2
0x04001002–0x04001003	Reserved		
0x040020D4	Timer Channel 7 Counter Register	TMR7CNTR	2
0x04001002–0x04001003	Reserved		
0x040020D8	Timer Channel 7 Control Register	TMR7CTL	2
0x04001002–0x04001003	Reserved		
0x040020DC	Timer Channel 7 Status and Control Register	TMR7SCTL	2
0x04001002–0x04001003	Reserved		
0x040020E0	Timer Channel 7 Compare 1 Load Register	TMR7CMPLD1	2
0x04001002–0x04001003	Reserved		
0x040020E4	Timer Channel 7 Compare 2 Load Register	TMR7CMPLD2	2
0x04001002–0x04001003	Reserved		
0x040020E8	Timer Channel 7 Comparator Status and Control Register	TMR7COMSC	2
0x040010EA–0x04002FFF	Reserved		
0x04003000	Master Priority Register 0	MPR0	4
0x04003004	Alternate Master Priority Register 0	AMPR0	4
0x04003008–0x0400300F	Reserved		
0x04003010	Slave General-Purpose Register 0	SGPCR0	4
0x04003014	Alternate Slave General-Purpose Register 0	ASGPCR0	4
0x04003018–0x040030FF	Reserved		
0x04003100	Master Priority Register 1	MPR1	4
0x04003104	Alternate Master Priority Register Port 1	AMPR1	4
0x04003108–0x0400310F	Reserved		
0x04003110	Slave General-Purpose Register 1	SGPCR1	4
0x04003114	Alternate Slave General-Purpose Register 1	ASGPCR1	4
0x04003118–0x040031FF	Reserved		
0x04003200	Master Priority Register 2	MPR2	4
0x04003204	Alternate Master Priority Register Port 2	AMPR2	4
0x04003208–0x0400320F	Reserved		
0x04003210	Slave General-Purpose Register 2	SGPCR2	4
0x04003214	Alternate Slave General-Purpose Register 2	ASGPCR2	4
0x04003218–0x040032FF	Reserved		
0x04003300	Master Priority Register 3	MPR3	4
0x04003304	Alternate Master Priority Register 3	AMPR3	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04003308–0x0400330F	Reserved		
0x04003310	Slave General-Purpose Register 3	SGPCR3	4
0x04003314	Alternate Slave General-Purpose Register 3	ASGPCR3	4
0x04003318–0x040033FF	Reserved		
0x04003400	Master Priority Register 4	MPR4	4
0x04003404	Alternate Master Priority Register 4	AMPR4	4
0x04003408–0x0400340F	Reserved		
0x04003410	Slave General-Purpose Register 4	SGPCR4	4
0x04003414	Alternate Slave General-Purpose Register 4	ASGPCR4	4
0x04003418–0x040034FF	Reserved		
0x04003500	Master Priority Register 5	MPR5	4
0x04003504	Alternate Master Priority Register 5	AMPR5	4
0x04003508–0x0400350F	Reserved		
0x04003510	Slave General-Purpose Register 5	SGPCR5	4
0x04003514	Alternate Slave General-Purpose Register 5	ASGPCR5	4
0x04003518–0x04003FFF	Reserved		
0x04004000	DMA Control Register	DMACR	4
0x04004004	DMA Error Status	DMAES	4
0x04004008–0x0400400B	Reserved		
0x0400400C	DMA Enable Request	DMAERQ	4
0x04004010–0x04004023	Reserved		
0x04004024	DMA Enable Error Interrupt	DMAEEI	4
0x04004018	DMA Set Enable Request	DMASERQ	1
0x04004019	DMA Clear Enable Request	DMACERQ	1
0x0400401A	DMA Set Enable Error Interrupt	DMASEEI	1
0x0400401B	DMA Clear Enable Error Interrupt	DMACEEI	1
0x0400401C	DMA Clear Interrupt Request	DMACINT	1
0x0400401D	DMA Clear Error	DMACERR	1
0x0400401E	DMA Set Start Bit	DMASSRT	1
0x0400401F	DMA Clear DONE Status Bit	DMACDNE	1
0x04004020–0x04004023	Reserved		
0x04004024	DMA Interrupt Request	DMAINT	4
0x04004028–0x0400402B	Reserved		
0x0400402C	DMA Error Register	DMAERR	4
0x04004030–0x040040FF	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04004100	DMA Channel 0 Priority	DCHPRI0	1
0x04004101	DMA Channel 1 Priority	DCHPRI1	1
0x04004102	DMA Channel 2 Priority	DCHPRI2	1
0x04004103	DMA Channel 3 Priority	DCHPRI3	1
0x04004104	DMA Channel 4 Priority	DCHPRI4	1
0x04004105	DMA Channel 5 Priority	DCHPRI5	1
0x04004106	DMA Channel 6 Priority	DCHPRI6	1
0x04004107	DMA Channel 7 Priority	DCHPRI7	1
0x04004108	DMA Channel 8 Priority	DCHPRI8	1
0x04004109	DMA Channel 9 Priority	DCHPRI9	1
0x0400410A	DMA Channel 10 Priority	DCHPRI10	1
0x0400410B	DMA Channel 11 Priority	DCHPRI11	1
0x0400410C	DMA Channel 12 Priority	DCHPRI12	1
0x0400410D	DMA Channel 13 Priority	DCHPRI13	1
0x0400410E	DMA Channel 14 Priority	DCHPRI14	1
0x0400410F	DMA Channel 15 Priority	DCHPRI15	1
0x04004110	DMA Channel 16 Priority	DCHPRI16	1
0x04004111	DMA Channel 17 Priority	DCHPRI17	1
0x04004112	DMA Channel 18 Priority	DCHPRI18	1
0x04004113	DMA Channel 19 Priority	DCHPRI19	1
0x04004114	DMA Channel 20 Priority	DCHPRI20	1
0x04004115	DMA Channel 21 Priority	DCHPRI21	1
0x04004116	DMA Channel 22 Priority	DCHPRI22	1
0x04004117	DMA Channel 23 Priority	DCHPRI23	1
0x04004118	DMA Channel 24 Priority	DCHPRI24	1
0x04004119	DMA Channel 25 Priority	DCHPRI25	1
0x0400411A	DMA Channel 26 Priority	DCHPRI26	1
0x0400411B	DMA Channel 27 Priority	DCHPRI27	1
0x0400411C	DMA Channel 28 Priority	DCHPRI28	1
0x0400411D	DMA Channel 29 Priority	DCHPRI29	1
0x0400411E	DMA Channel 30 Priority	DCHPRI30	1
0x0400411F	DMA Channel 31 Priority	DCHPRI31	1
0x04004120–0x04004FFF	Reserved		
0x04005000	Transfer Control Descriptor 0	TCD0	32
0x04005020	Transfer Control Descriptor 1	TCDx-1	32

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04005040	Transfer Control Descriptor 2	TCDx-2	32
0x04005060	Transfer Control Descriptor 3	TCDx-3	32
0x04005080	Transfer Control Descriptor 4	TCDx-4	32
0x040050A0	Transfer Control Descriptor 5	TCDx-5	32
0x040050C0	Transfer Control Descriptor 6	TCDx-6	32
0x040050E0	Transfer Control Descriptor 7	TCDx-7	32
0x04005100	Transfer Control Descriptor 8	TCDx-8	32
0x04005120	Transfer Control Descriptor 9	TCDx-9	32
0x04005140	Transfer Control Descriptor 10	TCDx-10	32
0x04005160	Transfer Control Descriptor 11	TCDx-11	32
0x04005180	Transfer Control Descriptor 12	TCDx-12	32
0x040051A0	Transfer Control Descriptor 13	TCDx-13	32
0x040051C0	Transfer Control Descriptor 14	TCDx-14	32
0x040051E0	Transfer Control Descriptor 15	TCDx-15	32
0x04005200	Transfer Control Descriptor 16	TCDx-16	32
0x04005220	Transfer Control Descriptor 17	TCDx-17	32
0x04005240	Transfer Control Descriptor 18	TCDx-18	32
0x04005260	Transfer Control Descriptor 19	TCDx-19	32
0x04005280	Transfer Control Descriptor 20	TCDx-20	32
0x040052A0	Transfer Control Descriptor 21	TCDx-21	32
0x040052C0	Transfer Control Descriptor 22	TCDx-22	32
0x040052E0	Transfer Control Descriptor 23	TCDx-23	32
0x04005300	Transfer Control Descriptor 24	TCDx-24	32
0x04005320	Transfer Control Descriptor 25	TCDx-25	32
0x04005340	Transfer Control Descriptor 26	TCDx-26	32
0x04005360	Transfer Control Descriptor 27	TCDx-27	32
0x04005380	Transfer Control Descriptor 28	TCDx-28	32
0x040053A0	Transfer Control Descriptor 29	TCDx-29	32
0x040053C0	Transfer Control Descriptor 30	TCDx-30	32
0x040053E0	Transfer Control Descriptor 31	TCDx-31	32
0x04005400–0x04005FFF	Reserved		
0x04006000	FEC Identification Register	FECID	4
0x04006004	Interrupt Event Register	IEVENT	4
0x04006008	Interrupt Enable Register	IMASK	4
0x0400600C	Descriptor Ring Poll Control Register	DRPC	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0400600D–0x0400600F	Reserved		
0x04006010	Receive Descriptor Active Register	RDA	4
0x04006014	Transmit Descriptor Active Register	TDA	4
0x04006018–0x04006023	Reserved		
0x04006024	Ethernet Control Register	ECTL	4
0x04006028–0x0400603F	Reserved		
0x04006040	MII Management Frame Register	MIIDATA	4
0x04006044	MII Speed Control Register	MIISPEED	4
0x04006048–0x04006063	Reserved		
0x04006064	MIB Control Register	MIBCTL	4
0x04006068–0x04006083	Reserved		
0x04006084	Receive Control Register	RCTL	4
0x04006088	Receive Hash	RHASH	4
0x0400608C–0x040060C3	Reserved		
0x040060C4	Transmit Control Register	TCTL	4
0x040060C8–0x040060E3	Reserved		
0x040060E4	Physical Address Low Register	PADDRL	4
0x040060E8	Physical Address High Register	PADDRH	4
0x040060EC	Opcode/Pause Duration Register	OPPAUSE	4
0x040060F0–0x04006117	Reserved		
0x04006118	Descriptor Individual Address 1	IADDR1	4
0x0400611C	Descriptor Individual Address 2	IADDR2	4
0x04006120	Descriptor Group Address 1	GADDR1	4
0x04006124	Descriptor Group Address 2	GADDR2	4
0x04006128–0x0400613F	Reserved		
0x04006140	FIFO ID Register	FIFOID	4
0x04006144	FIFO Transmit Watermark Register	TWMRK	4
0x04006148–0x0400614B	Reserved		
0x0400614C	FIFO Receive Bound Register	FRBND	4
0x04006150	FIFO Receive Start Register	FRST	4
0x04006154–0x0400617F	Reserved		
0x04006180	Receive Descriptor Ring Start Register	RDESST	4
0x04006184	Transmit Descriptor Ring Start Register	TDESST	4
0x04006188	Receive Buffer Size Register	RBSZ	4
0x0400618C–0x040061F3	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x040061F4	DMA Control Register	DMACTL	4
0x040061F8–0x040061FF	Reserved		
0x04006200	RMON Transmit Frames Dropped	RMON_T_DROP	4
0x04006204	RMON Transmit Packet Count	RMON_T_PACKETS	4
0x04006208	RMON Transmit Broadcast Packets	RMON_T_BC_PKT	4
0x0400620C	RMON Transmit Multicast Packets	RMON_T_MC_PKT	4
0x04006210	RMON Transmit Packets with CRC/Alignment Error	RMON_T_CRC_ALIGN	4
0x04006214	RMON Transmit Packets Undersized	RMON_T_UNDERSIZE	4
0x04006218	RMON Transmit Packets Oversized	RMON_T_OVERSIZE	4
0x0400621C	RMON Transmit Packets Fragmented	RMON_T_FRAG	4
0x04006220	RMON Transmit Packets Tabber	RMON_T_JAB	4
0x04006224	RMON Transmit Packet Collision Count	RMON_T_COL	4
0x04006228	RMON Transmit Packets 64 Bytes	RMON_T_P64	4
0x0400622C	RMON Transmit Packets 65–0x127 Bytes	RMON_T_P65TO127	4
0x04006230	RMON Transmit Packets 128–0x255 Bytes	RMON_T_P128TO255	4
0x04006234	RMON Transmit Packets 256–0x511 Bytes	RMON_T_P256TO511	4
0x04006238	RMON Transmit Packets 512–0x1023 Bytes	RMON_T_P512TO1023	4
0x0400623C	RMON Transmit Packets 1024–0x2047 Bytes	RMON_T_P1024TO2047	4
0x04006240	RMON Transmit Packets 2048 Bytes	RMON_T_PGTE2048	4
0x04006244	RMON Transmit Octets	RMON_T_OCTETS	4
0x04006248	Frames Not Transmitted Correctly	IEEE_T_DROP	4
0x0400624C	Frames Transmitted OK	IEEE_T_FRAME_OK	4
0x04006250	Frames Transmitted with Single Collision	IEEE_T_1COL	4
0x04006254	Frames Transmitted with Multiple Collisions	IEEE_T_MCOL	4
0x04006258	Frames Transmitted after Deferral Delay	IEEE_T_DEF	4
0x0400625C	Frames Transmitted with Late Collision	IEEE_T_LCOL	4
0x04006260	Frames Transmitted with Excessive Collision	IEEE_T_EXCOL	4
0x04006264	Frames Transmitted with Transmit FIFO Underrun	IEEE_T_MACERR	4
0x04006268	Frames Transmitted with Carrier Sense Error	IEEE_T_CSERR	4
0x0400626C	Frames Transmitted with SQE Error	IEEE_T_SQE	4
0x04006270	Flow Control Pause Frames Transmitted	T_FDXFC	4
0x04006274	Frames Transmitted without Error Octet Count	IEEE_T_OCTETS_OK	4
0x04006278–0x0400627F	Reserved		
0x04006280	RMON Receive Frames Dropped	RMON_R_DROP	4
0x04006284	RMON Receive Packet Count	RMON_R_PACKETS	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04006288	RMON Receive Broadcast Packets	RMON_R_BC_PKT	4
0x0400628C	RMON Receive Multicast Packets	RMON_R_MC_PKT	4
0x04006290	RMON Receive Packets with CRC/Alignment Error	RMON_R_CRC_ALIGN	4
0x04006294	RMON Receive Packets Undersized	RMON_R_UNDERSIZE	4
0x04006298	RMON Receive Packets Oversized	RMON_R_OVERSIZE	4
0x0400629C	RMON Receive Packets Fragmented	RMON_R_FRAG	4
0x040062A0	RMON Receive Packets Jabber	RMON_R_JAB	4
0x040062A4	RMON Receive Packet Collision Count	RMON_R_COL	4
0x040062A8	RMON Receive Packets 64 Bytes	RMON_R_P64	4
0x040062AC	RMON Receive Packets 65–0x127 Bytes	RMON_R_P65TO127	4
0x040062B0	RMON Receive Packets 128–0x255 Bytes	RMON_R_P128TO255	4
0x040062B4	RMON Receive Packets 256–0x511 Bytes	RMON_R_P256TO511	4
0x040062B8	RMON Receive Packets 512–0x1023 Bytes	RMON_R_P512TO1023	4
0x040062BC	RMON Receive Packets 1024–0x2047 Bytes	RMON_R_P1024TO2047	4
0x040062C0	RMON Receive Packets 2048 Bytes	RMON_R_PGTE2048	4
0x040062C4	RMON Receive Octets	RMON_R_OCTETS	4
0x040062C8	Frames Not Received Correctly	IEEE_R_DROP	4
0x040062CC	Frames Received OK	IEEE_R_FRAME_OK	4
0x040062D0	Frames Received with CRC Error	IEEE_R_CRC	4
0x040062D4	Frames Received with Alignment Error	IEEE_R_ALIGN	4
0x040062D8	Receive FIFO Overflow Count	IEEE_R_MACERR	4
0x040062DC	Flow Control Pause Frames Received	R_FDXFC	4
0x040062E0	Frames Received without Error Octet Count	IEEE_R_OCTETS_OK	4
0x040062E4–0x040063FF	Reserved		
0x04006400	MIIGSK Configuration Register	MIIGSKCFG	4
0x04006404–0x04006407	Reserved		
0x04006408	MIIGSK Enable Register	MIIGSKEN	4
0x0400640C–0x04007FFF	Reserved		
0x04008000	Chip Select 0 Memory Bounds Register	CSBR0	4
0x04008004–0x04008007	Reserved		
0x04008008	Chip Select 1 Memory Bounds Register	CSBR1	4
0x0400800C–0x0400807F	Reserved		
0x04008080	Chip Select 0 Configuration	CS0CFG	4
0x04008084	Chip Select 1 Configuration	CS1CFG	4
0x04008088–0x04008107	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x04008108	DDR SDRAM Timing Configuration Register 1	TCFG1	4
0x0400810C	DDR SDRAM Timing Configuration Register 2	TCFG2	4
0x04008110	DDR SDRAM Control Configuration Register	SCFG	4
0x04008114–0x04008117	Reserved		
0x04008118	DDR SDRAM Mode Configuration Register	SMCFG	4
0x0400811C–0x04008123	Reserved		
0x04008124	DDR SDRAM Interval Configuration Register	SICFG	4
0x04008128	Reserved		
0x04008130	DDR SDRAM Clock Configuration Register	SCLKCFG	4
0x04008134–0x04008E3F	Reserved		
0x04008E40	Memory Error Detect Register	MERRD	4
0x04008E44–0x04008E47	Reserved		
0x04008E48	Memory Error Interrupt Enable Register	ERRINT	4
0x04008E4C	Memory Error Attributes Capture Register	MEAC	4
0x04008E50	Memory Error Address Capture Register	MEADDC	4
0x04008E54	Memory Error Extended Address Capture Register	MEEAC	4
0x04008E58–0x04008FFF	Reserved		
0x04009000	I^2C Address Register	I2AR	2
0x04009002–0x04009003	Reserved		
0x04009004	I^2C Frequency Divider Register	I2FR	2
0x04009006–0x04009007	Reserved		
0x04009008	I^2C Control Register	I2CTLR	2
0x0400900A–0x0400900B	Reserved		
0x0400900C	I^2C Status Register	I2SR	2
0x0400900E–0x0400900F	Reserved		
0x04009010	I^2C Data Register	I2DR	2
0x04009012–0x04009FFF	Reserved		
0x0400A000	Bus Time-Out Control Register	BTMCTL	4
0x0400A004	Reserved		
0x0400A008	Bus Error Control Register	BERRCTL	4
0x0400A00C–0x0400A07F	Reserved		
0x0400A080	Device Configuration Register	DEVCFG	4
0x0400A084–0x0400AFFF	Reserved		
0x0400B000	Event Port Registers	EV_BASE	
0x0400B000	Event 0 Input Selection Register	EVIN0	

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0400B008	Event 1 Input Selection Register	EVIN1	
0x0400B010	Event 2 Input Selection Register	EVIN2	
0x0400B018	Event 3 Input Selection Register	EVIN3	
0x0400B020	Event 4 Input Selection Register	EVIN4	
0x0400B028	Event 5 Input Selection Register	EVIN5	
0x0400B030	Event 6 Input Selection Register	EVIN6	
0x0400B038	Event 7 Input Selection Register	EVIN7	
0x0400B040	Event A 0 Output Register	EVOUT0	
0x0400B048	Event A 1 Output Register	EVOUT1	
0x0400B050	Event A 2 Output Register	EVOUT2	
0x0400B058	Event A 3 Output Register	EVOUT3	
0x0400B060	Event A 4 Output Register	EVOUT4	
0x0400B068	Event A 5 Output Register	EVOUT5	
0x0400B070	Event A 6 Output Register	EVOUT6	
0x0400B078	Event A 7 Output Register	EVOUT7	
0x0400B0C0	Event Port Control	EVCTL	
0x0400B0C4–0x0400BFFF	Reserved		
0x0400C000	Clock Control Register	CLKCTL	4
0x0400C004–0x0400C007	Reserved		
0x0400C008	Stop Mode Control Register	STOPCTL	4
0x0400C00C–0x0400C00F	Reserved		
0x0400C010	Halt Request Register	HLTREQ	4
0x0400C014–0x0400C017	Reserved		
0x0400C018	Halt Acknowledge Status Register	HLTACK	4
0x0400C01C–0x0400C03F	Reserved		
0x0400C040	Reset Status Register	RSR	4
0x0400C044–0x0400CFFF	Reserved		
0x0400D000	Peripheral Address Detection Control Register 0	PADCTL0	4
0x0400D004	Peripheral Address Detection Control Register 1	PADCTL1	4
0x0400D008	Reserved		
0x0400D00C	Peripheral Address Detection Status Register	PADSR	4
0x0400D010	Reserved		
0x0400D014	Peripheral Address Detection AMDMA Lower bound Register 0	PADLWRD0	4
0x0400D018	Peripheral Address Detection AMDMA Lower bound Register 1	PADLWRD1	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0400D01C	Peripheral Address Detection AMDMA Lower bound Register 2	PADLWRD2	4
0x0400D020	Peripheral Address Detection AMDMA Lower bound Register 3	PADLWRD3	4
0x0400D024–0x0400D02B	Reserved		
0x0400D02C	Peripheral Address Detection AMDMA Upper bound Register 0	PADUPRD0	4
0x0400D030	Peripheral Address Detection AMDMA Upper bound Register 1	PADUPRD1	4
0x0400D034	Peripheral Address Detection AMDMA Upper bound Register 2	PADUPRD2	4
0x0400D038	Peripheral Address Detection AMDMA Upper bound Register 3	PADUPRD3	4
0x0400D03C–0x0400D043	Reserved		
0x0400D044	Peripheral Address Detection AMENT Lower Bound Register 0	PADLWRE0	4
0x0400D048	Peripheral Address Detection AMENT Lower Bound Register 1	PADLWRE1	4
0x0400D04C	Peripheral Address Detection AMENT Lower Bound Register 2	PADLWRE2	4
0x0400D050	Peripheral Address Detection AMENT Lower Bound Register 3	PADLWRE3	4
0x0400D054–0x0400D05B	Reserved		
0x0400D05C	Peripheral Address Detection AMENT Upper Bound Register 0	PADUPRX0	4
0x0400D060	Peripheral Address Detection AMENT Upper Bound Register 1	PADUPRE1	4
0x0400D064	Peripheral Address Detection AMENT Upper Bound Register 2	PADUPRE2	4
0x0400D068	Peripheral Address Detection AMENT Upper Bound Register 3	PADUPRE3	4
0x0400D06C–0x0400D073	Reserved		
0x0400D074	Peripheral Address Detection Capture AMDMA Address Register	PADCPTD	4
0x0400D078	Peripheral Address Detection Capture AMENT Address Register	PADCPTE	4
0x0400D07C–0x0400DFFF	Reserved		
0x0400E000	MCIF Control Register	MCIFCTL	4
0x0400E004	Reserved		
0x0400E008	DMA Read Buffer Channel Select Register	DCHSEL	4
0x0400E00C	Reserved		

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0400E010	Alternate Read Buffer Channel Select Register	ACHSEL	4
0x0400E014	Reserved		
0x0400E018	MCIF Status Register	MCIFSTAT	4
0x0400E01C–0x6000FFF	Reserved		
0x06001000	SWT Control Register	SWTCTL	4
0x06001004	SWT Time-Out Range Register	SWTTOR	4
0x06001008	SWT Current Counter Value Register	SWTCCV	4
0x0600100C	SWT Counter Restart Register	SWTCR	4
0x06001010	SWT Interrupt Status Register	SWTSTA	4
0x06001014	SWT End-of-Interrupt Register	SWTEOI	4
0x06001018–0x06003FFF	Reserved		
0x06004000	TDM0 General Interface Register	TDM0GIR	4
0x06004004	TDM0 Receive Interface Register	TDM0RIR	4
0x06004008	TDM0 Transmit Interface Register	TDM0TIR	4
0x0600400C	TDM0 Receive Frame Parameters Register	TDM0RFP	4
0x06004010	TDM0 Transmit Frame Parameters	TDM0TFP	4
0x06004014–0x0600401F	Reserved		
0x06004020	TDM0 Receive Channel 0 Enable Register	TDM0RCEN0	4
0x06004024	TDM0 Receive Channel 1 Enable Register	TDM0RCEN1	4
0x06004028	TDM0 Receive Channel 2 Enable Register	TDM0RCEN2	4
0x0600402C	TDM0 Receive Channel 3 Enable Register	TDM0RCEN3	4
0x06004030–0x0600403F	Reserved		
0x06004040	TDM0 Transmit Channel 0 Enable Register	TDM0TCEN0	4
0x06004044	TDM0 Transmit Channel 1 Enable Register	TDM0TCEN1	4
0x06004048	TDM0 Transmit Channel 2 Enable Register	TDM0TCEN2	4
0x0600404C	TDM0 Transmit Channel 3 Enable Register	TDM0TCEN3	4
0x06004050–0x0600405F	Reserved		
0x06004060	TDM0 Transmit Channel 0 Mask Register	TDM0TCMA0	4
0x06004064	TDM0 Transmit Channel 1 Mask Register	TDM0TCMA1	4
0x06004068	TDM0 Transmit Channel 2 Mask Register	TDM0TCMA2	4
0x0600406C	TDM0 Transmit Channel 3 Mask Register	TDM0TCMA3	4
0x06004070–0x0600407F	Reserved		
0x06004080	TDM0 Receive Control Register	TDM0RCR	4
0x06004084	TDM0 Transmit Control Register	TDM0TCR	4
0x06004088	TDM0 Receive Interrupt Enable Register	TDM0RIER	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x0600408C	TDM0 Transmit Interrupt Enable Register	TDM0TIER	4
0x06004090–0x0600409F	Reserved		
0x060040A0	TDM0 Receive Event Register	TDM0RER	4
0x060040A4	TDM0 Transmit Event Register	TDM0TER	4
0x060040A8	TDM0 Receive Status Register	TDM0RSR	4
0x060040AC	TDM0 Transmit Status Register	TDM0TSR	4
0x060040B0–0x06004FFF	Reserved		
0x06005000	TDM1 General Interface Register	TDM1GIR	4
0x06005004	TDM1 Receive Interface Register	TDM1RIR	4
0x06005008	TDM1 Transmit Interface Register	TDM1TIR	4
0x0600500C	TDM1 Receive Frame Parameters	TDM1RFP	4
0x06005010	TDM1 Transmit Frame Parameters	TDM1TFP	4
0x06005014–0x0600501F	Reserved		
0x06005020	TDM1 Receive Channel 0 Enable Register	TDM1RCEN0	4
0x06005024	TDM1 Receive Channel 1 Enable Register	TDM1RCEN1	4
0x06005028	TDM1 Receive Channel 2 Enable Register	TDM1RCEN2	4
0x0600502C	TDM1 Receive Channel 3 Enable Register	TDM1RCEN3	4
0x06005030–0x0600503F	Reserved		
0x06005040	TDM1 Transmit Channel 0 Enable Register	TDM1TCEN0	4
0x06005044	TDM1 Transmit Channel 1 Enable Register	TDM1TCEN1	4
0x06005048	TDM1 Transmit Channel 2 Enable Register	TDM1TCEN2	4
0x0600504C	TDM1 Transmit Channel 3 Enable Register	TDM1TCEN3	4
0x06005050–0x0600505F	Reserved		
0x06005060	TDM1 Transmit Channel 0 Mask Register	TDM1TCMA0	4
0x06005064	TDM1 Transmit Channel 1 Mask Register	TDM1TCMA1	4
0x06005068	TDM1 Transmit Channel 2 Mask Register	TDM1TCMA2	4
0x0600506C	TDM1 Transmit Channel 3 Mask Register	TDM1TCMA3	4
0x06005070–0x0600507F	Reserved		
0x06005080	TDM1 Receive Control Register	TDM1RCR	4
0x06005084	TDM1 Transmit Control Register	TDM1TCR	4
0x06005088	TDM1 Receive Interrupt Enable Register	TDM1RIER	4
0x0600508C	TDM1 Transmit Interrupt Enable Register	TDM1TIER	4
0x06005090–0x0600509F	Reserved		
0x060050A0	TDM1 Receive Event Register	TDM1RER	4
0x060050A4	TDM1 Transmit Event Register	TDM1TER	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x060050A8	TDM1 Receive Status Register	TDM1RSR	4
0x060050AC	TDM1 Transmit Status Register	TDM1TSR	4
0x060050B0–0x06005FFF	Reserved		
0x06006000	TDM2 General Interface Register	TDM2GIR	4
0x06006004	TDM2 Receive Interface Register	TDM2RIR	4
0x06006008	TDM2 Transmit Interface Register	TDM2TIR	4
0x0600600C	TDM2 Receive Frame Parameters	TDM2RFP	4
0x06006010	TDM2 Transmit Frame Parameters	TDM2TFP	4
0x06006014–0x0600601F	Reserved		
0x06006020	TDM2 Receive Channel 0 Enable Register	TDM2RCEN0	4
0x06006024	TDM2 Receive Channel 1 Enable Register	TDM2RCEN1	4
0x06006028	TDM2 Receive Channel 2 Enable Register	TDM2RCEN2	4
0x0600602C	TDM2 Receive Channel 3 Enable Register	TDM2RCEN3	4
0x06006030–0x0600603F	Reserved		
0x06006040	TDM2 Transmit Channel 0 Enable Register	TDM2TCEN0	4
0x06006044	TDM2 Transmit Channel 1 Enable Register	TDM2TCEN1	4
0x06006048	TDM2 Transmit Channel 2 Enable Register	TDM2TCEN2	4
0x0600604C	TDM2 Transmit Channel 3 Enable Register	TDM2TCEN3	4
0x06006050–0x0600605F	Reserved		
0x06006060	TDM2 Transmit Channel 0 Mask Register	TDM2TCMA0	4
0x06006064	TDM2 Transmit Channel 1 Mask Register	TDM2TCMA1	4
0x06006068	TDM2 Transmit Channel 2 Mask Register	TDM2TCMA2	4
0x0600606C	TDM2 Transmit Channel 3 Mask Register	TDM2TCMA3	4
0x06006070–0x0600607F	Reserved		
0x06006080	TDM2 Receive Control Register	TDM2RCR	4
0x06006084	TDM2 Transmit Control Register	TDM2TCR	4
0x06006088	TDM2 Receive Interrupt Enable Register	TDM2RIER	4
0x0600608C	TDM2 Transmit Interrupt Enable Register	TDM2TIER	4
0x06006090–0x0600609F	Reserved		
0x060060A0	TDM2 Receive Event Register	TDM2RER	4
0x060060A4	TDM2 Transmit Event Register	TDM2TER	4
0x060060A8	TDM2 Receive Status Register	TDM2RSR	4
0x060060AC	TDM2 Transmit Status Register	TDM2TSR	4
0x060060B0–0x06006FFF	Reserved		
0x06007000	Host Control Register	HCR	2

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x06007002–0x0600701F	Reserved		
0x06007020	Host Port Control Register	HPCR	2
0x06007022–0x0600703F	Reserved		
0x06007040	Host Status Register	HSR	2
0x06007042–0x0600705F	Reserved		
0x06007060	Host Command Vector Register	HCVR	2
0x06007062–0x0600707F	Reserved		
0x06007080	Host Transmit Data Register	HOTX	8
0x06007088–0x0600709F	Reserved		
0x060070A0	Host Receive Data Register	HORX	8
0x060070A8–0x06008003	Reserved		
0x06008004	Baud-Rate Divider Register	BRDR	1
0x06008005–0x06008007	Reserved		
0x06008008	Interrupt Identity Register	IIR	1
0x06008009–0x0600800B	Reserved		
0x0600800C	Line Control Register	LCR	1
0x0600800D–0x06008013	Reserved		
0x06008014	Line Status Register	LSR	1
0x06008015–0x0600801B	Reserved		
0x0600801C	Scratchpad Register	SCR	1
0x0600801D–0x06008FFF	Reserved		
0x06009000	Port A Data Register	GPADR	4
0x06009004	Port A Data Direction Register	GPADDR	4
0x06009008	Port A Control Register	GPACTL	4
0x0600900C	Port B Data Register	GPBDR	4
0x06009010	Port B Data Direction Register	GPBDDR	4
0x06009014	Port B Control Register	GPBCTL	4
0x06009018	Port C Data Register	GPCDR	4
0x0600901C	Port C Data Direction Register	GPCDDR	4
0x06009020	Port C Control Register	GPCCTL	4
0x06009024	Port D Data Register	GPDDR	4
0x06009028	Port D Data Direction Register	GPDDDR	4
0x0600902C	Port D Control Register	GPDCTL	4
0x06009030	Port A Interrupt Enable Register	GPAIEN	4
0x06009034	Port A Interrupt Mask Register	GPAIMSK	4

Table 5-2. MSC711x Detailed Memory Map (Continued)

Address	Register/Memory Region	Acronym	Size in Bytes
0x06009038	Port A Interrupt Detection Type Register	GPAITYP	4
0x0600903C	Port A Interrupt Polarity Register	GPAIPOL	4
0x06009040	Port A Interrupt Status Register	GPAISR	4
0x06009044	Port A Interrupt Raw Status Register	GPAIRSR	4
0x06009048–0x0600904B	Reserved		
0x0600904C	Clear Interrupt Register	GPAICLR	4
0x06009050	Port A External Port Register	GPAEXPR	4
0x06009054	Port B External Port Register	GPBEXPR	4
0x06009058	Port C External Port Register	GPCEXPR	4
0x0600905C	Port D External Port Register	GPDEXPR	4
0x06009060	Interrupt Synchronous Level Sensitive	PAISLS	4
0x06009064–0x06009FFF	Reserved		
0x0600A000	Non-Maskable Interrupt Pending Register	NMIPR	4
0x0600A004–0x0600A007	Reserved		
0x0600A008	Maskable Interrupt Pending Register	MIPR	4
0x0600A00C–0x0600AFFF	Reserved		
0x0600B000	Interrupt Priority Level Register 0	IPLR0	4
0x0600B004	Interrupt Priority Level Register 1	IPLR1	4
0x0600B008	Interrupt Priority Level Register 2	IPLR2	4
0x0600B00C	Interrupt Priority Level Register 3	IPLR3	4
0x0600B010	Interrupt Priority Level Register 4	IPLR4	4
0x0600B014	Interrupt Priority Level Register 5	IPLR5	4
0x0600B018	Interrupt Priority Level Register 6	IPLR6	4
0x0600B01C	Interrupt Priority Level Register 7	IPLR7	4
0x0600B020	Interrupt Priority Level Register 8	IPLR8	4
0x0600B024	Interrupt Priority Level Register 9	IPLR9	4
0x0600B028	Interrupt Priority Level Register 10	IPLR10	4
0x0600B02C	Interrupt Priority Level Register 11	IPLR11	4
0x0600B030	Interrupt Priority Level Register 12	IPLR12	4
0x0600B034	Interrupt Priority Level Register 13	IPLR13	4
0x0600B038	Interrupt Priority Level Register 14	IPLR14	4
0x0600B03C–0x1FFFFFFF	Reserved		
0x20000000–0xFFFFFFFF	External DDR Memory		

Table 5-3 summarizes the MSC711x memory map and shows the valid ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-3. Summary — MSC711x Memory Map

Address Range	Description	Size
Extended Core Memory (M1)		
064 KB: 0x00000000–0x0000FFFF 192 KB: 0x00000000–0x0002FFFF 256 KB: 0x00000000–0x0003FFFF	M1 memory when accessed by the SC1400 core	192 KB
064 KB: 0x00010000–0x00EFEFFF 192 KB: 0x00030000–0x00EFEFFF 256 KB: 0x00040000–0x00EFEFFF	Reserved	
Extended Core Registers, ICache Array		
0x00EFF000–0x00EFFFFF	OCE10 emulator registers	4 KB
0x00F00000–0x00F1FFFF	ICache registers Extended core interface registers Core address detection registers	—
0x00F20000–0x00F23FFF	ICache array contents	16 KB
0x00F24000–0x00FFFFFF	Reserved	
Internal Memory (M2, Boot ROM)		
0x0100000–0x0102FFFF	M2 Memory (for devices with M2 memory)	192 KB
0x01030000–0x013FFFFF	Reserved	
0x01400000–0x01401FFF	Boot ROM	8 KB
0x01402000–0x017FFFFF	Reserved	
Extended Core Memory (M1)— Accessible from Masters on the Crossbar Switch		
064 KB: 0x01800000–0x0180FFFF 192 KB: 0x01800000–0x0182FFFF 256 KB: 0x018000000–0x0183FFFF	M1 memory when accessed by the DMA controller via the AMDMA bus or the Ethernet MAC via the AMENT bus.	192 KB
064 KB: 0x01810000–0x01F7FFFF 192 KB: 0x01830000–0x01F7FFFF 256 KB: 0x01840000–0x01F7FFFF	Reserved	
Peripherals		
0x01F80000–0x03FFFFFF	TDM/HDI16 high speed ports (via ASTH bus)	—
0x04000000–0x05FFFFFF	IPBus peripheral registers (via ASSB bus)	32 MB
0x06000000–0x07FFFFFF	APB peripheral registers (via ASAPB bus)	32 MB
External Address Space		
0x08000000–0x1FFFFFFF	Reserved	
0x20000000–0xFFFFFFFF	External DDR memory (via ASEMI bus)	—

5.3 Address Space by Type of Access

The address space accessible by each type of master port access is shown in the following:

5.4 Program Accesses

Program accesses can be generated by the SC1400 or by the cache instruction fetch unit. Program fetches to the memory map occur as follows:

Table 5-4. Program Fetches from MSC711x Memory Map

Address Range	Description
064 KB: 0x00000000–0x0000FFFF 192 KB: 0x00000000–0x0002FFFF 256 KB: 0x00000000–0x0003FFFF	SC1400 fetches from M1 RAM. Non-cacheable, zero wait states.
064 KB: 0x00010000–0x00EFEFFF 192 KB: 0x00030000–0x00EFEFFF 256 KB: 0x00040000–0x00EFEFFF	Accesses not permitted. Generates an AORP_E NMI interrupt.
0x00F00000–0x00FFFFFF	Accesses not permitted. Generates an AORP_E NMI interrupt.
0x01000000–0x0102FFFF	**Cacheable:** IFU bursts from M2 RAM to ICache on cache misses. **Non-cacheable:** SC1400 fetches from M2 RAM through the IFU. For devices without M2 memory, this generates an AORP_AMIC NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x01400000–0x01401FFF	**Cacheable:** IFU bursts from Boot ROM to ICache on cache misses. **Non-cacheable:** SC1400 fetches from Boot ROM through the IFU.
0x01402000–0x017FFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x01800000–0x01F7FFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x01F80000–0x03FFFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x04000000–0x05FFFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x06000000–0x07FFFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORP_AMIC NMI interrupt.
0x20000000–0xFFFFFFFF	**Cacheable:** IFU bursts from external memory to ICache on cache misses. **Non-cacheable:** SC1400 core fetches from external memory through the IFU.

Cacheable regions are determined by the IRBSR and IRCR registers (see **Section 4.7.2**, *Instruction Cacheable Area*, on page 4-37). Note that when the SC1400 core accesses an address

in a cacheable region, the ICache processes the access. If the desired location is already in the cache, its value is provided to the SC1400 core. If the location is not in the cache, the IFU initiates an access to the correct location within a cacheable region. This is the reason why the SC1400 core does not directly access cacheable regions of the memory map. However, the SC1400 core can access any regions programmed as non-cacheable.

Note: Locations 0x01401800–0x01401BFF of the boot ROM are reserved for use by development tools.

5.4.1 SC1400 Read Data Accesses

Table 5-5 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-5. SC1400 Data Reads from MSC711x Memory Map

Address Range	Description
064 KB: 0x00000000–0x0000FFFF 192 KB: 0x00000000–0x0002FFFF 256 KB: 0x00000000–0x0003FFFF	SC1400 reads value from M1 RAM. Zero wait states.
064 KB: 0x00010000–0x00EFEFFF 192 KB: 0x00030000–0x00EFEFFF 256 KB: 0x00040000–0x00EFEFFF	Accesses not permitted. Generates an AORX_E NMI interrupt.
0x00EFF000–0x00EFFFFF	SC1400 core reads the value from OCE10 emulator registers. Zero wait states.
0x00F00000–0x00FFFFFF	SC1400 core reads the value from selected peripheral registers. Zero wait states.
0x01000000–0x0102FFFF	SC1400 core reads the value from M2 memory through the ECI. For devices without M2 memory, this generates an AORX_AMEC NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01400000–0x01401FFF	SC1400 core reads the value from Boot ROM through the ECI.
0x01402000–0x017FFFFF	• Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01800000–0x01F7FFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01F80000–0x01F83FFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01F84000–0x01F87FFF	SC1400 core reads the value from a TDM/HDI16 High Speed Port through the ECI
0x01F88000–0x03FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x04000000–0x0400EFFF	SC1400 core reads the value from a IPBus peripheral register through the ECI.
0x0400F000–0x05FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x06000000–0x0600BFFF	SC1400 core reads the value from an APB peripheral register through the ECI.
0x0600C000–0x07FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x20000000–0xFFFFFFFF	SC1400 core reads the value from external memory through the ECI.

5.4.2 SC1400 Core and Write Buffer Data Accesses

Table 5-6 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices. Writes outside the extended core are either directly performed by the SC1400 core through the bus switch or indirectly performed when the SC1400 core writes to the write buffer and the write buffer completes the access.

Table 5-6. SC1400 and WB Data Writes to MSC711x Memory Map

Address Range	Description
064 KB: 0x00000000–0x0000FFFF 192 KB: 0x00000000 –0x0002FFFF 256 KB: 0x00000000–0x0003FFFF	SC1400 core writes the value to M1 RAM. Zero wait states.
064 KB: 0x00010000–0x00EFEFFF 192 KB: 0x00030000–0x00EFEFFF 256 KB: 0x00040000–0x00EFEFFF	Accesses not permitted. Generates an AORX_E NMI interrupt.
0x00EFF000–0x00EFFFFF	SC1400 core writes the value to the OCE10 emulator registers. Zero wait states.
0x00F00000–0x00FFFFFF	SC1400 core writes the value to selected peripheral registers. Zero wait states.
0x01000000–0x0102FFFF	SC1400 core writes the value to M2 memory through the ECI. For devices without M2 memory, this generates an AORX_AMEC NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01400000–0x01401FFF	Accesses not permitted. Generates an ROM_WR NMI interrupt.
0x01402000–0x017FFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01800000 –0x01F7FFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01F80000–0x01F83FFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x01F84000–0x01F87FFF	ECore writes the value to a TDM/HDI16 high speed port through the ECI.
0x01F88000–0x03FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x04000000–0x0400DFFF	ECore writes the value to an IPBus peripheral register through the ECI.
0x0400E000–0x05FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x06000000–0x0600BFFF	ECore writes the value to an APB peripheral register through the ECI.
0x0600C000–0x07FFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMEC NMI interrupt.
0x20000000–0xFFFFFFFF	ECore writes the value to external memory through the ECI.

5.4.3 DMA Read Data Accesses

Table 5-7 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-7. DMA Reads from MSC711x Memory Map

Address Range	Description
0x00000000–0x00FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt. The DMA controller accesses M1 memory in a different address range.
0x01000000–0x0102FFFF	DMA controller reads the value from M2 memory through the crossbar switch. For devices without M2 memory, this generates an AORX_AMDMA NMI interrupt.
0x01030000–013FFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01400000–01401FFF	DMA controller reads the value from boot ROM through the crossbar switch.
0x01402000–017FFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
064 KB: 0x01800000–0x0180FFFF 192 KB: 0x01800000–0x0182FFFF 256 KB: 0x01800000–0x0183FFFF	DMA controller reads the value from M1 RAM through the crossbar switch.
064 KB: 0x01810000–0x01F7FFFF 192 KB: 0x01830000–0x01F7FFFF 256 KB: 0x01840000–0x01F7FFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01F80000–0x01F83FFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01F84000–0x01F87FFF	DMA controller reads the value from a TDM/HDI16 high speed port through the crossbar switch.
0x01F88000–0x03FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x04000000–0x0400DFFF	DMA controller reads value from IPBus peripheral register through the crossbar switch.
0x0400E000–0x05FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x06000000–0x0600BFFF	DMA controller reads value from APB peripheral register through the crossbar switch.
0x0600C000–0x07FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x20000000–0xFFFFFFFF	DMA controller reads value from external memory through the crossbar switch.

5.4.4 DMA Write Data Accesses

Table 5-8 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-8. DMA Writes to MSC711x Memory Map

Address Range	Description
0x00000000–0x00FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt. The DMA controller accesses M1 memory in a different address range.
0x01000000–0x0102FFFF	DMA controller writes the value to M2 Memory through the crossbar switch. For devices without M2 memory, this generates an AORX_AMDMA NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01400000–0x01401FFF	Accesses not permitted. Generates an ROM_WR NMI interrupt.
0x01402000–0x017FFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.

Table 5-8. DMA Writes to MSC711x Memory Map (Continued)

Address Range	Description
064 KB: 0x01800000–0x0180FFFF 192 KB: 0x01800000–0x0182FFFF 256 KB: 0x01800000–0x0183FFFF	DMA controller writes the value to M1 RAM through the crossbar switch.
064 KB: 0x01810000–0x01F7FFFF 192 KB: 0x01830000–0x01F7FFFF 256 KB: 0x01840000–0x01F7FFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01F80000–01F83FFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x01F84000–01F87FFF	DMA controller writes the value to a TDM/HDI16 High Speed Port through the crossbar switch.
0x01F88000–03FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x04000000–0400DFFF	DMA controller writes the value to IPBus peripheral register through the crossbar switch.
0x0400E000–05FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x06000000–0600BFFF	DMA controller writes the value to APB peripheral register through the crossbar switch.
0x0600C000–07FFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMDMA NMI interrupt.
0x20000000–0xFFFFFFFF	DMA controller writes the value to external memory through the crossbar switch.

5.4.5 Ethernet MAC Read Data Accesses

Table 5-9 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-9. Ethernet MAC DMA Reads from MSC711x Memory Map

Address Range	Description
0x00000000–0x00FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt. The Ethernet accesses M1 memory in a different address range.
0x01000000–0x0102FFFF	The Ethernet DMA reads the value from M2 memory through the crossbar switch. For devices without M2 memory, this generates an AORX_AMENT NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01400000–0x01401FFF	The Ethernet DMA reads the value from boot ROM through the crossbar switch.
0x01402000–0x017FFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
064 KB: 0x01800000–0x0180FFFF 192 KB: 0x01800000–0x0182FFFF 256 KB: 0x01800000–0x0183FFFF	The Ethernet DMA reads the value from M1 RAM through the crossbar switch.
064 KB: 0x01810000–0x01F7FFFF 192 KB: 0x01830000–0x01F7FFFF 256 KB: 0x01840000–0x01F7FFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01F80000–0x01F83FFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01F84000–0x01F87FFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.

Table 5-9. Ethernet MAC DMA Reads from MSC711x Memory Map (Continued)

Address Range	Description
0x01F88000–0x03FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x04000000–0x0400DFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x0400E000–0x05FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x06000000–0x0600BFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x0600C000–0x07FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x20000000–0xFFFFFFFF	The Ethernet DMA reads value from external memory through the crossbar switch.

5.4.6 Ethernet MAC Write Data Accesses

Table 5-10 shows valid address ranges for the different sizes of M1 memory on different MSC711x devices.

Table 5-10. Ethernet MAC DMA Writes to MSC711x Memory Map

Address Range	Description
0x00000000–0x00FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt. The Ethernet accesses M1 memory in a different address range.
0x01000000–0x0102FFFF	The Ethernet DMA writes value to M2 Memory through the crossbar switch. For devices without M2 memory, this generates an AORX_AMENT NMI interrupt.
0x01030000–0x013FFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01400000–0x01401FFF	Accesses not permitted. Generates an ROM_WR NMI interrupt.
0x01402000–0x017FFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
064 KB: 0x01800000–0x0180FFFF 192 KB: 0x01800000–0x0182FFFF 256 KB: 0x01800000–0x0183FFFF	The Ethernet DMA writes value to M1 RAM through the crossbar switch.
064 KB: 0x01810000–0x01F7FFFF 192 KB: 0x01830000–0x01F7FFFF 256 KB: 0x01840000–0x01F7FFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01F80000–0x01F83FFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01F84000–0x01F87FFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x01F88000–0x03FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x04000000–0x0400DFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x0400E000–0x05FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x06000000–0x0600BFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x0600C000–0x07FFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x08000000–0x1FFFFFFF	Accesses not permitted. Generates an AORX_AMENT NMI interrupt.
0x20000000–0xFFFFFFFF	The Ethernet DMA writes value to external memory through the crossbar switch.

5.5 Access Restrictions

There are restrictions on accesses allowed within MSC711x:

- Each master port may have restrictions on which slave ports it can access
- Each slave port may have restrictions on the size of accesses performed

This section shows permitted accesses and permitted access sizes throughout the system.

5.5.1 Master Port Restrictions

This section covers accesses that are restricted on MSC711x devices. Restricted accesses are detected by the illegal access detection blocks, which generate address out of range NMI requests. See **Section 7.2**, *Illegal Access Detection*, on page 7-3.

5.5.1.1 AMEC Port

The extended core interface port cannot access the ASM1 slave port. Instead, core accesses to M1 are more efficiently performed through the P, XA, and XB buses. The AMEC master cannot be used to access locations within the extended core, because these accesses reach the AMEC only if they are valid accesses. Invalid accesses to addresses within the extended core generate the AORP_E or the AORX_E exception. See **Table 5-5**, *SC1400 Data Reads from MSC711x Memory Map*, on page 5-32 and **Table 5-6**, *SC1400 and WB Data Writes to MSC711x Memory Map*, on page 5-33 for the complete set of addresses which cannot be accessed.

5.5.1.2 AMIC Port

The instruction fetch unit port may not access the following slave ports: ASM1, ASTH, ASAPB, and ASSB. Cache bursts are supported only from the components that would store program code: M2 memory and external memory. The AMIC master cannot be used to access locations within the extended core, because these accesses only reach the AMIC if they are valid accesses. Invalid program accesses to addresses within the extended core generate the AORP_E exception. See **Table 5-4**, *Program Fetches from MSC711x Memory Map*, on page 5-31 for the complete set of addresses that cannot be accessed.

5.5.1.3 AMDMA Port

The are no restrictions on the slave ports the DMA port can access. See **Table 5-7**, *DMA Reads from MSC711x Memory Map*, on page 5-34 and **Table 5-8**, *DMA Writes to MSC711x Memory Map*, on page 5-34 for the complete set of addresses that cannot be accessed. The DMA controller must not access addresses within the extended core (0x00000000–0x00FFFFFF). Invalid accesses generate an AORX_AMDMA exception.

5.5.1.4 AMENT Port

The Ethernet MAC unit port cannot access the following slave ports: ASTH, ASAPB, and ASSB.

Allowed access ranges are covered in **Section 5.4.5**, *Ethernet MAC Read Data Accesses*, on page 5-35 and **Section 5.4.6**, *Ethernet MAC Write Data Accesses*, on page 5-36. The Ethernet MAC must not access addresses within the extended core (0x00000000—0x00FFFFFF). Invalid accesses generate an AORX_AMENT exception.

5.5.2 Access Size Restrictions

Table 5-11 shows the allowed accesses and their allowed sizes for each block within MSC711x. The table is organized according to each of the AHB-Lite slave buses out of the crossbar switch.

Table 5-11. Permitted Accesses to MSC711x Blocks via Device-Level Buses

Block	Size of Block's Data Bus	IFU Bursts	Data Accesses: ECI, DMA, Ethernet MAC				Comments
		128-Bit	64-Bit	32-Bit	16-Bit	8-Bit	
ASM1 Accesses[4]							
M1 SRAM	64-bits	[AORP]	Yes[3]	Yes	Yes	Yes	—
ASM2 Accesses[5]							
M2 SRAM	128-bit	Read-Only	Yes[3]	Yes	Yes	Yes	—
Boot ROM	128-bit	Read-Only	Read-Only[3]	Read-Only	Read-Only	Read-Only	Read-only accesses permitted.
ASEMI Accesses[5]							
DDR Controller	128-bit	Read-Only	Yes[3]	Yes	Yes	Yes	—
ASTH Accesses[2]							
HDI16	64-bits	[AORP]	Yes[8]	Yes[8]	Yes[8]	Yes[8]	—
TDMx	64-bits	[AORP]	Yes	[ND]	Yes	Yes	32-byte DMA bursts not allowed.
ASAPB Accesses[2]							
HDI16	16-bits	[AORP]	[ISZ_PF]	[ND][1]	Yes	[ND]	—
TDMx	32-bits	[AORP]	[ISZ_PF]	Yes	Yes[7]	[ND]	—
Interrupt controller: IPL registers	32-bits	[AORP]	[ISZ_PF]	Yes	[ND]	[ND]	Registers using IPL_BASE.
Interrupt controller: All other	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	[ND]	Registers using ICTL_BASE.
Watchdog timer	32-bits	[AORP]	[ISZ_PF]	Yes	[ND]	[ND]	Watchdog registers.
UART	8-bits	[AORP]	[ISZ_PF]	[ND][1]	[ND][1]	Yes	—
GPIO: Port A registers	32-bits	[AORP]	[ISZ_PF]	Yes	Read-Only	[ND]	Includes GPIO interrupt regs.

Table 5-11. Permitted Accesses to MSC711x Blocks via Device-Level Buses (Continued)

Block	Size of Block's Data Bus	IFU Bursts	Data Accesses: ECI, DMA, Ethernet MAC				Comments
		128-Bit	64-Bit	32-Bit	16-Bit	8-Bit	
GPIO: All other	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	[ND]	Port B, C, and D registers.
ASSB Accesses[2]							
Crossbar	32-bits	[AORP]	[ISZ_PF]	Yes	[ND]	[ND]	—
DMA controller	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	Yes	—
DDR controller	32-bits	[AORP]	[ISZ_PF]	Yes	Read -Only	[ND]	[ND] on 16-bit write accesses.
Ethernet MAC	32-bits	[AORP]	[ISZ_PF]	Yes	Yes[6]	[ND]	—
Timer modules	16-bits	[AORP]	[ISZ_PF]	[ND][1]	Yes	[ND]	—
I^2C	16-bits	[AORP]	[ISZ_PF]	[ND][1]	Yes	[ND]	—
System control	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	[ND]	All except watchdog registers.
Event port	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	[ND]	—
PLL/Clock	32-bits	[AORP]	[ISZ_PF]	Yes	Yes	[ND]	Also contains reset registers.

Notation:

- [AORP] - Illegal access detected and AORP_AMIC exception asserted.
- [ISZ_PF] - Illegal access detected and ISZ_PF exception asserted.
- [ND] - Illegal access not detected.

Notes:

1. The data buses within the block limit the maximum size of the transfer.
2. ENET MAC cannot access the ASTH, ASAPB, or ASSB buses.
3. The largest access performed by the Ethernet MAC DMA is 32-bits.
4. ECI does not access M1 memory through the ASM1 bus but instead through the core buses.
5. IFU can only access ASM2 and ASEMI buses.
6. 16-bit accesses permitted to the IEVENT and IMASK registers only.
7. 16-bit accesses permitted to the TDMxRIER, TDMxTIER, TDMxRER, TDMxTER, TDMxRSR, and TDMxTSR registers only.
8. HDI16 8-bit mode supports 8, 16, and 32-bits. 16-bit mode supports 16, 32, and 64-bits.
   If 8 or 16 bits are accessed in 8-bit mode, the external host processor must also access the same size.
   If 16 or 32 bits are accessed in 16-bit mode, the external host processor must also access the same size.

Table 5-12 shows the sizes of accesses permitted to each block using MSC711x platform-level buses.

Table 5-12. Permitted Accesses to MSC711x Blocks via System-Level Buses

Block	Program Accesses	Data Accesses — SC1400				Comments
	128-bit	64-bit	32-Bit	16-Bit	8-Bit	
P Accesses (on SC1400's PDB bus)						
M1 SRAM	Read-Only	N/A	N/A	N/A	N/A	SC1400 program accesses.

Table 5-12. Permitted Accesses to MSC711x Blocks via System-Level Buses

Block	Program Accesses	Data Accesses — SC1400				Comments
	128-bit	64-bit	32-Bit	16-Bit	8-Bit	
XA, XB Accesses (on SC1400 Core XDBA, XDBB buses)						
M1 SRAM	N/A	Yes	Yes	Yes	Yes	SC1400 data accesses.
OCE10 Registers	N/A	[ND]	Yes	Yes	Yes	SC1400 data accesses.
Internal Bus[1] Accesses						
ECI Registers	N/A	[ND]	Yes[2]	Yes	Yes	SC1400 data accesses.
ICache Registers	N/A	[ND]	Yes	Yes	[ND]	SC1400 data accesses.
ICache Array	N/A	Yes	Yes	Yes	Yes	—

Notes: **1.** The Internal bus is a dedicated bus used for accessing ECI and ICache registers, as shown in **Figure 4-5**, *Extended Core Interface Block Diagram,* on page 4-11.

2. The 16-bit IRBSRx and IRCRx registers can be accessed as a 32-bit pair using move.l instructions as noted in **Section 4.8.2**, *ICache Registers,* on page 4-45.

5.6 Misaligned Access Detection on AHB Masters

Misaligned accesses are detected on all of the AHB-Lite master buses: AMEC, AMIC, AMDMA, and AMENT. Upon detection, an appropriate non-maskable interrupt is generated (see **Table 12-3**, *MSC711x Non-Maskable Interrupt Sources from the SC1400 Core*, on page 12-6).

5.7 Bit Field Operations and Restricted Accesses

The SC1400 core has a set of bit field instructions for testing, clearing, setting, or changing bits within a location. These instructions are useful for programming peripherals. These instructions perform 16-bit accesses to memory so they can only be used on blocks listed in **Table 5-11** that allow 16-bit accesses.

For blocks that allow only 32-bit accesses, these instructions cannot be performed on the locations directly. When bit field instructions are desired on these blocks, it is recommended that:

1. The 32-bit value is first read from its location into a SC1400 data register.
2. The bit field instruction is performed on the value the SC1400 data register.
3. The 32-bit value is then stored back to its original location.

5.8 Big-Endian Operation

MSC711x devices are built for big-endian operation:

- All programs should be assembled/compiled for big-endian operation.
- All data is stored in the M1, M2, or DDR memories in big-endian format.
- The SC1400 core BEM bit within the EMR register is set to 1.

Note: For details on endian operation for the SC1400 core, see the *SC1000-Family Processor Core Reference Manual.*

Big-endian operation can also have an effect on how peripherals are accessed. Specifically, when a peripheral's registers are not the full size of the bus they are located on, attention must be paid to endian operation. For example, the HDI16 has a set of 16-bit registers accessible through the 32-bit APB bus. The address offsets in the HDI16 are already correctly specified for access in a big-endian system. Also, the UART has a set of 8-bit registers that are accessible through the 32-bit APB bus.

A different type of example is the event port, which supports both 16- and 32-bit accesses. The address offsets for 32-bit accesses are specified in the event port chapter. For accesses to a 16-bit portion of the 32-bit register, however, the correct address offset must be used for the upper or lower portion, as described in the next section.

5.9 16-bit Accesses to 32-bit Peripheral Registers

When performing 16-bit accesses to 32-bit peripheral registers within MSC711x, it is important to use the correct offset when accessing the upper or lower portion of the register. The correct offset is determined by what is required within a big endian system. 16-bit read-write accesses to 32-bit registers is possible in the following peripheral blocks:

- DMA controller
- Clock synthesis module
- Reset module
- Interrupt controller; all registers except the IPL registers
- TDM — APB port (only to a subset of registers)
- Ethernet MAC
- System control (all except the watchdog timer)
- Event port
- GPIO: port B, C, and D registers

16-bit read-only accesses to 32-bit registers are possible in the following peripheral blocks:

- DDR controller
- GPIO: port A and interrupt registers

When a program needs to access the *upper* 16-bits, big endian operation requires that the offset to access the register be the same as for 32-bit accesses. When a program needs to access the *lower*

16-bits, big endian operation requires that the offset used to access the register be the value for 32-bit accesses plus 2.

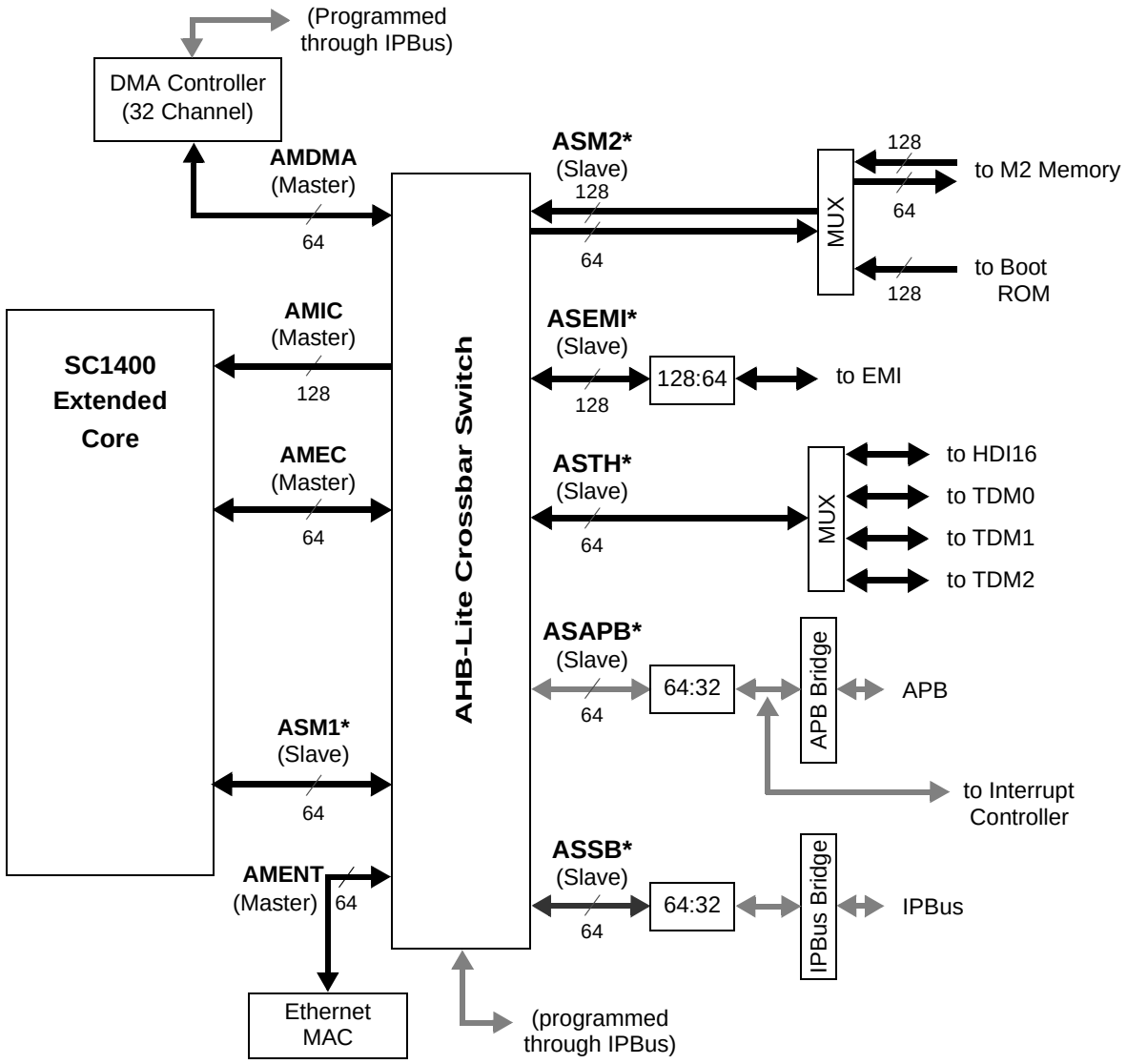
Crossbar Switch 6

The MSC711x extended core and DMA controller connect to the internal resources through the multi-layer AHB crossbar switch. The crossbar switch handles parallel transfers at the system level so that the various masters, including the DMA controller and the extended core interface, can get immediate service when they request resources. The crossbar switch ensures that the SC1400 core executes efficiently within the MSC711x extended core without long stalls while the system waits for cache lines to be filled or data to be processed. All buses connecting to the switch are AHB-Lite. The crossbar switch also facilitates communication between MSC711x modules and ensures that the SC1400 core can focus on the intensive computational work while the DMA controller brings in data for processing. The following features promote flexibility in the crossbar switch:

- Four master ports supporting up to four parallel data transfers.
- Six slave ports, each independently programmable for:
  - Fixed-priority or round-robin arbitration.
  - Dynamically configurable master port priorities.
  - Programmable parking.
- Automatic resizing when source and destination bus widths for a transfer differ.
- Low power park mode.
- Error detection capabilities:
  - Address out-of-range detection on each master port access.
  - Bus error detection units on each master port bus.
  - Bus time-out units on each slave port bus.

6.1 Architecture

Figure 6-1 shows a block diagram of the crossbar switch, associated buses, ports, and multiplexes. The connections of the crossbar switch and hardware bus monitors to the APB are required for accessing its programmable registers.



***** Each crossbar switch master bus has a bus error detection unit with a time-out.
* Each crossbar switch slave bus has a dedicated time-out monitor.

Note: The arrows show the direction of the transfer.

Figure 6-1. Block Diagram of Crossbar Switch System

6.1.1 Master and Slave Ports

The following four MSC711x modules are masters to the crossbar switch:

- Instruction fetch unit (128-bit read path)
- Extended core interface (64-bit read/64-bit write path)
- DMA controller (64-bit read/write path)
- Ethernet MAC (64-bit read/write path)

Table 6-1 shows the master port assignments on MSC711x. You can change these priorities by programming new priorities via the crossbar MPRx registers.

Table 6-1. Master Ports on the Crossbar Switch

Master	Master Port AHB-Lite Bus	Size: Read Path	Size: Write Path	Crossbar Port	Priority from Reset
Extended core interface	AMEC	64 bits	64 bits	0	Highest
DMA	AMDMA	64 bits	64 bits	1	—
Instruction fetch unit	AMIC	128 bits	—	2	—
Ethernet MAC	AMENT	32 bits	32 bits	3	Lowest

The following modules are slaves to the crossbar switch, each with a dedicated AHB-Lite bus:

- M1 memory (64-bit read/write path)
- M2 memory (128-bit read/64-bit write path)
- External memory interface (64-bit read/write path)
- TDMs/HDI16 (64-bit read/write path)
- APB peripherals (32-bit read/write path)
- IPBus peripherals (32-bit read/write path)

Table 6-2 shows the slave port assignments on MSC711x devices.

Table 6-2. Slave Ports on the Crossbar Switch

Slave	Slave Port AHB-Lite Bus	Size: Read Path	Size: Write Path	Crossbar Port	Comments
M1 memory	ASM1	64 bits	64 bits	0	—
M2 memory	ASM2	128 bits	64 bits	1	128-bit path supports full sized accesses from the instruction fetch unit (IFU).
External memory interface	ASEMI	64 bits	64 bits	2	—
TDM/HDI data port	ASTH	64-bits	64-bits	3	Bus for reading/writing data values to the peripheral. Peripherals are configured using ASAPB or ASSB.
IPBus peripherals	ASSB	32 bits	32 bits	4	—
APB peripherals	ASAPB	32 bits	32 bits	5	—

Downsizers match master ports with a defined bus width to a slave port with a smaller bus width. There are downsizers on the following MSC711x buses:

- ASEMI. Supports 128-bit program accesses on the 64-bit ASEMI bus.
- ASAPB. Converts a 64-bit bus width from the crossbar into 32 bits for the ASAPB bus.
- ASSB. Converts a 64-bit bus width from the crossbar into 32 bits for the ASSB bus.

The crossbar switch routes bus transactions initiated on the master ports to the appropriate slave ports. There are no provisions for routing transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol, and the crossbar switch assumes it is the sole master of each slave port. The crossbar switch does not support the bus request/bus grant protocol, so multiple masters are not supported.

Each master and slave port fully complies with AHB-Lite + AMBA extensions. The ports are not fully AHB-compliant because the crossbar switch does not support SPLITs or RETRYs.

6.1.2 Buses

Bus time-out monitors on all AHB-Lite slave buses provide the following advantages:

- Quicker response to a time-out.
- Monitoring is active only after arbitration in the crossbar switch.
- The time-out value can be programmed separately for each slave port. This is significant because different slaves often have different response characteristics. Placing the monitors on the slave buses manages the system latencies more efficiently.
- Bus error monitors detect an AHB ERROR condition on the bus and generate a non-maskable interrupt. Illegal access detection helps trap incorrect system operation.

Bus error monitors on all AHB-Lite master buses detect an AHB error condition on the bus and generate a non-maskable interrupt. Also, a time-out value is associated with each master bus.

Multiplexers on the ASM2 bus allow both the M2 memory and boot ROM to connect to a single slave port on the crossbar switch. Multiplexers on the ASTH bus allow both the TDMs and the host port high-speed port to connect to a single slave port on the crossbar switch.

6.1.3 System-Level Parallelism

Because the crossbar switch is a multi-layer switch, system-level transfers can occur in parallel. MSC711x devices support four master ports with a corresponding four-layer switch so that the following transfers can occur in parallel:

- Cache fills from M2 memory or external memory initiated by the instruction fetch unit.
- One DMA access (read or write).
- Accesses from the SC1400 core through the ECI or writes from the extended core write buffer to system-level resources such as the MSC711x device peripherals.
- One DMA access from the Ethernet MAC.

Following are examples of system-level parallelism. **Figure 6-2** shows a view of the crossbar switch in the MSC711x system to illustrate the various interfaces to the switch.

- Cache bursting from M2 memory:
  - ICache bursts in a portion of a cache line from M2 memory via the instruction fetch unit.
  - DMA bursts of new data from external memory to M1 memory.
  - SC1400 core accesses a peripheral via the extended core interface.
- Cache bursting from external memory:
  - Cache bursts in a portion of a cache line from external memory via the fetch unit.

— DMA bursts of new data from the TDM to M1 memory.
— Write buffer writes to M2 via the extended core interface

- Cache bursting from external memory:
  — Cache bursts in a portion of a cache line from external memory via the fetch unit.
  — Ethernet MAC DMA bursts in new data to M1 memory.
  — Write buffer writes to M2 memory via the extended core interface.
- Transfers on all masters:
  — Cache bursts in a portion of a cache line from M2 via the fetch unit.
  — Ethernet MAC DMA bursts in new data to M1 memory.
  — DMA writes a value to external memory.
  — Write buffer writes to a HDI16 control register via the extended core interface.

6.2 Crossbar Switch Operation

This section describes the functionality of the crossbar switch, including arbitration, priority assignment (including alternate priority), master port functionality, slave port functionality, and halting the crossbar switch.

6.2.1 Arbitration

The crossbar switch supports both fixed-priority and round-robin arbitration, which is independently programmable for each slave port. Alternate priority capability is also supported.

6.2.1.1 Alternate Priority Capability

Alternate priority capability is triggered from the event port. When the event port trigger arrives, all slave ports on the crossbar switch are switched to the priorities in the alternate register set. The alternate priorities remain in effect until the event port signal is deasserted. See **Section A.1.7.6**, *Alternate Priorities*, on page A-15.

6.2.1.2 Context Switching

A hardware input per slave port selects the registers from which the master priority levels and general-purpose control bits are taken, either the MPR and SGPCR or the AMPR and the ASGPCR. This hardware input is useful for context switching because you do not have to rewrite the MPR or SGPCR if a particular slave port would temporarily benefit from modifying the master priority levels or functions affected by the bits in the SGPCR.

6.2.1.3 Fixed-Priority Arbitration

In fixed-priority mode, each master is assigned a unique priority level in the Master Priority Register (MPR) and the Alternate Master Priority Register (AMPR). If two masters request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

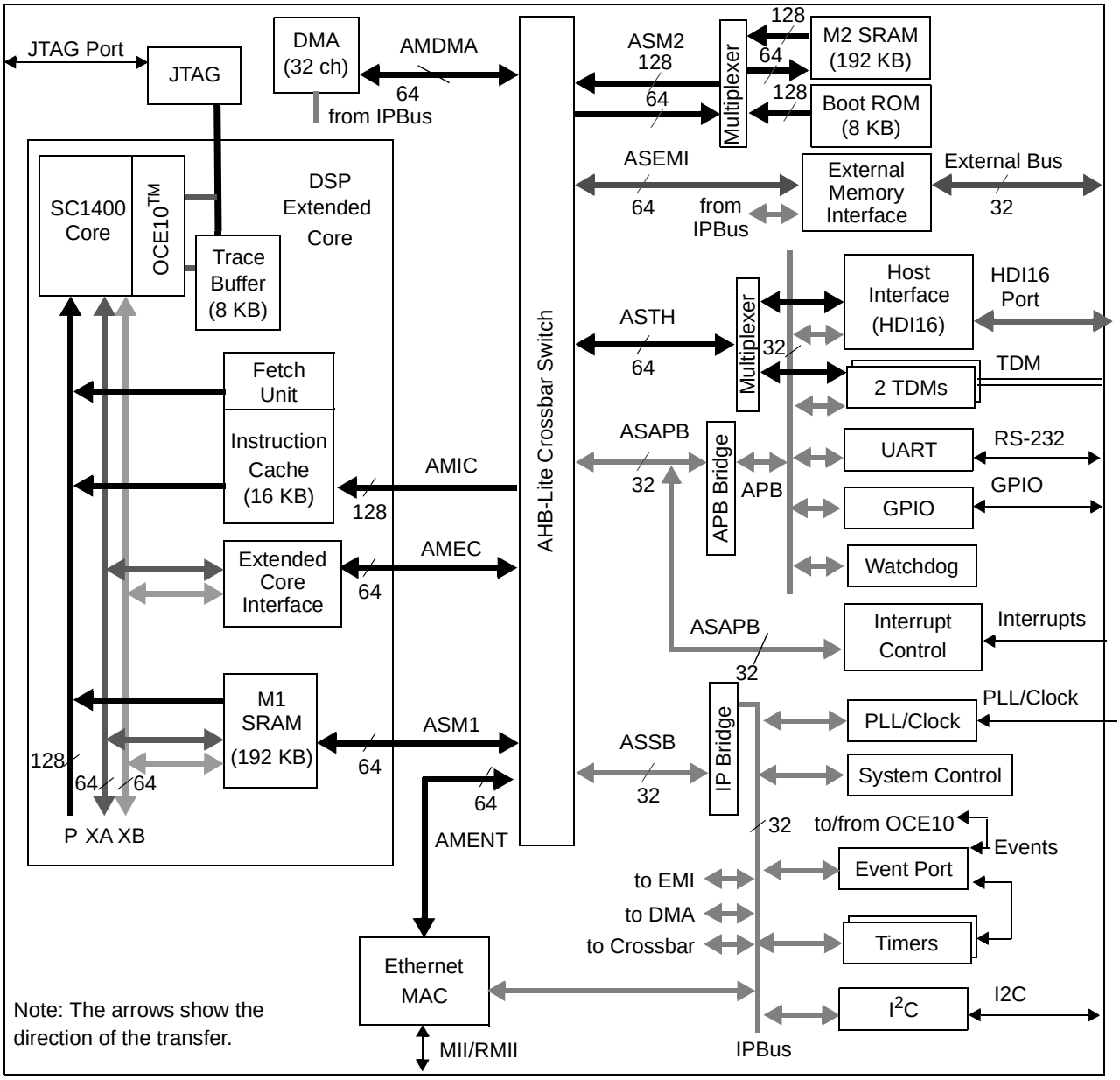


Figure 6-2. View of the Crossbar Switch in the MSC711x System (MSC7116 Device)

When a master makes a request to a slave port, the slave port verifies whether the requesting master's priority level is higher than that of the master currently in control of it (unless the slave port is in a parked state). The slave port performs an arbitration check at every clock edge to ensure that the proper master (if any) has control of it. If the new requesting master's priority level is higher than that of the current master, the requesting master is granted control over the slave port at the next clock edge. However, if the current master is running a fixed-length burst transfer or a locked transfer, the requesting master must wait until the end of the transfer before it is granted control of the slave port. If the current master is running a burst transfer of undefined length, the requesting master must wait until an arbitration point in the transfer before it is granted control of the slave port.

If the requesting master's priority level is lower than that of the current master, the requesting master must wait until the current master either runs an IDLE cycle or runs a non-IDLE cycle to a location other than the current slave port.

6.2.1.4 Round-Robin Priority Arbitration

In round-robin mode, each master is assigned a relative priority based on the master number. This relative priority is compared with the ID of the last master to perform a transfer on the slave bus. The highest-priority requesting master becomes owner of the slave bus as the next transfer boundary (accounting for locked and fixed-length burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number).

After it is granted access to a slave port, a master can perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port at the completion of the current transfer, or possibly on the next clock cycle if the current master has no pending access request. For example, if the last master of the slave port was master 1, and master 0, 2, and 3 make simultaneous requests, they are serviced in the order 2, 3, and then 0.

Parking can be used in a round-robin mode, but it does not affect the round-robin pointer unless the parked master actually performs a transfer. Hand-off occurs to the next master in line after one cycle of arbitration. If the slave port is put into low power park mode, the round-robin pointer is reset to point at master port 0, giving it the highest priority.

Each master port has a high-priority input that can be enabled by writing the correct data to the SGPCR or ASGPCR. If a master's high-priority input is enabled for a slave port programmed for round-robin mode, that master can force the slave port into fixed-priority mode by asserting its high priority input. While that (or any enabled) master's high-priority input is asserted, the slave port remains in fixed-priority mode. After that (or any enabled) master's high-priority input is negated, the slave port reverts to round-robin priority mode, and the pointer is set on the last master to access the slave port.

6.2.2 Priority Assignment

Each master port must be assigned a unique 3-bit priority level in the MPR and AMPR. If an attempt is made to program multiple master ports with the same priority level, the crossbar switch responds with an error and the registers are not updated.

A hardware input per master port can temporarily elevate the master's priority level on all slave ports. In this case, the master port automatically has a higher priority than all other master ports, regardless of the priority levels programmed in the MPR and AMPR. If multiple master ports have their priority elevated, they have higher priority than all master ports that do not. The MPR or AMPR priority level determines which master port of elevated priority has the highest priority

on a per slave port basis. This functionality is useful because it allows you automatically to elevate a master port's priority level throughout the crossbar switch in order to perform temporary tasks such as servicing interrupts quickly. The HPEx bits must be set in the slave port SGPCR or ASGPCR to elevate the priority of a master. The priority of the IFU, DMA, and ECI masters can be elevated. Priority elevation for the Ethernet MAC is not supported.

See **Section 4.6.1**, *Cache Bursting Parameters*, on page 4-31 for information on elevating the IFU priority. ECI priority is elevated in the following cases:

- Any read access through the ECI
- A write immediate access (freezes the SC1400 core)
- A write immediate access with no freeze
- The write buffer is flushed
- A write access when the write buffer is disabled
- Atomic operations

The ECI returns to its normal programmed priority when the SC1400 core writes to the write buffer (when no flush occurs) or the write buffer performs writes to locations outside the platform but not in response to a flush.

The priority of the DMA controller can be elevated when the bandwidth control field in the TCDs specifies that the priority should be elevated for this transfer. **Table 6-3** shows the priorities of crossbar switch slave port out of reset. You can reorder the priority among the four master ports, if desired, by programming the slave port SGPCRx register. The settings out of reset are not recommended for best performance.

Table 6-3. Crossbar Switch Master Port Priorities

Crossbar Master Port	Priority Elevated?	Priority	Comments
ECI	yes	Highest	Elevated on write buffer flushes.
DMA controller	yes	—	Elevated when selected in the TCD.
IFU	yes	—	Bursts in primary set on initial cache miss.
Ethernet MAC	yes	—	Elevated if DEVCDF[ENTP] is set.
ECI	no	—	—
DMA controller	no	—	Normal use.
IFU	no	—	Used for prefetch bursts for the cache.
Ethernet MAC	no	Lowest	Used when DEVCFG[ENTP] is cleared.

Table 6-4 shows an example of a recommended settings for the ASEMI slave port for fixed-priority arbitration. This setting helps to ensure that the SC1400 core does not freeze for a long time.

Table 6-4. ASEMI Recommended Master Port Priorities

Crossbar Master Port	Priority Elevated?	Priority	Comments
IFU	yes	Highest	Bursts in primary set on initial cache miss. The SC1400 core can freeze during these accesses
ECI	yes	—	The SC1400 core can freeze during these accesses.
DMA controller	yes	—	Elevated when selected in the TCD.
Ethernet MAC	yes	—	Elevated if DEVCFG[ENTP] is set.
IFU	no	—	Used for prefetch bursts for the cache.
ECI	no	—	—
DMA controller	no	—	Normal use.
Ethernet MAC	no	Lowest	Used when DEVCFG[ENTP] is cleared.

6.2.3 Master Port Functionality

Each master port consists of two decoders, a capture unit, a register slice, a multiplex, and a small state machine, which controls all aspects of the master port, including which slave port is to receive a request when that request is made. It also has information about whether the slave port is ready to accept an access from the master port. Therefore, it can determine whether the slave port will immediately take the request from the master or whether the master port must capture the master's request and queue it at the slave port boundary.

The state machine has six states:

- *Busy*. The master runs a BUSY cycle to the master port. The master port maintains its request to the slave port if it currently owns the slave port; however, if it loses control of the slave port it no longer maintains its request. It is not allowed to make another request to the slave port until it runs a NSEQ or SEQ cycle. No master generates this state on an MSC711x device.
- *Idle*. The master runs a valid IDLE cycle to the master port. The master port makes no requests to the slave ports (disables the slave port decoder) and terminates the IDLE cycle.
- *Waiting*. The master is running valid cycles to a local slave other than the crossbar switch. In this case, the crossbar switch disables the slave port decoder.
- *Stalled*. The master makes a request to a slave port that is not immediately ready to receive the request. The state machine directs the capture unit to send out the captured address and control signals and enables the slave port decoder to indicate a pending request to the appropriate slave port.
- *Steady state*. The master and slave ports are in fully asynchronous mode, making the crossbar switch completely transparent in the access. The state machine selects the appropriate slave's signals to pass back to the master.

- *First cycle error response and second cycle error response*. These states are self explanatory. The crossbar switch responds with an error response to the master if the master tries to access an unavailable memory location.

The main goal of the master-side state machine is to handle the move of a master from one slave port access to another slave port access, while minimizing or eliminating any extra clock cycles that might be inserted into the access during the switch. The state machine does not allow the master to request access to another slave port until the current access terminates. This prevents a single master from owning two slave ports at the same time. The state machine also maintains watch on the slave port the master is accessing as well as the slave port targeted for the next access. If the targeted slave port is parked on the master, then the master can make the switch without incurring any delays. The termination of the current access also acts as the launch of the new access on the new slave port. If the new slave port is not parked on the master, there is a minimum of one clock delay before the master can launch its access on the targeted slave port.

The same holds true for switching from the busy, idle or waiting states to actively accessing a slave port. If the slave port is parked on the master, the state machine goes to the steady state, and the access begins immediately. If the slave port is not parked on the master because it is serving another master, parked on another master, or in low-power park mode, the state machine transitions to the stalled state, and at least a one clock penalty must be paid.

6.2.4 Slave Port Functionality

The state machine handles the main slave port arbitration. It determines which master is in control of the slave port and which master is to control the slave port in the next bus cycle. **Figure 6-3** shows a block diagram of a slave port, ASEMI, which is useful in understanding the capabilities of the crossbar switch. On the left are the buses from each master on an MSC711x device. These buses are arbitrated separately on each slave port. The arbitration parameters are specified for each slave port on the crossbar.

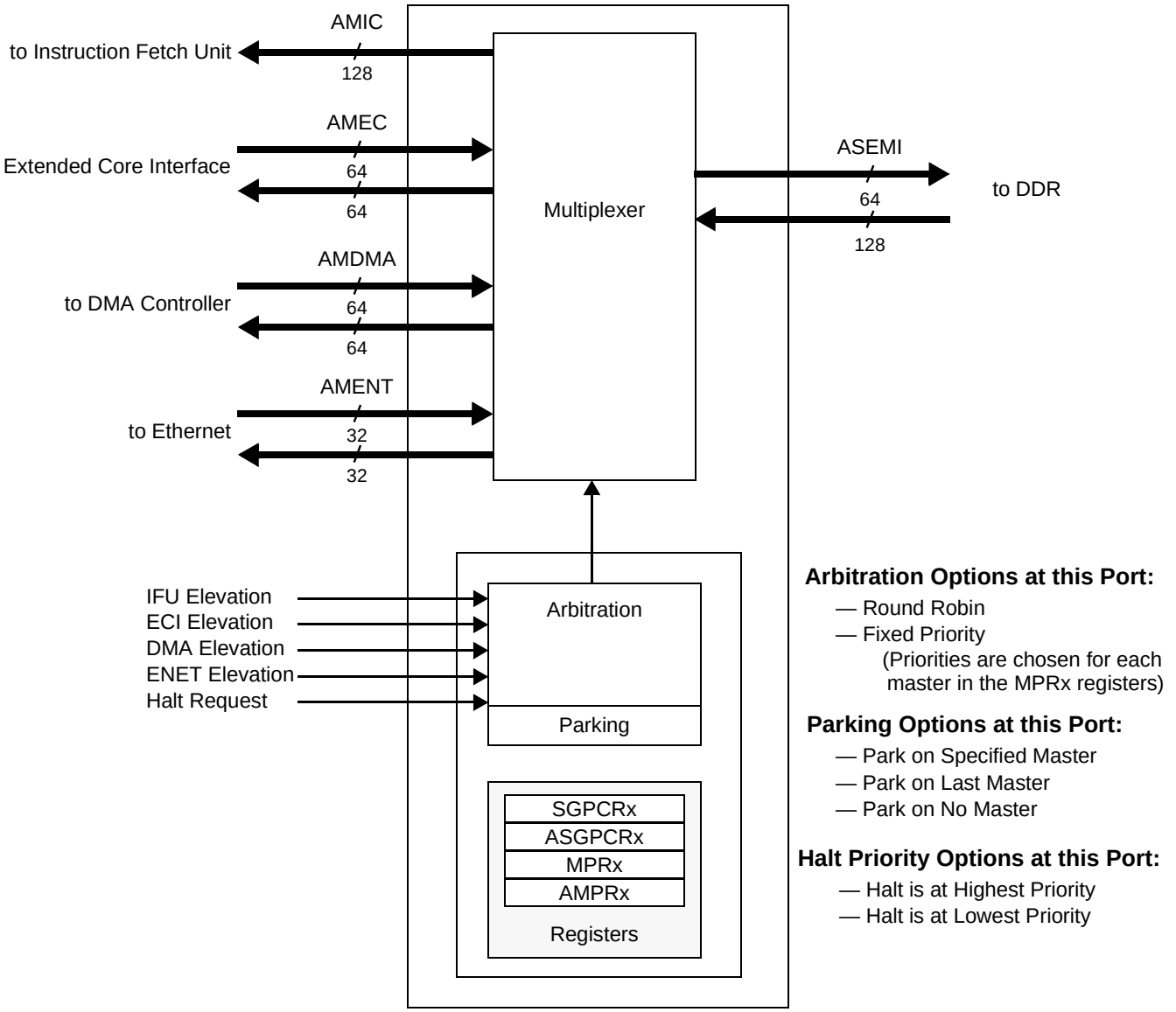


Figure 6-3. Detailed View of One Crossbar Slave Port, ASEMI

6.2.4.1 Slave Port Registers

A register control block at the same level as the master and slave port instantiations in the crossbar switch ensures that all accesses are 32-bit supervisor accesses before it passes them on to the master and slave ports. The registers in a slave port are only those associated with that particular slave port. The read and write interface for the registers is not a full IPBus interface at this level because not all the IPBus signals are routed this deep in the design. The register outputs directly connect to the slave state machine, and an internal slave port input signal notifies the crossbar switch to use the AMPR rather than the MPR and the ASGPCR rather than the SGPCR to define priority levels, halt priority values, and pass the arbitration algorithm and parking control bits to the state machine. The registers can be read from an unlimited number of times.

The registers can be written only as long as the RO bit has a value of 0 in the SGPCR. After a value of 1 is written to this bit, only a hardware reset can allow the registers to be written again.

6.2.4.2 Slave Port State Machine

At the heart of the slave port is the state machine, which requires only four states: steady state, transition state, transition hold state, and hold state. Either the slave port is owned by the same master that owned it in the last clock cycle (either by active use or by parking), it is transitioning to a new master (either for active use or parking), it is transitioning to a new master during wait states, or it is held on the same master pending a transition to a new master.

The main task of the state machine is the arbitration to determine which master port is to control the slave port in the next clock cycle. Each master is programmed with a fixed 3-bit priority level. A fourth priority bit is derived from the input signal on each master port that notifies the crossbar switch to elevate the master port's priority above that of all other masters. Effectively, each master's priority level is a 4-bit field with the priority elevation as the MSB. The crossbar switch uses these bits in to determine priority levels when it is programmed for fixed-priority arbitration or when one of the enabled priority elevation inputs is asserted.

Arbitration occurs on a clock edge, but only when a change in mastership does not violate AHB-Lite protocols. Valid arbitration points include any clock cycle in which a slave indicates it is ready (if the master is not performing a burst or locked cycle) and any wait state in which the master owning the bus indicates a transfer type of IDLE (if the master is not performing a locked cycle). Since arbitration can occur on every clock cycle, the slave port masks off all master requests if the current master is performing a locked transfer or a protected burst transfer, guaranteeing that no matter how low its priority level, the master can finish its locked or protected portion of a burst sequence.

When it is programmed for fixed-priority arbitration, a slave port switches masters only when a higher-priority master makes a request or the current master has the highest priority and it gives up the slave port by either running and IDLE cycle to the slave port or running a valid access to a location other than the slave port. If the current master loses control of the slave port because a higher-priority master takes it, the slave port does not incur any wasted cycles. The slave port terminates the current cycle of the current master port at the same time it recognizes the new master's address and control information. This transition is seamless to the slave port.

If the current master is wait stated when the higher-priority master makes its request, the current master is allowed to make one more transaction on the slave bus before giving it up to the new master. **Figure 6-4** illustrates the effect of a higher-priority master taking control of the bus when the slave port is programmed for a fixed priority mode of operation.

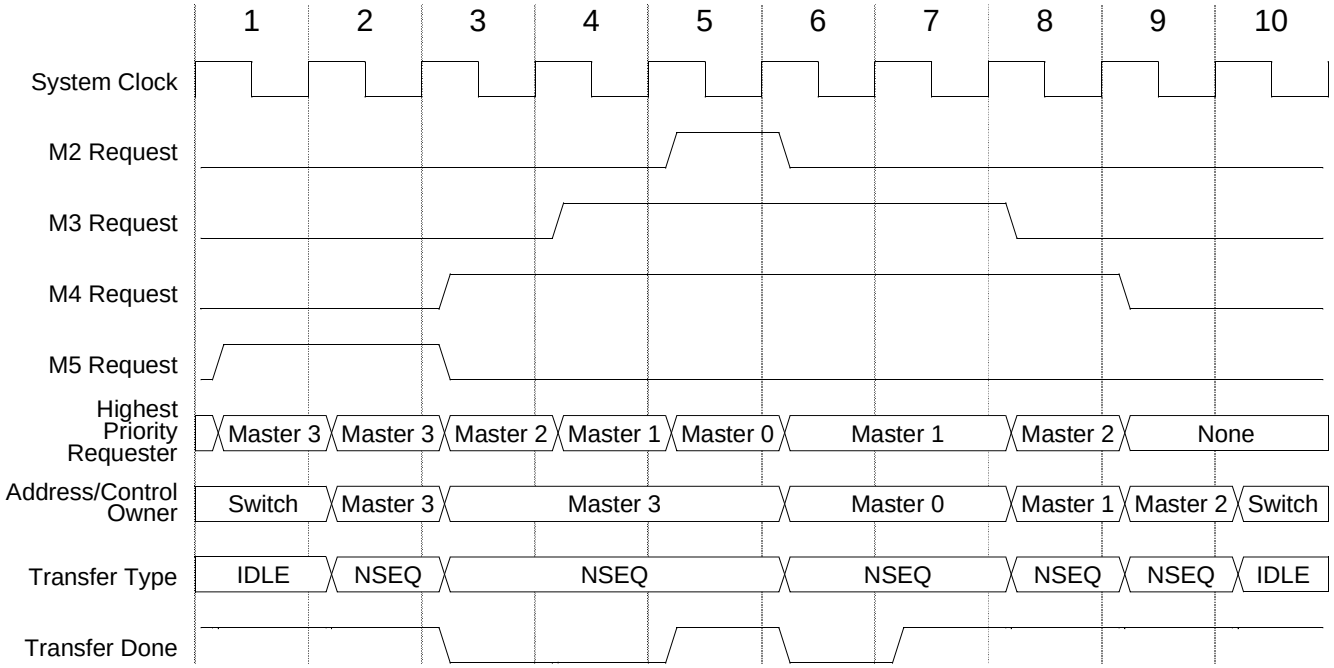


Figure 6-4. Low to High Priority Mastership Change

If the current master gives up the slave port, the master with the next highest priority gains control of the slave port. If the current access incurs any wait states, the transition is seamless, and no bandwidth is lost. However, if the current transaction terminates without wait states, the crossbar switch forces one IDLE cycle onto the slave bus before the new master takes control. If no other master is requesting the bus, the crossbar switch runs IDLE cycles, but no bandwidth is lost. **Figure 6-5** illustrates the effect of a higher-priority master giving up control of the bus.

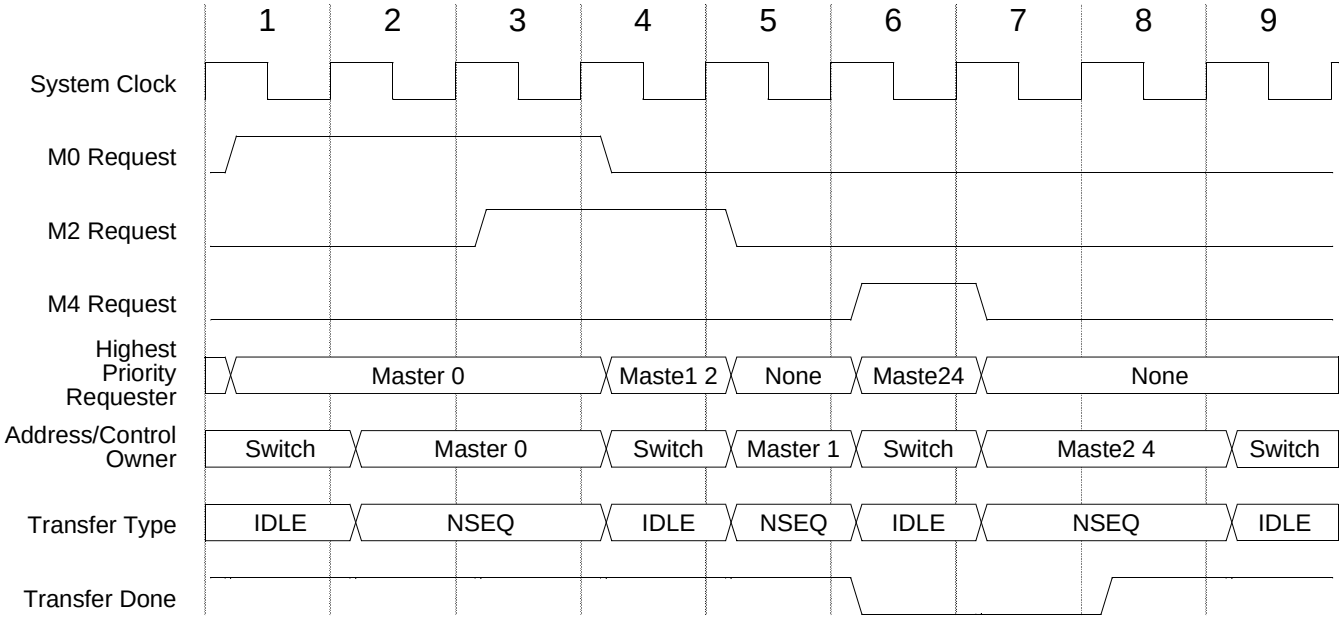


Figure 6-5. High to Low Priority Mastership Change

When the slave port is programmed for round-robin arbitration, the slave port switches masters any time more than one master makes a request to the slave port. This happens because any master other than the one with current control of the bus is considered to have higher priority. **Figure 6-6** shows an example of round-robin arbitration.

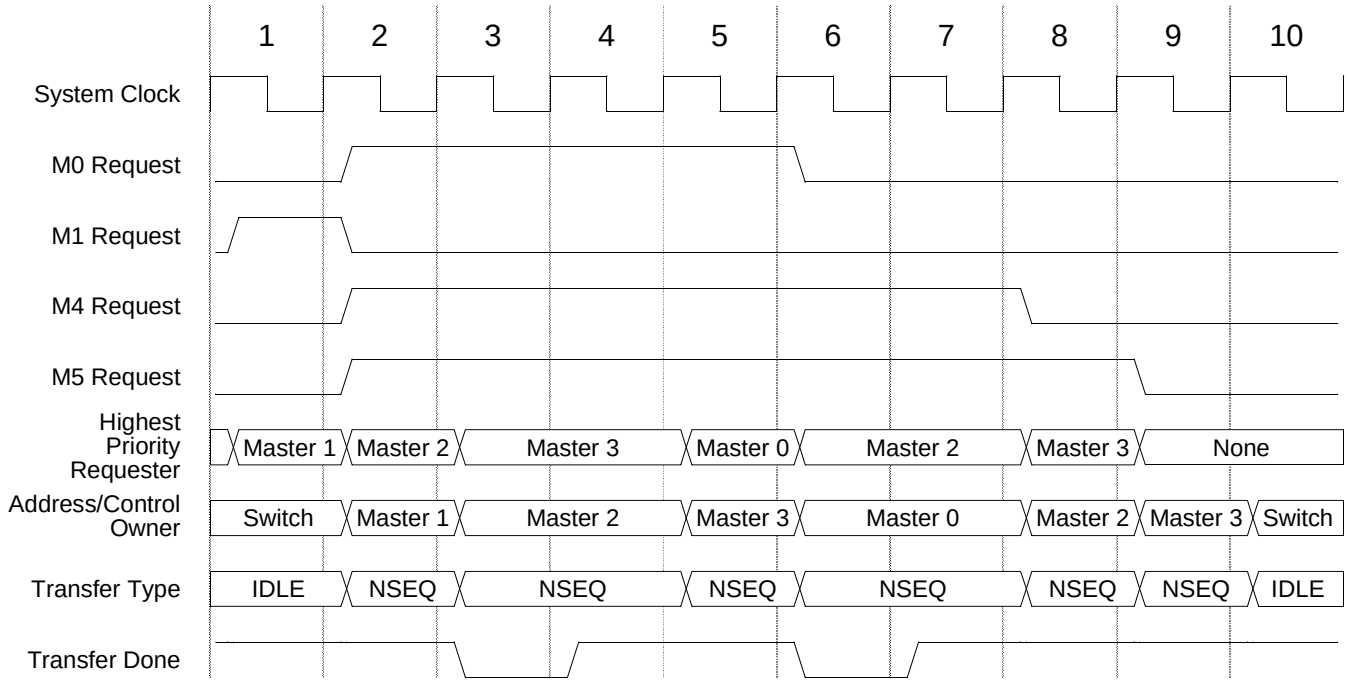


Figure 6-6. Round-Robin Mastership Change

If no master requests access to the slave port, the slave port is parked. It parks in one of four places, dictated by the PCTL and PARK bits in the GPCR or AGPCR and the locked state of the last master to access it. If the last master to access the slave port ran a locked cycle and continues to run locked cycles even after leaving the slave port, the slave port parks on that master without regard to the bit settings in the GPCR or to pending requests from other masters. Therefore, the master can run a locked transfer to the slave port, leave it, and return to it and be guaranteed that no other master has accessed it, if the master maintains all transfers as locked transfers. If locking is not an issue for parking, the GPCR bits indicate the parking method.

If the PCTL bits are set for a low-power park, the slave port does not recognize any master as in control of it and it does not select any master signals to pass to the slave bus. All slave bus activity effectively halts because all slave bus signals driven from the crossbar switch have a value of 0. The result can be a significant power savings if the slave port is not in use for some time. However, there is a penalty of a one clock cycle delay when a master does make a request to the slave port because it must arbitrate to acquire ownership of the slave port.

If the PCTL bits are set to park on last mode, the slave port parks on the last master to access it, passing all the master signals to the slave bus. The crossbar switch asynchronously forces internal

signals to 0 for all access that the master does not run to the slave port. When that master accesses the slave port again, it does not pay any arbitration penalty. However, if any other master accesses the slave port, an arbitration penalty of one clock cycle is imposed.

If the PCTL bits are set to use PARK/APARK mode, the slave port parks on the master designated by the PARK bits. The behavior here is the same as for park on last mode, with the exception that the slave port parks on a specific master instead of the last master to access it. If the master designated by the PARK bits accesses the slave port, it does not pay an arbitration penalty. However, any other master must pay a penalty of one clock cycle. **Figure 6-7** illustrates parking on a specific master.

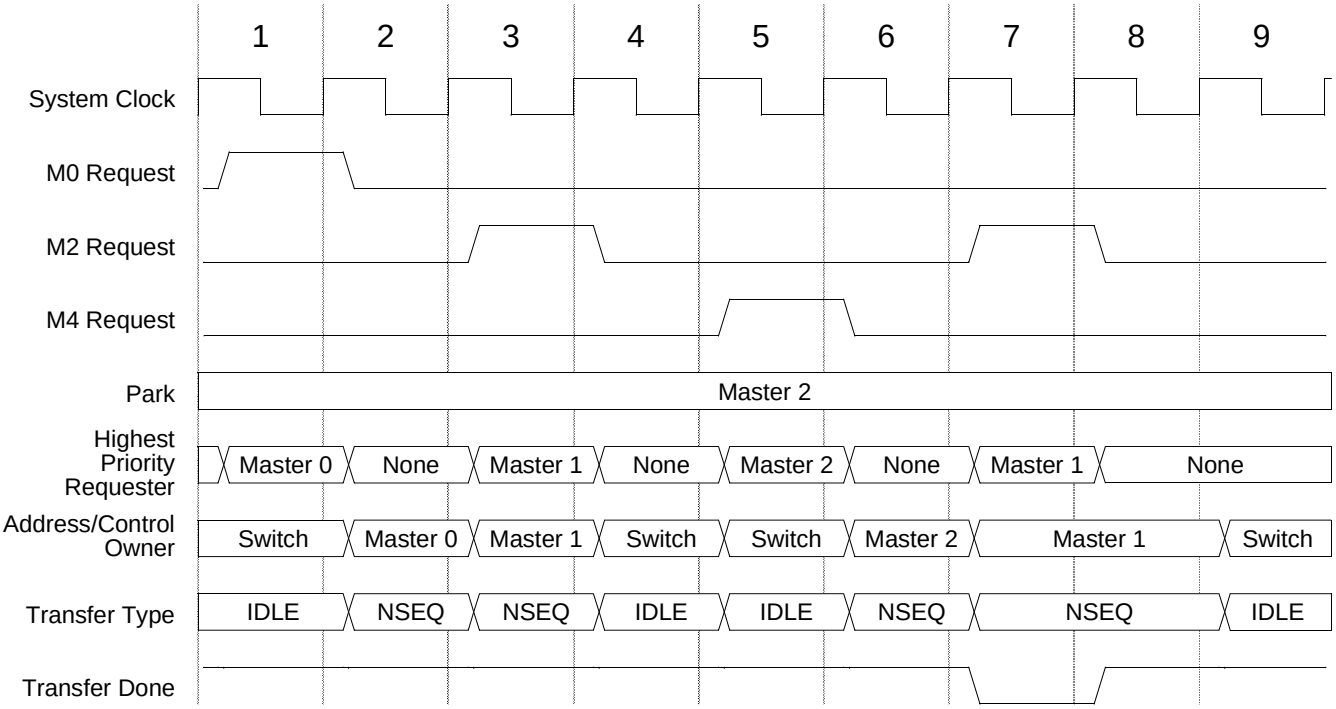


Figure 6-7. Parking on a Specific Master

Figure 6-8 illustrates parking on the last master. Notice that in cycle 6 simultaneous requests are made by master 1 and master 2. Although master 1 has a higher priority, the slave bus is parked on master 2, so master 2's access is taken first. The slave port parks on master 1 when it gives control to master 1. This same situation can occur for parking on a specific master.

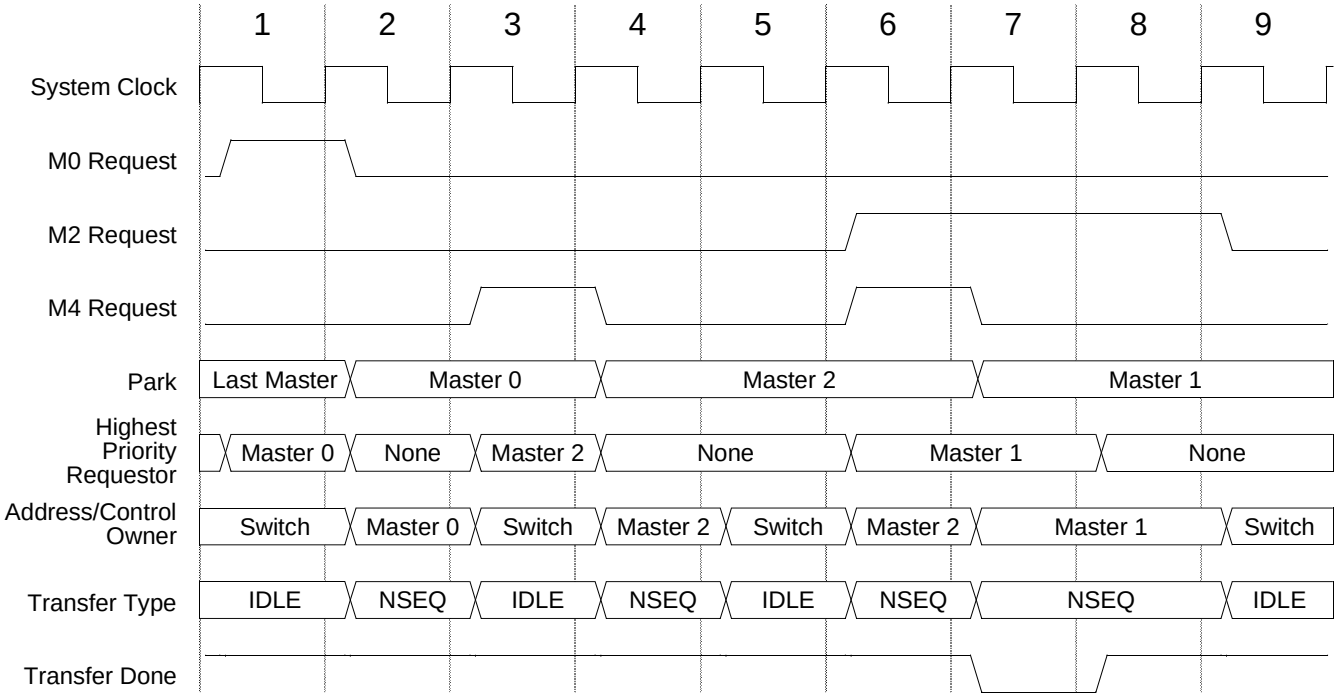


Figure 6-8. Parking on Last Master

6.2.5 Halting the Crossbar Switch

The crossbar switch is halted using the technique described in **Section 11.4.3**, *AHB Subsystem Low-Power Operation*, on page 11-16. There must first be a request for a halt via the HLTREQ[XBRHRQ] bit. This signal is passed to each slave port in the crossbar switch where it must arbitrate for control of the slave port based on the priority programmed in the slave port via its SGPCR[HLP] bit. All slave ports must successfully arbitrate for the halt request before the crossbar switch halts. This procedure promotes a graceful shut down so that the system clock can be stopped for low power mode.

If the halt request input is asserted, the slave port eventually halts all slave bus activity and enters halt mode, which is almost identical to low-power park mode. In halt mode no master is selected to own the slave port, so all the outputs of the slave port are set to 0. The GPCR[HLP] bit controls the priority level of the halt request. If the HLP bit is cleared, the halt requestor has the highest priority of any master and gains control of the slave port at the next arbitration point, which is probably the next bus cycle unless the current master is running a locked or fixed-length burst transfer. If the HLP bit is set, the slave port waits until no masters are actively making requests before moving to halt mode.

Regardless of the state of the HLP bit, when the slave port enters halt mode as a result of a halt request, it remains in halt mode until the halt request is negated, regardless of the priority level of any masters that may make requests.

6.3 Data Throughput for Masters and Slaves

The main goal of the crossbar switch interface is to increase overall system performance by allowing multiple masters to communicate in parallel with multiple slaves. To maximize data throughput, arbitration delays must be kept to a minimum. This section examines data throughput from the point of view of masters and slaves, detailing when the crossbar switch interface stalls the masters or inserts extra clock cycles on the slave side.

6.3.1 Master Ports

Master accesses receive one of the following responses from the crossbar switch:

- *Ignored*. A master access is ignored if the switch select input of the crossbar switch is not asserted. The crossbar switch responds to IDLE transfers when the switch select input is asserted but does not allow the access to pass through it.
- *Terminated*. A master access terminates if the switch select input of the crossbar switch is asserted and the transfer type is IDLE. The crossbar switch terminates the access and it is not allowed to pass through the crossbar switch.
- *Taken*. A master access is accepted if the switch select input of the crossbar switch is asserted, the transfer type is non-IDLE, and the slave port to which the access decodes is either servicing the master or is parked on the master. The crossbar switch is completely transparent, the master access appears immediately on the slave bus, and no arbitration delays are incurred.
- *Stalled*. A master access stalls if the switch select input of the crossbar switch is asserted, the transfer type is non-IDLE, and the access decodes to a slave port that is busy serving another master, parked on another master, or in low-power park mode. The crossbar switch indicates to the master that the address phase of the access is accepted and queued to the appropriate slave port. If the slave port is parked on another master or in low-power park mode and no other master is requesting access to the slave port, only one clock of arbitration penalty is incurred. If the slave port is serving another master of a lower priority and the master has a higher priority than all other requesting masters, the master gains control over the slave port as soon as the data phase of the current access completes (burst and locked transfers excluded). If the slave port is servicing another master of a higher priority, the requesting master gain control of the slave port once the other master releases control, if no other higher-priority master is also waiting for the slave port.
- *Error response terminated*. A master access gets an error response if the switch select input of the crossbar switch is asserted, the transfer type is non-IDLE, and the access decodes to a location not occupied by a slave port. This is the only time the crossbar switch issues an error response. All other error responses received by the master are the slave port error responses passed through the crossbar switch.

6.3.2 Slave Ports

A goal of the crossbar switch interface is to keep the slave ports 100 percent saturated when masters are actively making requests. To achieve this goal, the crossbar switch must not insert any extra clock cycles onto the slave bus unless absolutely necessary.

However, the crossbar switch forces a bubble onto the slave bus when a higher-priority master has control of the slave port and runs single clock (zero wait state) accesses while a lower-priority master is stalled and waiting for control of the slave port. When the higher-priority master either leaves the slave port or runs an IDLE cycle to the slave port, the crossbar switch takes control of the slave bus and runs a single IDLE cycle before giving the slave port to the lower-priority master.

In addition, the crossbar switch takes control of the slave port when the crossbar switch halts or when no masters request access to the slave port and the crossbar switch is forced either to park the slave port on a specific master or to put the slave port into low-power park mode.

Usually, when the crossbar switch has control of the slave port, it indicates IDLE for the transfer type, negates all control signals, and indicates ownership of the slave bus. One exception occurs when a master running locked cycles leaves the slave port but continues to run locked cycles. The crossbar switch controls the slave port and indicates IDLE for the transfer type but it does not affect any other signals.

Note: When a master runs a locked cycle through the crossbar switch, the master is guaranteed ownership of all slave ports it accesses for one cycle beyond when it finishes running locked cycles.

6.4 Crossbar Switch Programming Model

Four IPBus-compliant registers reside in each slave port of the crossbar switch, and one register resides in each master port. **Figure 6-9** and **Figure 6-10** show the programming model at each master port and slave port, respectively. Read and write transfers require two IPBus clock cycles. The registers can be read and written only in Supervisor mode and in 32-bit accesses. The registers are fully decoded and an error response is returned if an unavailable location is accessed within the crossbar switch. Since the content of the registers has a real-time effect on the crossbar switch operation, it is important to keep in mind that any register modifications take effect as soon as the register is written. The values of the registers do not track with slave port related AHB accesses but instead track only with IPBus accesses.

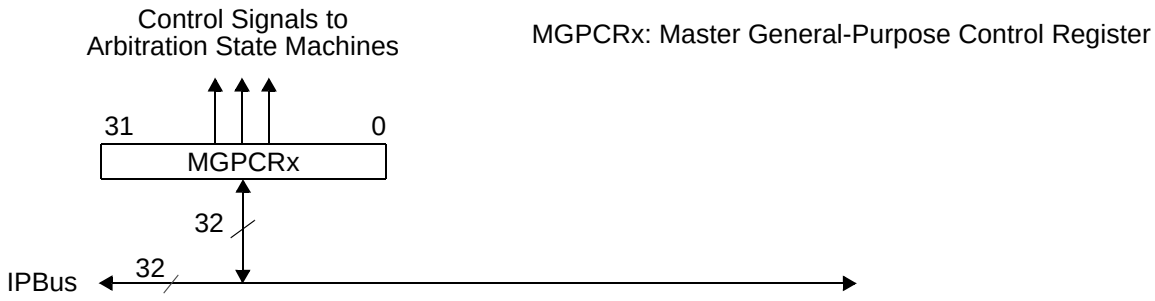


Figure 6-9. Programming Model at Each Master Port

MPRx: Master Priority Register
AMPRx: Alternate Master Priority Register
SGPCRx: Slave General-Purpose Control Register
ASGPCRx: Alternate Slave General-Purpose Control Register

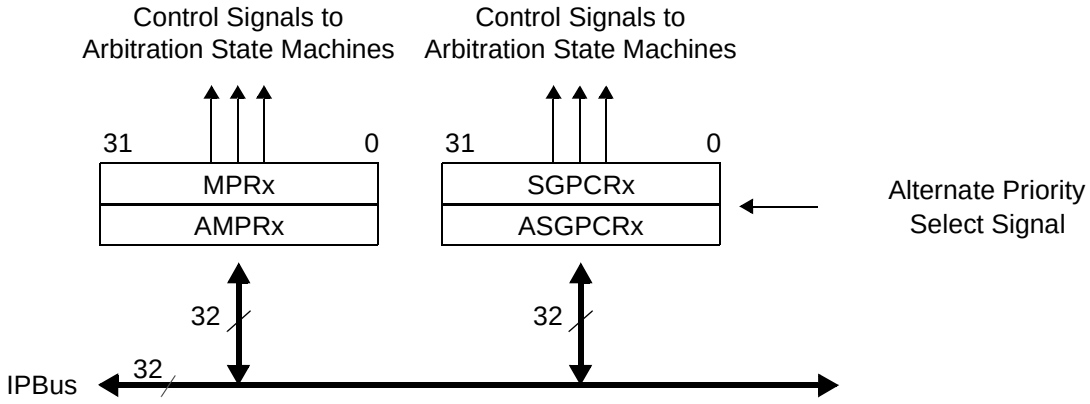


Figure 6-10. Programming Model at Each Slave Port

The slave registers also feature a bit that, when written with a 1, prevents the registers from being written again. The registers are still readable, but write attempts have no effect and terminate with an error response. Following is the list of ECI registers and the pages on which they are discussed:

- Master Priority Register x (MPR[0–5]), **page 6-20**.
- Alternate Master Priority Register x (AMPR[0–5], **page 6-20**.
- Slave General-Purpose Control Register x (SGPCR[0–5]), **page 6-22**.
- Alternate Slave General-Purpose Control Register x (ASGPCR[0–5]), **page 6-22**.

MPR*x* Master Priority Register

MPR0	0x000
MPR1	0x100
MPR2	0x200
MPR3	0x300

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	MSTR3			—	MSTR2			—	MSTR1			—	MSTR0		
TYPE	R	R/W			R	R/W			R	R/W			R	R/W		
RESET	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

MPR sets the priority of each master port on a per slave port basis and resides in each slave port. MPR can be accessed only in Supervisor mode with 32-bit accesses. When a read-only bit is set in the slave General-Purpose Control Register, the MPR can only be read. Attempts to write to it have no effect on the MPR and result in an error response. Additionally, no two available master ports can be programmed with the same priority level. Attempts to do so result in an error response, and the MPR is not updated. See **Table 6-5** for bit descriptions.

AMPR*x* Alternate Master Priority Register

AMPR0	0x004
AMPR1	0x104
AMPR2	0x204
AMPR3	0x304

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	MSTR3			—	MSTR2			—	MSTR1			—	MSTR0		
TYPE	R	R/W			R	R/W			R	R/W			R	R/W		
RESET	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

AMPR sets the alternate priority of each master port on a per slave port basis. The functionality of the AMPR is identical to that of the MPR. The purpose of the AMPR is to allow you to set up an alternate set of priorities for context switching. A hardware input to the crossbar switch controls, on a per slave port basis, whether the slave port uses the MPR or the AMPR. See **Table 6-5** for definitions of the AMPR bits. AMPR*x* can be accessed only in Supervisor mode with 32-bit accesses. When the read-only bit is set in the General-Purpose Control Register, the AMPR*x* can only be read. Attempts to write to it have no effect on the AMPR*x*, and they result in an error response. Additionally, no two available master ports can be programmed with the same priority level. Attempts to do so result in an error response, and the AMPR is not be updated.

Table 6-5. MPR*x* and AMPR*x* Bit Descriptions

Name	Reset	Description	Settings
— 31–15	0	Reserved. Write to zero for future compatibility.	
MSTR3 14–12	0b011	**Master 3 Priority** Sets the arbitration priority for master port 3 on the associated slave port. These bits are initialized by hardware reset.	000 Highest priority when accessing the slave port. 011 Lowest priority when accessing the slave port.
— 11	0	Reserved. Write to zero for future compatibility.	
MSTR2 10–8	0b010	**Master 2 Priority** Sets the arbitration priority for master port 2 on the associated slave port. These bits are initialized by hardware reset.	000 Highest priority when accessing the slave port. 011 Lowest priority when accessing the slave port.
— 7	0	Reserved. Write to zero for future compatibility.	
MSTR1 6–4	0b001	**Master 1 Priority** Sets the arbitration priority for master port 1 on the associated slave port. These bits are initialized by hardware reset.	000 Highest priority when accessing the slave port. 011 Lowest priority when accessing the slave port.
— 3	0	Reserved. Write to zero for future compatibility.	
MSTR0 2–0	0x000	**Master 0 Priority** Sets the arbitration priority for master port 0 on the associated slave port. These bits are initialized by hardware reset.	000 Highest priority when accessing the slave port. 011 Lowest priority when accessing the slave port.

SGPCR*x* Slave General-Purpose Register

SGPCR0	0x010
SGPCR1	0x110
SGPCR2	0x210
SGPCR3	0x310

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RO	HLP	—										HPE3	HPE2	HPE1	HPE0
TYPE	R/W		R										R/W			
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—						ARB		—		PCTL		—	PARK		
TYPE	R						R/W		R		R/W		R	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SGPCR controls several features of each slave port. The SGPCR can be accessed only in supervisor mode with 32-bit accesses. When the RO (Read Only) bit is set, the SGPCR can only be read. Attempts to write to it have no effect on the SGPCR and result in an error response.

ASGPCR*x* Alternate Slave General-Purpose Register

ASGPCR0	0x014
ASGPCR1	0x114
ASGPCR2	0x214
ASGPCR3	0x314

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	HLP	—										HPE3	HPE2	HPE1	HPE0
TYPE	R/W		R										R/W			
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—						ARB		—		PCTL		—	PARK		
TYPE	R						R/W		R		R/W		R	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ASGPCR controls several features of each slave port. The function of ASGPCR is identical to that of the SGPCR, with the notable exception that it lacks the Read Only (RO) bit. The purpose of the ASGPCR is to allow you to set up an alternate set of general control fields for context switching. A hardware input to the crossbar switch controls, on a per slave port basis, whether the

slave port uses the SGPCR or ASGPCR. See **Table 6-6** for descriptions of the bit fields in the ASGPCR because they are identical except for the RO bit. The ASGPCR can be accessed only in Supervisor mode with 32-bit accesses. When the SGPCR[RO] bit is set, the ASGPCR can only be read. Attempts to write to it have no effect on the ASGPCR, and they result in an error response.

Table 6-6. SGPCR*x* and ASGPCR*x* Bit Descriptions

Name	Reset	Description	Setting
RO 31	0	**Read Only** Prevents any registers associated with this slave port from being written after they are set. You can write this bit with a value of 0 as many times as you want, but after you write a value of 1 to it, only a reset condition allows it to be written again. **Note:** The ASGPCR does not have an RO bit. Otherwise, the functionality of ASGPCR is identical to that of SGPCR.	0 All this slave port's registers can be written. 1 All this slave port's registers are read-only and cannot be written (attempted writes have no effect and result in an error response).
HLP 30	0	**Halt Low Priority** Sets the initial arbitration priority of the halt request input. This bit is initialized by a hardware reset. Setting this bit does not prevent the halt request from attaining highest priority when it has control of the slave ports.	0 The halt request input has the highest priority for arbitration on this slave port. 1 The halt request input has the lowest initial priority for arbitration on this slave port.
— 29–20	0	Reserved. Write to zero for future compatibility.	
HPEx 19–16	0	**High Priority Enable** Enables the high priority inputs for the respective master. These bits are initialized by hardware reset.	0 The high priority input is disabled on this slave port. 1 The high priority input is enabled on this slave port.
— 15–10		Reserved. Write to zero for future compatibility.	
ARB 9–8	0b00	**Arbitration Mode** Selects the arbitration policy for the slave port. These bits are initialized by hardware reset.	00 Fixed priority. 01 Round robin (rotating) priority. 10 Reserved. 11 Reserved.
— 7–6	0	Reserved. Write to zero for future compatibility.	

Table 6-6. SGPCR*x* and ASGPCR*x* Bit Descriptions (Continued)

Name	Reset	Description	Setting
PCTL 5–4	0b00	**Parking Control** Determines the parking control used by this slave port. That is, PCTL determines how the slave port parks when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low-power park mode to force all the outputs of the slave port to inactive states when no master is requesting an access. The low-power park feature can result in a power savings if the slave port is not saturated. However, it forces an extra clock cycle of latency when any master tries to access it while it is not in use because it is not parked on any master. These bits are initialized by hardware reset.	00 Parks the slave port on the master port defined by the PARK bit field. 01 Parks the slave port on the last master to be in control of the slave port. 10 Parks the slave port on no master and drives all outputs to a constant safe state. 11 Reserved.
— 3	0	Reserved. Write to zero for future compatibility.	
PARK 2–0	0b000	**Park** Determines which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset.	000 Park on Master Port 0. 001 Park on Master Port 1. 010 Park on Master Port 2. 011 Park on Master Port 3.

System Control 7

The MSC711x system control unit provides system-level control and protection, as well as high-level device configuration, as follows:

- System protection:
  - Bus time-out monitors
  - Bus error detection
  - Illegal access detection
- Software watchdog timer
- Device configuration:
  - Selecting between secondary and additional pin usage for GPIO pins in port D
  - Software watchdog timer enable via SWTE pin
  - Selecting 16- or 32-pin operation for the DDR controller
- Device identification

7.1 System Protection

This section covers all aspects of system protection, including bus time-out monitors, bus error detection, and illegal access detection.

7.1.1 Bus Time-Out Monitors (Slave Buses)

The bus time-out monitors check all AHB-Lite buses connected to the slave ports of the crossbar switch and detect cases where an access takes too long. When a monitor detects that an access has not been serviced correctly, the AHB-Lite transaction terminates with an error response and an NMI interrupt is generated. **Table 7-1** shows a block diagram of one bus time-out monitors.

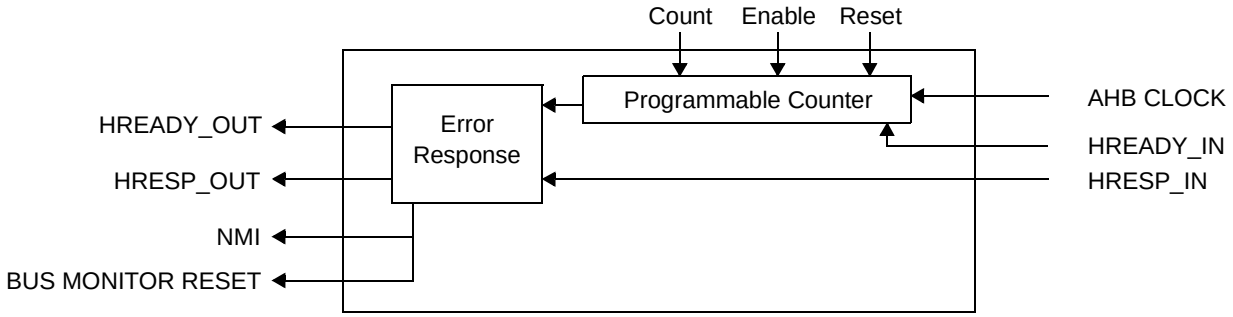


Figure 7-1. Bus Time-out Monitor Block Diagram

The counter is reset to 0x7FF. When active, the counter is programmed for a count value of 31, 127, 511, or 2047 clock ticks. When a bus time-out occurs, an AHB error response is generated according to the standard two-cycle protocol and an $\overline{\text{NMI}}$ is asserted. The time-out steps occur are outlined as follows:

1. A bus master begins a transaction.
2. The transaction passes through the crossbar switch to a slave.
3. The slave accepts the transaction and holds the transaction via internal signals that enter the bus time-out monitor.
4. The bus time-out monitor passes these signals back to the master through the crossbar switch. In addition, the bus time-out monitor begins down-counting to zero.
5. The slave never responds and the counter in the monitor reaches zero.
6. The bus time-out monitor generates an AHB-Lite error response, which terminates the transaction and asserts the internal ready again. An $\overline{\text{NMI}}$ is generated as well.
7. The transaction at the slave is terminated.

7.1.2 Bus Time-Out and Error Detection (Master Buses)

The MSC711x AHB-Lite buses and IPBus can return error responses for an access, such as an access to an invalid memory location. A set of bus error detection units on each AHB master port (AMEC, AMIC, AMDMA, and AMENT) responds to these conditions. When an error response is detected on an AHB-Lite bus, a dedicated non-maskable interrupt is asserted to determine which master port caused the incorrect access. The bus error interrupts are listed in **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8.

An error response on the IPBus passes to the ASSB bus. The response is then passed back to the original master port, which flags the error. The bus error interrupt also asserts if there is a time-out on an AHB master bus, which occurs if an access is not serviced after a preprogrammed number of AHB clock cycles. For example, a bus error interrupt occurs when a master is locked out of the crossbar switch for a long time by the other device masters.

Any non-maskable interrupt asserts the device-level DEVCFG[CNMI] bit, which ensures that the SC1400 core can gain access to the crossbar switch, with all other masters having lower priority. In this case, the AHB error condition terminates the access to the master that timed out. This master can then issue subsequent transfers, which assert as HREADY due to the time-out. The master treats these transactions as having succeeded, but the transactions are discarded by the system. Recovery occurs through servicing the $\overline{\text{NMI}}$ interrupt.

7.2 Illegal Access Detection

Illegal accesses to restricted areas in the MSC711x memory map are detected and generate $\overline{\text{NMI}}$ interrupt requests. Illegal accesses are categorizes as follows:

- Program accesses:
  - Address out-of-range from SC1400 core (PBus) and IFU (AMIC bus)
  - Programmable address out-of-range detection on the SC1400 P bus, ASM2, and ASEMI
  - Misaligned accesses from: SC1400 core (PBus) and IFU (AMIC bus)
- Data accesses:
  - Address Out-of-Range from:
    - SC1400 XA or XB buses
    - ECI (AMEC bus)
    - DMA controller (AMDMA bus)
    - Ethernet MAC DMA (AMENT bus)
  - Programmable address out-of-range detection on:
    - SC1400 XA and XB buses
    - ASM2
    - ASEMI
  - Misaligned accesses from:
    - SC1400 XA or XB buses
    - ECI (AMEC bus)
    - DMA controller (AMDMA bus)
    - Ethernet MAC DMA (AMENT bus)
  - Writes to the boot ROM

7.2.1 Fixed Illegal Access Detection

The fixed detection units detect address out-of-range and misaligned accesses on each master port bus. Address out-of-range detection on each master port is described in **Section 5.3**, *Address Space by Type of Access*, on page 5-31. For example, if the instruction fetch unit attempts to access the APB peripheral space, this attempt is detected as an illegal access. For details on illegal access interrupts, see **Table 12-3** on page 12-6.

7.2.2 Programmable Access Detection

For effective debugging and system recovery, you can define invalid access regions on important MSC711x buses. The access detection units provide protection beyond that of the fixed address out-of-range access detection hardware so that you can define illegal accesses ranges on the basis of how the M1, M2, and DDR memories are partitioned in an application for program code, read-only data, and read-write data. These units also provide a mechanism for a ROM patch capability for the device. For details on the programmable access detection units, see **Chapter 17**, *Programmable Address Detection*.

7.2.3 Misaligned Access Detection

Misaligned accesses from the SC1400 core are discussed in **Section 4.3.2**, *Errors, Exceptions, and Events*, on page 4-8. Also, there is misaligned detection logic on all AHB master buses.

7.3 Software Watchdog Timer

The software watchdog timer (SWT) prevents system lock if the software becomes trapped in loops with no controlled exit. Hardware protects this timer from corruption, ensuring that time-outs are correctly detected. The SWT requires a special service sequence to execute periodically. Without the periodic servicing, the SWT times out and issues a reset or a non-maskable interrupt. The SWT is programmed either to reset the MSC711x device or to generate a dedicated non-maskable interrupt. SWT features are as follows:

- Disabled out of reset. Manually enabled by the user (SWTCTL[WDEN] = 1).
- 32-bit watchdog counter.
- Programmable time-out range (period).
- Counter counts down from a pre-set value to zero to indicate a time-out.
- If a time-out occurs, the watchdog timer can be programmed for one of the following operations:
  - Generate a system reset.
  - Generate an interrupt, which the service routine must clear by the time a second time-out occurs or a system reset is generated.
- Accidental restart of the SWT counter is prevented.
- Accidental disabling of the SWT is prevented.
- Pause mode with the use of external pause enable signal:
  - Optional pause upon entering the SC1400 core stop mode.
  - Pause upon entering SC1400 Debug mode or pause for single-stepping through SC1400 instructions.
- SWT can be disabled via an external pin.

7.3.1 Software Watchdog Timer Operation

The SWT is an APB slave peripheral that can prevent system lock-up caused by conflicting modules or programs in an MSC711x device. The SWT is composed of the following sub-modules:

- APB slave interface.
- Register file with read coherency for the current count register.
- Reset detection logic.
- 32-bit decrementing counter.
- Interrupt/system reset generation block.

Figure 7-2 shows the SWT block diagram.

7.3.1.1 Counter

The SWT counts from a preset time-out value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. The counter wraps to the selected time-out value and continues to decrement. You can also restart the counter to its initial value by writing to the restart register at any time. As a safety feature to prevent accidental restarts, a value of 0x76 must be written to the SWT Current Counter Value Register (SWTCCV) (see **page 7-14**).

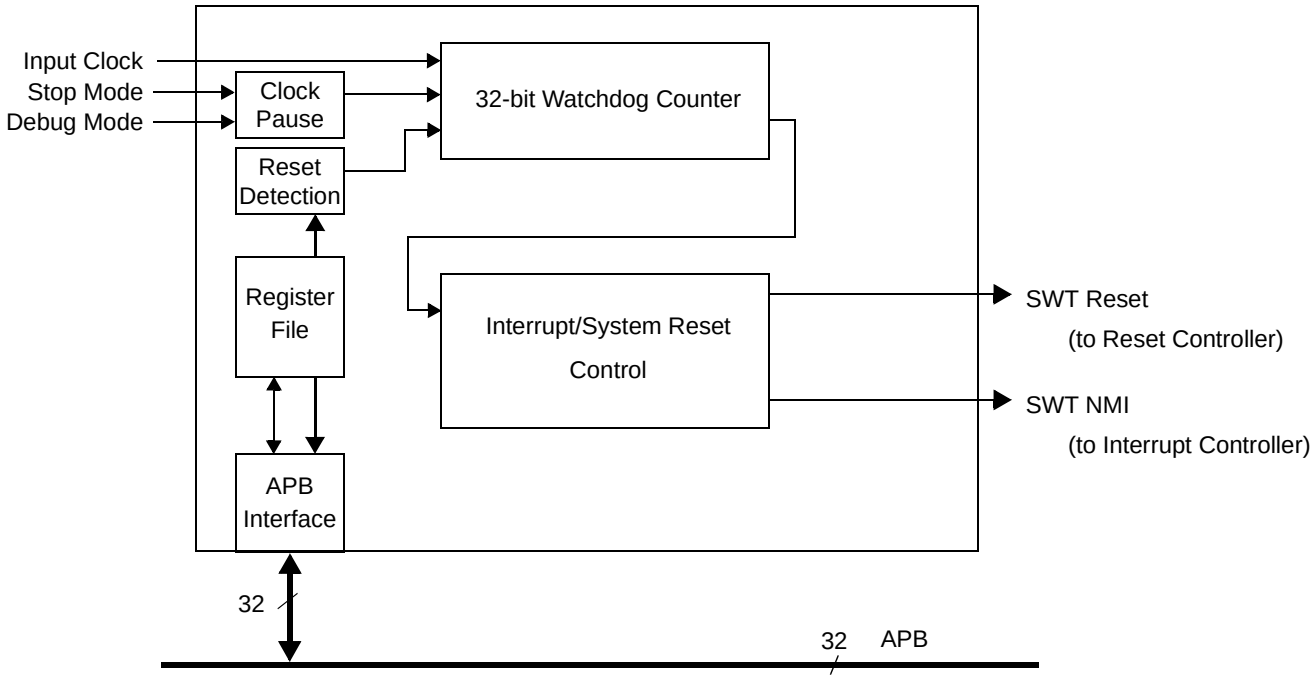


Figure 7-2. Software Watchdog Timer Block Diagram

7.3.1.2 Pause Mechanism

The SWT supports a pause mechanism which “freezes” the watchdog counter while the system is being paused. The SWT begins running when it is enabled via the SWTCTL[WDEN] bit. It is clocked with MSC711x watchdog clock. The SWT counter is paused and counting is suspended when the device enters Debug mode. Optionally, the SWT can be paused when the device enters Stop mode via the STPDIS5 or PLLSTP bits in the CLKCTL0 register (**Section 11.5**, *Clock Programming Model*, on page 11-24). Also, the development tools can pause the SWT by setting the ECI GPSCTL[INDBG] bit.

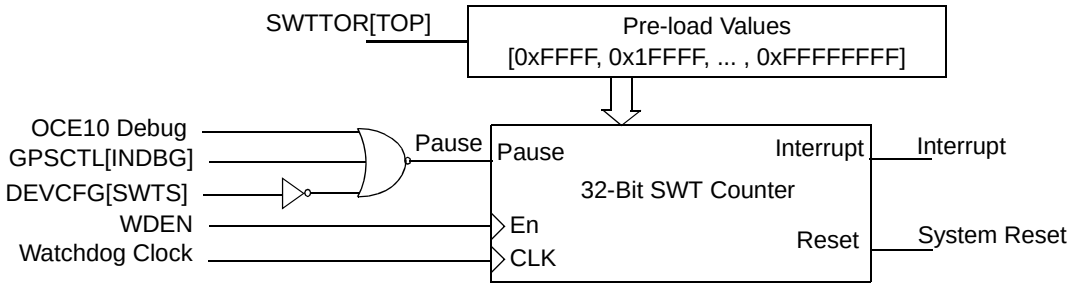


Figure 7-3. Watchdog Timer Pause Circuitry

When the SWT enters pause mode, the counter is normally not at zero, so the counter simply resumes counting when the pause is removed, and no interrupt or system reset is generated. If the counter is frozen at the zero count, no interrupt or system reset is generated. When the pause is removed, the interrupt or system reset is asserted on the next rising edge of the clock.

7.3.1.3 Interrupt and System Reset Response

The SWT can be programmed via the SWTCL[RMOD] bit discussed on **page 7-12** to generate an interrupt and then a system reset when a time-out occurs. The first time the watchdog counter expires, the SWT generates an interrupt. If it is not cleared by the time a second time-out occurs, then it generates a system reset. If a restart occurs at the same time the watchdog counter reaches zero, an interrupt is not generated.

When the SWTCL[RMOD] bit is cleared, the software watchdog timer generates a system reset when a time-out occurs. If a restart occurs when the watchdog counter reaches zero, a system reset is not generated.

7.3.2 Configuring the Watchdog Timer out of Reset

The SWT comes out of power-on reset in a disabled state. When power-on reset is deasserted, the DEVCFG[SWTS] bit (**page 7-17**) captures the value of the SWTE signal. If this bit is set, normal SWT operation occurs when the SWT is properly enabled via the SWTCTL[WDEN] bit (**page 7-14**). If the SWTS bit is cleared, the SWT remains disabled even if the WDEN enable bit is set. The SWTS bit can be written only on the deassertion of power-on reset.

Before the SWT is enabled, you must configure it for desired operation in the SWT Control Register (SWTCTL) (**page 7-12**) and SWT Time-out Range (SWTTOR) register (**page 7-13**).

When the SWT is configured via the SWTCL[RMOD] bit to generate a system reset, the SWT must be serviced before its counter reaches zero. If the SWT times out, a hard reset occurs. When the SWT is configured via the SWTCL[RMOD] bit to generate a non-maskable interrupt, the SWT must either be serviced or the SWTEOI[CLRI] bit (**page 7-15**) must be read to clear the interrupt before the counter reaches zero. If the SWT times out, a non-maskable interrupt is asserted. If a second time-out occurs before the interrupt is cleared, a hard reset occurs.

7.3.3 Servicing the Watchdog Timer

Normal SWT operation requires an application to restart the watchdog counter periodically to prevent watchdog counter time-out. To restart the watchdog counter, the program must write a value of 0x76 to the SWTCR register. The software watchdog timer can be configured in the SWTTOR register to have a user-defined time-out period range.

7.4 System Control Programming Model

The system control registers are listed as follows, along with the number of the page where each is discussed:

- Bus time-out monitor and error registers:
  - Bus Time-Out Control Register (BTMCTL), **page 7-8**.
  - Bus Error Control Register (BERRCTL), **page 7-10**.
- Software watchdog timer registers:
  - SWT Control Register (SWTCTL), **page 7-12**.
  - SWT Time-Out Range Register (SWTTOR), **page 7-13**.
  - SWT Current Counter Value Register (SWTCCV), **page 7-14**.
  - SWT Counter Restart Register (SWTCR), (**page 7-14**).
  - SWT Interrupt Status Register (SWTSTA), **page 7-15**.
  - SWT Interrupt Clear Register (SWTEOI), **page 7-15**.
- Device identification and configuration registers:
  - Device ID Register (DEVID), **page 7-16**
  - Device Configuration Register (DEVCFG), **page 7-17**.

7.4.1 Bus Time-Out Monitor and Bus Error Registers

BTMCTL Bus Time-Out Control Register BTM_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RSTEN	—								TMDASM1			—	TMDASM2		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	TMDASEMI			—	TMDASTH			—	TMDASAPB			—	TMDASSB		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BTMCTL controls the time-out period for different buses.

Table 7-1. BTMCTL Bit Descriptions

Bit	Reset	Description	Description
RSTEN 31	0	**Reset Enable** Select action for all BTMs.	0 Bus time-out asserts non-maskable interrupt. 1 Bus time-out asserts bus monitor reset.
— 30–23	0	Reserved. Write to zero for future compatibility.	
TMDASM1 22–20	0	**Time-out Delay for ASM1 Bus**	000 Detection after 31 ticks. 001 Detection after 127 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.
— 19	0	Reserved. Write to zero for future compatibility.	
TMDASM2 18–16	0	**Time-out Delay for ASM2 Bus**	000 Detection after 31 ticks. 001 Detection after 127 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.

Table 7-1. BTMCTL Bit Descriptions (Continued)

Bit	Reset	Description	Description
— 15	0	Reserved. Write to zero for future compatibility.	
TMDASEMI 14–12	0	**Time-out Delay for ASEMI Bus** The DDR memory controller is an exceptional case. When the DDR is disabled, any subsequent writes to the DDR would normally time out on ASEMI. However, the MCIF peripheral captures up to eight subsequent writes to the DDR in its write buffer, with no time-outs occurring. A time-out occurs only when there are more than eight writes.	000 Detection after 511 ticks. 001 Detection after 511 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.
— 11	0	Reserved. Write to zero for future compatibility.	
TMDASTH 10–8	0	**Time-out Delay for ASTH Bus**	000 Detection after 31 ticks. 001 Detection after 127 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.
— 7	0	Reserved. Write to zero for future compatibility.	
TMDASAPB 6–4	0	**Time-out Delay for ASAPB Bus**	000 Detection after 31 ticks. 001 Detection after 127 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.
— 3	0	Reserved. Write to zero for future compatibility.	
TMDASSB 2–0	0	**Time-out Delay for ASSB Bus**	000 Detection after 31 ticks. 001 Detection after 127 ticks. 010 Detection after 511 ticks. 011 Detection after 2047 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Disable detection.
Note: A time-out of 31 or 127 ticks is not supported on the ASEMI bus because 31 or 127 ticks is not long enough to process all possible cases.			

BERRCTL Bus Error Control Register BTM_BASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved				TMEL_AMIC			TMNE_AMIC			TMEL_AMEC			TMNE_AMEC		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				TMEL_AMDMA			TMNE_AMDMA			TMEL_AMENT			TMNE_AMENT		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-2. BERRCTL Bit Descriptions

Bit x	Reset	Description	Description
— 31–28	0	Reserved. Write to zero for future compatibility.	
TMEL_AMIC[2–0] 27–25	0	**Time-out Delay for AMIC Bus** When priority is elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
TMNE_AMIC 24–22	0	**Time-out Delay for AMIC Bus** When priority is not elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
TMEL_AMEC 21–19	0	**Time-out Delay for AMEC Bus** When priority is elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.

Table 7-2. BERRCTL Bit Descriptions (Continued)

Bit x	Reset	Description	Description
TMNE_AMEC 18–16	0	**Time-out Delay for AMEC Bus** When priority is not elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
— 15–12	0	Reserved. Write to zero for future compatibility.	
TMEL_AMDMA 11–9	0	**Time-out Delay for AMDMA Bus** When priority is elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
TMNE_AMDMA[2:0] 8–6	0	**Time-out Delay for AMDMA Bus** When priority is not elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
TMEL_AMENT 5–3	0	**Time-out Delay for AMENT Bus** When priority is elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
TMNE_AMENT[2:0] 2–0	0	**Time-out Delay for AMENT Bus** When priority is not elevated.	000 Disable detection. 001 Detection after 256 ticks. 010 Detection after 1024 ticks. 011 Detection after 4096 ticks. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.

7.4.2 Software Watchdog Timer Registers

The value of the base address for this SWT file, SWT_BASE, is provided in **Section 5.1**, *Register Base Addresses*, on page 5-4.

SWTCTL Watchdog Control Register SWT_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—														RMOD	WDEN
TYPE	R/W															
RESET	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

SWTCTL configures the watchdog timer.

Table 7-3. SWTCTL Bit Descriptions

Name	Reset	Description	Settings
— 31–2	0x0	Reserved. Write to zero for future compatibility.	
RMOD 1	1	**Output Response Mode** Selects the action performed at watchdog counter time-out.	0 Generate a reset. 1 Generate a non-maskable interrupt. If not cleared by the time a second time-out occurs, generate a reset.
WDEN 0	0	**Watchdog Timer Enable** Enables the watchdog timer when it is properly configured. When this bit is enabled, it can be cleared only by a system reset.	0 Watchdog timer disabled. 1 Watchdog timer enabled.

SWTTOR Watchdog Time-out Range Register SWT_BASE + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—												TOP			
TYPE	R/W															
RESET	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SWTTOR specifies the preload value of the watchdog counter.

Table 7-4. SWTTOR Bit Descriptions

Name	Reset	Description	Settings
— 31–4	0x0	Reserved. Write to zero for future compatibility.	
TOP 3–0	0x0	**Time-out Period Count** Preload value for watchdog counter.	

TOP	32-bit Preload Value
0000	0x0000 FFFF
0001	0x0001 FFFF
0010	0x0003 FFFF
0011	0x0007 FFFF
0100	0x000F FFFF
0101	0x001F FFFF
0110	0x003F FFFF
0111	0x007F FFFF
1000	0x00FF FFFF
1001	0x01FF FFFF
1010	0x03FF FFFF
1011	0x07FF FFFF
1100	0x0FFF FFFF
1101	0x1FFF FFFF
1110	0x3FFF FFFF
1111	0x7FFF FFFF

SWTCCV Watchdog Current Counter Value Register SWT_BASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CCVR															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CCVR															
TYPE	R/W															
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

SWTCCV reads the current value of the watchdog counter.

Table 7-5. SWTCCV Bit Descriptions

Name	Reset	Description	Settings
CCVR 31–0	0x0000FFFF	**Current Counter Contents** Reading this register gives the current value of the internal watchdog counter.	

SWTCR Watchdog Counter Restart Register SWT_BASE + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—								RSTVAL							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SWTCR triggers the watchdog timer, which restarts the watchdog counter.

Table 7-6. SWTCR Bit Descriptions

Name	Reset	Description	Settings
— 31–8	0x0	Reserved. Write to zero for future compatibility.	
RSTVAL 7–0	0x0	**Restart Value** Writing the value of 0x76 to this field restarts the watchdog counter. This field is read as 0x00.	

SWTSTA Watchdog Interrupt Status Register SWT_BASE + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															STAT
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SWTSTA indicates whether a watchdog interrupt has occurred.

Table 7-7. SWTSTA Bit Descriptions

Name	Reset	Description	Settings
— 31–1	0x0	Reserved. Write to zero for future compatibility.	
STAT 0	0	**Time-out Interrupt** Reads as a 1 when a non-maskable interrupt occurs upon watchdog counter time-out.	0 No interrupt. 1 Interrupt.

SWTEOI Watchdog End of Interrupt Register SWT_BASE + 0x14

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															CLRI
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SWTEOI clears a pending watchdog interrupt.

Table 7-8. SWTEOI Bit Descriptions

Name	Reset	Description	Settings
— 31–1	0x0	Reserved. Write to zero for future compatibility.	
CLRI 0	0	**Clear Time-out Interrupt** Reading this register clears the watchdog time-out interrupt. This bit reads as a zero.	

7.4.3 Device Identification and Configuration

DEVID Device ID Register BTM_BASE + 0x88

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	VER				FSLID						DEVNBR					
Type	R															
Reset																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DEVNBR				FSLMFRID											1
Type	R															
Reset	Device-Dependant				0	0	0	0	0	0	0	0	1	1	1	0

DEVID provides information to identify the particular MSC711x device.

Table 7-9. DEVID Bit Descriptions

Name	Reset	Description
VER 31–28	xxxx????? xxxx	**Version** Contains the version number for the device.
FSLID 27–22	0x21	**Freescale Design Center ID** Specifies design center within Freescale: 100001
DEVNBR 21–12	0x21	**Device Number** Identifies the device among the different family members: • MSC7110 = 0x3 • MSC7112 = 0x7 • MSC7113 = 0x6 • MSC7115 = 0x1 • MSC7116 = 0x2 • MSC7118 = 0x9 • MSC7119 = 0xA
FSLMFRID 11–1	0x01B	**Manufacturer Identity** Freescale = 0b 00000001110.
— 0	1	This bit is hardwired to a value of 1.

DEVCFG Device Configuration Register BTM_BASE + 0x80

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	HCOV	ENTP	CNMI	—	SWTS	DDR	—	PAS	—	—	PDS
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	*	0	0	0	0	0	0

* These bits are sampled from MSC711x pins at power-on reset and loaded here.

Table 7-10. DEVCFG Bit Descriptions

Name	Reset	Description	Settings
— 31–11	0	Reserved. Write to zero for future compatibility.	
HCOV 10	0	**HCS2 Override** Determines whether the HDI module uses the HCS2 pin. When this bit is cleared, the host port is selected via both the HCS[1–2] pins.When this bit is set, the host port is selected with only the HCS1 pin. The HDI port ignores the value on the HCS2 pin.	0 HCS2 pin is available to the HDI port. 1 HCS2 pin is not available to the HDI port.
ENTP 9	0	**Ethernet MAC Priority Level** Control bit for raising the access priority for all Ethernet MAC accesses through the crossbar switch.	0 Priority is not elevated on all accesses from the Ethernet MAC. 1 Priority is elevated on all accesses from the Ethernet MAC.
CNMI 8	0	**Device NMI Detected** Indicates a device-level non-maskable interrupt. When this bit is set, the priority elevation signals on the AMEC and AMIC buses are forced to an asserted state, while the priority elevation signal on all other master buses is forced to a deasserted state. Software cannot set this bit, but it can be cleared by writing it with a value of 0. CNMI must be cleared only *after* all device non-maskable interrupt requests are cleared in the NMIPR.	0 No device-level NMI detected. 1 Device-level NMI detected.
— 7	0	Reserved. Write to zero for future compatibility.	
SWTS 6	0	**SWTE Pin Status** Contains the value of the SWTE signal when sampled at power-on reset. SWTS is used to disable the software watchdog timer, as described in **Section 7.3.2**, *Configuring the Watchdog Timer out of Reset,* on page 7-6. This bit can be loaded only when power-on reset is deasserted on exit from power-on reset.	0 Signal deasserted. 1 Signal asserted.

Table 7-10. DEVCFG Bit Descriptions (Continued)

Name	Reset	Description	Settings
DDR 5	0	**DDR Number of Pins Select** A write-once bit that determines whether the DDR interface is configured for 16- or 32-pin operation. When this bit is set after reset, it can no longer be reset.	0 Selects 16-pin DDR Interface. 1 Selects 32-pin DDR Interface.
— 4	0	Reserved. Write to zero for future compatibility.	
PAS 3	0	**Port A Select** When the signal is not configured as GPIO, this bit selects between the secondary and additional functions of the pin. PAS affects only a subset of the port pins. For a listing of pins affected by PAS, see **Table 7-11**. PAS does not affect the CLKO pin, which is selected when it is enabled in the Clock Control Register and overrides the value of this bit. Refer to **Section 24.3.2.2**, *Port A,* on page 24-4 and to **Table 24-1**, *Port A GPIO Signal Pin Assignments for Devices With Ethernet MAC,* on page 24-6.	0 Selects the secondary function of the signal. 1 Selects the additional multiplexing function of the signal.
— 2–1	0	Reserved. Write to zero for future compatibility.	
PDS 0	0	**Port D Select** When the pins are not configured as GPIO, this bit selects between the secondary and additional functions of the signals. PDS affects only a subset of the port pins. For a listing of pins affected by PDS, see **Table 7-11**. Refer to **Section 24.3.2.5**, *Port D,* on page 24-9 and to **Table 24-5**, *Port D GPIO Signal Pin Assignments for Devices With an Ethernet MAC,* on page 24-10.	0 Selects the secondary function of the signal. 1 Selects the additional multiplexing function of the signal.

The PAS and PDS bits affect only a subset of the signals for each port, as listed in **Table 7-11**. Signals not listed in this table are always configured for secondary functions when they are not configured as GPIO. Similarly, signals that are always sampled out of reset (SWTE, BM1, BM0, and H8BIT) are not affected.

Table 7-11. Pins Affected

Control Bit	Port	Affected Bits	Settings
PAS	A	29–27	0 Selects MII functionality of signals: RXD3, TX_ER, TXD3. 1 Selects TDM2 functionality of signals: T2TFS, T2RD, T2RCK.
PDS	D	6–4	0 Selects MII functionality of signals: RXD2, RXCLK, TXD2. 1 Selects TDM2 functionality of signals: T2TD, T2TCK, T2RFS.

DMA Controller 8

The DMA controller performs complex data transfers on 32 programmable channels with minimal intervention from a host processor. The hardware micro-architecture includes a DMA engine that calculates the source and destination addresses and moves the data and a local memory containing the transfer control descriptors (TCD) for the channels. This SRAM-based implementation minimizes overall module size. For best use of the DMA controller, you should read not only this chapter but also **Section A.1.6**, *DMA Controller*, on page A-4.

8.1 Features

Features of the DMA controller are as follows:

- A 64-bit data path.
- All data movement via dual-address transfers: read from source, write to destination. The source, destination addresses, transfer size are programmable, and there is support for enhanced addressing modes.
- TCD to handle two-deep, nested transfer operations:
  - An *inner* data transfer loop defined by a minor byte transfer count
  - An *outer* data transfer loop defined by a major iteration count
- Channel activation via one of three methods, all of which require one activation per execution of the minor loop:
  - Explicit software initiation.
  - Initiation via a channel-to-channel linking mechanism for continuous transfers (independent channel linking at the end of the minor loop and/or major loop).
  - Peripheral-initiated hardware requests, one per channel.
- Fixed-priority and round-robin channel arbitration.
- Channel completion reported via optional interrupt requests:
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error termination is optionally enabled per channel and logically summed together to form a small number of error interrupt outputs.
- Optional scatter/gather DMA processing.

8.2 DMA Architecture

The DMA controller is for applications in which the size of the data transfer is statically known and is *not* defined within the data packet itself (see **Figure 8-1**). The DMA hardware supports:

- Connections to the AMDMA-AHB crossbar switch for bus mastering the data movement and to the IPBus interface for programming the module.
- 32-byte TCD per channel stored in local memory.
- 32 bytes of data registers to store data temporarily for burst transfers.

Throughout this chapter, data sizes are defined as byte (8-bit), half word (16-bit), word (32-bit) and double word (64-bit).

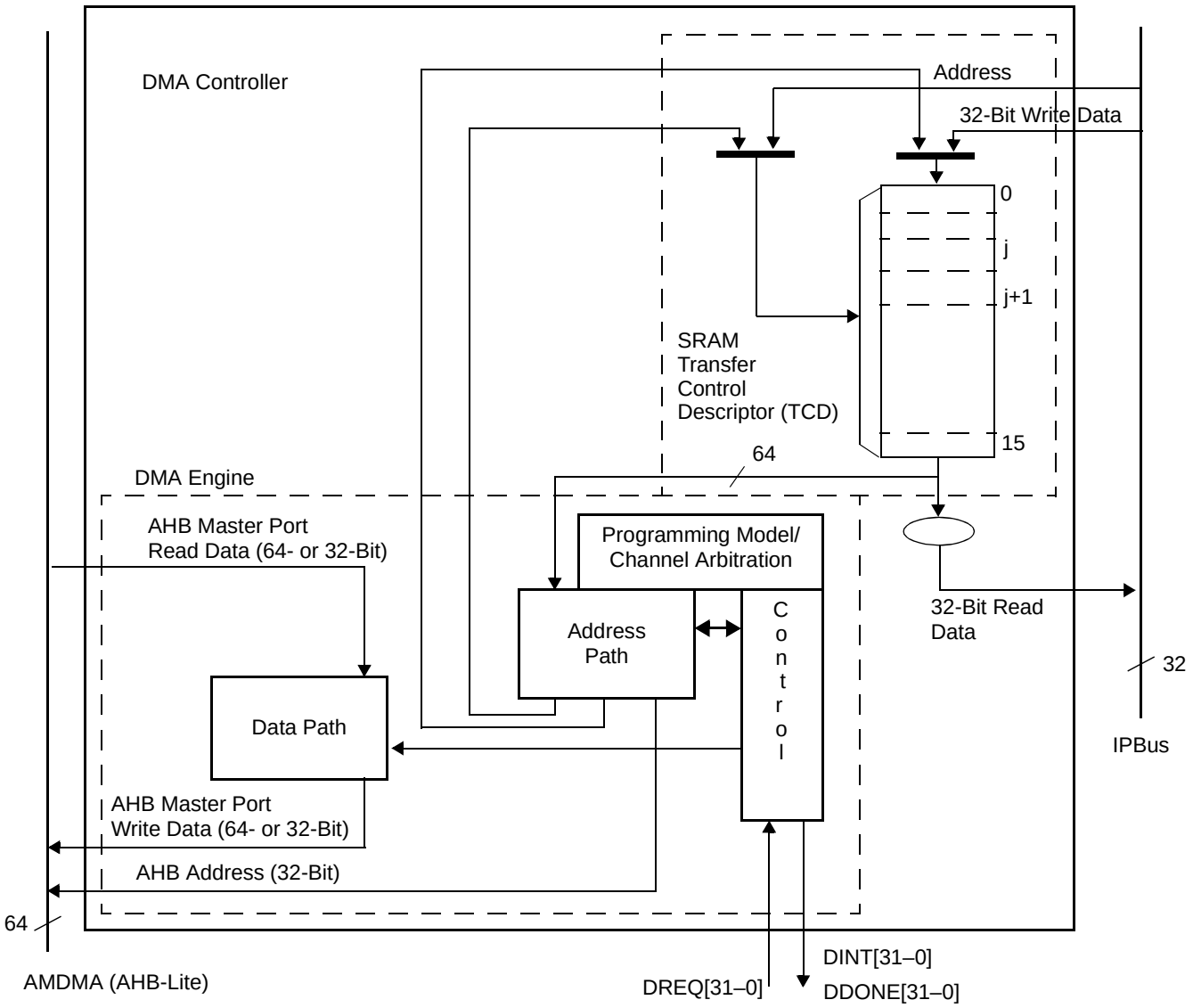


Figure 8-1. DMA Controller Block Diagram

8.2.1 DMA Engine

The DMA engine is partitioned into an address path, a data path, a module for the programming model and channel arbitration, and control functions.

The address path contains registered versions of two channel transfer control descriptors: channel *x* and channel *y*, and calculates the master bus addresses. All channels provide the exact same functionality. The data transfers in one channel can be preempted after a read/write sequence completes if a higher-priority channel is activated while the first channel is active. Therefore, a large data move operation can be preempted to minimize the time another channel is blocked from execution (optionally enabled by DCHPRIx[ECP] (see **Table 8-22**, *DCHPRIx Bit Descriptions*, on page 8-39). When any channel is activated, the contents of its TCD are read from local memory. When execution of the inner minor loop completes, the new values of the TCD registers are written back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDx-5[CITER] field, and a possible fetch of the next TCD*x* from memory as part of a scatter/gather operation.

The bus master read/write data path includes 32 bytes of register storage to match the maximum transfer size and the necessary multiplex logic to support any required data alignment. The AMBA-AHB read data bus is the primary input, and the AHB write data bus is the primary output. The address and data path modules directly support the 2-stage pipelined AMBA-AHB bus. The address path represents the first stage of the bus pipeline, and the data path is the second stage of the pipeline.

The programming model registers are connected to the IPBus. The DREQ register inputs and DINT register outputs also connect to this module via the control logic. The control module provides the control functions for the DMA engine. For data transfers with equal source and destination sizes, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count are transferred. For descriptors of unequal sizes, multiple accesses to the smaller data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, followed by one 32-bit write.

8.2.2 Transfer Control Descriptor (TCD)

TCD local memory includes a memory controller and a memory array. The dual-ported memory controller handles accesses from both the DMA engine as well as references from the IPBus. For simultaneous accesses, the DMA engine has priority, and the IPBus transaction stalls. The TCD memory array is a single-ported, synchronous compiled RAM memory array.

The structure of the DMA transfer control descriptor is fundamental to the operation of the DMA module. It is defined as follows in a C pseudo-code specification. `int` refers to a 32-bit variable

unless noted otherwise, and **short** is a 16-bit variable. To compile these structures, change any periods (.) in the variable name to underscores (_).

```
typedef union {
    struct {                                /* CITERE = 1 */
        unsigned short CITERH:6;            /* link channel number, */
        unsigned short citer:9;             /* current (“major”) iteration count */
    } minor_link_enabled;                   /* channel link at end of the minor loop */
    struct {                                /* CITERE = 0 */
        unsigned short citer:15;            /* current (“major”) iteration count */
    } minor_link_disabled;                  /* no linking at end of the minor loop */
} t_minor_link_citer;


typedef union {
    struct {                                /* BITERE = 1 */
        unsigned short BITERH:6;            /* link channel number, */
        unsigned short biter:9;             /* beginning (“major”) iteration count */
    } init_minor_link_enabled;              /* channel link at end of the minor loop */
    struct {                                /* BITERE = 0 */
        unsigned short biter:15;            /* beginning (“major”) iteration count */
    } init_minor_link_disabled;             /* no linking at end of the minor loop */
} t_minor_link_biter;

typedef struct {
     unsigned int        saddr;             /* source address */
     unsigned int        smod:5;            /* source address modulo */
     unsigned int        ssize:3;           /* source transfer size */
     unsigned int        dmod:5;            /* destination address modulo */
     unsigned int        dsize:3;           /* destination transfer size */
     short               soff;              /* signed source address offset */
     unsigned int        nbytes;            /* inner (“minor”) byte count */
     int                 slast;             /* last source address adjustment */
     unsigned int        daddr;             /* destination address */
     unsigned short      CITERE:1;          /* enable channel linking on minor loop */
     t_minor_link_citer  minor_link_citer;  /* conditional current iteration count */
     short               doff;              /* signed destination address offset */
     int                 dlast_sga;         /* last destination address adjustment, or
                                               scatter/gather address (if ESG = 1) */
     unsigned short      BITERE:1;          /* beginning channel link enable */
     t_minor_link_biter  minor_link_biter;  /* beginning (“major”) iteration count */
     unsigned int        bwc:2;             /* bandwidth control */
     unsigned int        LCNUM:6;           /* link channel number */
     unsigned int        done:1;            /* channel done */
     unsigned int        active:1;          /* channel executing */
     unsigned int        CLE:1;             /* enable channel linking on major loop*/
     unsigned int        ESG:1;             /* enable scatter/gather descriptor */
     unsigned int        DREQ:1;            /* disable ipd_req when done */
     unsigned int        INTH:1;            /* interrupt on citer = (biter >> 1) */
```

```
    unsigned int        INTM:1;          /* interrupt on major loop completion */
    unsigned int        start:1;         /* explicit channel start */
} tcd                                    /* transfer_control_descriptor */
```

8.3 Data Transfer Overview

The DMA controller performs single transfers of 8, 16, 32, or 64 bits. Bursts are 4 beats of 64 bits (32 bytes). The DMA controller requires correct alignment in order to initiate a transfer. This applies to the addresses and offsets of both the source and destination. **Table 8-1** summarizes this alignment requirement for the different types of data transfers.

Table 8-1. DMA Alignment Requirements

Transfer Size	AHB Burst Type	Address or Offset: Must be aligned on a	Comments
8 bits	Single	Byte boundary	—
16 bits	Single	2 byte boundary	—
32 bits	Single	4 byte boundary	—
64-bits	Single	8 byte boundary	—
4 × 64-bits (32 bytes)	WRAP4	32 byte boundary	Although a WRAP4 is issued, wrapping is not supported. MSC711x devices require the start address to be aligned to the total number of bytes in the burst. Therefore, although the DMA controller issues a WRAP4 burst attribute, this is equivalent to an INCR4 burst attribute.
Note: This table presents all legal transactions permitted on the MSC711x DMA controller.			

If a transfer is not correctly aligned as described in **Table 8-1**, the DMA issues one of the following errors, which are indicated in the DMA Error Status Register (DMAES) (see **Table 8-9**, *DMAES Bit Descriptions*, on page 8-29):

- Source address error
- Source offset error
- Destination address error
- Destination offset error

The range of addresses used by the DMA controller to access M1 memory over the AMDMA bus differs from the range of addresses used by the SC1400 core to access these same locations. For example, an SC1400 core access to address 0x0000 0000 accesses the same location that the DMA accesses at address 0x0180 0000. See **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4, which demonstrates this difference, and **Table 5-2**, *MSC711x Detailed Memory Map*, on page 5-5, which shows the correct address ranges.

8.3.1 Channel Assignments

Table 8-2 shows the MSC711x channel assignments.

Table 8-2. DMA Channel Assignments

Group 0 Channels	
DMA Channel	**Assigned to Channel**
0	TDM0 TX
1	TDM0 RX
2	TDM1 TX
3	TDM1 RX
4	HDI16 TX
5	HDI16 RX
6	Available for memory transfers
7	Available for memory transfers
8	Available for memory transfers
9	Available for memory transfers
10	Available for memory transfers
11	Available for memory transfers
12	TDM2 TX
13	TDM2 RX
14	EVDMA0
15	EVDMA1

Group 1 Channels	
DMA Channel	**Assigned to Channel**
16	Available for memory transfers
17	Available for memory transfers
18	Available for memory transfers
19	Available for memory transfers
20	Available for memory transfers
21	Available for memory transfers
22	Available for memory transfers
23	Available for memory transfers
24	Available for memory transfers
25	Available for memory transfers
26	Available for memory transfers
27	Available for memory transfers
28	Available for memory transfers
29	Available for memory transfers
30	Available for memory transfers
31	Available for memory transfers

Note: DMA transfers to/from the Ethernet MAC are not performed by this DMA controller but are instead handled by a dedicated DMA unit within the Ethernet MAC.

8.3.2 DMA Arbitration

The 32 DMA channels are organized into the following groups:

- Group 0: Channels 0 through 15
- Group 1: Channels 16 through 31

There is a group arbitration scheme and a channel arbitration scheme. Group arbitration is selected with the DMACR[ERGA] bit (see **Table 8-8**, on page 8-27). Group arbitration is handled as either fixed-priority or round robin-priority. The priority of a group programmed for fixed-priority arbitration is set with the DMACR[GRP*x*PRI] bit (see **Table 8-8**, on page 8-27). Each group must be programmed with a unique priority. Different groups cannot have the same priority level.

8.3.2.1 Channel Arbitration within a Group

When the arbiter selects a group, the channel arbitration scheme is selected with the DMACR[ERCA] bit (see **Table 8-8**, on page 8-27). For fixed-priority arbitration, one of 16 priority levels is assigned to each channel in the DMA Channel Priority Registers (DCHPRIx) discussed on **page 8-39**. Each channel must be programmed with a unique priority. Different channels cannot have the same priority level. When all channels are programmed for fixed-priority arbitration via the DMACR[ERCA] bit, each channel can be individually programmed to enable or disable preemption by a higher-priority channel. If preemption is enabled, a request from a higher-priority channel preempts the current active channel and switches to the new one. If preemption is disabled, higher priority channels gain control only upon completion of the current channel's minor loop. Nested preemption is not permitted. That is, a DMA channel that has preempted another channel cannot be preempted. Preemption is available only in fixed-priority mode. It is not available in round-robin mode. **Table 8-3** shows the permitted scenarios for configuring DMA arbitration.

Table 8-3. Arbitration Capabilities

Scenario	Group Arbitration	Channel Arbitration	Preemption Allowed?	Description
1	Round robin	Fixed priority	No	When there are one or more DMA requests from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced from the highest group number to the lowest group number. After a channel request is serviced, the group round-robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or just skipping a group if it has no pending requests. If a channel requests service at a rate that equals or exceeds the round-robin service rate, that channel is always serviced before lower-priority channels in the same group, and the lower-priority channels may never receive service. The advantage of this scenario is that no one group can consume all the DMA bandwidth. The highest-priority channel selection latency is potentially greater than that for fixed/fixed arbitration. However, excessive request rates on high-priority channels could prevent servicing of lower-priority channels in the same group.
2	Round robin	Round robin	No	Groups are serviced as in scenario 1, except that channels are serviced in channel number order. Only one channel is serviced from each requesting group for each round robin pass through the groups. Within each group, channel service starts at the highest channel number and rotates through to the lowest channel number without regard to channel priority levels. Any channel that generates DMA requests more quickly than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests not serviced are simply lost, but at least one channel is serviced. All channels are treated equally. Priority levels are not used in round robin/round robin mode. This scenario guarantees that all channels get service at some point, regardless of the request rates. However, the potential latency could be quite high.

Table 8-3. Arbitration Capabilities (Continued)

Scenario	Group Arbitration	Channel Arbitration	Preemption Allowed?	Description
3	Fixed priority	Round robin	No	The highest-priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups. Within each group, channel service starts with the highest channel number and rotates through to the lowest channel number, without regard to channel priority levels assigned within the group. This scenario can cause the same bandwidth consumption problem as indicated in scenario 1, but all the channels in the highest-priority group get service. Service latency is short on the highest-priority group, but it can increase as the group priority decreases.
4	Fixed priority	Fixed priority	Yes	The channel service request from the highest-priority channel in the highest-priority group is selected to execute. If the DMA is programmed so that channels within one group use fixed priorities and that group is assigned the highest fixed priority of all groups, that group can take all the bandwidth of the DMA controller. That is, no other groups are serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need quick service. Preemption is available only in this scenario.

8.3.2.2 Prioritization through the Crossbar Switch

The DMA controller also has a programmable priority for accessing the crossbar switch. Each slave port of the switch individually programs the DMA priority via the AMDMA bus (see **Section 6.4**, *Crossbar Switch Programming Model*, on page 6-18). The DMA controller uses these programmable priorities for normal DMA transactions on the AMDMA bus through the switch. The DMA controller can elevate its priority through the switch for a particular TCD within a DMA channel by programming the bandwidth control bits for the desired TCD as shown in **Table 8-32**, *TCDx-7 Bit Descriptions*, on page 8-49. Elevating the priority raises the AMDMA master above all other crossbar master ports with no elevated priorities. Each time a DMA channel is activated by reloading its parameters from the DMA local memory, the first transaction does not have elevated priority for the first cycle while the DMA loads the priority field from the TCD memory. In some cases, an additional cycle may be required before priority elevation occurs. Then, the priority of all subsequent cycles for the specific channel is elevated.

8.3.3 DMA Interrupt Vectors

The MSC711x architecture allocates interrupt vectors differently for different DMA channels. This may be important when DMA channels are allocated within an application:

- Channels 0–7. Each has its own dedicated interrupt vector.
- Channels 8–16. Each shares an interrupt vector with one other DMA channel.
- Channels 16–31. All of these channel share a single interrupt vector

8.4 Channel Operation and Data Flow

This section covers channel operation and overall DMA data flow.

8.4.1 Channel Operation

1. A channel is initialized when software loads the transfer control descriptor (TCD) into the DMA programming model, which is memory-mapped through the IPBus space and implemented as local memory.
2. The channel is activated either explicitly by software, by a peripheral request, or by channel linking.

   One iteration of the major loop is executed per activation. One iteration of the major loop is equal to the complete execution of the minor loop. The number of iterations in the minor loop varies and can be computed by dividing the value of the TCDx-2[NBYTES] field by the larger of the TCDx-1[SSIZE, DSIZE] field values. For example, transferring 16 bytes, source size is byte, destination size is word:

   minor loop iterations = 16 bytes/larger[1 byte, 4 bytes] = 4 iterations of the sequence [byte read, byte read, byte read, byte read → word write].
3. The contents of the TCD for the activated channel are read from local memory and loaded into the DMA engine registers.
4. The DMA engine executes the data transfer defined by the inner minor loop, reading from the source and writing to the destination.
5. After the minor loop executes, certain TCD fields are restored in local memory.

Steps 2–5 repeat until the outer major loop iteration count is exhausted. Then additional processing steps are completed, such as the optional assertion of an interrupt request signaling the transfer's completion, final adjustments to the source and destination addresses, and so on. For more information, see **Section 8.4.3**, *Pseudo-Code Description of DMA Channel Processing*, on page 8-12 and **Section 8.7**, *DMA Programming Model*, on page 8-25.

8.4.2 DMA Data Flow

The flow of a DMA data transfer is partitioned into channel activation, data movement, and field updates.

8.4.2.1 Channel Activation

As **Figure 8-2** shows, a peripheral sets the TCDx-7[START] or DREQ bit to activate a channel. In the next cycle, channel arbitration is performed, either fixed-priority or round-robin. Then the activated channel number is sent through the address path and converted into the required address to access TCD local memory. When TCD memory is accessed, the TCD for the activated channel

is read from local memory and loaded into the DMA engine channel registers. TCD memory is 64-bits wide to minimize the time needed to fetch and load the activated channel's descriptor.

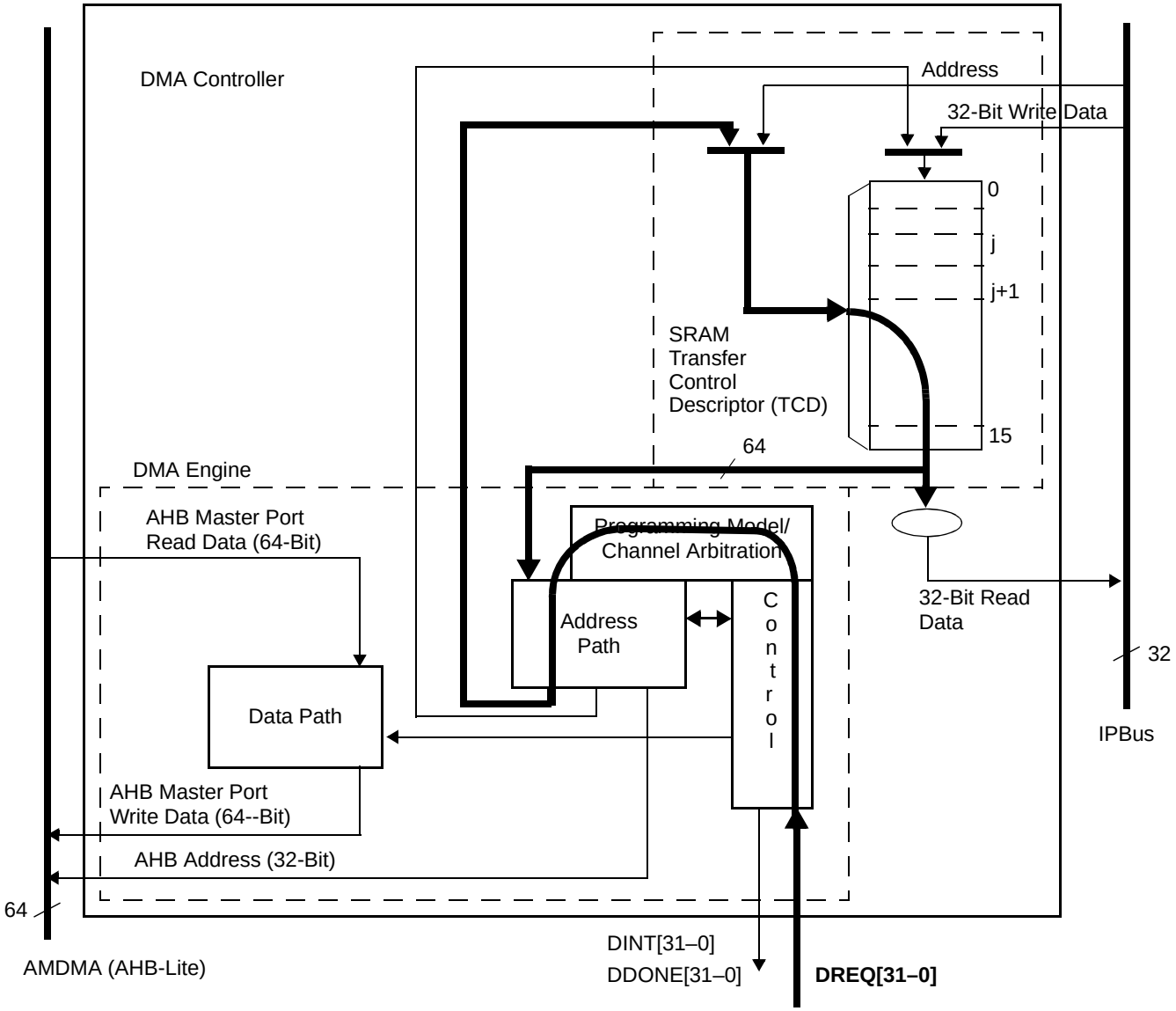


Figure 8-2. Channel Activation

8.4.2.2 Data Movement

Figure 8-3 shows the modules associated with the data transfer sequence through the required source reads and destination writes to move the data. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the AMBA-AHB bus during the destination write. This source read/destination write processing continues until it reaches the end of the inner minor byte count.

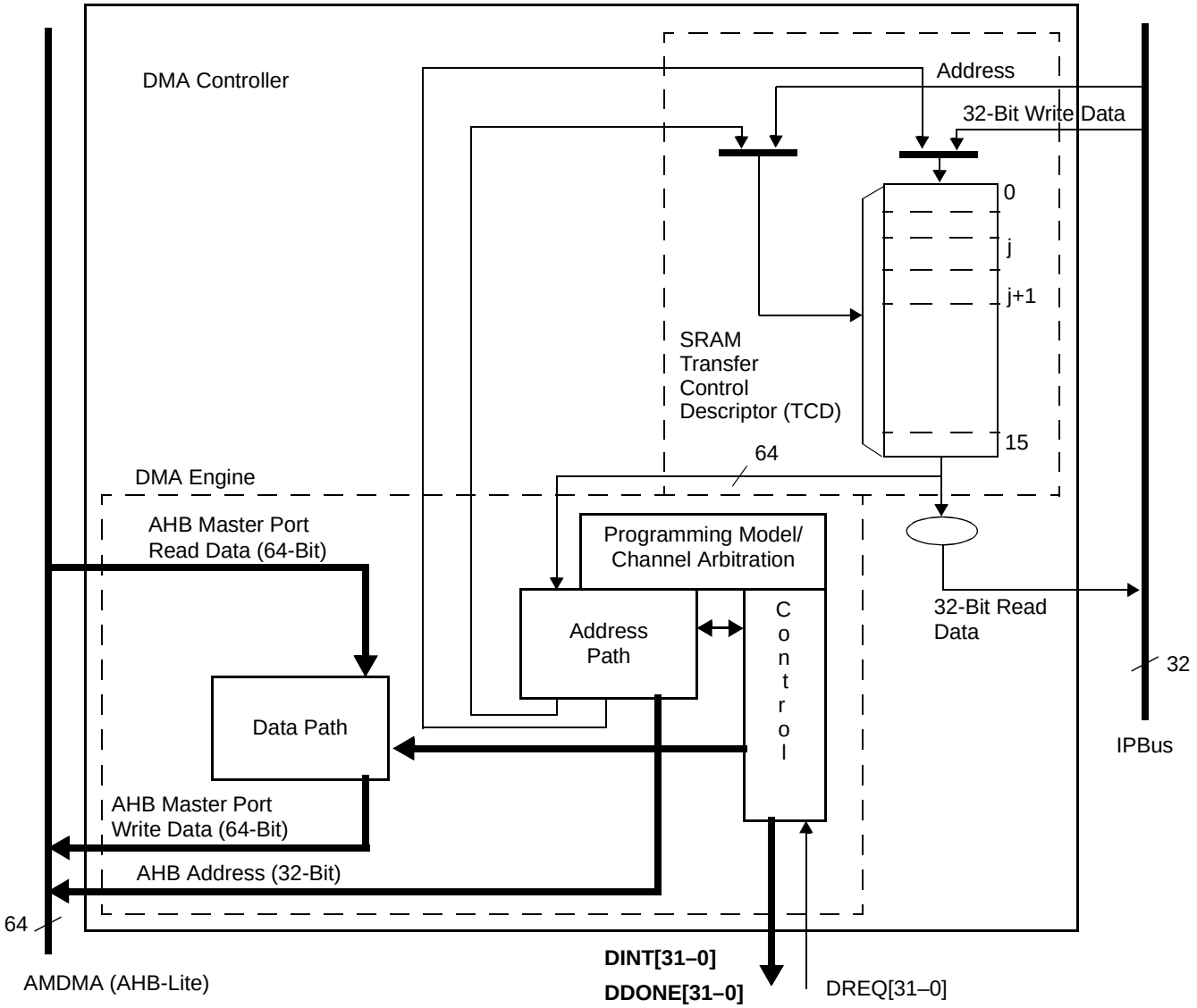


Figure 8-3. Data Movement

8.4.2.3 Field Updates

The final phase of the DMA data flow is the required updates to certain fields in the channel's TCD, for example, the TCD0[SADDR], TCDx-4[DADDR], and TCDx-5[CITER] fields described in **Section 8.7.2**, *Transfer Control Descriptor (TCD) Registers*, on page 8-40. If the outer major iteration count is exhausted, additional operations are performed, including final address adjustments and updating the TCDx-5[CITER] field. Additionally, an optional interrupt request is asserted, as well as a possible fetch of a new TCD from main memory using the scatter/gather address pointer in the descriptor. Updates to TCD memory and the assertion of an interrupt request are shown in **Figure 8-4**.

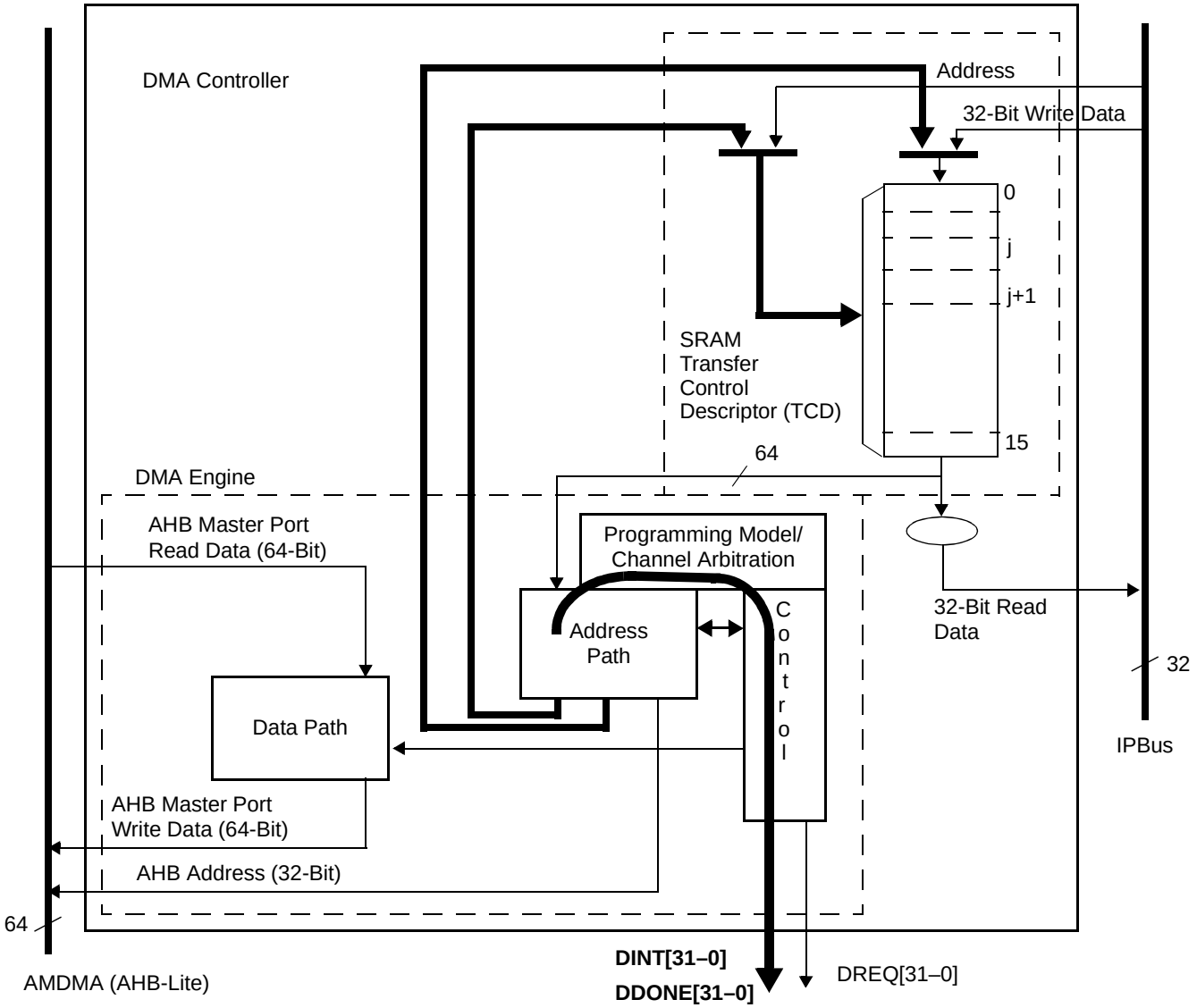


Figure 8-4. Field Updates

8.4.3 Pseudo-Code Description of DMA Channel Processing

The pseudo-code in this section describes channel processing in detail. This simplified example represents the basic data transfers. Detailed processing associated with the error handling is omitted.

```
/* the given DMAchannel is requesting service by the software assertion of the
   tcd[channel].start bit, the assertion of an enabled ipd_req from a device, or
   the implicit assertion of a channel-to-channel link */

/* begin by reading the transfer control descriptor from the local RAM
   into the local dma_engine registers */
dma_engine        = read_from_local_memory [channel];
dma_engine.active = 1;                  /* set active flag */
dma_engine.done   = 0;                  /* clear done flag */
```

```
/* check the transfer control descriptor for consistency */
if (dma_engine.config_error == 0) {

/ * begin execution of the inner "minor" loop transfers */
    while (dma2_engine.nbytes > 0) {
        dma2_engine.active = 1;          /* set   active flag */
        dma2_engine.done   = 0;          /* clear done   flag */

        /* convert the source transfer size into a byte count */
        switch (dma_engine.ssize) {
        case 0:                          /* 8-bit transfer */
            src_xfr_size = 1;
            break;
        case 1:                          /* 16-bit transfer */
            src_xfr_size = 2;
            break;
        case 2:                          /* 32-bit transfer */
            src_xfr_size = 4;
            break;
        case 3:                          /* 64-bit transfer */
            src_xfr_size = 8;
            break;
        case 4:                          /* 16-byte burst transfer */
            src_xfr_size = 16;
            break;
        case 5:                          /* 32-byte burst transfer */
            src_xfr_size = 32;
            break;
        }

        /* convert the destination transfer size into a byte count */
        switch (dma_engine.dsize) {
        case 0:                          /* 8-bit transfer */
            dest_xfr_size = 1;
            break;
        case 1:                          /* 16-bit transfer */
            dest_xfr_size = 2;
            break;
        case 2:                          /* 32-bit transfer */
            dest_xfr_size = 4;
            break;
        case 3:                          /* 64-bit transfer */
            dest_xfr_size = 8;
            break;
        case 4:                          /* 16-byte burst transfer */
            dest_xfr_size = 16;
            break;
        case 5:                          /* 32-byte burst transfer */
            dest_xfr_size = 32;
            break;
```

```
}

/* determine the larger of the two transfer sizes, this value reflects */
/* the number of bytes transferred per read->write sequence. */
/* number of iterations of the minor loop = nbytes / xfer_size */
if (dma_engine.ssize < dma_engine.dsize)
    xfr_size = dest_xfer_size;
 else
    xfr_size = src_xfer_size;

/* process the source address, READ data into the buffer*/

/* read "xfr_size" bytes from the source */
/* if the ssize < dsize, do multiple reads to equal the dsize */
/* if the ssize => dsize, do a single read of source data */
number_of_source_reads = xfer_size / src_xfer_size;

for (number_of_source_reads) {
    dma_engine.data = read_from_amba-ahb (dma_engine.saddr, src_xfr_size);

    /* generate the next-state source address */
    /* sum the current saddr with the signed source offset */
    ns_addr = dma_engine.saddr + (int) dma_engine.soff; }

    /* if enabled, apply the power-of-2 modulo to the next-state addr */
    if (dma_engine.smod != 0)
        address_select = (1 << dma_engine.smod) - 1; }
    else
        address_select = 0xffff_ffff;

    dma_engine.saddr = ns_addr           &  address_select
                     | dma_engine.saddr & ~address_select; }
}


/* process the destination address, WRITE data from buffer */

/* write "xfr_size" bytes to the destination */
/* if the dsize < ssize, do multiple writes to equal the ssize */
/* if the dsize => ssize, do a single write of dest data */
number_of_dest_writes = xfer_size / dest_xfer_size;

for (number_of_dest_writes) {
    write_to_amba-ahb (dma_engine.daddr, dest_xfr_size) = dma_engine.data;

    /* generate the next-state destination address */
    /* sum the current daddr with the signed destination offset */
    ns_addr = dma_engine.daddr + (int) dma_engine.doff;

   /* if enabled, apply the power-of-2 modulo to the next-state dest addr */
    if (dma_engine.dmod != 0)
        address_select = (1 << dma_engine.dmod) - 1;
```

```
        else
            address_select = 0xffff_ffff;

        dma_engine.daddr = ns_addr           &  address_select
                         | dma_engine.daddr & ~address_select;
    }

    /* check for a higher priority channel to service if: */
    /* 1) preemption is enabled */
    /* 2) in fixed arbitration mode */
    /* 3) a higher priority channel is requesting service */
    /* 4) not already servicing a preempting channel */
    if ((DCHPRIn.ecp = 1)  &  fixed_arbitration_mode
        higher_pri_request & ~current_channel_is_preempt)
        service_preempt_channel;

    /* the bandwidth control field determines when the next read/write occurs */
    if (dma_engine.bwc > 1)
        stall_dma_engine (1 << dma_engine.bwc);

    /* decrement the minor loop byte count */
    dma_engine.nbytes = dma_engine.nbytes - xfr_size;


}   /* end of minor inner loop */

dma_engine.citer--;            /* decrement major loop iteration count */

/* if the major loop is not yet exhausted, update certain TCD values in the RAM */
if (dma_engine.citer != 0) {
    write_to_local_memory [channel].saddr = dma_engine.saddr;
    write_to_local_memory [channel].daddr = dma_engine.daddr;
    write_to_local_memory [channel].citer = dma_engine.citer;

    /* if minor loop linking is enabled, make the channel link */
    if (dma_engine.citer.e_link)
        TCD[citer.linkch].start = 1;     /* specified channel service req */

    /* check for interrupt assertion if half of the major iterations are done */
    if (dma_engine.int_half && (dma_engine.citer == (dma_engine.biter >> 1)))
        generate_interrupt (channel);

    dma_engine.active = 0;         /* clear the channel busy flag */

 }
 else { /* major loop is complete, dma_engine.citer == 0 */
   /* since the major loop is complete, perform the final address adjustments */

   /* sum the current {src,dst} addresses with "last" adjustment */
   write_to_local_memory [channel].saddr = dma_engine.saddr + dma_engine.slast;
   write_to_local_memory [channel].daddr = dma_engine.daddr + dma_engine.dlast;
   /* restore the major iteration count to the beginning value */
```

```
        write_to_local_memory [channel].citer = dma_engine.biter;

        /* check for interrupt assertion at completion of the major iteration */
        if (dma_engine.int_maj)
          generate_interrupt (channel);

       /* check if the ipd_req is to be disabled at completion of the major iteration */
        if (dma_engine.d_req)
            DMAERQ [channel] = 0;

        /* check for a scatter/gather transfer control descriptor */
        if (dma_engine.e_sg) {
        /* load new transfer control descriptor from the address defined by dlast_sga */
            write_to_local_memory [channel] =
                                      read_from_amba-ahb(dma_engine.dlast_sga,32);
        }
        if (dma_engine.major.e_link)
            TCD[major.linkch].start = 1;       /* specified channel service req */

        dma_engine.active = 0;                /* clear the channel busy flag */
        dma_engine.done = 1;                  /* set the channel done flag */

}
else { /* configuration error detected, abort the channel */
        dma_engine.error_status = error_type; /* record the error */
        dma_engine.active = 0;                /* clear the channel busy flag */
        /* check for interrupt assertion on error */
        if (dma_engine.int_err)
            generate_interrupt (channel);
}
```

8.5 DMA Performance

Obtaining optimal performance requires that the DMA is properly set up for the application. The different aspects of setting up the DMA optimally for an application are covered in **Section A.1.6**, *DMA Controller*, on page A-4. The crossbar switch can also have a considerable effect on DMA performance so this too must be configured as needed by the application as discussed in **Section A.1.7**, *Crossbar Switch*, on page A-8.

Examples of transfer times possible with the DMA between sources and destinations on the device are presented in **Section A.3**, *DMA Burst Times*, on page A-19 for reference. It can also be helpful to see **Section A.4**, *DMA Burst Efficiency*, on page A-20.

8.6 DMA Initialization/Applications

The DMA initialization sequence proceeds as follows:

1. Write to the DMACR if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRI*x* registers if a configuration other than the default is desired.
3. Enable error interrupts in the DMAEEI registers if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the DMAERQ register.
6. Request channel service by setting the TCDx-7[START] bit.

When a channel requests service, it is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD for the selected channel into its internal address path module. As the TCD is read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD0[SADDR]) to the destination (as defined by the destination address, TCDx-4[DADDR]) continue until the specified number of bytes (TCDx-2[NBYTES]) have been transferred. Then the DMA engine's local TCD0[SADDR], TCDx-4[DADDR], and TCDx-5[CITER] are written back to the main TCD memory, and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, such as interrupts, major loop channel linking, and scatter/gather operations, if enabled.

8.6.1 DMA Programming Errors

The DMA controller performs various tests on the TCD to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors, Group Priority Error and Channel Priority Error, GPE and CPE bits in the DMAES register, respectively.

For all error types other than group or channel priority errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

The typical application enables error interrupts for all channels, so you may see an error interrupt for which the channel number for the DMAERR register and the error interrupt request line are wrong because they reflect the selected channel. Channel priority errors are identified within a group when that group is selected as the active group, as shown in the following example:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group0 has the highest priority, and all channels are unique in that group.
3. Group1 has the next highest priority and has two channels with the same priority level.

4. If Group0 has any service requests, those requests are executed.
5. Once all of Group0 requests complete, Group1 is the next active group.
6. If Group1 has a service request, an undefined channel in Group1 is selected and a channel priority error occurs.
7. This process repeats until all Group1 requests are removed or a higher-priority Group0 request occurs.

A group priority error is global, and any request in any group causes a group priority error. If priority levels are not unique, the highest (channel/group) priority with an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA controller. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

8.6.2 Single-Request DMA Data Transfer Example

To transfer *n* bytes of data with one activation, set the major loop to one (TCDx-5[CITER] = TCDx-7[BITER] = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. When the transfer completes, the TCDx-7[DONE] bit is set and an interrupt is generated, if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The DMA controller is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port at address 0x1000. The destination memory has a word-wide port at address 0x2000. The address offsets are programmed in increments to match the size of the transfer, one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values. Following are the TCD parameters for the data transfer, with the page number of the register in which this bit is defined.

Table 8-4. TCD Parameters for a Single-Request Data Transfer

TCD Bit/Field	Value	Page
TCDx-5[CITER] = TCDx-7[BITER]	1	**page 8-46** (CITER), **page 8-48** (BITER)
TCDx-2[NBYTES]	16	**page 8-44**
TCD0[SADDR]	0x1000	**page 8-42**
TCDx-1[SOFF]	1	**page 8-42**
TCDx-1[SSIZE]	0	**page 8-42**
TCDx-3[SLAST]	–16	**page 8-44**
TCDx-4[DADDR]	0x2000	**page 8-45**
TCDx-5[DOFF]	4	**page 8-46**
TCDx-1[DSIZE]	2	**page 8-42**
TCDx-6[DLAST]	–16	**page 8-47**

Table 8-4. TCD Parameters for a Single-Request Data Transfer (Continued)

TCD Bit/Field	Value	Page
TCDx-7[INTM]	1	**page 8-48**
TCDx-7[START]	1	**page 8-48**
All other TCD fields	0	
Note: TCDx-7 should be written last, after all other fields are initialized.		

These values generate the following sequence of events:

1. An IPBus write to the TCDx-7[START] bit requests channel service.
2. The arbiter selects the channel for servicing.
3. The DMA engine writes TCDx-7[DONE] = 0, TCDx-7[START] = 0, TCDx-7[ACTIVE] = 1.
4. The DMA engine reads channel TCD data from local memory to the internal register file.
5. The source-to-destination transfers execute as follows:
   a. read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003).
   b. write_word(0x2000) → *First iteration of the minor loop.*
   c. read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007).
   d. write_word(0x2004) → *Second iteration of the minor loop.*
   e. read_byte(0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B).
   f. write_word(0x2008) → *third iteration of the minor loop*
   g. read_byte(0x100C), read_byte(0x100d), read_byte(0x100E), read_byte(0x100F).
   h. write_word(0x200C) → *Last iteration of the minor loop → major loop complete.*
6. The DMA engine writes TCD0[SADDR] = 0x1000, TCDx-4[DADDR] = 0x2000, TCDx-5[CITER] = 1 (TCDx-7[BITER]).
7. The DMA engine writes TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 1, DMAINT*x* = 1.
8. The DMA channel retires.

The DMA controller goes idle or services the next channel.

8.6.3 Multiple-Request DMA Data Transfer Example

The multiple-request data transfer example is the same as that for a single request, with the exception that 32 bytes are transferred via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The DMA controller is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's

hardware requests is enabled in the DMAERQ register, channel service requests are initiated by the slave device, with the values shown in **Table 8-5**.

Table 8-5. TCD Parameters for a Multiple-Request Data Transfer

TCD Bit/Field	Value	Page
TCDx-5[CITER] = TCDx-7[BITER]	2	**page 8-46** (CITER) **page 8-48** (BITER)
TCDx-3[SLAST]	–32	
TCDx-7[DLAST]	–32	

These values generate the following sequence of events:

1. The first hardware request for channel service occurs.
2. The arbiter selects the channel for servicing.
3. The DMA engine writes TCDx-7[DONE] = 0, TCDx-7[START] = 0, TCDx-7[ACTIVE] = 1.
4. The DMA engine reads channel TCD data from local memory to the internal register file.
5. The source-to-destination transfers execute as follows:
   a. read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003).
   b. write_word (0x2000) → *First iteration of the minor loop.*
   c. read_byte (0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007).
   d. write_word (0x2004) → *Second iteration of the minor loop.*
   e. read_byte (0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B).
   f. write_word (0x2008) → *Third iteration of the minor loop.*
   g. read_byte (0x100C), read_byte(0x100D), read_byte(0x100E), read_byte(0x100F).
   h. write_word (0x200C) → *Last iteration of the minor loop.*
6. The DMA engine writes TCD0[SADDR] = 0x1010, TCDx-4[DADDR] = 0x2010, and TCDx-5[CITER] = 1.
7. The DMA engine writes TCDx-7[ACTIVE] = 0.
8. The channel retires → *One iteration of the major loop.*

The DMA controller goes idle or services the next channel, as follows:

1. The second hardware request for channel service occurs.
2. The arbiter selects the channel for servicing.
3. The DMA engine writes TCDx-7[DONE] = 0, TCDx-7[START] = 0, TCDx-7[ACTIVE] = 1.

4. The DMA engine reads channel TCD data from local memory to the internal register file.
5. The source-to-destination transfers execute as follows:
   a. read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013).
   b. write_word(0x2010) → *First iteration of the minor loop.*
   c. read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017).
   d. write_word(0x2014) → *Second iteration of the minor loop.*
   e. read_byte(0x1018), read_byte(0x1019), read_byte(0x101A), read_byte(0x101B).
   f. write_word(0x2018) → *Third iteration of the minor loop.*
   g. read_byte(0x101C), read_byte(0x101D), read_byte(0x101E), read_byte(0x101F).
   h. write_word(0x201C) → *Last iteration of the minor loop → Major loop complete.*
6. The DMA engine writes TCD0[SADDR] = 0x1000, TCDx-4[DADDR] = 0x2000, and TCDx-5[CITER] = 2 (TCDx-7[BITER]).
7. The DMA engine writes TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 1, DMAINT[n] = 1.
8. The channel retires → *major loop complete.*

The DMA controller goes idle or services the next channel.

8.6.4 TCD Status

This section discusses how to check TCD status in terms of loop completion, true TCD values, and preemption.

8.6.4.1 Minor Loop Completion

For software-initiated service requests, there are two ways to test for minor loop completion. The first is to read the TCDx-5[CITER] field and test for a change. The second is to test the TCDx-7[START] bit *and* the TCDx-7[ACTIVE] bit. The minor loop complete condition is indicated when both bits read zero after the TCDx-7[START] bit is written with a value of one. Polling the TCDx-7[ACTIVE] bit can be inconclusive because the active status can be missed if channel execution time is short. The TCD status bits execute the following sequence for a software-activated channel:

1. TCDx-7[START] = 1, TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 0 (channel service request via software).
2. TCDx-7[START] = 0, TCDx-7[ACTIVE] = 1, TCDx-7[DONE] = 0 (channel is executing).
3. TCDx-7[START] = 0, TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 0 (channel has completed the minor loop and is idle) or TCDx-7[START] = 0, TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 1 (channel has completed the major loop and is idle).

For hardware-initiated service requests, the best way to test for minor loop completion is to read the TCDx-5[CITER] field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model. The TCD status bits execute the following sequence for a hardware activated channel:

4. A hardware request for channel service occurs.
5. TCDx-7[START] = 0, TCDx-7[ACTIVE] = 1, TCDx-7[DONE] = 0 (channel is executing).
6. TCDx-7[START] = 0, TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 0 (channel has completed. the minor loop and is idle) or TCDx-7[START] = 0, TCDx-7[ACTIVE] = 0, TCDx-7[DONE] = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCDx-7[DONE] bit. The TCDx-7[START] bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

8.6.4.2 Active Channel TCD Reads

The DMA controller reads back the true TCD0[SADDR], TCDx-4[DADDR], and TCDx-2[NBYTES] values if a read occurs while a channel is executing. The true values are the values the DMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES decrement to zero as the transfer progresses) can indicate the progress of the transfer. All other values are read back from TCD local memory.

8.6.4.3 Preemption Status

Preemption is available only when *fixed* arbitration is selected for *both* group and channel arbitration modes. Preemption becomes possible when a preempt-enabled channel is running and a higher-priority request becomes active. When the DMA engine is not operating in fixed group, fixed channel arbitration mode, the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCDx-7[ACTIVE] bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCDx-7[ACTIVE] bits set at the same time in the overall TCD map indicates that a higher-priority channel is actively preempting a lower-priority channel.

The worst-case latency during a switch to a preempting channel is the summation of the following:

- Arbitration latency (2 cycles).
- Bandwidth control stalls, if enabled.

- The time to execute two read/write sequences, including AHB bus holds. This is a system dependency driven by the slave devices or the crossbar switch.

8.6.5 Channel Linking

Channel linking (or chaining) occurs when one channel sets the TCDx-7[START] bit of another channel (or itself) to initiate a service request for that channel. The DMA engine automatically performs this operation at the end of the major or minor loop, when properly enabled, or after one iteration of the major loop.

The TCDx-5[CITERE] field is used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop, except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine whether a channel link should be made. For example, **Table 8-6** shows the initial fields for executing loops.

Table 8-6. Example TCD Field Values for Loop Execution

TCD Bit/Field	Value	Page
TCDx-5[CITERE]	1	**page 8-46**
TCDx-5[CITERH]	0xC	**page 8-46**
TCDx-5[CITER]	0x4	**page 8-46**
TCDx-7[CLE]	1	**page 8-48**
TCDx-7[LCNUM]	0x7	**page 8-48**

These values result in execution of the following sequence:

1. Minor loop done → set channel 12 TCDx-7[START] bit.
2. Minor loop done → set channel 12 TCDx-7[START] bit.
3. Minor loop done → set channel 12 TCDx-7[START] bit.
4. Minor loop done, major loop done → set channel 7 TCDx-7[START] bit.

When minor loop linking is enabled (TCD[CITERE] = 1), the TCDx-5[CITER] field uses a 9-bit vector to form the current iteration count. When minor loop linking is disabled (TCDx-5[CITERE] = 0), the TCDx-5[CITER] field uses a 15-bit vector to form the current iteration count. The bits associated with the TCDx-5[CITERH] field are concatenated onto the CITER value to increase the range of the CITER.

Note: The TCDx-5[CITERE] bit and the TCDx-7[BITERE] bit must be equal, in order to calculate the major loop halfway done interrupt point, or a configuration error is reported by setting DMAES[NCE] = 1.

8.6.6 Dynamic Programming

The following two options are recommended for dynamically changing *channel* priority levels:

1. Switch to round-robin channel arbitration mode, change the channel priorities, then switch back to fixed arbitration mode.
2. Disable all the channels within a group, change the channel priorities within that group only, then enable the appropriate channels.

The following two options are available for dynamically changing *group* priority levels:

1. Switch to round-robin group arbitration mode, change the group priorities, then switch back to fixed arbitration mode.
2. Disable *all* channels, change the group priorities, then enable the appropriate channels.

Dynamic channel linking and dynamic scatter/gather is the process of changing the values of the TCDx-7[CLE] or TCDx-7[ESG] bits during channel execution. These bits are read from the TCD local memory at the *end* of channel execution so that you can enable either feature during channel execution.

If you were to attempt to execute a dynamic channel link by enabling the TCDx-7[CLE] bit at the same time the DMA engine is retiring the channel, the bit would be set in the programmer's model, but it would be unclear whether the actual link occurred before the channel retired. Therefore, the following coherency model is recommended when you execute a dynamic channel link or dynamic scatter/gather request:

1. Set the TCDx-7[CLE] bit.
2. Read back the TCDx-7[CLE] bit.
3. Test the TCDx-7[CLE] request status:
   a. If the bit is set, the dynamic link attempt was successful.
   b. If the bit is cleared, the attempted dynamic link did not succeed because the channel was already retiring.

This coherency model holds true for dynamic scatter/gather operations. For both dynamic requests, after the channel's TCDx-7[DONE] bit is set to indicate that the major loop is complete, the TCD local memory controller forces the TCDx-7[CLE] and TCDx-7[ESG] bits to zero on any writes to a channel's TCDx-7 register.

Note: You must clear the TCDx-7[DONE] bit before writing the TCDx-7[CLE] or TCDx-7[ESG] bits. The DMA engine automatically clears the TCDx-7[DONE] bit when a channel begins execution.

8.7 DMA Programming Model

The DMA programming model is partitioned into two sections, both mapped into the IPBus address space. The first section defines control registers, and the second defines the local TCD memory. For all unused or reserved register bits: reads return zeroes, and writes are ignored. The DMA module does not include any access control. Rather, standard access control is provided by the IPBus controller. **Table 8-7** shows a 32-bit view of the DMA memory map.

Table 8-7. DMA 32-Bit Memory Map

DMA Offset	Register			
0x0000	DMA Control Register (DMACR)			
0x0004	DMA Error Status (DMAES)			
0x0008	Reserved			
0x000C	DMA Enable Request (DMAERQ)			
0x0010	Reserved			
0x0014	DMA Enable Error Interrupt (DMAEEI)			
0x0018	DMA Set Enable Request (DMASERQ)	DMA Clear Enable Request (DMACERQ)	DMA Set Enable Error Interrupt (DMASEEI)	DMA Clear Enable Error Interrupt (DMACEEI)
0x001C	DMA Clear Interrupt Request (DMACINT)	DMA Clear Error (DMACERR)	DMA Set Start Bit, Activate Channel (DMASSRT)	DMA Clear Done Status Bit (DMACDNE)
0x0020	Reserved			
0x0024	DMA Interrupt Request (DMAINT)			
0x0028	Reserved			
0x002C	DMA Error (DMAERR)			
0x0030–0x00FC	Reserved			
0x0100	DMA Channel 0 Priority (DCHPRI0)	DMA Channel 1 Priority (DCHPRI1)	DMA Channel 2 Priority (DCHPRI2)	DMA Channel 3 Priority (DCHPRI3)
0x0104	DMA Channel 4 Priority (DCHPRI4)	DMA Channel 5 Priority (DCHPRI5)	DMA Channel 6 Priority (DCHPRI6)	DMA Channel 7 Priority (DCHPRI7)
0x0108	DMA Channel 8 Priority (DCHPRI8)	DMA Channel 9 Priority (DCHPRI9)	DMA Channel 10 Priority (DCHPRI10)	DMA Channel 11 Priority (DCHPRI11)
0x010C	DMA Channel 12 Priority (DCHPRI12)	DMA Channel 13 Priority (DCHPRI13)	DMA Channel 14 Priority (DCHPRI14)	DMA Channel 15 Priority (DCHPRI15)
0x0110	DMA Channel 16 Priority (DCHPRI16)	DMA Channel 17 Priority (DCHPRI17)	DMA Channel 18 Priority (DCHPRI18)	DMA Channel 19 Priority (DCHPRI19)
0x0114	DMA Channel 20 Priority (DCHPRI20)	DMA Channel 21 Priority (DCHPRI21)	DMA Channel 22 Priority (DCHPRI22)	DMA Channel 23 Priority (DCHPRI23)
0x0118	DMA Channel 24 Priority (DCHPRI24)	DMA Channel 25 Priority (DCHPRI25)	DMA Channel 26 Priority (DCHPRI26)	DMA Channel 27 Priority (DCHPRI27)
0x011C	DMA Channel 28 Priority (DCHPRI28)	DMA Channel 29 Priority (DCHPRI29)	DMA Channel 30 Priority (DCHPRI30)	DMA Channel 31 Priority (DCHPRI31)

Table 8-7. DMA 32-Bit Memory Map (Continued)

DMA Offset	Register
0x0120-0x0ffc	Reserved
0x1000-0x11fc	TCD00-TCD15
0x1200-0x13fc	TCD16-TCD31
0x1400-0x1800	Reserved

The DMA registers are listed as follows, along with the number of the page on which each register is discussed:

- Control registers:
  - DMA Control Register (DMACR), **page 8-27**.
  - DMA Error Status Register (DMAES), **page 8-28**.
  - DMA Enable Request Register (DMAERQ), **page 8-31**.
  - DMA Enable Error Interrupt (DMAEEI), **page 8-32**.
  - DMA Set Enable Request Register (DMASERQ), **page 8-32**.
  - DMA Clear Enable Request Register (DMACERQ), **page 8-33**.
  - DMA Set Enable Error Interrupt Register (DMASEEI), **page 8-33**.
  - DMA Clear Enable Error Interrupt Register (DMACEEI), **page 8-34**.
  - DMA Clear Interrupt Request Register (DMACINT), **page 8-34**.
  - DMA Clear Error Register (DMACERR), **page 8-35**.
  - DMA Set Start Register (DMASSRT), **page 8-36**.
  - DMA Clear Done Status Register (DMACDNE), **page 8-36**.
  - DMA Interrupt Request Register (DMAINT), **page 8-37**.
  - DMA Error Register (DMAERR), **page 8-38**.
  - DMA Channel Priority (DCHPR[0–31]), **page 8-39**.
- Each of the thirty-two DMA channels is controlled by eight Transfer Control Descriptor Words (TCD) that are defined in the following set of 32-bit registers:
  - Transfer Control Descriptor Word 0 (TCD[0–31]-0), **page 8-42**.
  - Transfer Control Descriptor Word 1 (TCD[0–31]-1), **page 8-42**.
  - Transfer Control Descriptor Word 2 (TCD[0–31]-2), **page 8-44**.
  - Transfer Control Descriptor Word 3(TCD[0–31]-3), **page 8-44**.
  - Transfer Control Descriptor Word 4 (TCD[0–31]-4), **page 8-45**.
  - Transfer Control Descriptor Word 5 (TCD[0–31]-5), **page 8-46**.
  - Transfer Control Descriptor Word 6 (TCD[0–31]-6), **page 8-47**.
  - Transfer Control Descriptor Word 7 (TCD[0–31]-7), **page 8-48**.

8.7.1 Control Registers

DMACR DMA Control Register DMA_Base + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—					GRP1PRI	—	GRP0PRI	—				ERGA	ERCA	EDBG	EBW
TYPE	R					R/W		R/W	R				R/W			
RESET	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

DMACR defines the basic operating configuration of the DMA controller. When multiple channels are activated at the same time, the DMA controller arbitrates channel execution using either fixed mode or round robin. In fixed-mode arbitration, the activated channel with the highest priority is executed. In round robin arbitration, the activated channels are cycled through without regard to priority.

Table 8-8. DMACR Bit Descriptions

Name	Reset	Description	Setting
— 31–11	0	Reserved. Write to zero for future compatibility.	
GRP1PRI 10	1	**Channel Group 1 Priority** Priority level when fixed-priority group arbitration is enabled.	0 Group priority is level 0 (lowest priority). 1 Group priority is level 1 (highest priority).
— 9	0	Reserved. Write to zero for future compatibility.	
GRP0PRI 8	0	**Channel Group 0 Priority** Priority level when fixed priority group arbitration is enabled.	0 Group priority is level 0 (lowest priority). 1 Group priority is level 1 (highest priority).
— 7–4	0	Reserved. Write to zero for future compatibility.	
ERGA 3	0	**Enable Round Robin Group Arbitration** Specifies whether fixed arbitration or round robin arbitration is used among the groups.	0 Fixed priority arbitration is used for selection among the groups. 1 Round robin arbitration is used for selection among the groups.

Table 8-8. DMACR Bit Descriptions (Continued)

Name	Reset	Description	Setting
ERCA 2	0	**Enable Round Robin Channel Arbitration** Specifies whether fixed arbitration or round robin arbitration is used among the channels in a group.	0 Fixed priority arbitration is used for channel selection within each group. 1 Round robin arbitration is used for channel selection within each group.
EDBG 1	0	**Enable Debug** Enables/disables DMA debug mode. When this bit is set, the DMA controller stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the internal debug input is negated or the EDBG bit is cleared.	0 No Debug mode. 1 DMA debug mode.
EBW 0	0	**Enable Buffered Writes** Enables/disables buffered writes.	0 No buffered write during AMBA AHB writes. 1 Buffered on all AMBA AHB writes except for the last write sequence.

DMAES DMA Error Status Register DMA_Base + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	VLD	—														
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GPE	CPE	—	ERRCHN					SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMAES provides information on the last recorded channel error. Channel errors can be caused by a configuration error, which is an illegal setting in the TCD or an illegal priority register setting in fixed-arbitration mode, or an error termination to a bus master read or write cycle. A configuration error occurs when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size.

In fixed arbitration mode, a configuration error is caused when any two channel priorities are equal. All channel priority levels must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled at channel completion, a configuration error is reported if the

scatter/gather address is not aligned on a 32-byte boundary. If minor loop channel linking is enabled at channel completion, a configuration error is reported when the link is attempted if the TCDx-5[CITERE] bit does not equal the TCDx-7[BITERE] bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and asserts an error interrupt request if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write terminates with an error, the data transfer stops and the appropriate bus error flag is set. The DMA controller updates the state of the channel's TCD with the current source address, destination address, and iteration count at the point of the fault. When a system bus error occurs, the channel terminates after completion of the read or write transaction already pipelined. If a bus error occurs on the last read before the write sequence, the write uses the data captured during the bus error. If a bus error occurs on the last write before the read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

Any type of error causes the DMA controller to stop immediately, the appropriate channel bit in the DMA Error Register is asserted, and the details of the error condition are loaded into the DMAES register. The normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

Table 8-9. DMAES Bit Descriptions

Name	Reset	Description	Setting
VLD 31	0	**Logical OR of DMAERRH and DMAERRL Status** Indicates whether DMA error bits are set.	0 No DMAERR bits are set. 1 At least one DMAERR bit is set, indicating a valid error that has not been cleared.
— 30–16	0	Reserved. Write to zero for future compatibility.	
GPE 15	0	**Group Priority Error** Indicates whether there is a group priority error.	0 No group priority error. 1 The last recorded error was a configuration error among the group priorities. All group priorities are not unique.
CPE 14	0	**Channel Priority Error** Indicates whether there is a channel priority error. Channel priority errors are identified within a group when that group is selected as the active group.	0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. All channel priorities are not unique.
— 13	0	Reserved. Write to zero for future compatibility.	

Table 8-9. DMAES Bit Descriptions

Name	Reset	Description	Setting
ERRCHN 12–8	0	**Error Channel Number** The channel number of the last recorded error, excluding GPE and CPE errors.	
SAE 7	0	**Source Address Error** Indicates whether there is a source address configuration error. When this bit is set, TCD0[SADDR] is inconsistent with TCDx-1[SSIZE]. See **Section 8.7.2**, *Transfer Control Descriptor (TCD) Registers*, on page 8-40.	0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD[SADDR] field.
SOE 6	0	**Source Offset Error** Indicates whether there is a source offset error. When this bit is set, TCDx-1[SOFF] is inconsistent with TCDx-1[SSIZE].	0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD[SOFF] field.
DAE 5	0	**Destination Address Error** Indicates whether there is a destination address error. If this bit is set, TCDx-4[DADDR] is inconsistent with TCDx-1[DSIZE].	0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCDx-4[DADDR]field.
DOE 4	0	**Destination Offset Error** Indicates whether there is a destination offset error. If this bit is set, TCDx-5[DOFF] is inconsistent with TCDx-1[DSIZE].	0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCDx-5[DOFF] field.
NCE 3	0	**NBYTES/CITER Configuration Error** Indicates whether there is an error caused by a disparity in the TCDx-2[NBYTES] and the TCDx-5[CITER] fields. If this bit is set, one of the following conditions applies: • TCDx-2[NBYTES] is not a multiple of TCDx-1[SSIZE] and TCDx-1[DSIZE] • TCDx-5[CITER] bit is equal to zero • TCDx-5[CITERE] is not equal to TCDx-7[BITERE].	0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCDx-2[NBYTES] or TCDx-5[CITER] fields.
SGE 2	0	**Scatter/Gather Configuration Error** Indicates whether there is a scatter/gather configuration error. The TCDx-6[DLAST] field is checked at the beginning of a scatter/gather operation after a major loop executes if TCDx-7[ESG] is enabled. TCDx-6[DLAST] is not on a 32-byte boundary.	0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCDx-6[DLAST] field.
SBE 1	0	**Source Bus Error** Indicates whether there is a bus error on a source read.	0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE 0	0	**Destination Bus Error** Indicates whether there is a bus error on a destination write.	0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

DMAERQ DMA Enable Request Register DMA_Base + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ERQ31	ERQ30	ERQ29	ERQ28	ERQ27	ERQ26	ERQ25	ERQ24	ERQ23	ERQ22	ERQ21	ERQ20	ERQ19	ERQ18	ERQ17	ERQ16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ9	ERQ8	ERQ7	ERQ6	ERQ5	ERQ4	ERQ3	ERQ2	ERQ1	ERQ0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMAERQ provide a bit mapping to enable the request signal for each channel. The state of any given channel enable is directly affected by writes to this register. The state is also affected by writes to the DMASERQ and DMACERQ registers, which handle the request enable for a *single* channel without the need to perform a read-modify-write sequence to DMAERQ. Both the DMA request input signal and this enable request flag must be asserted before a channel is activated. The state of the DMA enable request flag does *not* affect a channel activated through an explicit software initiation or a linked channel request. As a given channel finishes processing its major iteration count, a flag in the TCD may affect the ending state of the DMAERQ bit for that channel. If the TCDx-7[DREQ] bit is set, the corresponding DMAERQ bit is cleared, disabling the DMA request. If the DREQ bit is cleared, the state of the DMAERQ bit is unaffected.

Table 8-10. DMAERQ Bit Descriptions

Name	Description	Setting
ERQ[31–0] 31–0	**Enable DMA Request** Enables/disables the DMA request signal for each channel.	0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

DMAEEI DMA Enable Error Interrupt Register DMA_Base + 0x24

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	EE31	EE30	EE29	EE28	EE27	EE26	EE25	EE24	EE23	EE22	EE21	EE20	EE19	EE18	EE17	EE16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ERQ15	EE14	EE13	EE12	EE11	EE10	EE9	EE8	EE7	EE6	EE5	EE4	EE3	EE2	EE1	EE0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMAEEI provides a bit mapping to enable the error interrupt signal for each channel. The state of any channel's error interrupt enable is directly affected by writes to this register. The state is also affected by writes to the DMASEEI and DMACEEI registers, which handle the error interrupt enable for a *single* channel without the need to perform a read-modify-write sequence to the DMAEEI register. Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Table 8-11. DMAEEI Bit Descriptions

Name	Description	Setting
EEI[31–0] 31–0	**Enable Error Interrupt** Specifies whether the error signal for a channel generates an error interrupt.	0 The error signal for the channel does not generate an error interrupt. 1 The assertion of the error signal for the channel generates an error interrupt request.

DMASERQ DMA Set Enable Request Register DMA_Base + 0x18

BIT	7	6	5	4	3	2	1	0
	—	SAER	—	SERQ				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMASERQ provides a simple memory-mapped mechanism to set a given bit in the DMAERQ registers to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ register to be set. A data value of 64 to 127 provides a global set function, forcing the entire contents of DMAERQ to be asserted. Reads of this register return all zeroes.

Table 8-12. DMASERQ Bit Descriptions

Name	Description	Setting	
SAER 6	**Set All Enable Requests** Sets all the enable bits in the DMAERQ.	0 1	Set the specified bit in DMAERQ. Set all bits in DMAERQ.
SERQ 4–0	**Set Enable Request** Sets each enable bit in the DMAERQ.	0–31	Set the corresponding bit in DMAERQ

DMACERQ DMA Clear Enable Request Register DMA_Base + 0x19

BIT	7	6	5	4	3	2	1	0
	—	CAER	—	CERQ				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMACERQ provides a simple memory-mapped mechanism to clear a given bit in the DMAERQ to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ to be cleared. A data value of 64 to 127 provides a global clear function, forcing the entire contents of the DMAERQ to zero, disabling all DMA request inputs. Reads of this register return all zeroes.

Table 8-13. DMACERQ Bit Descriptions

Name	Description	Setting	
CAER 6	**Clear All Enable Requests** Clears all bits in the DMAERQ.	0 1	Clear the specified bit in DMAERQ. Clear all bits in DMAERQ.
CERQ 4–0	**Clear Enable Request** Clears the corresponding bit in the DMAERQ.	0–31	Clear the corresponding bit in DMAERQ

DMASEEI DMA Set Enable Error Interrupt Register DMA_Base + 0x1A

BIT	7	6	5	4	3	2	1	0
	—	SAEE	—	SEE				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMASEEI provides a simple memory-mapped mechanism to set a given bit in the DMAEEI register to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI register to be set. A data value of 64 to 127 provides a

global set function, forcing the entire contents of DMAEEI to be asserted. Reads of this register return all zeroes.

Table 8-14. DMASEEI Bit Descriptions

Name	Description	Setting
SAEE 6	**Set All Enable Error Interrupts** Sets all bits in the DMAEEI.	0 Set the specified bit in DMAEEI. 1 Set all bits in DMAEEI.
SEEI 4–0	**Set Enable Error Interrupt** Sets the corresponding bit in the DMAEEI.	0–31 Set the corresponding bit in DMAEEI

DMACEEI — DMA Clear Enable Error Interrupt Register — DMA_Base + 0x1B

BIT	7	6	5	4	3	2	1	0
	—	CAEE	—	CEE				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMACEEI provides a simple memory-mapped mechanism to clear a given bit in the DMAEEI register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI register to be cleared. A data value of 64 to 127 (regardless of the number of channels) provides a global clear function, forcing the entire contents of the DMAEEI to zero, disabling all DMA request inputs. Reads of this register return all zeroes.

Table 8-15. DMACEEI Bit Descriptions

Name	Description	Value
CAEE 6	**Clear All Enable Error Interrupts** Clears all the bits in the DMAEEI register.	0 Clear the specified bit in DMAEEI. 1 Clear all bits in DMAEEI.
CEEI 4–0	**Clear Enable Error Interrupt** Clears the corresponding bit in the DMAEEI register.	0–31 Clear the corresponding bit in DMAEEI.

DMACINT — DMA Clear Interrupt Request Register — DMA_Base + 0X1C

BIT	7	6	5	4	3	2	1	0
	—	CAIR	—	CINT				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMACINT provides a simple memory-mapped mechanism to clear a given bit in the DMAINT registers to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMAINT register to be cleared. A data value of 64 to 127 (regardless of the number of channels) provides a global clear function, forcing the entire contents of the DMAINT register to zero, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Table 8-16. DMACINT Bit Descriptions

Name	Description	Value
CAEE 6	**Clear All Enable Error Interrupts** Clears all bits in the DMAINT register.	0 Clear the specified bit in DMAINT. 1 Clear all bits in DMAINT.
CINT 4–0	**Clear Interrupt Request** Clears the corresponding bit in the DMAINT register.	0–31 Clear the corresponding bit in DMAINT.

DMACERR DMA Clear Error DMA_Base + 0X1D

BIT	7	6	5	4	3	2	1	0
	—	CAER	—	CERR				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMACEER provides a simple memory-mapped mechanism to clear a given bit in the DMAERR registers to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMAERR register to be cleared. A data value of 64 to 127 (regardless of the number of channels) provides a global clear function, forcing the entire contents of the DMAERR to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Table 8-17. DMACERR Bit Descriptions

Name	Description	Setting
CAER 6	**Clear All Error Indicators** Clears all bits in the DMAERR to disable the error condition flags.	0 Clear the specified bit in DMAERR. 1 Clear all bits in DMAERR.
CERR 4–0	**Clear Error Indicator** Clears the corresponding bit in the DMAERR.	0–31 Clear the corresponding bit in DMAERR.

DMASSRT DMA Set Start DMA_Base + 0X1E

BIT	7	6	5	4	3	2	1	0
	—	SAST	—	SSRT				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMASSRT provides a simple memory-mapped mechanism to set the TCDx-7[START] bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding TCD to be set. A data value of 64 to 127 (regardless of the number of channels) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Table 8-18. DMA Set START Bit Register (DMASSRT) Field Descriptions

Name	Description	Value	
SAST 6	**Set All START Bits** Activates all the channels.	0 1	Set the specified channel's TCDx-7[START] Set all TCDx-7[START] bits
SSRT 4–0	**Set START Bit (Activate Channel)** Activates only the specified channel.	0–31	Set the corresponding channel's TCDx-7[START]

DMACDNE DMA Clear Done Status Register DMA_Base + 0X1F

BIT	7	6	5	4	3	2	1	0
	—	CADN	—	CDNE				
TYPE	R	R/W	R	R/W				
RESET	0	0	0	0	0	0	0	0

DMACDNE provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding TCD to be cleared. A data value greater than 63 (regardless of the number of channels) provides a global clear function, forcing all DONE bits to be cleared. Reads of this register return all zeroes.

Table 8-19. DMACDNE Field Descriptions

Name	Description	Value
CADN 6	**Clear All DONE Status Bits** Clears the TCDx-7[DONE] bits for all channels.	0 Clear the specified channel's TCDx-7[DONE] bit. 1 Clear all TCDx-7[DONE] bits.
CDNE 4–0	**Clear DONE Status Bit** Clears the TCDx-7[DONE] bit of the corresponding channel.	0–31 Clear the corresponding channel's TCDx-7[DONE] bit.

DMAINT DMA Interrupt Request Register DMA_Base + 0x24

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ERQ15	INT14	INT13	INT12	INT11	INT10	INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMAINT provides a bit mapping to signal the presence of an interrupt request for each channel. The DMA controller sets the appropriate bit in this register to signal a programmed interrupt when a data transfer completes, as defined in the TCD. The outputs of this register are directly routed to the interrupt controller. During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose. The state of any given channel's interrupt request is directly affected by writes to this register. The state is also affected by writes to the DMACINT register. During writes to the DMAINT, a one in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The DMACINT register handles the interrupt request for a *single* channel without the need to perform a read-modify-write sequence to DMAINT.

Table 8-20. DMAINT Bit Descriptions

Name	Description	Value
INT[31–0] 31–0	**DMA Interrupt Request** Activates an interrupt request for a specified channel.	0 The interrupt request for the specified channel is cleared. 1 The interrupt request for the specified channel is active.

DMAERR DMA Error Register DMA_Base + 0x2C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ERR31	ERR30	ERR29	ERR28	ERR27	ERR26	ERR25	ERR24	ERR23	ERR22	ERR21	ERR20	ERR19	ERR18	ERR17	ERR16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ERQ15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR9	ERR8	ERR7	ERR6	ERR5	ERR4	ERR3	ERR2	ERR1	ERR0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMAERR provides a bit mapping to signal the presence of an error for each channel. The DMA controller signals an error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, then logically summed across all 32 channels to form a single error interrupt request that is routed to the interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, software must clear the appropriate bit, negating the error interrupt request. Typically, a write to the DMACERR in the interrupt service routine is used for this purpose. The normal DMA channel completion indicators, setting the TCD done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled. A non-zero value indicates the presence of a channel error, regardless of the state of the DMAEEI register. The state of any given channel's error indicators is affected by writes to this register. The state is also affected by writes to the DMACERR. During writes to DMAERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no effect on the corresponding channel's current error status. The DMACERR register is provided so that the error indicator for a *single* channel can easily be cleared.

Table 8-21. DMAERR Bit Descriptions

Name	Description	Value
ERR[31–0] 31–0	**DMA Error** Indicates whether an error has occurred in the specified channel.	0 An error in the specified channel has not occurred. 1 An error in the specified channel has occurred.

DCHPRI[0–31] DMA Channel Priority Registers 0–31 DMA_Base + 0x100–0x11F

BIT	7	6	5	4	3	2	1	0
	ECP	0	0	GRPPRI	CHPRI			
TYPE	R/W	R			R/W			
RESET	0	0	Defaults to Channel n After Reset					

When fixed-priority channel arbitration is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities of each channel. The channel priorities are evaluated by numeric value, that is, 0 is the lowest priority, 1 is the next priority, then 2, 3, and do on. Software must program the channel priorities with unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per channel basis when the DCHPRI[ECP] bit is set. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher-priority channel. Once the preempting channel completes all of its programmed minor loop transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is eligible to be preempted again if any higher-priority channel is activated. Multiple ECP bits can be set, but the DMA controller does not perform nested preemption (attempt to preempt a preempting channel). Once a preempting channel begins execution, it cannot be preempted. Preemption is available in fixed arbitration mode only.

Table 8-22. DCHPRI*x* Bit Descriptions

Name	Description	Value
ECP 7	**Enable Channel Preemption** Enables/disables suspension of a specified channel by activating a channel with a higher priority.	0 The specified channel cannot be suspended by activation of a higher-priority channel. 1 The specified channel can be temporarily suspended by activation of a higher-priority channel.
0 6–5	Reserved. Write to zero for future compatibility.	
GRPPRI 4	**Channel Current Group Priority** Group priority assigned to this channel group when fixed-priority arbitration is enabled.	
CHPRI 3–0	**Channel *n* Arbitration Priority** Channel priority when fixed-priority arbitration is enabled.	

8.7.2 Transfer Control Descriptor (TCD) Registers

Each channel requires a 32-byte TCD for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel[0–31]. The definitions of the TCD are presented as eight 32-bit values. **Table 8-23** shows a 32-bit view of the basic TCD structure.

Note: TCDx-1 is the mnemonic for the TCD for channel x, word 1, where x is the channel number from 0–31.

Table 8-23. DMA Channel x Priority (DCHPRIx) Field Descriptions

DMA Offset	TCDx Field	
0x1000 + (32 × x) + 0x00	Source Address (SADDR)	
0x1000 + (32 × x) + 0x04	Transfer Attributes (SMOD, SSIZE, DMOD, DSIZE)	Signed Source Address Offset (SOFF)
0x1000 + (32 × x) + 0x08	Inner Minor Byte Count (NBYTES)	
0x1000 + (32 × x) + 0x0c	Last Source Address Adjustment (SLAST)	
0x1000 + (32 × x) + 0x10	Destination Address (DADDR)	
0x1000 + (32 × x) + 0x14	Current Major Iteration Count (CITER)	Signed Destination Address Offset (DOFF)
0x1000 + (32 × x) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (DLAST)	
0x1000 + (32 × x) + 0x1C	Beginning Major Iteration Count (BITER)	Channel Control/Status

Table 8-24 shows this TCD information organized by functionality.

Table 8-24. TCD Information Organized by Function

Field	Field Name	Field Size	TCDx Word	TCDx Offset	Description
Source Transfer Fields					
Source Address	SADDR	32	0	0x00	Address for DMA read accesses.
Source Address Offset	SOFF	16	1	0x04	Updates source address once per minor loop.
Source Transfer Attributes	SMOD, SSIZE	5, 3	1	0x04	Modulo capability and source data transfer size.
Source Final Addr Adjust	SLAST	32	3	0x0C	For final update of source address when last iteration of major loop completes.
Destination Transfer Fields					
Destination Address	DADDR	32	4	0x10	Address for DMA write accesses.
Destination Address Offset	DOFF	16	5	0x14	Updates the destination address once per minor loop.
Destination Transfer Attributes	DMOD, DSIZE	5, 3	1	0x04	Modulo capability and destination data transfer size.

Table 8-24. TCD Information Organized by Function (Continued)

Field	Field Name	Field Size	TCDx Word	TCDx Offset	Description
Destination Final Addr Adjust	DLAST	32	6	0x18	For final update of the source address when last iteration of the major loop completes. Alternatively, it provides the address of the next TCD to load for a scatter-gather.
Major / Minor Looping Fields					
Minor Loop Byte Count	NBYTES	32	2	0x08	The largest value should be 0x1FFF FFFF.
Major Loop Begin Count	BITER	15	7	0x1C	Used in two different ways: • If linking, provides 9-bit major loop beginning count and 6-bit number of minor loop linked channel. • If not linking, provides 15-bit major loop beginning count.
Major Loop Current Count	CITER	15	5	0x14	Used in two different ways: • If linking, provides 9-bit major loop current count and 6-bit number of minor loop linked channel. • If not linking, provides 15-bit major loop current count.
Channel Control Fields					
Channel Start	START	1	7	0x1C	Activates a channel
Bandwidth Control	BWC	2	7	0x1C	Allows following for channel: • Priority elevation. • Optional stalls.
Enable Major Loop Linking	CLE	1	7	0x1C	Enables channel linking on competion of major loop.
Major Loop Link Channel	LCNUM	6	7	0x1C	Channel linked to when major loop linking is enabled.
Enable Scatter-Gather	ESG	1	7	0x1C	Enable for scatter-gather capability.
Disable Request	DREQ	1	7	0x1C	Enables auto-clearing of the DMAERQ bit when the other major loop completes.
Interrupt Control Fields					
Interrupt: Major Counter Half Complete	INTH	1	7	0x1C	Enables interrupts for the associated channel when the major loop counter reaches its halfway point.
Interrupt: Major Counter Complete	INTM	xx-bits	7	0x1C	Enables interrupts for the associated channel when the major loop counter reaches zero.
Channel Status Fields					
Channel Active	ACTIVE	1	7	0x1C	Flag indicating that this channel is currently in execution.
Channel Done	DONE	1	7	0x1C	Flag indicating that this channel has completed.

TCDx-0 Transfer Control Descriptor Word 0 DMA_Base + 0x1000 + (32 × n) + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	SADDR															
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SADDR															
TYPE	R/W															
RESET	Undefined															

TCD0 defines the source address of the DMA transfer.

Table 8-25. TCD0 Bit Descriptions

Name	Description	Value
SADDR 31–0	**Source Address** Memory address pointing to the source data.	

TCDx-1 Transfer Control Descriptor Word 1 DMA_Base + 0x1000 + (32 × n) + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	SMOD					SSIZE			DMOD					DSIZE		
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SOFF															
TYPE	R/W															
RESET	Undefined															

TCDx-1 defines such aspects of the DMA transfer as the source and destination address modulo, the size of the source data transfer, size of the destination data transfer, and source address signed offset.

Table 8-26. TCDx-1 Field Descriptions

Name	Description	Setting
SMOD 31–27	**Source Address Modulo** Defines a specific address bit that is selected to be either the value after a SADDR + SOFF calculation or the original register value. This feature makes it easy to implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should be based at a 0-modulo-size address and the SMOD field set to the appropriate value to freeze the upper address bits. The bit select is defined as ((1 << SMOD) – 1), where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, SOFF is typically set to the transfer size to implement post-increment addressing with SMOD constraining the addresses to a 0-modulo-size range.	0 Source address modulo feature is disabled.
SSIZE 26–24	**Source Data Transfer Size** Specifies the size of the source data transfer.	000 8-bit. 001 16-bit. 010 32-bit. 011 64-bit. 100 Reserved. 101 32-byte (if supported by the platform). 110 Reserved. 111 Reserved.
DMOD 23–19	**Destination Address Modulo** See the SMOD definition.	0 Destination address modulo feature is disabled.
DSIZE 18–16	**Destination Data Transfer Size** See the SSIZE definition.	
SOFF 15–0	**Source Address Signed Offset** Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.	

TCDx-2 Transfer Control Descriptor Word 2 DMA_Base + 0x1000 + (32 × n) + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	NBYTES															
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NBYTES															
TYPE	R/W															
RESET	Undefined															

Table 8-27. TCDx-2 Bit Descriptions

Name	Description	Value
NBYTES 31–0	**Inner Minor Byte Transfer Count** Number of bytes to transfer in each activation of the channel. As a channel is activated, the contents of the appropriate TCD are loaded into the DMA controller, and the appropriate reads and writes are performed until the complete byte transfer count is transferred. This operation cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into local memory, and the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.	The NBYTES value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.

TCDx-3 Transfer Control Descriptor Word 3 DMA_Base + 0x1000 + (32 × n) + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	SLAST															
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SLAST															
TYPE	R/W															
RESET	Undefined															

Table 8-28. TCDx-3 Bit Descriptions

Name	Description	Setting
SLAST 31–0	**Last Source Address Adjustment** The adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to restore the source address to the initial value or to adjust the address to reference the next data structure.	

TCDx-4 Transfer Control Descriptor Word 4 DMA_Base + 0x1000 + (32 × n) + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DADDR															
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DADDR															
TYPE	R/W															
RESET	Undefined															

Table 8-29. TCDx-4 Bit Descriptions

Name	Description	Setting
DADDR 31–0	**Destination Address** Memory address pointing to the destination data.	.

TCDx-5 Transfer Control Descriptor Word 5 DMA_Base + 0x1000 + (32 × n) + 0x14

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CITERE	CITERH						CITER								
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DOFF															
TYPE	R/W															
RESET	Undefined															

Table 8-30. TCDx-5 Bit Descriptions

Name	Description	Value
CITERE 31	**Channel-to-Channel Link Enable** As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITERH. The link target channel is activated by an internal mechanism that sets the TCDx-7[START] bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the CLE channel linking. The value of this bit must be equal to that of the BITERE bit. Otherwise, a configuration error is reported.	0 Channel-to-channel linking is disabled. 1 Channel-to-channel linking is enabled.
CITERH 30–25	**Current Iteration Count or Link Channel Number** If CITERE = 0, the CITERH bits form bits 14–9 appended with CITER to create a 15-bit CITER field. Otherwise, after the minor loop is exhausted, the DMA controller activates the channel defined by CITERH by setting that channel's TCDx-7[START] bit. The value in CITERH must not exceed the number of channels.	0 No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted.

Table 8-30. TCDx-5 Bit Descriptions

Name	Description	Value
CITER 24–16	**Current Major Iteration Count** This 9 or 15-bit count (if CITERE = 1, append CITERH to CITER) represents the current major loop count for the channel. It decrements each time the minor loop completes and updates in the transfer control descriptor memory. Once the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. When software initially loads CITER, it must be set to the same value as that contained in the TCDx-7[BITER] field. If the channel is configured to perform a single activation, the initial value of BITER and CITER should be 0x0001.	
DOFF 15–0	**Destination Address Signed Offset** Sign-extended offset applied to the current destination address to form the next-state value as each destination write completes.	

TCDx-6 Transfer Control Descriptor Word 6 DMA_Base + 0x1000 + (32 × n) + 0x18

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DLAST															
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DLAST															
TYPE	R/W															
RESET	Undefined															

Table 8-31. TCDx-6 Bit Descriptions

Name	Description	Setting
DLAST 31–0	**Last Destination Address** The last destination address adjustment or the memory address for the next TCD to be loaded into the channel (scatter/gather). If (TCD.ESG = 0), DLAST is the adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value or to adjust the address to reference the next data structure. Otherwise, this address points to the beginning of a 0-modulo-32 region containing the next TCD to be loaded into the channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32. Otherwise, a configuration error is reported.	

TCDx-7 Transfer Control Descriptor Word 7 DMA_Base + 0x1000 + (32 × n) + 0x1C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BITERE	BITERH							BITER							
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BWC		LCNUM						DONE	ACTIVE	CLE	ESG	DREQ	INTH	INTM	START
TYPE	R/W															
RESET	Undefined								0	0	Undefined					0

Table 8-32. TCDx-7 Bit Descriptions

Name	Reset	Description	Setting
BITERE 31	Undefined	**Channel-to-Channel Link Enable** As the channel completes the inner minor loop, this flag enables linking to another channel defined by BITERH. The link target channel is activated by an internal mechanism that sets the TCDx-7[START] bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of CLE channel linking. The value of this bit must be equal to that of the CITERE bit. Otherwise, a configuration error is reported.	0 Channel-to-channel linking is disabled. 1 Channel-to-channel linking is enabled.
BITERH 30–25	Undefined	**Major Iteration Count or Link Channel Number** If BITERE = 0, no channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. The BITERH bits form bits 14–9 appended with BITER to create a 15-bit BITER field. Otherwise, after the minor loop is exhausted, the DMA controller activates the channel defined by BITERH by setting that channel's TCDx-7[STRAR] bit. The value in BITERH must not exceed the number of channels.	
BITER 24–16	Undefined	**Beginning Major Iteration Count** This 9 or 15-bit count (if BITERE = 1, append BITERH to BITER) represents the beginning of the major loop count for the channel. Once the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the BITER field. When software initially loads BITER, it must be set to the same value as that contained in the TCDx-5[CITER] field.	
BWC 15–14	Undefined	**Bandwidth Control** Provides a mechanism to throttle the amount of bus bandwidth consumed by the DMA controller. In general, as the DMA controller processes the inner minor loop, it continuously generates read/write, read/write sequences until the minor count is exhausted. BWC forces the DMA controller to stall after completing each read/write access to control the bus request bandwidth as it appears to the crossbar switch.To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the DMA controller as it appears to the crossbar switch arbitrator for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles.	00 No DMA controller stalls. 01 Dynamic priority elevation. 10 DMA controller stalls for 4 cycles after each read/write. 11 DMA controller stalls for 8 cycles after each read/write.

Table 8-32. TCDx-7 Bit Descriptions (Continued)

Name	Reset	Description	Setting
LCNUM 13–8	Undefined	**Link Channel Number** If TCDx-7[CLE] = 0, no channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise, after the major loop counter is exhausted, the DMA controller activates the channel defined by LCNUM by setting that channel's TCDx-7[START] bit. The value in LCNUM must not exceed the number of implemented channels.	
DONE 7	0	**Channel Done** This flag indicates that the DMA controller has finished processing the outer major loop. The DMA controller sets this bit as the CITER count reaches zero. Software or hardware clears this bit when the channel is activated.	0 The channel is not done. 1 The channel is done.
ACTIVE 6	0	**Channel Active** This flag signals that the channel is currently executing. This bit is set as each inner minor loop starts to execute, and the DMA controller clears it as the inner minor loop completes or a configuration error is detected	0 The channel is not active. 1 The channel is active.
CLE 5	Undefined	**Channel-to-Channel Linking** As a channel completes the outer major loop, this flag enables the linking to another channel, defined by LCNUM. The link target channel is activated by an internal mechanism that sets the TCDx-7[START] bit of the specified channel. This field cannot be modified when the TCDx-7[DONE] bit is set.	0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
ESG 4	Undefined	**Enable Scatter/Gather Processing** As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If this bit is enabled, the DMA controller uses TCDx-6[DLAST] as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure that is loaded as the TCD into local memory. This field cannot be modified when the TCDx-7[DONE] bit is set.	0 The current channel's TCD has the normal format. 1 The current channel's TCD specifies a scatter/gather format.
DREQ 3	Undefined	**Disable Request** If this flag is set, the DMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero.	0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the outer major loop completes.

Table 8-32. TCDx-7 Bit Descriptions (Continued)

Name	Reset	Description	Setting
INTH 2	Undefined	**Half-Point Interrupt** Enables/disables an interrupt when the major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the DMA controller is (CITER == (BITER >> 1)). This halfway point interrupt request supports double-buffered schemes or other types of data movement in which the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when the BITER values are less than two.	0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
INTM 1	Undefined	**Interrupt Major** Enables/disables an interrupt when the major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero.	0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
START 0	0	**Channel Start** If this flag is set, the channel is activated. The DMA hardware automatically clears this flag bit after the channel is active.	0 The channel is not explicitly started. 1 The channel is explicitly activated.

Memory Controller 9

The memory controller provides a glueless interface between the internal MSC711x bus and the external double data rate (DDR) SDRAM memory modules. The memory controller establishes an interface between MSC711x devices and external memories by translating internal bus accesses to appropriate address, data, and control signals for DDR SDRAMs. This fully programmable DDR SDRAM controller supports most JEDEC standard x8 or x16 DDR memories available today, including buffered and unbuffered DIMMs (although mixing unbuffered and registered DIMMS in the same system is not supported). Dynamic power management and auto-precharge modes simplify memory system design.

9.1 Features

The DDR memory controller includes these distinctive features:

- Glueless interface to JEDEC-compliant first generation DDR SDRAMs (x8, x16, or x32 devices).
- 16-bit or 32-bit SDRAM data bus, configured by the DEVCFG[DDR] bit as described in **Section 7.4.3**, *Device Identification and Configuration*, on page 7-16.
- Clock ratio of 2:1 between the memory controller clock frequency and SDRAM clock.
- Programmable settings for all SDRAM timing parameters.
- 14-bit SDRAM address.
- Two chip selects for up to two physical banks.
- Support for unbuffered and registered DIMMs.
- Single access or bursting.
- Data mask signals and read-modify-write.
- Open page management (dedicated entry for each sub-bank).
- Optional auto-precharge mode.
- Sleep power management mode.
- Two-entry input request queue.

9.2 DDR Memory Controller Signal Description

This section summarizes the DDR memory controller external signals. For detailed descriptions of these signals, consult **Section 2.3**, *Memory System Interface (DDR Controller)*, on page 2-11.

Table 9-1 lists the signals of the DDR memory controller, showing how the signals are grouped.

Table 9-1. DDR Memory Interface Pin Summary

Name	Function/Description	Reset	Pins	I/O
Address and Chip Selects				
A[13:0]	Address bus	All zeros	14	O
BA[1:0]	Logical bank address	All zeros	2	O
$\overline{\text{CS}}$[1–0]	Chip selects	All ones	2	O
Data				
D[31–0]	Data bus	All zeros	32	I/O
Data Strobes and Masks				
DQS[3–0]	Data strobes	All zeros	4	I/O
DQM[3–0]	Data mask	All zeros	4	O
Memory Control				
$\overline{\text{RAS}}$	Row address strobe	Deasserted	1	O
$\overline{\text{CAS}}$	Column address strobe	Deasserted	1	O
$\overline{\text{WE}}$	Write enable	Deasserted	1	O
Clocking				
CKE	Clock enable	Deasserted	1	O
CK	DDR clock	—	1	—
$\overline{\text{CK}}$	DDR Clock — Inverted Sense	—	1	—

Note: A bar over a signal name indicates that the signal is active low, such as $\overline{\text{RAS}}$ (row address strobe). Active-low signals are referred to as asserted (active) when they are low and deasserted when they are high. Signals that are not active low are referred to as asserted when they are high and deasserted when they are low.

Table 9-2 shows the mapping from the DDR memory controller address pins to the JEDEC standard.

Table 9-2. Memory Address Signal Mappings

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
		SDRAM 168–Pin DIMM
MSB	A13	A13
	A12	A12
	A11	A11
	A10	A10(AP)
	A9	A9
	A8	A8
	A7	A7
	A6	A6
	A5	A5
	A4	A4
	A3	A3
	A2	A2
	A1	A1
	A0	A0
	BA1	BA1
LSB	BA0	BA0

The manner in which an address from the MSC711x device maps to a multiplexed DDR address and then drives these pins is covered in **Section 9.6.3**, *DDR SDRAM Address Multiplexing*, on page 9-25.

9.3 Architecture

The DDR SDRAM memory controller controls processor and I/O interactions with system memory. It supports JEDEC-compliant first generation dual-data rate DDR SDRAMs. Within the memory system, a wide range of memory devices can be mapped to any arbitrary chip select. However, registered DIMMs cannot be mixed with unbuffered DIMMs.

Figure 9-1 shows an example system-level view of an MSC711x device connected to a single x16 DDR device. The MSC711x masters access the memory controller through the crossbar switch and the memory controller interface (MCIF). See **Chapter 10**, *Memory Controller Interface*, which directly controls the DDR pins. All accesses to the DDR memory controller occur from AHB masters through the crossbar switch and the MCIF.

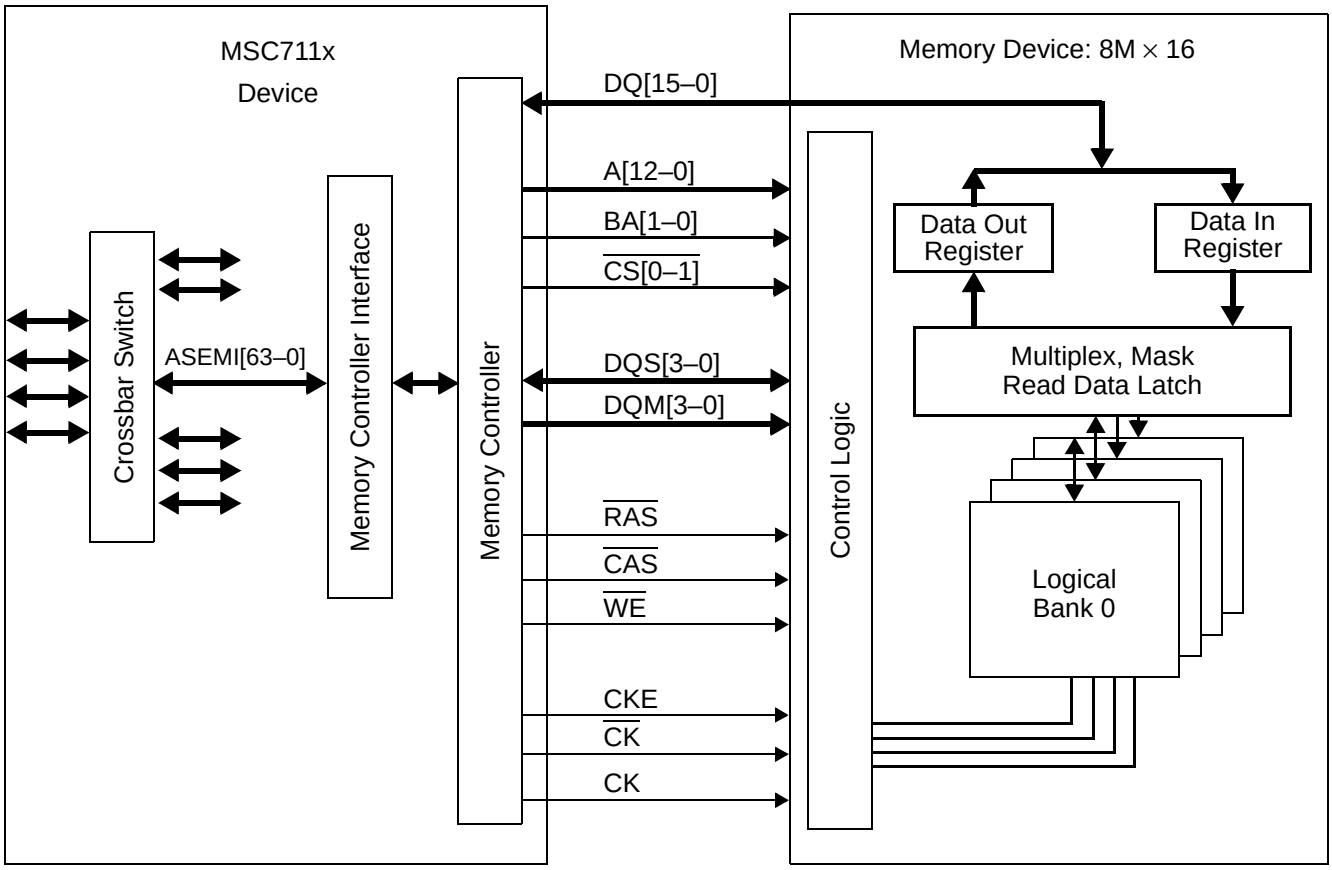


Figure 9-1. System-Level View of the DDR Memory Controller

The DDR SDRAM interface supports two physical banks of 32 bit-wide memory. A physical bank is a portion of the address map assigned to a memory controller chip select, and a logical bank is one of the banks within a DDR SDRAM device, which is specified by a bank address. **Figure 9-2** shows a block diagram of the memory controller.

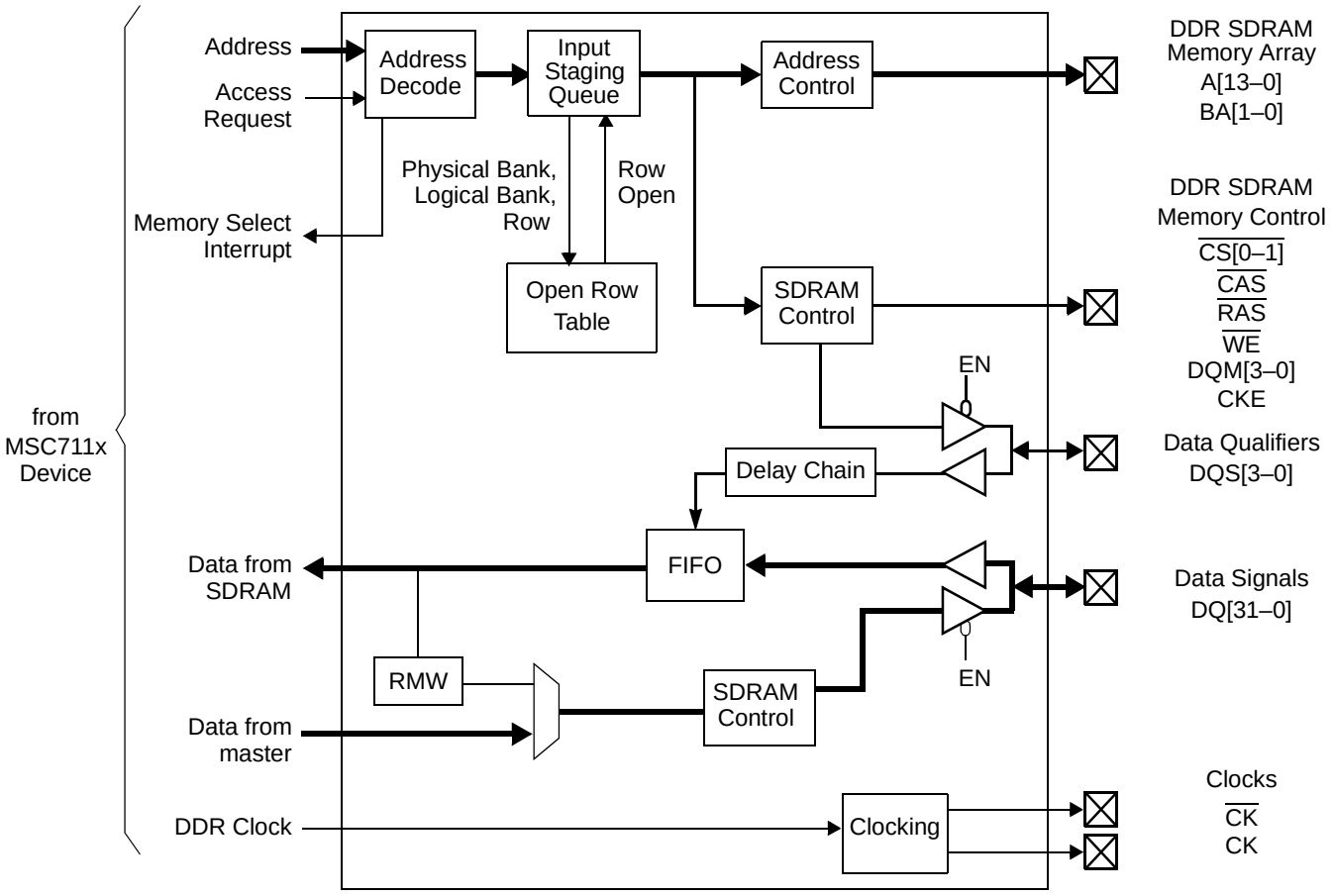


Figure 9-2. DDR Memory Controller Block Diagram

The memory controller processes accesses from the MSC711x device as follows:

1. Receives requests from the internal mastering device.
2. Decodes the address to generate the physical bank, logical bank, row, and column addresses.
3. Loads the transaction into the input staging queue with the decoded information.
4. Compares the two entries of the input queue with values in the open row table to determine whether the address maps to an open page.
5. If the address from either entry does not map to an open page, issues an ACTIVATE command for the entry, and the lowest entry has priority over the next lowest. Commands are always issued from the lowest entry in the input queue.

Read and write accesses to the DDR SDRAM are burst oriented; accesses start at a selected location and continue for a programmed number of higher locations (2, 4, or 8) in a programmed sequence, (sequential or interleaved).

9.3.1 DDR SDRAM Configurations

The DDR memory controller supports many different DDR SDRAM configurations, according to the following characteristics:

- Number of physical banks (that is, number of chip selects used): 1 or 2
- Data path size: 16-pin or 32-pin
- Number of data pins on the DDR SDRAM device(s): x8, x16, or x32

Physical bank sizes range from 8 MB to 512 MB. SDRAMs of different sizes can be used in the same system. Device densities range from 64 Mbit to 1 Gbit and are specified by 14 multiplexed address signals and two logical bank select signals. The DDR SDRAM physical banks can be built from standard memory modules or directly-attached memory devices. Byte lane selection for memory accesses is provided by 4 data qualifier ($\overline{\text{DQM}}$) signals.

Note: An 8-bit DDR SDRAM device has a $\overline{\text{DQM}}$ signal and eight data signals (DQ[7–0]). A 16-bit DDR SDRAM device has two $\overline{\text{DQM}}$ signals associated with specific halves of the sixteen data signals (DQ[15–8] and DQ[7–0]).

Although the DDR memory controller multiplexes the row and column address bits onto 14 memory address signals and two logical bank select signals, individual physical banks can be implemented with memory devices requiring fewer than 28 address bits. Each physical bank may be individually configured to provide from 12 to 14 row address bits, plus two logical bank-select bits and from eight to eleven column address bits. **Table 9-3** shows different configurations of the DDR memory controller in 16-pin mode.

Table 9-3. Configurations in 16 Data Pin Mode

SDRAM Device Size (in Bits)	Device Configuration	Row × Column Bits	Number of Devices per Physical Bank[1]	Size (in MB) of One Physical Bank[1]	Size (in MB) of Two Physical Banks[2]
64Mb	8M × 8	12 × 9	2	16	32
64Mb	4M × 16	12 × 8	1	8	16
128Mb	16M × 8	12 × 10	2	32	64
128Mb	8M × 16	12 × 9	1	16	32
256Mb	32M × 8	13 × 10	2	64	128
256Mb	16M × 16	13 × 9	1	32	64
512Mb	64M × 8	13 × 11	2	128	256
512Mb	32M × 16	13 × 10	1	64	128
1Gb	128M × 8	14 ×11	2	256	512
1Gb	64M × 16	14 × 10	1	128	256

Notes: 1. One physical bank corresponds to a set of devices that share a single chip select.
2. Two physical banks correspond to two sets of devices, where each set has a single chip select.

Table 9-4 shows the different configurations of the DDR memory controller in 32-pin mode.

Table 9-4. Supported Configurations, 32 Data Pin Mode

SDRAM Device Size (in bits)	Device Configuration	Row x Column Bits	Number of Devices per Physical Bank[1]	Size (in MB) of One Physical Bank[1]	Size (in MB) of Two Physical Banks[2]
64Mb	8M × 8	12 × 9	4	32	64
64Mb	4M × 16	12 × 8	2	16	32
128Mb	16M × 8	12 × 10	4	64	128
128Mb	8M × 16	12 × 9	2	32	64
256Mb	32M x 8	13 x 10	4	128	256
256Mb	16M × 16	13 × 9	2	64	128
512Mb	64M × 8	13 × 11	4	256	512
512Mb	32M × 16	13× 10	2	128	256
1Gb	128M × 8	14 × 11	4	512	1 GB
1Gb	64M × 16	14 × 10	2	256	512

Notes: **1.** One physical bank corresponds to a set of devices that share a single chip select.
2. Two physical banks correspond to two sets of devices, where each set has a single chip select.

If a transaction request is issued to the DDR memory controller and the address does not lie within a programmed address range for an enabled chip select, a memory select error is flagged. Errors are described in detail in **Section 9.6.5**, *Error Detection and Management*, on page 9-30.

If the starting and ending addresses of a disabled bank overlap with the address space of an enabled bank, there may be system memory corruption in the overlapping address range. The starting and ending addresses of unused memory banks should be mapped to unused memory space.

Using a memory-polling algorithm at power-on reset or querying the JEDEC serial presence detect capability of memory modules, system firmware configures the DDR memory controller to map the size of each bank in memory using the memory-boundary registers. The memory controller uses its bank map to assert the appropriate $\overline{\text{CS}n}$ signal for memory accesses according to the provided bank starting and ending addresses. The memory banks are not required to be mapped to a contiguous address space.

9.3.2 Configuration Examples

The examples presented in this section demonstrate the capabilities of the DDR memory controller. **Figure 9-3** shows a typical byte-wide DDR device; its internal organization is shown in **Figure 9-4**.

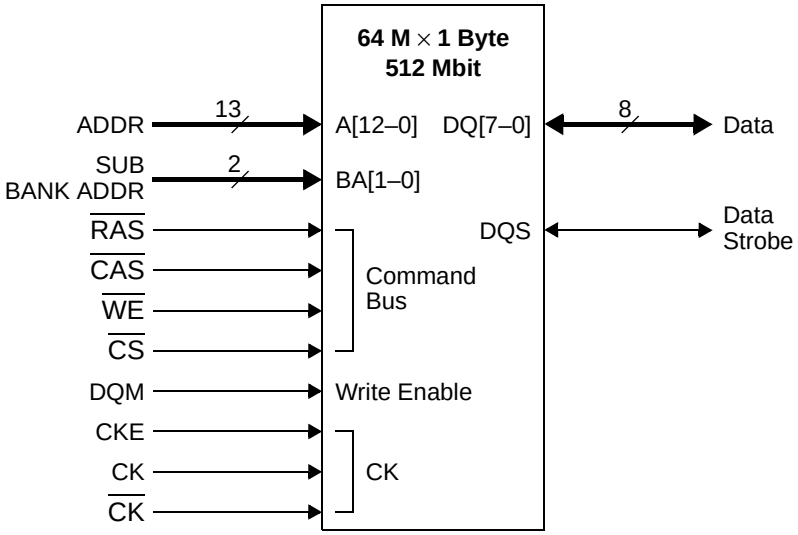


Figure 9-3. Typical DDR SDRAM Device

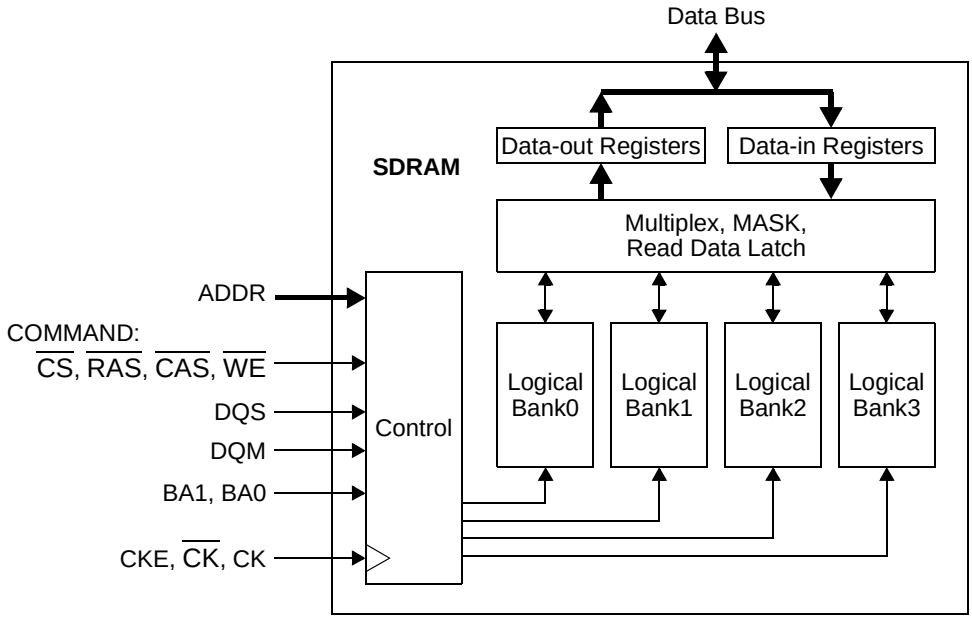


Figure 9-4. Typical Dual Data Rate SDRAM Internal Organization

Different configurations may require buffering of certain address and control lines. Analysis of the device's AC timing specifications, desired memory operating frequency, capacitive loads, and board routing loads can assist you in determining signal buffering requirements. While the DDR memory controller drives 14 address pins, the following examples use only 12 bits.

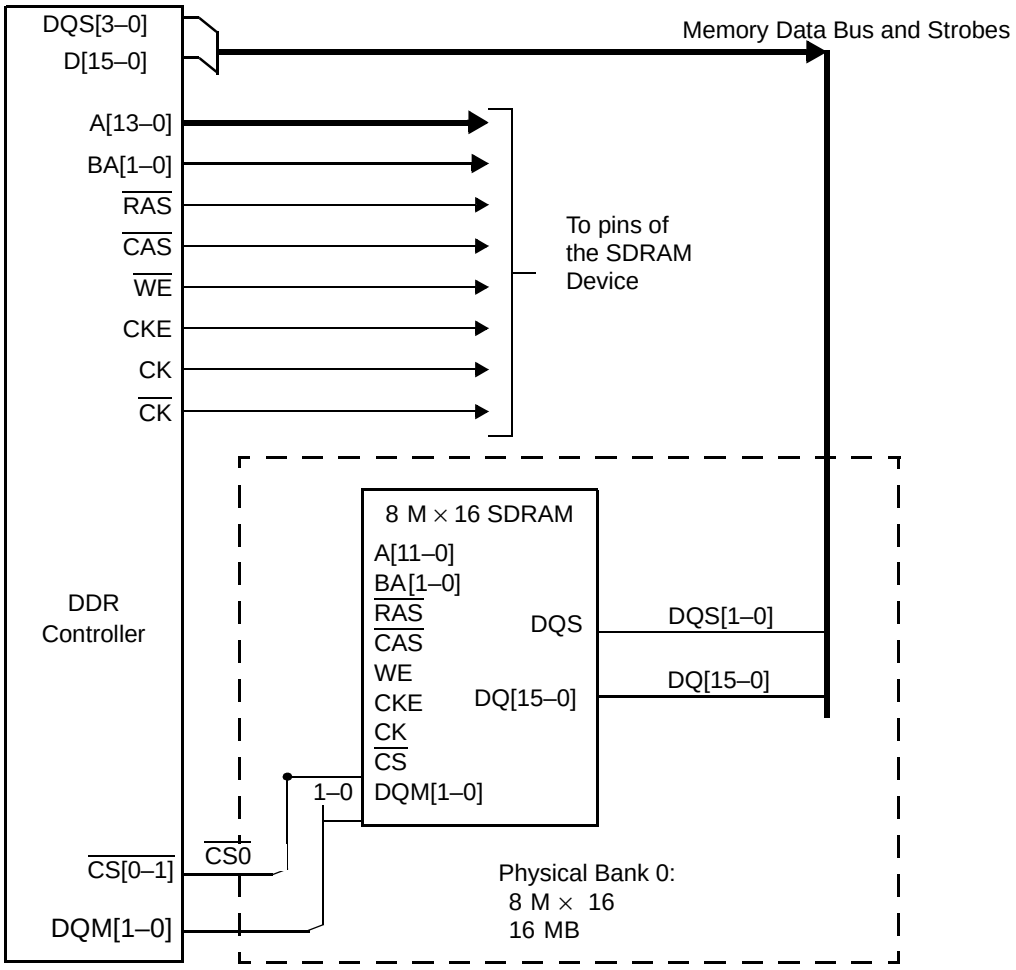
9.3.2.1 Fan-Out and Termination

Fan-out and termination depends on the MSC711x controller drive strength, output slew rate, and board topology. The typical drive on a single signal is up to six devices. Some stacked modules drive up to eight loads, although careful simulation is required.

Keep the following points in mind when setting up your system to interface with DDR SDRAM devices:

- If there are several SDRAM devices, use zero-delay PLL clock buffers (JEDEC-JESD82 standard). These buffers are designed for DDR applications.
- The CK/$\overline{CK}$ signal pair typically drives no more than six devices.
- PCB traces for DDR clock signals should be short and all on the same layer.
- There should be an effective 100–120 ohms of termination between CK and $\overline{CK}$. It may be useful to add a small compensating capacitor between CK and $\overline{CK}$.
- DDR SDRAM manufacturers provide detailed information on PCB layout/termination.

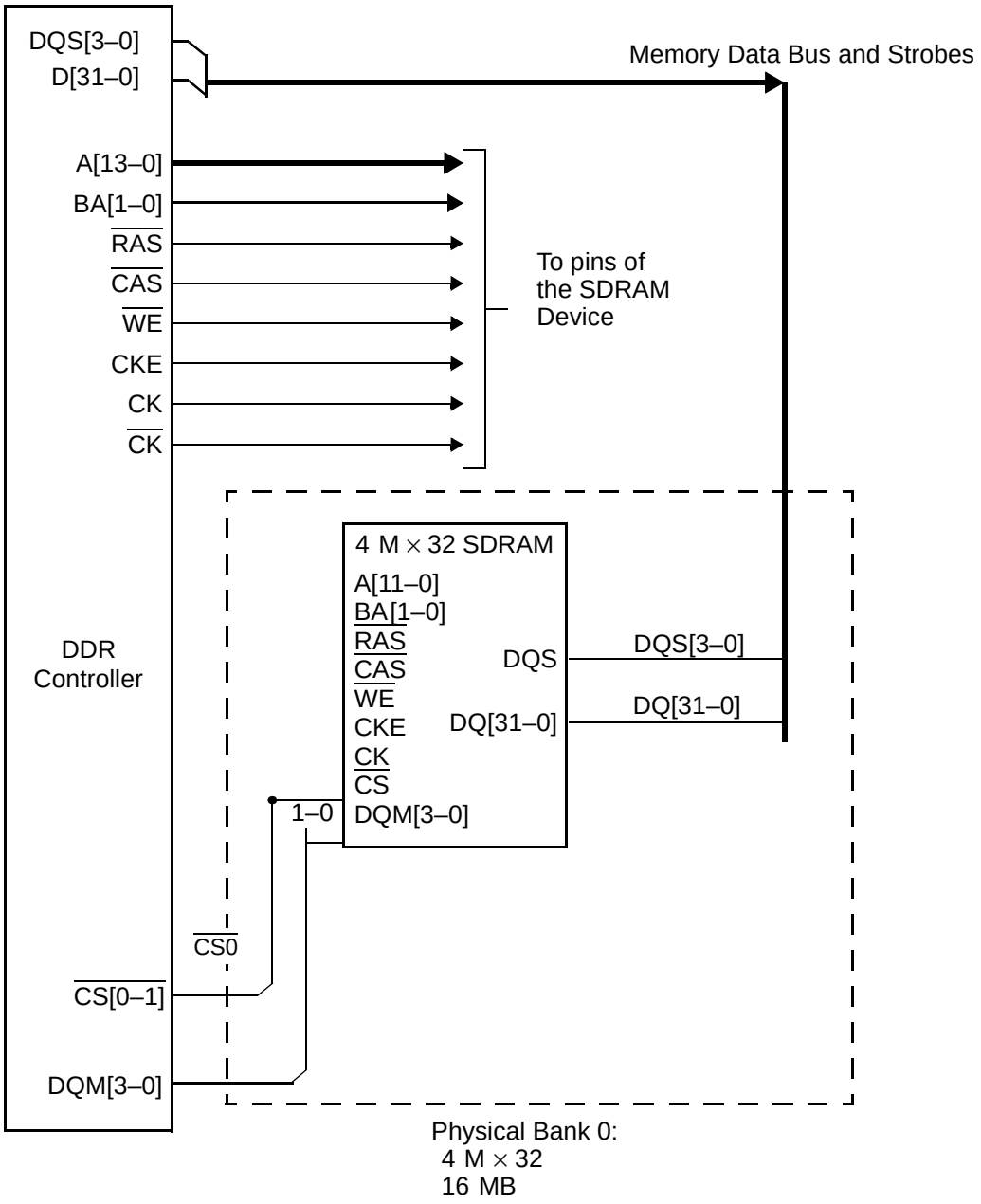
It is important to simulate the design of the MSC711x and DDR devices. Fan-out and termination is high speed and sensitive. Specialized termination is required. See the *Technical Data* sheet and design checklist application note for details on termination. **Figure 9-5** shows a simple DDR SDRAM configuration with a single physical bank containing one 8M × 16 DDR module for a total of 16 MB of system memory. The memory controller is configured for 16-pin operation.



1. All signals are connected in common (in parallel) except for the DQM signals.
2. $\overline{CK}$ is not shown on the device but must be connected.

Figure 9-5. Simple 16-Pin DDR SDRAM Configuration (16 MB)

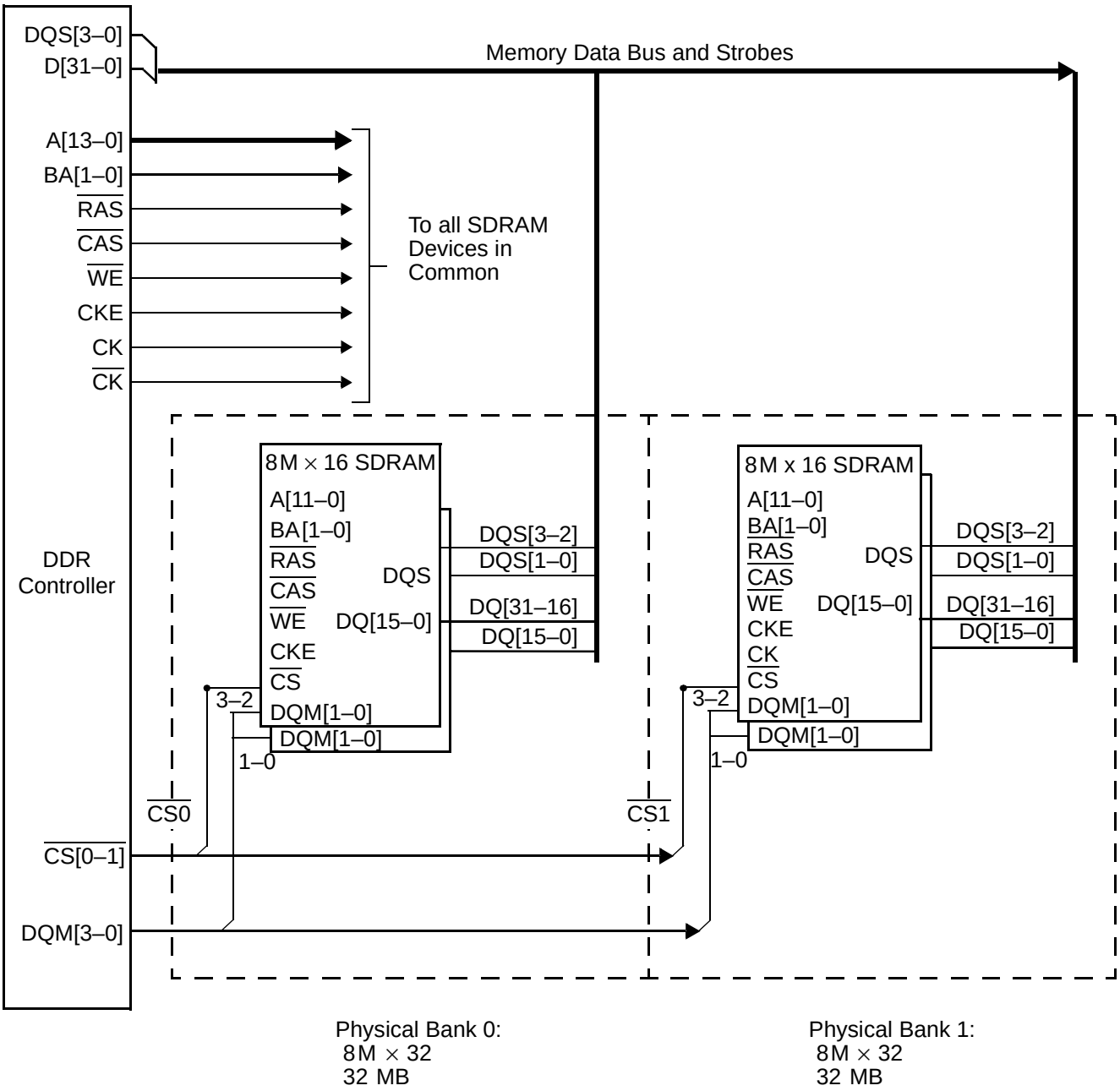
Figure 9-6 shows a simple DDR SDRAM configuration with a single physical bank containing one 4M × 32 DDR module for a total of 16 MB of system memory. The memory controller is configured for 32-pin operation.



1. All signals are connected in common (in parallel) except for the DQM signals.
2. $\overline{CK}$ is not shown on the memory device but must be connected.

Figure 9-6. Simple 32-Pin DDR SDRAM Configuration (16 MB)

Figure 9-7 shows a DDR SDRAM configuration with two physical banks, each composed of two 8 M × 16 DDR modules for a total of 64 MB of system memory. The memory controller is configured for 32-pin operation.



1. All signals are connected in common (in parallel) except for $\overline{CS0}$ and $\overline{CS1}$, the DQM signals, and the data bus signals.
2. Each chip select signal corresponds to a separate physical bank of memory.
3. Buffering may be needed for large memory arrays.
4. $\overline{CK}$ is not shown on the these memory devices but must be connected.

Figure 9-7. Example Two-Bank 64 MB DDR SDRAM Configuration

9.4 JEDEC-Standard DDR SDRAM Interface Commands

The DDR memory controller performs all read or write accesses to DDR SDRAM using JEDEC-standard DDR SDRAM interface commands. The SDRAM device samples command and address inputs on rising edges of the memory clock; data is sampled using both the rising and falling edges of DQS. Data read from the DDR SDRAM is also sampled on both edges of DQS.

The DDR SDRAM interface commands are sent from the MSC711x memory controller to the DDR SDRAM devices. The ACTIVATE command (row activate) opens a row for a new access to the SDRAM device. The PRECHARGE command closes a row after accesses to the row are complete. The commands are listed here and summarized in **Table 9-5**. All actions for these commands are described from the perspective of the SDRAM device.

- ROW ACTIVATE. Latches row address and initiates a memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored by a PRECHARGE command before another row activate occurs.
- PRECHARGE. Restores data from the sense amplifiers to the appropriate row. Also initializes the sense amplifiers in preparation for reading another row in the memory array, (performing another activate command). Precharge must occur after a read or write, if the row address changes on the next Open Page mode access.
- READ. Latches the column address and transfers data from the selected sense amplifier to the output buffer as determined by the column address. During each succeeding clock edge, additional data is driven without additional read commands. The amount of data transferred is determined by the burst size, which defaults to 4.
- WRITE. Latches the column address and transfers data from the data pins to the selected sense amplifier as determined by the column address. During each succeeding clock edge, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data transferred is determined by the burst size, which is set to four by the DDR memory controller.
- REFRESH (similar to $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$). Causes a row to be read in all logical banks (JEDEC SDRAM) as determined by the refresh, row address counter. This refresh row address counter is internal to the SDRAM. After it is read, the row is automatically rewritten in the memory array. All logical banks must be in a precharged state before a refresh executes.
- MODE REGISTER SET (for configuration). Sets DDR SDRAM options, which are $\overline{\text{CAS}}$ latency, burst type, and burst length. $\overline{\text{CAS}}$ latency can be chosen as provided by the preferred SDRAM. Some SDRAMs provide $\overline{\text{CAS}}$ latency {1,2,3}, some provide $\overline{\text{CAS}}$ latency {1,2,3,4}, and so on. Burst type is always sequential. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, and page size, this memory controller supports only a burst length of 4.

  The mode register set command is performed during system initialization. Software sets parameters such as mode register data, $\overline{\text{CAS}}$ latency, burst length, and burst type in

SMCFG[SDMOD] and the DDR memory controller transfers them to the SDRAM array after SCFG[MEMEN] is set.

- SELF REFRESH (for long periods of standby). Automatically generates internal refresh cycles to keep the data in all memory banks refreshed. Before this command executes, all logical banks are in a precharged state.

Table 9-5. SDRAM Command Table

Operation	$\overline{CS}$	CKE Previous	CKE Current	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	BA	A10	A
Precharge and Activate Commands									
Activate	L	H	H	L	H	H	Logical bank select	Row	Row
Precharge select logical bank	L	H	H	L	H	L	Logical bank select	L	X
Precharge all logical banks	L	H	H	L	H	L	X	H	X
Read and Write Commands									
Read	L	H	H	H	L	H	Logical bank select	L	Column
Read with auto precharge	L	H	H	H	L	H	Logical bank select	H	Column
Write	L	H	H	H	L	L	Logical bank select	L	Column
Write with auto precharge	L	H	H	H	L	L	Logical bank select	H	Column
Mode and Refresh Commands									
Mode register set	L	H	H	L	L	L	Opcode	Opcode	Opcode and mode
Auto refresh	L	H	H	L	L	H	X	X	X
Self refresh	L	H	L	L	L	H	X	X	X
No Operation (NOP)	L	H	H	H	H	H	X	X	X

9.5 Operating Modes

The DDR memory controller runs in Open Page mode and an Auto-Precharge mode. In Open Page mode, there are up to 8 open pages, one for each of the four logical banks associated with each of the two chip selects. In Auto-Precharge mode, pages are closed after the read or write access. Accesses to closed pages start with the an ACTIVATE command ($\overline{RAS}$ signal assertion) followed by a READ or WRITE command ($\overline{CAS}$ assertion). Accesses to open pages eliminate the need for the ACTIVATE command. Instead, the access starts with a READ or WRITE command ($\overline{CAS}$ assertion). The address bits registered coincident with the READ or WRITE READ or WRITE command ($\overline{CAS}$ signal) specify the logical bank and starting column for the burst access.

The address bits registered coincident with the ACTIVATE command ($\overline{\text{RAS}}$ signal) specifies the logical bank and row to be accessed. The address coincident with the READ or WRITE command ($\overline{\text{CAS}}$ signal) specifies the logical bank and starting column for the burst access.

9.5.1 Open Page Mode

Page mode allows efficient accesses. After the DDR memory controller opens a page, subsequent accesses no longer need to issue an ACTIVATE command. This mode is most efficient in bursting and is commonly used instead of Auto-Precharge mode. When the DDR memory controller operates in Open Page mode, it retains the currently active SDRAM page by not issuing a PRECHARGE command. The page remains opens until one of the following conditions occurs:

- Refresh interval is attained.
- The user-programmable value controlled by SICFG[REFINT] is exceeded.
- Another transaction is issued to the same physical bank and logical bank but to a different row.

The amount of time (in DDR clock cycles) that each pages remain open is programmable via SICFG[REFINT].

9.5.2 Auto-Precharge Mode

In Auto-Precharge mode, the memory controller issues a PRECHARGE command to a logical bank after every read or write transaction. Auto-Precharge mode can be selected for all chip selects for a global auto precharge by clearing SICFG[PI] to 0, or it can be set individually on physical banks using CSxCFG[APxEN].

This mode is useful only when most accesses to the DDR are not bursted and the accesses occur over many different rows. It is less efficient than Page mode for bursting. In Auto-Precharge mode, an application sometimes configures one portion of the DDR address space (that is, one chip select) for Page mode for efficient bursting, and another portion for Auto-precharge mode.

In closed page mode, the DDR memory controller uses the auto-precharge feature so that the SDRAM automatically closes the page after the read or write access. An automatic close uses A[10] of the address during the command phase of the access to enable Auto-Precharge mode. Auto-Precharge is non-persistent in that it is either enabled or disabled for each individual read or write command. However, it can be enabled or disabled separately for each chip select. This mode is for applications in which many random accesses are performed to the DDR memory and the accesses occur over many different rows.

Open Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save two to three clock cycles for subsequent burst accesses that hit in an active page. Also, better performance can be obtained by using more banks, especially in systems with many different channels.

9.5.3 DDR SDRAM 2T Timing Mode

For heavily loaded systems, the 2T Timing mode is available. In this mode, the address and control signals are held for two full cycles for every DRAM command used. For example, if two unbuffered DIMMs are used, there can be 36 loads on the address, resulting in very slow slew rates and delay on these signals. The board design must be very carefully managed to ensure that this configuration meets all DDR SDRAM specifications. Another option is to enable 2T timing to give the address and control more margin. The chip selects are still asserted for only one cycle, but each chip select has at most 9 loads.

9.5.4 Low-Power Modes

Modes for reducing power consumption are:

- Normal operation: Dynamic Power Management mode
- Stop mode operation: Self Refresh or No Refresh modes

In normal operation, the DDR memory controller can reduce power consumption by negating the SDRAM CKE signal when no transactions are pending to the SDRAM. The dynamic power-saving mode uses the CKE DDR SDRAM pin to power down dynamically when there is no system memory activity. The CKE pin is deasserted when both no memory refreshes and no memory accesses are scheduled. CKE is reasserted when a new access or refresh is scheduled or the dynamic power mode is disabled. This mode is controlled with SCFG[DPWR]. Dynamic power management mode gives you tight control of the power consumption by trading power for performance using SICFG[REFINT]. Powering up the DDR SDRAM when a new memory reference is scheduled causes a latency penalty of one clock, as shown in **Figure 9-8**.

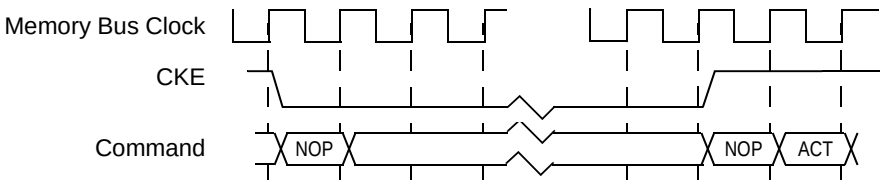


Figure 9-8. DDR SDRAM Power-Down Mode

In normal operation, the DDR memory controller supplies the normal auto refresh to SDRAM. In Stop mode, the DDR memory controller can be configured to take advantage of self-refreshing SDRAMs or to provide no refresh support. Self-refresh support is enabled with the SCFG[SREN] bit. In the absence of refresh support, system software must first preserve DDR SDRAM data, such as by copying the data to disk, before it enters power-saving mode. When the system enters Debug mode, the memory controller continues to refresh the DDR SDRAM devices. It is not affected by SCFG[SREN]. **Figure 9-9** and **Figure 9-10** show the entry and exit timing for self-refreshing SDRAMs.

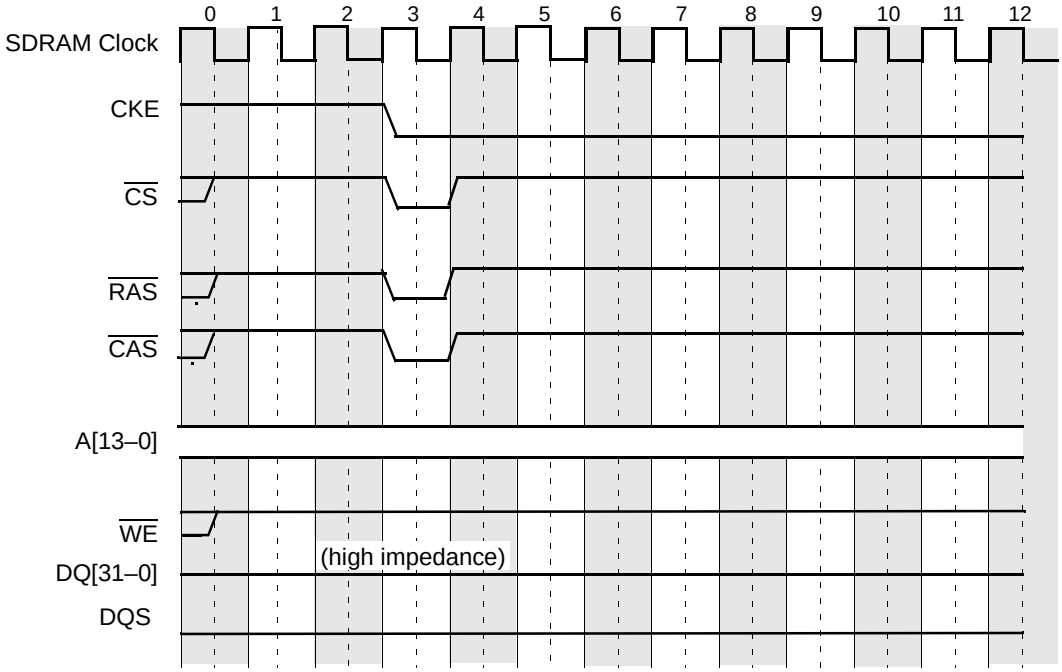


Figure 9-9. DDR SDRAM Self-Refresh Entry Timing

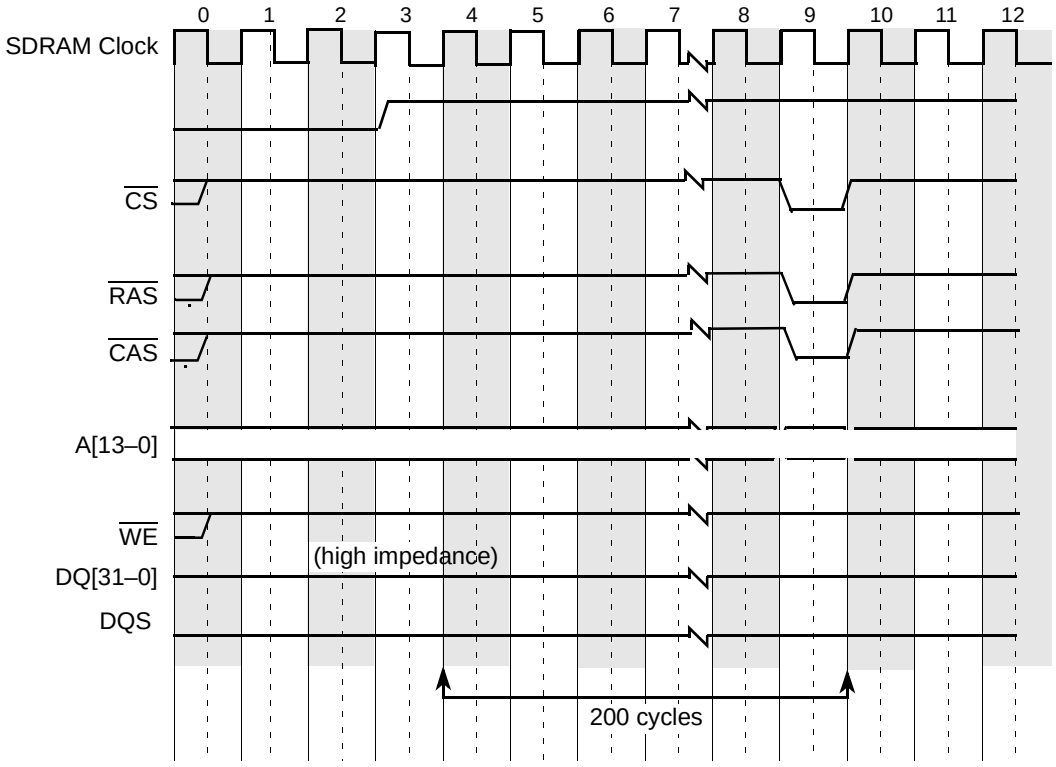


Figure 9-10. DDR SDRAM Self-Refresh Exit Timing

9.6 Interface Characteristics

The DDR memory controller is clocked with the DDR clock, which is converted with a 2:1 ratio to the frequency required for the DDR clock pins. The data interface is source synchronous, so the data source provides a clocking signal to synchronize data reception. These bidirectional data strobes (DQS[3–0]) are inputs to the controller during reads and are outputs during writes. The DDR SDRAM specification requires the data strobe signals to be centered within the data tenure during writes and to be offset by the controller to the center of the data tenure during reads. Therefore, there are delay chains for the data strobe signals during reads and a delay chain on the data multiplexer select during writes.

Note: The DQS signals should not be intentionally delayed on the board. Memory controller circuitry already handles this function.

9.6.1 SDRAM Interface Timing

There are four-beat bursts to SDRAM. For single-beat reads, the DDR memory controller performs a four-beat burst read, but ignores the last three beats, as shown in the first row of **Table 9-10** and **Table 9-11**. Single-beat writes are performed by masking the last three beats of the four-beat burst using the data mask signals (DQM). In 16-pin mode, writes smaller than 16-bits are performed by appropriately activating the data mask. In 32-pin mode, writes smaller than 32-bits are performed by appropriately activating the data mask.

Note: If a second read or write is pending, reads shorter than four beats are not terminated early, even when some of the data is irrelevant.

To accommodate available memory technologies across a wide spectrum of operating frequencies, the DDR memory controller allows the setting of the intervals defined in **Table 9-6** with a granularity of one memory clock cycle, except for CASLAT, which can be programmed with half clock granularity.

Table 9-6. DDR SDRAM Interface Timing Intervals

Timing Intervals	Definition
ACTACT	The number of clock cycles from an active command to a logical bank until another active command is allowed for any logical bank within that bank. This interval is listed in the SDRAM AC specifications.
ACTPRE	The number of clock cycles from an activate command until a precharge command is allowed. This interval is listed in the SDRAM AC specifications.
ACTRW	The number of clock cycles from an activate command until a read or write command is allowed. This interval is listed in the SDRAM AC specifications.
PI	The number of DDR clock cycles to maintain a page open after an access. A subsequent access can generate a page hit during this interval. A page hit reloads the PI counter. When the interval expires, a precharge is issued to the page as soon as possible, closing the associated row on the DDR SDRAM device.

Table 9-6. DDR SDRAM Interface Timing Intervals (Continued)

Timing Intervals	Definition
CASLAT	The READ latency (CASLAT) is the delay, in clock cycles, between the SDRAM registration of a READ command and the availability of the first piece of output data. If a READ command is registered at clock edge n, and the latency is m clocks, the data is available nominally coincident with clock edge $n + m$.
PREACT	The number of clock cycles from a precharge command until an activate or a refresh command is allowed. This interval is listed in the SDRAM AC specifications.
REFINT	Refresh interval. The number of memory bus clock cycles between refresh cycles. One row is refreshed in each SDRAM bank during each refresh cycle. The value of REFINT depends on the specific SDRAMs used and the frequency of the interface.
REFREC	The number of clock cycles from the refresh command until an activate command is allowed. This can be calculated by referring to the SDRAM AC specification, which indicates a maximum refresh to activate the interval in nanoseconds.
WRDD	Provides different options for the timing between a write command and the write data strobe. Therefore, write data can be sent later than the nominal time to meet the SDRAM timing requirement between the registration of a write command and the reception of a data strobe associated with the write command. The specification states that the data strobe cannot be received earlier than 75 percent of a cycle or later than 125 percent of a cycle from the registration of a write command. This parameter is not defined in the SDRAM specification. It is implementation-specific, defined for the DDR memory controller in TCFG2.
WRREC	The number of clock cycles from the last beat of a write until a precharge command is allowed. This interval, write recovery time, is listed in the SDRAM AC specifications.
WRRD	Last write pair to read command issue for DDR and DDRII. Controls the number of clock cycles from the last write data pair to the subsequent read command to the same bank.

Note: Also, see the TCFG2[CPO] field, which adjusts parameter timing.

The value of the parameters in **Table 9-6** (in whole DDR clock cycles) must be programmed into the appropriate registers by boot code at system start-up. At reset, system software must optimally configure the SDRAM timing parameters, for both read and write timing. The configuration process must be completed and the DDR SDRAM initialized before any accesses to SDRAM are attempted.

9.6.2 DDR Access Timings

This section shows the timing for different memory controller accesses to SDRAM. All signal transitions occur on the rising edge of the memory bus clock. All waveforms use 1T timing, which is programmed in the SCFG[2TEN] field.

Figure 9-11 shows a case in which two single-beat read operations are requested. The memory controller issues an ACTIVATE command, indicated by the assertion of $\overline{\text{RAS}}$, and opens the page in the appropriate logical bank of the DDR SDRAM device, if the page is not already open. Next, it issues two read commands because it is requesting two read accesses. These commands are indicated by the assertion of $\overline{\text{CAS}}$. A preamble of one DDR clock cycle occurs after the first request, which is the first assertion of $\overline{\text{CAS}}$, in which the DQS pins are driven low from their

tri-state condition. After this preamble, the first data is read from the DDR SDRAM device using the delay specified by the TCFG1[CASLAT] bit.

Since this example demonstrates two read requests, the waveform in **Figure 9-5** shows two 4-beat bursts issued back-to-back. These are single read requests instead of bursts, so only the first beat of each 4-beat burst (D0 and D4) contains the desired data. The remaining beats are ignored. Note that single-beat read operations are identical to burst reads, except that the remaining beats are not ignored in the burst. After the last data is transferred, the DQS signals return to their tri-state condition after a short postamble. The DQM pins are not used in read operations, so they are not shown in this figure.

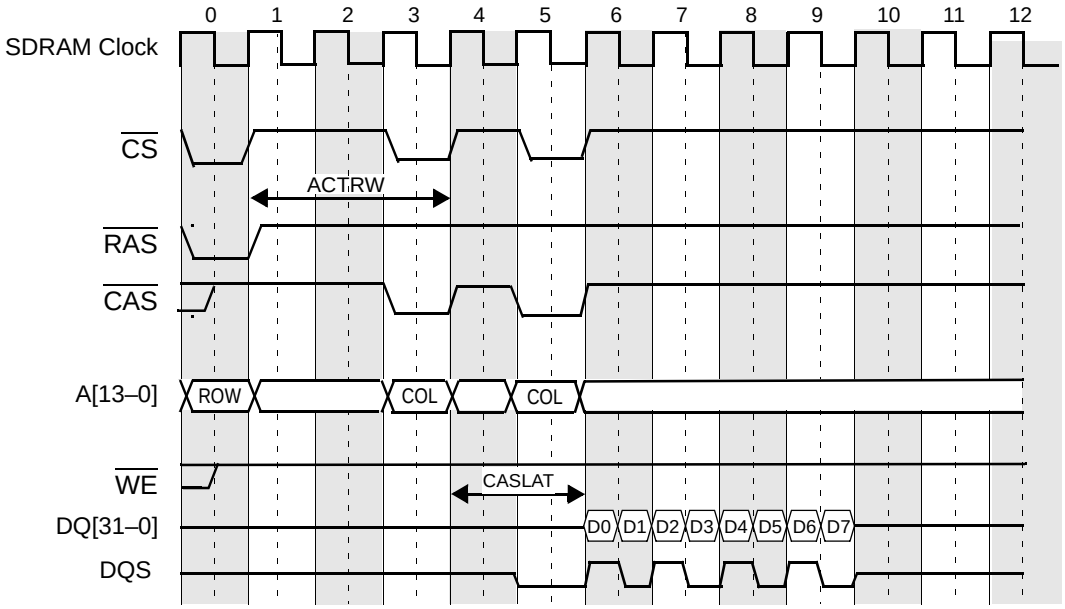


Figure 9-11. DDR SDRAM Burst Read Timing—ACTRW = 3, $\overline{\text{CAS}}$ Latency = 2

Figure 9-12 shows a single-beat write operation. The memory controller issues an ACTIVATE command, indicated by the assertion of $\overline{\text{RAS}}$, which opens the page in the appropriate logical bank of the DDR SDRAM device, if the page is not already open. Next is a single WRITE command, indicated by the assertion of $\overline{\text{CAS}}$, since the memory controller is requesting one write access. The preamble is not shown in **Figure 9-12**. The first data is then immediately written to the DDR SDRAM device (the CASLAT parameter applies only to reads).

Because this example demonstrates a single write request, the waveform in **Figure 9-12** shows one 4-beat burst. Since these are single access requests instead of bursts, only the first beat of the 4-beat burst (D0) contains the desired data. The remaining beats are ignored. Single-beat write operations are identical to burst writes, except that the remaining beats are not ignored in a burst. The DQM signals indicate which bytes for each contain valid data. This example shows the write of a 4-byte value because the DQM[3–0] signals are all asserted on the first beat, D0. On all other beats, the DQM signals indicate that the data is not valid.

After the write, the DDR SDRAM device issues a PRECHARGE command to close the page. In Open Page mode, this may occur after many accesses and clocks, as programmed in SICFG[REFINT]). **Figure 9-12** shows the fastest time that a page can be closed in Auto-Precharge mode.

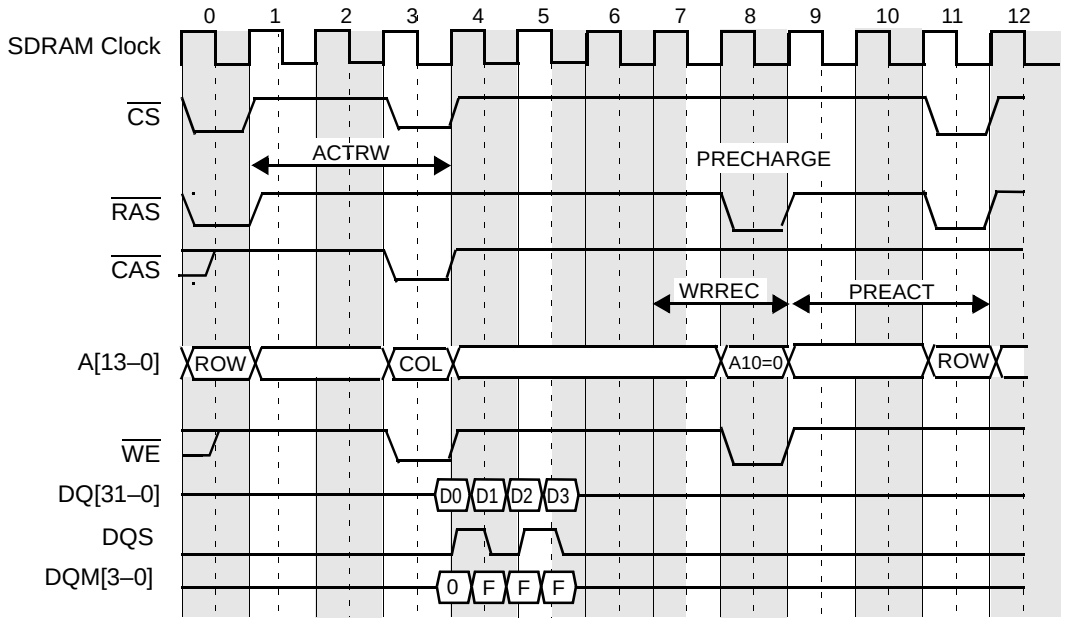


Figure 9-12. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTRW = 3

In **Figure 9-13**, four 4-beat bursts are issued. All bytes in all beats contain valid data, as indicated by the DQM[3–0] signals, which are all asserted on every beat.

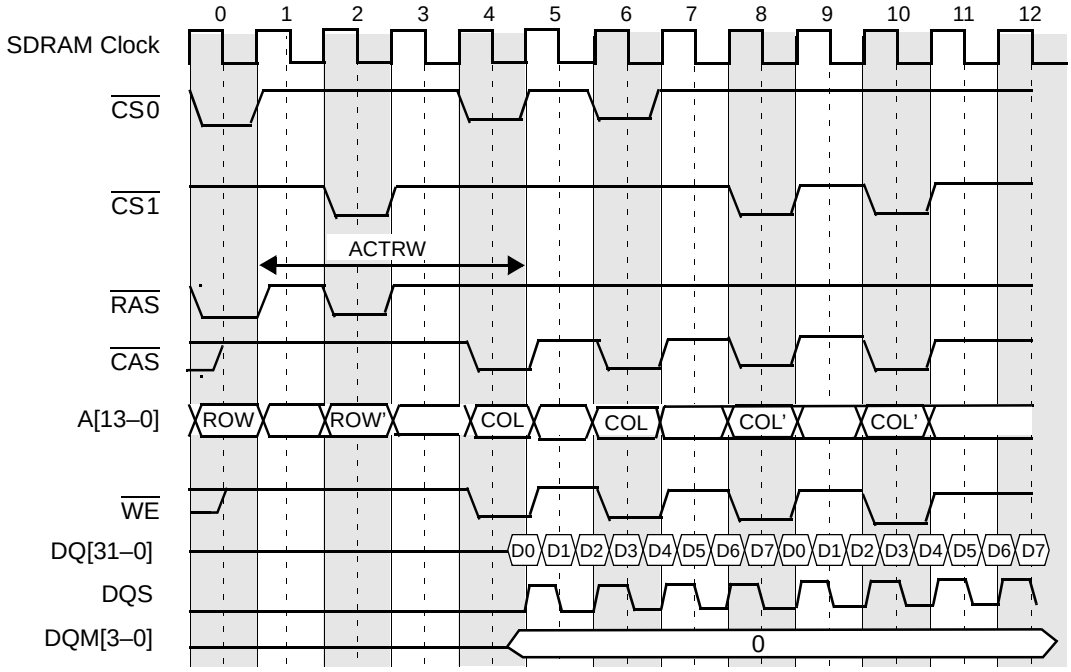


Figure 9-13. DDR SDRAM 8-Beat Burst Write Timing—ACTRW = 4

9.6.2.1 Adjustments to Read Timing

Figure 9-14 shows a read timing adjusted with TCFG2[CPO]. This is the same as the timing diagram in **Figure 9-11** but with additional detail to show how the CPO parameter can be tweaked for more reliable read operations. The memory controller issues an ACTIVATE command followed by two READ commands. The SDRAM device recognizes the $\overline{\text{CAS}}$ and drives the DQS and DQ pins after the specified latency (CASLAT). The DQS pins are driven from their previous tri-state with a preamble of one DDR clock cycle, and the first read data is driven onto the pins of the SDRAM device. There can be some delay before these signals arrive at the pins of the MSC711x device, as shown for the "DQ at Chip" and "DQS at Chip" waveforms. The DQS signals are further delayed by 1/4 DDR clock before they arrive at the memory controller to ensure that the data at the MSC711x pins is sampled at its center. The CPO parameter indicates to the memory controller when it should begin using the DQS signals for sampling the DQ data. The CPO must be adjusted so that the DQS is enabled in the middle of the one cycle preamble on the DQS pins, as shown in **Figure 9-14**.

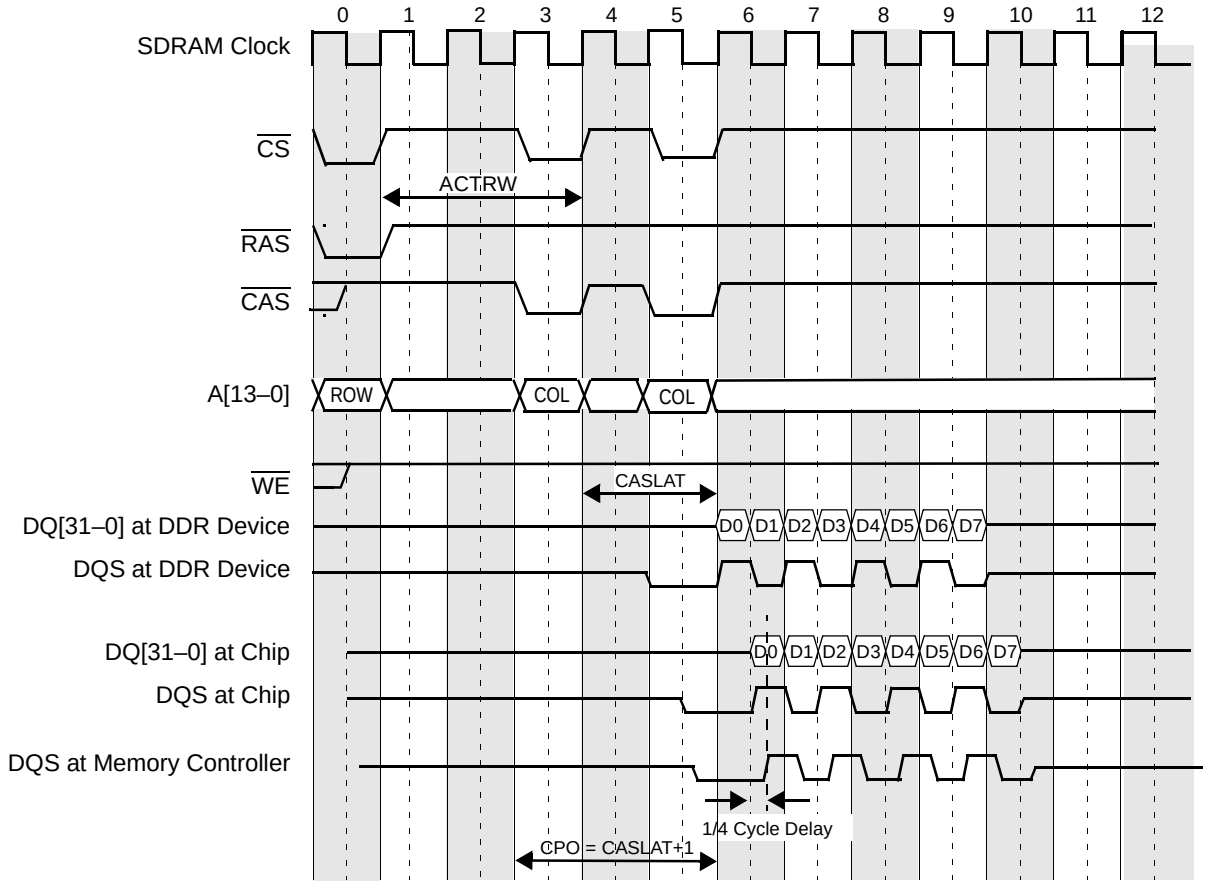


Figure 9-14. Read Timing Adjustments—ACTRW = 3, $\overline{\text{CAS}}$ Latency = 2, CPO = 0

9.6.2.2 DDR SDRAM Mode-Set Command Timing

The DDR memory controller transfers the extended mode register data SMCFG[ESDMOD] and the base mode register data SMCFG[SDMOD] to the SDRAM array by issuing two mode-set

commands separated by two SDRAM clock periods. **Figure 9-15** shows the timing of the mode-set command. The first transfer corresponds to the ESDMOD code, and the second one corresponds to the SDMOD code. Commands that follow must wait two SDRAM cycles.

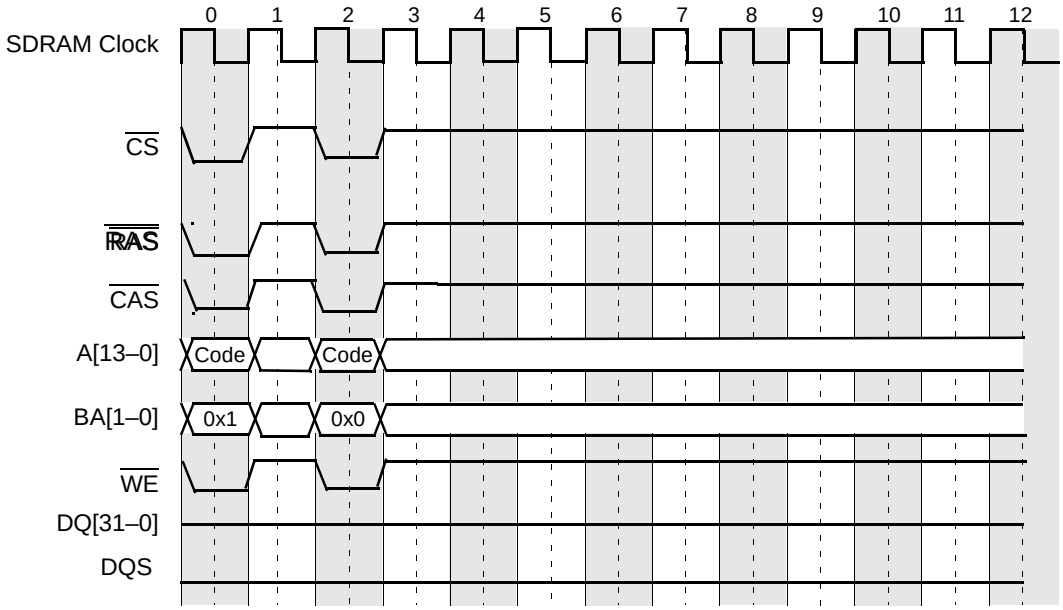


Figure 9-15. DDR SDRAM Mode-Set Command Timing

9.6.2.3 Configurable Timing Parameters

Table 9-7 shows the pins on which timing is configurable and can be adjusted in increments of the DDR fast clock for complete flexibility in tuning a system for robustness. The DDR fast clock is the clock labelled “DDR clock” in **Figure 11-2**, *Clock Generation,* on page 11-6. It runs at the frequency of the core clock.

Table 9-7. Adjusting Edges on DDR Pins

Pins	Adjustable by:	Adjusted in:	More Information
CK, $\overline{CK}$	0, 1, 2, or 3 DDR fast clocks	SCLKCTL[SHFT]	—
$\overline{CS}$, $\overline{RAS}$, $\overline{CAS}$, A[13–0], $\overline{WE}$	0 or 1 DDR fast clocks	TCFG2[ACSM]	—
DQ, DQS (outputs)	0, 1/2, 1, 3/2, or 2 DDR fast clocks	TCFG2[WRDD]	**Section 9.6.2.5**, *DDR SDRAM Write Timing Adjustments,* on page 9-23
DQS (inputs)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or 10 DDR fast clocks	TCFG2[CPO]	**Section 9.6.2.1**, *Adjustments to Read Timing,* on page 9-21

9.6.2.4 DDR SDRAM Registered DIMM Mode

To reduce loading, registered DIMMs latch the DDR SDRAM control signals internally before they are used to access the array. Enabling the registered DIMM mode (SCFG[RDEN] = 1) compensates for this delay on the DIMM control bus by delaying the data and data mask writes

on SDRAM buses by one additional SDRAM clock cycle. Enabling registered DIMM mode has no effect on bus timing for DDR reads. **Figure 9-16** shows the registered DDR SDRAM DIMM single-beat write timing.

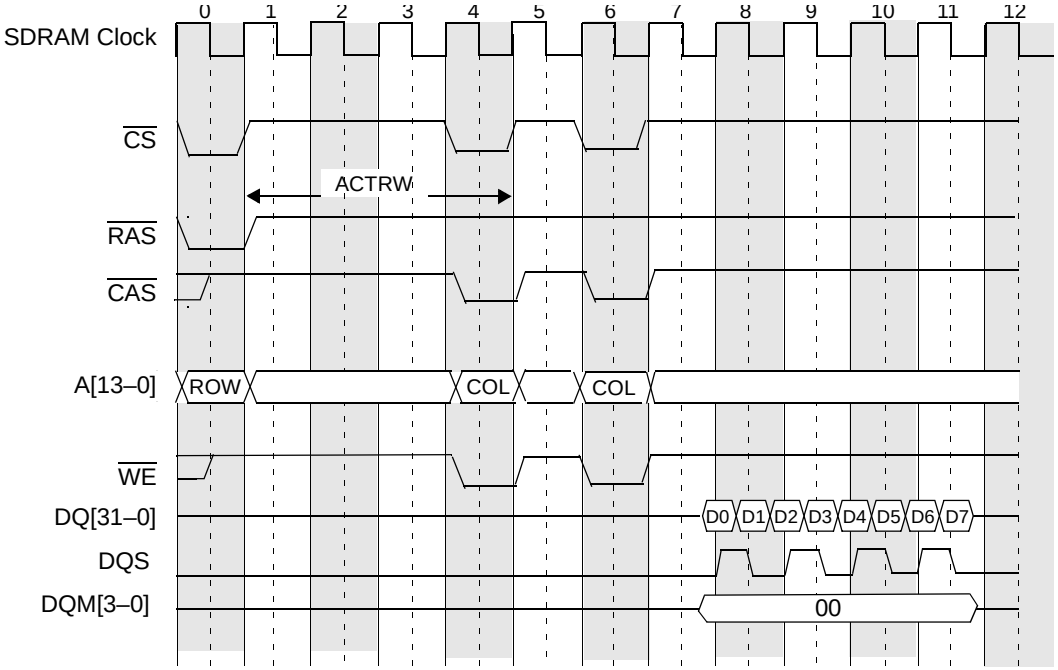


Figure 9-16. Registered DDR SDRAM DIMM Burst Write Timing

9.6.2.5 DDR SDRAM Write Timing Adjustments

The DDR memory controller provides a write timing adjustment parameter, write data delay (TCFG2[WRRD]), for data and DQS. The DDR SDRAM specification requires DQS to be received no sooner than 75 percent of an SDRAM clock period, and no later than 125 percent of a clock period, from the capturing clock edge of the command/address at the SDRAM. The write data delay (WRRD) parameter can be used to meet this timing requirement for a variety of system configurations, ranging from a system with one DIMM to a fully populated system with DIMMs. TCFG2[WRRD] specifies how much to delay the launching of DQS and data from the first clock edge one SDRAM clock cycle after the command is launched. The delay increment step sizes are in quarter SDRAM clock periods, starting with the default value of a one fourth period delay. **Figure 9-17** shows the use of the TCFG2[WRRD] write data delay parameter.

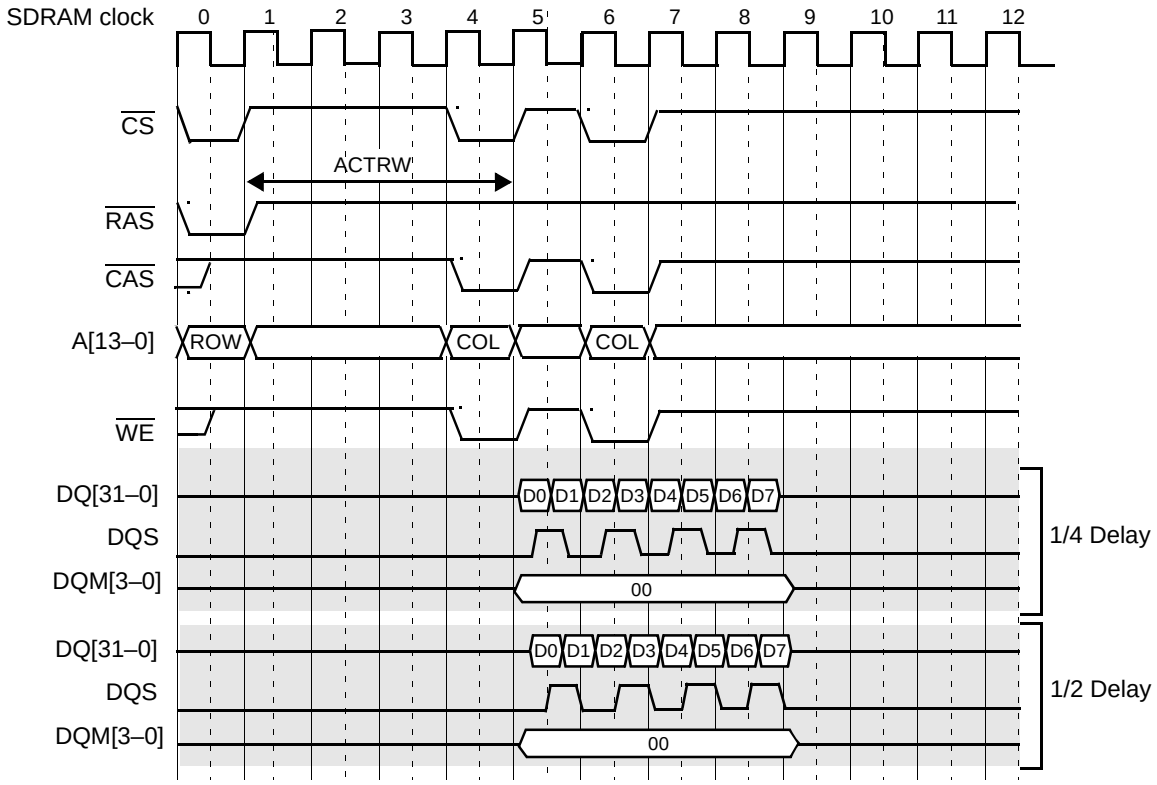


Figure 9-17. Write Timing Adjustments Example

9.6.2.6 DDR SDRAM Refresh

The DDR memory controller supports auto-refresh and self-refresh. Auto refresh is used during normal operation and is controlled by the value stored in SICFG[REFINT]. Self-refresh is used only when the DDR memory controller is set to enter a sleep power management state or a soft-stop state. The value in SICFG[REFINT] represents the number of memory bus clock cycles between refresh cycles. It is the value allowed for outstanding transactions to complete before a refresh request is sent to the memory after the SICFG[REFINT] value is reached. If a memory transaction is in progress, the refresh cycle waits for the transaction to complete. In the worst case, the refresh cycle must wait the number of bus clock cycles required by the longest programmed access. To ensure that the latency caused by a memory transaction does not violate the device refresh period, it is recommended that the programmed value of SICFG[REFINT] be less than that required by the SDRAM. When a refresh cycle is required, the DDR memory controller performs the following actions:

1. Completes all current memory requests.
2. Closes all open pages with a PRECHARGE-ALL command to each DDR SDRAM bank with an open page (as indicated by the row open table).
3. Issues an auto-refresh command to each DDR SDRAM bank (as identified by its chip select) to refresh one row in each of the logical banks in the selected physical bank.

The auto-refresh commands are staggered across the banks to reduce the system's instantaneous power requirements. Three sets of auto refresh commands should be issued on consecutive cycles when the memory is fully populated with DIMMs. The initial PRECHARGE-ALL commands are staggered in three groups for convenience. When self-refresh mode starts, only one refresh command is issued simultaneously to all physical banks. CKE is deasserted at this time, so it is not possible to stagger two more refresh commands. For this entire refresh sequence, no cycle optimization occurs for the usual case where fewer than four banks are installed. After the refresh sequence completes, any pending memory request is initiated after an inactive period specified by TCFG1 [REFREC].

9.6.2.6.1 DDR SDRAM Refresh Timing

The refresh timing for DDR SDRAM is controlled by the programmable timing parameter TCFG1 [REFREC], which specifies the number of memory bus clock cycles from the refresh command until a logical bank activate command is allowed. The DDR memory controller implements bank staggering for refreshes, as shown in **Figure 9-18** (TCFG1 [REFREC] = 10 in this example). At reset, system software must configure TCFG1[REFREC]. Configuration must be completed before any DDR SDRAM accesses are attempted.

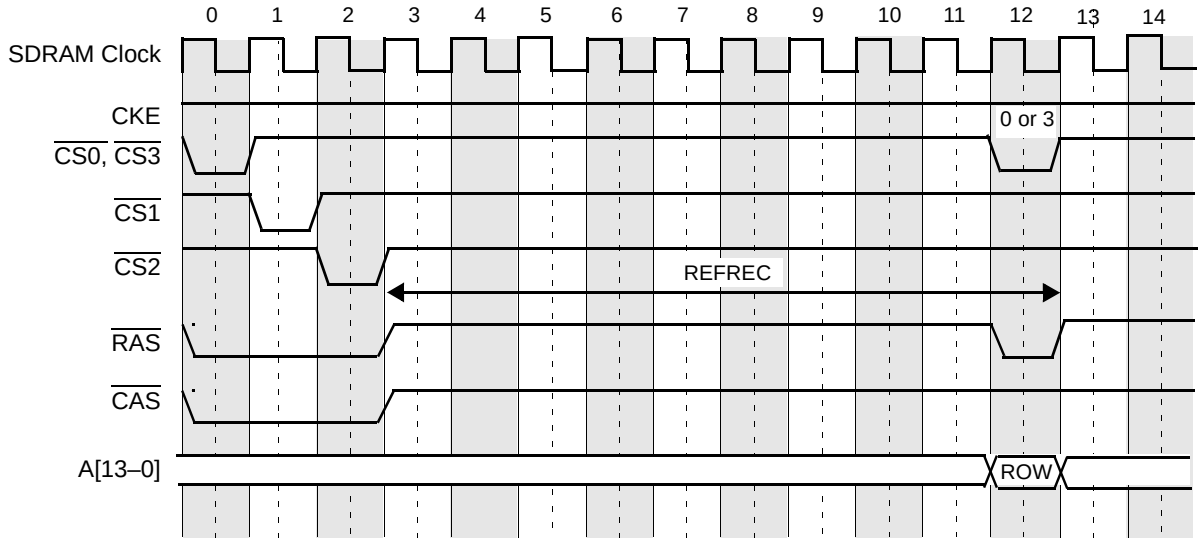


Figure 9-18. DDR SDRAM Bank Staggered Auto Refresh Timing

9.6.3 DDR SDRAM Address Multiplexing

Table 9-8 shows the address bit encodings for each DDR SDRAM configuration, where A0 refers to the least-significant bit on the address pins of the device.

Table 9-8. DDR SDRAM Address Multiplexing in 16-Pin Mode

Row × Column		MSB	ASEMI Address from MSC711x Device																												LSB
		31–30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
14 × 11	$\overline{RAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	BA																		1	0											
	$\overline{CAS}$																				11	9	8	7	6	5	4	3	2	1	0
14 × 10	$\overline{RAS}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0										
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0
13 × 11	$\overline{RAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	BA																		1	0											
	$\overline{CAS}$																				11	9	8	7	6	5	4	3	2	1	0
13 × 10	$\overline{RAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0										
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0
13 × 9	$\overline{RAS}$							12	11	10	9	8	7	6	5	4	3	2	1	0												
	BA																				1	0									
	$\overline{CAS}$																						8	7	6	5	4	3	2	1	0
12 × 10	$\overline{RAS}$							11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0										
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0
12 × 9	$\overline{RAS}$								11	10	9	8	7	6	5	4	3	2	1	0												
	BA																				1	0									
	$\overline{CAS}$																						8	7	6	5	4	3	2	1	0
12 × 8	$\overline{RAS}$									11	10	9	8	7	6	5	4	3	2	1	0											
	BA																					1	0								
	$\overline{CAS}$																							7	6	5	4	3	2	1	0

Note: A10 is used as the auto-precharge bit for reads and writes, so the column address can never use A10. Notice that for the N × 11 configurations in **Table 9-8**, the lowest 11 address pins for the column address are numbered as 11,9,8,...,0, skipping A10.

Table 9-9. DDR SDRAM Address Multiplexing in 32-Pin Mode

Row × Column		MSB	ASEMI Address from MSC711x Device																												LSB
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1–0
14 × 11	$\overline{RAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	BA																		1	0												
	$\overline{CAS}$																				11	9	8	7	6	5	4	3	2	1	0	
14 × 10	$\overline{RAS}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0											
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0	
13 × 11	$\overline{RAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	BA																		1	0												
	$\overline{CAS}$																				11	9	8	7	6	5	4	3	2	1	0	
13 × 10	$\overline{RAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0											
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0	
13 × 9	$\overline{RAS}$							12	11	10	9	8	7	6	5	4	3	2	1	0												
	BA																				1	0										
	$\overline{CAS}$																						8	7	6	5	4	3	2	1	0	
12 × 10	$\overline{RAS}$							11	10	9	8	7	6	5	4	3	2	1	0													
	BA																			1	0											
	$\overline{CAS}$																					9	8	7	6	5	4	3	2	1	0	
12 × 9	$\overline{RAS}$								11	10	9	8	7	6	5	4	3	2	1	0												
	BA																				1	0										
	$\overline{CAS}$																						8	7	6	5	4	3	2	1	0	
12 × 8	$\overline{RAS}$									11	10	9	8	7	6	5	4	3	2	1	0											
	BA																					1	0									
	$\overline{CAS}$																							7	6	5	4	3	2	1	0	

Note: A10 is the auto-precharge bit for reads and writes, so the column address can never use A10. Notice that for the N × 11 configurations in **Table 9-9**, the lowest 11 address pins used for the column address are numbered as 11,9,8,...,0, skipping A10.

9.6.4 Data Beats to DDR SDRAM Devices

Transfers between the DDR memory controller and the DDR SDRAM devices are always performed in four-beat bursts. In 16-pin mode, this corresponds to 8 bytes, and in 32-pin mode, this corresponds to 16 bytes. For transfer sizes smaller than four beats, the data transfers are still operated as four-beat bursts. The data mask pins (DQM) prevent writing of unwanted data to

SDRAM. Data masks prevent all unintended beats from writing to SDRAM. For example, if a 16-bit write transaction is desired in 16-pin mode (as shown in first row of **Table 9-10**), the write is performed on the first of 4 beats. The second, third, and fourth beats of data are not written to DRAM but instead are masked with the data mask signals (DQM).

In 16-pin mode, writes smaller than 16-bits are performed by appropriately activating a subset of the DQM[1–0] pins on the appropriate beat. In 32-pin mode, writes smaller than 32-bits are performed by appropriately activating a subset of the DQM[3–0] pins on the appropriate beat.

Table 9-10 and **Table 9-11** demonstrate the sequencing capabilities of the data beats on transfers between the DDR SDRAM and the memory controller data queues. All are non-wrapping bursts except the last and the valid start addresses of the burst are indicated by the column with a portion of the address bits. These addresses, in turn, are directly tied to accesses on the MSC711x architecture in **Table 9-12** and **Table 9-13**.

Table 9-10. Data Beat Ordering to DDR Pins in 16-Pin Mode

Requested Transfer Size	Address Bits [3–1]	Wrapping Burst?	Desired Burst(s)	Actual Sequence of Bursts to/from DRAM and Queues[1]	Comments
1 × 16-bits	000* 001* 010* 011* 100* 101* 110* 111*	N N N N N N N N	0 1 2 3 4 5 6 7	**0** - [1 - 2 - 3] **1** - [2 - 3 - 0] **2** - [3 - 0 - 1] **3** - [0 - 1 - 2] **4** - [5 - 6 - 7] **5** - [6 - 7 - 4] **6** - [7 - 4 - 5] **7** - [4 - 5 - 6]	Needed: 1 Beat. Number issued: 4 Beats
2 × 16-bits	000* 001 010* 011 100* 101 110*	N N N N N N N	0 - 1 1 - 2 2 - 3 3 - 4 4 - 5 5 - 6 6 - 7	**0** - **1** - [2 - 3] **1** - **2** - [3 - 0] **2** - **3** - [0 - 1] **3** - [0 - 1 - 2] - **4** - [5 - 6 - 7] **4** - **5** - [6 - 7] **5** - **6** - [7 - 4] **6** - **7** - [4 - 5]	Needed: 2 Beats. Number issued: 4 or 8 Beats
3 × 16-bits	000 001 010 011 100 101	N N N N N N	0 - 1 - 2 1 - 2 - 3 2 - 3 - 4 3 - 4 - 5 4 - 5 - 6 5 - 6 - 7	**0** - **1** - **2** - [3] **1** - **2** - **3** - [0] **2** - **3** - [1 - 2] - **4** - [5 - 6 - 7] **3** - [0 - 1 - 2] - **4** - **5** - [6 - 7] **4** - **5** - **6** - [7] **5** - **6** - **7** - [0]	Needed: 3 Beats. Number issued: 4 or 8 Beats
4 × 16-bits	000* 001 010 011 100*	N N N N N	0 - 1 - 2 - 3 1 - 2 - 3 - 4 2 - 3 - 4 - 5 3 - 4 - 5 - 6 4 - 5 - 6 - 7	**0** - **1** - **2** - **3** **1** - **2** - **3** - [0] - **4** - [5 - 6 - 7] **2** - **3** - [0 - 1] - **4** - **5** - [6 - 7] **3** - [0 - 1 - 2] - **4** - **5** - **6** - [7] **4** - **5** - **6** - **7**	Needed: 4 Beats. Number issued: 4 or 8 Beats
5 × 16-bits	000 001 010 011	N N N N	0 - 1 - 2 - 3 - 4 1 - 2 - 3 - 4 - 5 2 - 3 - 4 - 5 - 6 3 - 4 - 5 - 6 - 7	**0** - **1** - **2** - **3** - **4** - [5 - 6 - 7] **1** - **2** - **3** - [0] - **4** - **5** - [6 - 7] **2** - **3** - [0 - 1] - **4** - **5** - **6** - [7] **3** - [0 - 1 - 2] - **4** - **5** - **6** - **7**	Needed: 5 Beats. Number issued: 8 Beats

Table 9-10. Data Beat Ordering to DDR Pins in 16-Pin Mode (Continued)

Requested Transfer Size	Address Bits [3–1]	Wrapping Burst?	Desired Burst(s)	Actual Sequence of Bursts to/from DRAM and Queues[1]	Comments
6 × 16-bits	000 001 010	N N N	0 - 1 - 2 - 3 - 4 - 5 1 - 2 - 3 - 4 - 5 - 6 2 - 3 - 4 - 5 - 6 - 7	**0 - 1 - 2 - 3 - 4 - 5** - [6 - 7] **1 - 2 - 3** - [0] - **4 - 5 - 6** - [7] **2 - 3** - [0 - 1] - **4 - 5 - 6 - 7**	Needed: 6 Beats. Number issued: 8 Beats
7 × 16-bits	000 001	N N	0 - 1 - 2 - 3 - 4 - 5 - 6 1 - 2 - 3 - 4 - 5 - 6 - 7	**0 - 1 - 2 - 3 - 4 - 5 - 6** - [7] **1 - 2 - 3** - [0] - **4 - 5 - 6 - 7**	Needed: 7 Beats. Number issued: 8 Beats
8 × 16-bits	000* 001 010 011 100* 101 110 111	Yes Yes Yes Yes Yes Yes Yes Yes	0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 1 - 2 - 3 - 4 - 5 - 6 - 7 - 0 2 - 3 - 4 - 5 - 6 - 7 - 0 - 1 3 - 4 - 5 - 6 - 7 - 0 - 1 - 2 4 - 5 - 6 - 7 - 0 - 1 - 2 - 3 5 - 6 - 7 - 0 - 1 - 2 - 3 - 4 6 - 7 - 0 - 1 - 2 - 3 - 4 - 5 7 - 0 - 1 - 2 - 3 - 4 - 5 - 6	**0 - 1 - 2 - 3 - 4 - 5 - 6 - 7** **1 - 2 - 3 - 4 - 5 - 6 - 7 - 0** **2 - 3 - 4 - 5 - 6 - 7 - 0 - 1** **3 - 4 - 5 - 6 - 7 - 0 - 1 - 2** **4 - 5 - 6 - 7 - 0 - 1 - 2 - 3** **5 - 6 - 7 - 0 - 1 - 2 - 3 - 4** **6 - 7 - 0 - 1 - 2 - 3 - 4 - 5** **7 - 0 - 1 - 2 - 3 - 4 - 5 - 6**	Needed: 8 Beats. Number issued: 8 Beats

Notes:
1. In the Actual Sequence column, there is never a transition from 3 to 4 or from 7 to 8.
2. The wrapping burst must always be aligned on a 16-bit boundary, that is, Bit 0 = 0. The non-wrapping bursts can begin on any byte address.
3. Only accesses marked with an asterisk (*) are valid on the MSC711x architecture because of alignment requirements.

Table 9-11. Data Beat Ordering to DDR Pins in 32-Pin Mode

Requested Transfer Size	Address Bits [4–2]	Wrapping Burst?	Desired Burst(s)	Actual Sequence of Bursts to/from DRAM and Queues[1]	Comments
1 × 32-bits	000* 001* 010* 011* 100* 101* 110* 111*	N N N N N N N N	0 1 2 3 4 5 6 7	**0** - [1 - 2 - 3] **1** - [2 - 3 - 0] **2** - [3 - 0 - 1] **3** - [0 - 1 - 2] **4** - [5 - 6 - 7] **5** - [6 - 7 - 4] **6** - [7 - 4 - 5] **7** - [4 - 5 - 6]	Needed: 1 Beat. Number issued: 4 Beats
2 × 32-bits	000* 001 010* 011 100* 101 110*	N N N N N N N	0 - 1 1 - 2 2 - 3 3 - 4 4 - 5 5 - 6 6 - 7	**0 - 1** - [2 - 3] **1 - 2** - [3 - 0] **2 - 3** - [0 - 1] **3** - [0 - 1 - 2] - **4** - [5 - 6 - 7] **4 - 5** - [6 - 7] **5 - 6** - [7 - 4] **6 - 7** - [4 - 5]	Needed: 2 Beats. Number issued: 4 or 8 Beats
3 × 32-bits	000 001 010 011 100 101	N N N N N N	0 - 1 - 2 1 - 2 - 3 2 - 3 - 4 3 - 4 - 5 4 - 5 - 6 5 - 6 - 7	**0 - 1 - 2** - [3] **1 - 2 - 3** - [0] **2 - 3** - [1 - 2] - **4** - [5 - 6 - 7] **3** - [0 - 1 - 2] - **4 - 5** - [6 - 7] **4 - 5 - 6** - [7] **5 - 6 - 7** - [0]	Needed: 3 Beats. Number issued: 4 or 8 Beats

Table 9-11. Data Beat Ordering to DDR Pins in 32-Pin Mode

Requested Transfer Size	Address Bits [4–2]	Wrapping Burst?	Desired Burst(s)	Actual Sequence of Bursts to/from DRAM and Queues[1]	Comments
4 × 32-bits	000* 001 010 011 100*	N N N N N	0 - 1 - 2 - 3 1 - 2 - 3 - 4 2 - 3 - 4 - 5 3 - 4 - 5 - 6 4 - 5 - 6 - 7	**0 - 1 - 2 - 3** **1 - 2 - 3** - [0] - **4** - [5 - 6 - 7] **2 - 3** - [0 - 1] - **4 - 5** - [6 - 7] **3** - [0 - 1 - 2] - **4 - 5 - 6** - [7] **4 - 5 - 6 - 7**	Needed: 4 Beats. Number issued: 4 or 8 Beats
5 × 32-bits	000 001 010 011	N N N N	0 - 1 - 2 - 3 - 4 1 - 2 - 3 - 4 - 5 2 - 3 - 4 - 5 - 6 3 - 4 - 5 - 6 - 7	**0 - 1 - 2 - 3 - 4** - [5 - 6 - 7] **1 - 2 - 3** - [0] - **4 - 5** - [6 - 7] **2 - 3** - [0 - 1] - **4 - 5 - 6** - [7] **3** - [0 - 1 - 2] - **4 - 5 - 6 - 7**	Needed: 5 Beats. Number issued: 8 Beats
6 × 32-bits	000 001 010	N N N	0 - 1 - 2 - 3 - 4 - 5 1 - 2 - 3 - 4 - 5 - 6 2 - 3 - 4 - 5 - 6 - 7	**0 - 1 - 2 - 3 - 4 - 5** - [6 - 7] **1 - 2 - 3** - [0] - **4 - 5 - 6** - [7] **2 - 3** - [0 - 1] - **4 - 5 - 6 - 7**	Needed: 6 Beats. Number issued: 8 Beats
7 × 32-bits	000 001	N N	0 - 1 - 2 - 3 - 4 - 5 - 6 1 - 2 - 3 - 4 - 5 - 6 - 7	**0 - 1 - 2 - 3 - 4 - 5 - 6** - [7] **1 - 2 - 3** - [0] - **4 - 5 - 6 - 7**	Needed: 7 Beats. Number issued: 8 Beats
8 × 32-bits	000* 001 010 011 100* 101 110 111	Yes Yes Yes Yes Yes Yes Yes Yes	0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 1 - 2 - 3 - 4 - 5 - 6 - 7 - 0 2 - 3 - 4 - 5 - 6 - 7 - 0 - 1 3 - 4 - 5 - 6 - 7 - 0 - 1 - 2 4 - 5 - 6 - 7 - 0 - 1 - 2 - 3 5 - 6 - 7 - 0 - 1 - 2 - 3 - 4 6 - 7 - 0 - 1 - 2 - 3 - 4 - 5 7 - 0 - 1 - 2 - 3 - 4 - 5 - 6	**0 - 1 - 2 - 3 - 4 - 5 - 6 - 7** **1 - 2 - 3 - 4 - 5 - 6 - 7 - 0** **2 - 3 - 4 - 5 - 6 - 7 - 0 - 1** **3 - 4 - 5 - 6 - 7 - 0 - 1 - 2** **4 - 5 - 6 - 7 - 0 - 1 - 2 - 3** **5 - 6 - 7 - 0 - 1 - 2 - 3 - 4** **6 - 7 - 0 - 1 - 2 - 3 - 4 - 5** **7 - 0 - 1 - 2 - 3 - 4 - 5 - 6**	Needed: 8 Beats. Number issued: 8 Beats

Notes:
1. In the Actual Sequence column, there is never a transition from 3 to 4 or from 7 to 8.
2. The wrapping burst must always be aligned on a 16-bit boundary, that is, Bit 0 = 0. The non-wrapping bursts can begin on any byte address.
3. Only accesses marked with an asterisk (*) are valid on the MSC711x architecture because of alignment requirements.

9.6.5 Error Detection and Management

Memory select errors are accesses to addresses that lie outside the valid chip select regions. The status of the error is captured in the MERRD[MSE] bit, and interrupts can be enabled using the ERRINT[MSEE] bit. In addition to the status bit, the address that caused the memory select error is captured in the Memory Error Address Capture Register (MEADDC), and the attributes of the access are captured in the Memory Error Attributes Capture Register (MEAC).

The DDR memory controller can detect a memory select error, which causes the DDR memory controller to log the error and generate a machine check or critical interrupt. This error is detected if the address from the memory request does not fall into any of the enabled, programmed chip select address ranges.

For all memory select errors, the DDR memory controller does not issue any transactions onto the pins after sample points are used for sampling (the first read has returned data strobes). If the DDR memory controller is not using sample points, a dummy transaction is issued to DDR

SDRAM with the first enabled chip select. The source port on the pins is forced to 0x1F to show that the transaction is not real. **Table 9-12** shows the errors with their descriptions.

It is recommended that the memory select error be set up to generate a non-maskable interrupt to the MSC711x device if an access is performed incorrectly. An error interrupt is enabled via the ERRINT[MSEE] bit. If this error interrupt does occur, the interrupt must be serviced by clearing the MERRD[MSE] bit so that it can detect subsequent memory select errors.

Table 9-12. Memory Controller Errors

Category	Error	Description(s)	Action	Detect Register
Access error	Memory select error	Read, or write, address does not fall within the address range of any of the memory banks.	The error is reported via machine check or critical interrupt if enabled.	The error control register logs only read versus write, not full type

When the DDR controller is disabled, it flags errors on accesses to its address space, as follows:

- *Burst read or write access*. Bus time-out and bus error.
- *Single read access*. Bus time-out and bus error.
- *Single write access*:
  - No error detected on this access.
  - If the next access to the DDR is a read or write burst, the result is a bus time-out and bus error.
  - If the next access to the DDR is a single read, the result is a bus time-out and bus error.
  - If next access to the DDR is a single write, no error is detected, and any subsequent access (read or write) to the DDR address space results in a bus time-out and bus error.

9.7 Initialization and Set-Up

System software must configure the DDR memory controller using a memory polling algorithm at system start-up to map the size of each bank in memory correctly. Then the DDR memory controller uses its bank map to assert the appropriate $\overline{\text{CS}}$[0–1] signal for memory accesses according to the provided bank depths. System software must also configure the DDR memory controller to multiplex appropriately the row and column address bits for each bank. At system reset, initialization software must set up the programmable parameters in the memory interface configuration registers described in **Section 9.8.2**, *Configuration Registers*, on page 9-35.

After all parameters are configured, system software must set the memory controller SCFG[MEMEN] bit to enable the memory interface. Setting this bit asserts the CKE signal. The DDR memory controller then automatically performs the initialization sequence to prepare the JEDEC-compliant DDR SDRAM array for accesses.

9.8 DDR Memory Controller Programming Model

The DDR memory controller registers are accessed via the IPBus interface. These registers configure the memory controller. The value of the base address for this register file, DDR_BASE, is found in **Table 5-3**, *Summary — MSC711x Memory Map*, on page 5-30. There are three sets of registers in the DDR memory controller programming model:

- Chip-select registers
- Configuration registers
- Error handling registers

The DDR address space is located between 0x2000 0000 and 0xFFFF FFFF, as shown in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. Two chip selects are available within this range.

9.8.1 Chip Select Registers

The memory controller chip-select registers are as follows:

- Chip-Select Memory Bounds Registers (CSBRx), **page 9-32**.
- Chip-Select Configuration (CSxCFG), **page 9-34**.

CSBR*x* Chip Select Memory Bounds Register, 16-Pin Operation

CSBR0 DDR_BASE + 0x000

CSBR1 DDR_BASE + 0x008

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—						SA*x*									
TYPE	R						R/W									
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—						EA*x*									
TYPE	R						R/W									
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CSBR*x* defines the starting and ending addresses of the memory space that corresponds to the individual chip selects. This register is configured differently, depending on whether the memory controller is configured for 16- or 32-pin operation.

Table 9-13. CSBRx Bit Descriptions, 16-Pin Operation

Bits	Reset	Description
— 31–26	0	Reserved. Write to zero for future compatibility.
SA*x* 25–16	0	**Starting Address** Specifies the starting address for chip select (bank) *n.* This value is compared against the most significant 10 bits of the 32-bit address.
— 15–10	0	Reserved. Write to zero for future compatibility.
EA*x* 9–0	0	**Ending Address** Specifies the ending address for chip select (bank) *n.* This value is compared against the most significant 10 bits of the 32-bit address.

CSBR*x* Chip Select Memory Bounds Register, 32-Pin Operation

CSBR0 DDR_BASE + 0x000

CSBR1 DDR_BASE + 0x008

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—							SA*x*								
TYPE	R							R/W								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—							EA*x*								
TYPE	R							R/W								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-14. CSBRx Bit Descriptions, 32-Pin Operation

Bits	Reset	Description
— 31–25	0	Reserved. Write to zero for future compatibility.
SA*x* 24–16	0	**Starting Address** Specifies the starting address for chip select (bank) *x.* This value is compared against the most significant 9 bits of the 32-bit address.
— 15–9	0	Reserved. Write to zero for future compatibility.
EA*x* 8–0	0	**Ending Address** Specifies the ending address for chip select (bank) *x.* This value is compared against the most significant 9 bits of the 32-bit address.

CS*x*CFG Chip Select Configuration Registers

CS0CFG DDR_BASE + 0x080

CS1CFG DDR_BASE + 0x084

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CSxEN	—							APxEN	—						
TYPE	R/W	R							R/W	R						
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—					RBCSx			—					CBCSx		
TYPE	R					R/W			R					R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CS*x*CFG enable the DDR chip selects and set the number of row and column bits used for each chip select. CS*x*CFG should be loaded with the correct number of row and column bits for each physical bank of SDRAM devices. Because address multiplexing is established by the bits in these registers, you must take great care to set these values correctly.

Table 9-15. CSxCFG Field Descriptions

Bits	Reset	Description	Settings
CSxEN 31	0	**Chip Select x Enable** Enables/disables chip select.	0 Chip select *x* is not active. 1 Chip select *x* is active and assumes the state set in CSBRx.
— 30–24	0	Reserved. Write to zero for future compatibility.	
APxEN 23	0	**Chip Select *x* Auto-Precharge Enable** Specifies when auto-precharge is enabled.	0 Chip select *x* is auto-precharged only if global auto-precharge mode is enabled (REFINT = 0). 1 Chip select *x* always issues an auto-precharge for read and write transactions.
— 22–11	0	Reserved. Write to zero for future compatibility.	
RBCSx 10–8	0	**Number of Row Bits for SDRAM** Specifies the number of SDRAM row bits on chip select x.	000 12 row bits. 001 13 row bits. 010 14 row bits. 011–111 Reserved.

Table 9-15. CSxCFG Field Descriptions (Continued)

Bits	Reset	Description	Settings
— 7–3	0	Reserved. Write to zero for future compatibility.	
CBCSx 2–0	0	Number of column bits for SDRAM on chip select x.	000 8 column bits. 001 9 column bits. 010 10 column bits. 011 11 column bits. 100–111 Reserved.

9.8.2 Configuration Registers

The DDR memory controller configuration registers are as follows:

- DDR SDRAM Timing Configuration Register 1 (TCFG1), **page 9-35**.
- DDR SDRAM Timing Configuration Register 2 (TCFG2), **page 9-38**.
- DDR SDRAM Control Configuration Register (SCFG), **page 9-39**.
- DDR SDRAM Mode Configuration Register (SMCFG), **page 9-41**.
- DDR SDRAM Interval Configuration Register (SICFG), **page 9-42**.
- DDR SDRAM Clock Configuration Register (SCLKCFG), **page 9-43**.

Keep in mind that the chip select registers must be initialized. See **Section 9.8.1**, *Chip Select Registers*, on page 9-32.

TCFG1 DDR SDRAM Timing Configuration Register 1 DDR_BASE + 0x108

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	PREACT			ACTPRE				—	ACTRW			—	CASLAT		
TYPE	R	R/W							R	R/W			R	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	REFREC				—	WRREC			—	ACTACT			—	WRRD		
TYPE	R/W				R	R/W			R	R/W			R	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TCFG1 sets the number of clock cycles between various SDRAM control commands.

Table 9-16. TCFG1 Bit Descriptions

Bits	Reset	Description	Settings
— 31	0	Reserved. Write to zero for future compatibility.	
PREACT 30–28	0	**Precharge to Activate Interval (t_{rp})** Determines the number of clock cycles from a precharge command until an activate or refresh command is allowed.	000 Reserved. 001 1 clock. 010 2 clocks. 011 3 clocks. 100 4 clocks. 101 5 clocks. 110 6 clocks. 111 Reserved.
ACTPRE 27–24	0	**Activate to precharge interval (t_{ras})** Determines the number of clock cycles from an activate command until a precharge command is allowed.	0000 Reserved 0001 1 clocks 0010 2 clocks 0011 3 clocks . . . 1110 14 clocks 1111 15 clocks
— 23	0	Reserved. Write to zero for future compatibility.	
ACTRW 22–20	0	**Activate to Read/Write Interval for SDRAM (t_{rcd})** Controls the number of clock cycles from an activate command until a read or write command is allowed.	000 Reserved. 001 1 clock. 010 2 clocks. 011 3 clocks. 100 4 clocks. 101 5 clocks. 110 6 clocks. 111 7 clocks.
— 19	0	Reserved. Write to zero for future compatibility.	
CASLAT 18–16	0	**$\overline{\text{CAS}}$ Latency From Read Command** Specifies read latency. The read latency is the delay, in clock cycles, between the time the SDRAM registers a READ command and time the first output data is available. If a READ command is registered at clock edge n, and the latency is m clocks, the data is available nominally coincident with clock edge $n + m$. This value must also be programmed into the DDR SDRAM at initialization (via the DDR SDRAM mode register described on **page 9-41**.	000 Reserved. 001 1 clock. 010 1.5 clocks. 011 2 clocks. 100 2.5 clocks. 101 3 clocks. 110 3.5 clocks. 111 4 clocks.

Table 9-16. TCFG1 Bit Descriptions (Continued)

Bits	Reset	Description	Settings
REFREC 15–12	0	**Refresh Recovery Time (t_{rfc})** Controls the number of clock cycles from a refresh command until an activate command is allowed. Refresh recovery time is equal to eight plus the value entered into this field.	0000 24 clocks. 0001 9 clocks. 0010 10 clocks. 0011 11 clocks. . . . 1110 22 clocks. 1111 23 clocks.
— 11	0	Reserved. Write to zero for future compatibility.	
WRREC 10–8	0	**Last Data to Precharge Minimum Interval (t_{wr})** Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed.	000 Reserved. 001 1 clock. 010 2 clocks. 011 3 clocks. 100 4 clocks. 101 5 clocks. 110 6 clocks. 111 7 clocks.
— 7	0	Reserved. Write to zero for future compatibility.	
ACTACT 6–4	0	**Activate to Activate Interval (t_{rrd})** Specifies the number of clock cycles from an activate command until another activate command is allowed for a different logical bank within the same physical bank (chip select).	000 Reserved. 001 1 clock. 010 2 clocks. 011 3 clocks. 100 4 clocks. 101 Reserved. 110 Reserved. 111 Reserved.
— 3	0	Reserved. Write to zero for future compatibility.	
WRRD 2–0	0	**Last Write Data Pair to Read Command Interval (t_{wtr})** Determines the number of clock cycles between the last write data pair and the subsequent read command to the same physical bank.	000 Reserved. 001 1 clock. 010 2 clocks. 011 3 clocks. 100 4 clocks. 101 5 clocks. 110 6 clocks. 111 7 clocks.

TCFG2 DDR SDRAM Timing Configuration Register 2 DDR_BASE + 0x10C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—				CPO				—				ACSM	—		
TYPE	R				R/W				R				R/W	R		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—			WRDD			—									
TYPE	R			R/W			R									
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TCFG2 sets the amount of clock delay to data for writes. The CPO and ACSM fields allow system designers to tweak designs for more reliable operation.

Table 9-17. TCFG2 Register Bit Descriptions

Bits	Reset	Description	Settings
— 31–28	0	Reserved. Write to zero for future compatibility.	
CPO 27–24		**$\overline{\text{CAS}}$ to Preamble Override** Allows software to override the default value for $\overline{\text{CAS}}$ to preamble. This defines the number of DRAM cycles between the time a read is issued and the time the corresponding DQS preamble is valid for the memory controller. This parameter affects only read accesses.	0000 Default. $\overline{\text{CAS}}$ to preamble is defined as ⌈CASLAT⌉ + 1. 0001 ⌈CASLAT⌉. 0010 ⌈CASLAT⌉ + 1/2. 0011 ICASLAT⌉ + 1. 0100 ⌈CASLAT⌉ + 3/2. 0101 ⌈CASLAT⌉ + 2. 0110 ⌈CASLAT⌉ + 5/2. 0111 ⌈CASLAT⌉ + 3. 1000 ⌈CASLAT⌉ + 7/2. 1001 ⌈CASLAT⌉ + 4. 1010 ⌈CASLAT⌉ + 9/2. 1011 ⌈CASLAT⌉ + 5. 1100 –1111: Reserved.
— 23–20	0	Reserved. Write to zero for future compatibility.	
ACSM 19	0	**Address and Control Shift Mode** Enables/disables address and control shift mode.	0 The DRAM address and control buses are output in the default mode. 1 The DRAM address and control buses are shifted by half a DRAM cycle before they are driven onto the pins.
— 18–13	0	Reserved. Write to zero for future compatibility.	

Table 9-17. TCFG2 Register Bit Descriptions (Continued)

Bits	Reset	Description	Settings
WRDD 12–10	0	**Write Command to Write Data Strobe Timing Adjustment** Controls the amount of delay applied to the data and data strobes for writes. If a value of 0 clock delay is chosen, the memory controller automatically adds an extra turn-around cycle between reads and writes to avoid violating the write preamble requirement.	000 0 clock delay . 001 2/8 clock delay (recommended). 010 4/8 clock delay. 011 6/8 clock delay. 100 1 clock delay. 101 Reserved. . . . 111 Reserved.
— 9–0	0	Reserved. Write to zero for future compatibility.	

SCFG DDR SDRAM Control Configuration Register DDR_BASE + 0x110

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	MEMEN	SREN	—	RDEN	—	—	STYPE		—	—	DPWR	—	—	—	NCAP	—
TYPE	R/W		R	R/W	R		R/W		R		R/W	R			R/W	R
RESET	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	2TEN	—														
TYPE	R/W	R														
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SCFG enables the interface logic and specifies operating features such as self refreshing, error checking and correcting, registered DIMMS, and dynamic power management.

Table 9-18. SCFG Bit Descriptions

Bit	Reset	Description	Setting
MEMEN 31	0	**DDR SDRAM Interface Logic Enable** Enables/disables SDRAM interface logic. This bit must not be set until all other memory configuration parameters are appropriately configured by initialization code.	0 SDRAM interface logic is disabled. 1 SDRAM interface logic is enabled.
SREN 30	0	**Self Refresh Enable** Enables/disabled SDRAM self refresh. When self refresh is disabled, the system must preserve the integrity of SDRAM.	0 SDRAM self refresh is disabled during sleep or soft-stop mode. 1 SDRAM self refresh is enabled during sleep or soft-stop

Table 9-18. SCFG Bit Descriptions (Continued)

Bit	Reset	Description	Setting
— 29	0	Reserved. Write to zero for future compatibility.	
RDEN 28	0	**Registered DIMM Enable** Specifies the type of DIMM used in the system (see **Section 9.6.2.4**, *DDR SDRAM Registered DIMM Mode*).	0 Unbuffered DIMMs. 1 Registered DIMMs.
— 27–26		Reserved. Write to zero for future compatibility.	
STYPE 25–24	10	**Type of SDRAM Device** Specifies the type of SDRAM.	00 Reserved. 01 Reserved. 10 DDR SDRAM. 11 Reserved.
— 23–22	0	Reserved. Write to zero for future compatibility.	
DYNPWR 21	0	**Dynamic Power Management Mode** Enables/disables dynamic power management mode. The dynamic power-saving mode uses the CKE DDR SDRAM pin to power down dynamically when there is no system memory activity. When DYNPWR is enabled, if there is no ongoing memory activity, the SDRAM CKE signal is deasserted. See **Section 9.5.4**, *Low-Power Modes,* on page 9-15.	0 Dynamic power management mode is disabled. 1 Dynamic power management mode is enabled.
— 20–18	0	Reserved. Write to zero for future compatibility.	
NCAP 17	0	**Non-Concurrent Auto-Precharge** Specifies whether the SDRAM devices support concurrent auto-precharge. See **Section 9.5.2**, *Auto-Precharge Mode,* on page 9-14.	0 DDR SDRAMs support concurrent auto-precharge. 1 DDR SDRAMs do not support concurrent auto-precharge.
— 16	0	Reserved. Write to zero for future compatibility.	
2TEN 15	0	**2T Timing Enable** Specifies whether 1T or 2T timing is used on address and control signals. When 2TEN is set, chip selects still assert for only one cycle.	0 1T timing is used on address and control signals. 1 2T timing is used on address and control signals.
— 14–0	0	Reserved. Write to zero for future compatibility.	

SMCFG DDR SDRAM Mode Configuration Register DDR_BASE + 0x118

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ESDMOD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SMMOD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SMCFG sets the values loaded into the DDR mode registers.

Table 9-19. SMCFG Bit Descriptions

Bits	Reset	Description
ESDMOD 31–16	0	**Extended SDRAM Mode** Specifies the initial value loaded into the DDR SDRAM Extended Mode Register. The range of legal values and the meaning of each is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus during the DDR SDRAM initialization sequence, A0 presents the least significant bit of ESDMOD, which, in the big-endian convention shown in Figure 25 corresponds to bit 31 of the ESDMOD field. The most significant bit of the SDRAM extended mode register value must be stored at bit 16 of the ESDMOD field.
SDMOD 15–0	0	**SDRAM Mode** Specifies the initial value loaded into the DDR SDRAM mode register. The range of legal values and the meaning of each is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus during the DDR SDRAM initialization sequence, A0 presents the least significant bit of SDMOD, which, in the big-endian convention corresponds to bit 15 in the SDMOD field. The most significant bit of the SDRAM extended mode register value must be stored at bit 16 of the ESDMOD field. Because the memory controller forces SDMOD[13–8] to certain values depending upon the state of the initialization sequence (for resetting the SDRAM DLL), the memory controller ignores the corresponding bits of this field.

SICFG DDR SDRAM Interval Configuration Register DDR_BASE + 0x124

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	REFINT													
TYPE	R		R/W													
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PI															
TYPE	R		R/W													
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SICFG sets the number of DRAM clock cycles between bank refreshes issued to the DDR SDRAM devices. In addition, it specifies the number of DRAM cycles to maintain a page after it is accessed.

Table 9-20. SICFG Bit Descriptions

Bits	Reset	Description
— 31–30	0	Reserved. Write to zero for future compatibility.
REFINT 29–16	0	**Refresh Interval** Represents the number of memory bus clock cycles between refresh cycles. One row is refreshed in each DDR SDRAM physical bank during each refresh cycle. The value for REFINT depends on the specific SDRAMs used and the interface clock frequency.
— 15–14	0	Reserved. Write to zero for future compatibility.
PI 13–0	0	**Precharge Interval** Sets the duration (in memory bus clocks) that a page is retained as an open page after a DDR SDRAM access. If REFINT is cleared, the DDR memory controller uses auto-precharge read and write commands rather than Open Page mode. This is referred to as global auto-precharge mode. See **Section 9.5**, *Operating Modes,* on page 9-13.

SCLKCFG DDR SDRAM Clock Configuration Register DDR_BASE + 0x130

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	SSEN	—				SHFT		ADJUST	—							
TYPE	R/W	R				R/W		R/W	R							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SCLKCFG provides a source synchronous option and a 1/2 cycle clock adjustment.

Table 9-21. SCLKCFG Bit Descriptions

Bits	Reset	Description	Settings
SSEN 31	0	**Source Synchronous Enable** Specifies whether the address and command are sent to the DDR SDRAMs source synchronously.	0 Reserved. 1 Command sent synchronously.
— 30–27	0	Reserved. Write to zero for future compatibility.	
SHFT 26–25	0	**Clock Shift** Specifies when the clock is started in relationship to the address/command.	00 Clock start is aligned with the address/command. 01 Clock starts one DDR fast clock after the address/command. 10 Clock starts two DDR fast clocks after the address/command. 11 Clock starts three DDR fast clocks after the address/command.
ADJUST 24	0	**Clock Adjust** Allows you to adjust the clock shift to make it an extra DDR fast clock later.	0 Clock does not start an extra DDR fast clock later. 1 Clock starts an extra DDR fast clock later.
— 23–0	0	Reserved. Write to zero for future compatibility.	

9.8.3 Error Handling Registers

The DDR memory controller error handling registers are as follows:

- Memory Error Detect Register (MERRD), **page 9-44**.
- Memory Error Interrupt Enable Register (ERRINT), **page 9-44**.
- Memory Error Attributes Capture Register (MEAC), **page 9-45**.
- Memory Error Address Capture Register (MEADDC), **page 9-46**.

- Memory Error Extended Address Capture Register (MEEAC), **page 9-47**.

MERRD Memory Error Detect Register DDR_BASE + 0xE40

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															MSE
TYPE	R															R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MERRD stores the detection bits for multiple memory errors and memory select errors. A bit is cleared by writing a high value to it. System software can determine the type of memory error by examining the contents of this register. If an error is disabled with ERR_DISABLE, the corresponding error is never detected and captured in MERRD.

Table 9-22. MERRD Field Descriptions

Bits	Name	Description	Settings
— 31–1	0	Reserved. Write to zero for future compatibility.	
MSE 0	0	**Memory Select Error** Indicates whether a memory select error has been detected. This bit is cleared when software writes a value of 1 to it.	0 No memory select error. 1 Memory select error.

ERRINT Memory Error Interrupt Enable Register DDR_BASE + 0xE48

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															MSEE
TYPE	R															R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ERRINT enables memory select error interrupts.

Table 9-23. ERRINT Bit Descriptions

Bits	Reset	Description	Settings
— 31–1	0	Reserved. Write to zero for future compatibility.	
MSEE 0	0	**Memory Select Error Interrupt Enable** Enables/disables interrupts when memory select errors occur.	0 Memory select errors do not cause interrupts. 1 Memory select errors generate interrupts.

MEAC Memory Error Attributes Capture Register DDR_BASE + 0xE4C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—					TSIZ			—							
TYPE	R					R/W			R							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—		TTYP													VLD
TYPE	R		R/W													R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MEAC captures the attributes of the transaction that caused the error.

Table 9-24. MEAC Bit Descriptions

Bits	Reset	Description	Settings
— 31–27	0	Reserved. Write to zero for future compatibility.	
TSIZ 26–24	0	**Transaction Size for the Error** In 16-pin mode, TSIZ captures the transaction size in N × 16-bit transfers. (See the leftmost column of **Table 9-10**, *Data Beat Ordering to DDR Pins in 16-Pin Mode,* on page 9-28). In 32-pin mode, TSIZ captures the transaction size in N × 32-bit transfers. (See the leftmost column of **Table 9-11**, *Data Beat Ordering to DDR Pins in 32-Pin Mode,* on page 9-29).	
— 23–14	0	Reserved. Write to zero for future compatibility.	
TTYP 13–12	0	**Transaction Type for the Error** Specifies the error transaction type; that is, whether it is a read or a write.	00 Reserved. 01 Write. 10 Read. 11 Reserved.

Table 9-24. MEAC Bit Descriptions (Continued)

Bits	Reset	Description	Settings
— 11–1	0	Reserved. Write to zero for future compatibility.	
VLD 0	0	**Valid** Set when valid information is captured in the error capture registers.	

MEADDC Memory Error Address Capture Register, 16-Pin Operation DDR_BASE + 0xE50

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CADDR															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CADDR													T		L
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MEADDC holds the least significant bits of the address when an error is detected. It captures the address differently, depending on whether the memory controller is configured for 16- or 32-pin operation.

Table 9-25. MEADDC Bit Descriptions, 16-Pin Operation

Bits	Reset	Description
CADDR 31–3	0	**Capture Address** Captures bits [29–1] of the transaction address when an error is detected.
T 2–1	0	**Capture Access Size** Captures 11 if the access size is less than 16-bits. Otherwise, it captures 00.
L 0	0	**LSB of Captured Address** Captures bit 0 of the transaction address when an error is detected.

MEADDC Memory Error Address Capture Register, 32-Pin Operation DDR_BASE + 0xE50

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CADDR															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CADDR													T	L	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-26. MEADDC Bit Descriptions, 32-Pin Operation

Bits	Reset	Description
CADDR 31–3	0	**Capture Address** Captures bits [30–2] of the transaction address when an error is detected.
T 2	0	**Capture Access Size** Captures 1 if the access size is less than 32-bits. Otherwise, it captures 0.
L 1–0	0	**LSBs of Captured Address** Captures bits [1–0] of the transaction address when an error is detected.

MEEAC Memory Error Extended Address Capture Register, 16-Pin Operation
DDR_BASE + 0xE54

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—														CEADDR	
TYPE	R														R/W	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MEEAC holds the most significant transaction bits of the address when an error is detected. It captures the address differently, depending on whether the memory controller is configured for 16- or 32-pin operation.

Table 9-27. MEEAC Bit Descriptions, 16-Pin Operation

Bits	Reset	Description
— 31–2	0	Reserved. Write to zero for future compatibility.
CEADDR 1–0	0	**Capture Extended Address** Captures the most significant two bits of the transaction address, bits 31–30, when an error is detected.

MEEACMemory Error Extended Address Capture Register, 32-Pin Operation
DDR_BASE + 0xE54

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															CEADDR
TYPE	R															R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-28. MEEAC Bit Descriptions, 32-Pin Operation

Bits	Reset	Description
— 31–1	0	Reserved. Write to zero for future compatibility.
CEADDR 0	0	**Capture Extended Address** Captures the most significant bit of the transaction address, bit 31, when an error is detected.

Memory Controller Interface 10

The memory controller interface (MCIF) increases the efficiency of accesses through the DDR memory controller to external DDR memory. It processes accesses from the crossbar switch to the DDR memory controller as shown in **Figure 10-1**

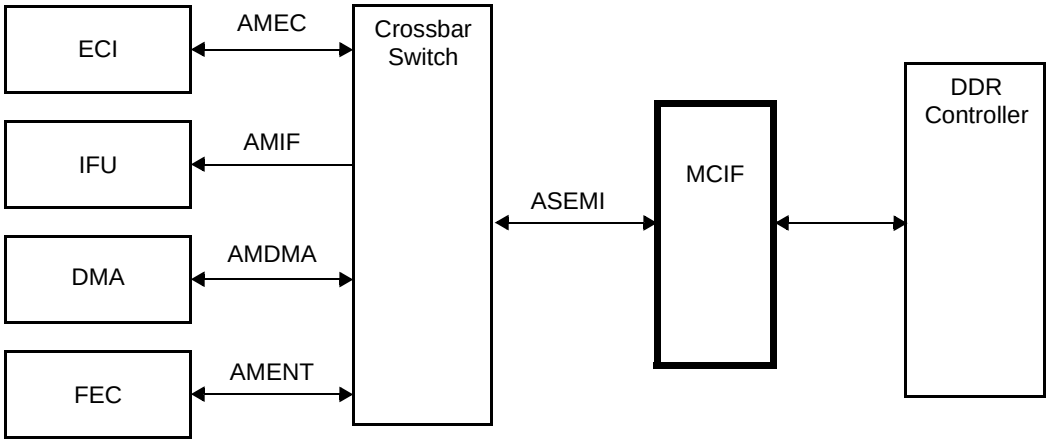


Figure 10-1. System View of Memory Controller Interface

10.1 Features

The features of the MCIF are as follows:

- 8-location write buffer to reduce latency on ASEMI write accesses.
- 4-location IFU program read buffer.
- 4-location DMA data read buffer.
- 4-location configurable-master alternate data read buffer.
- 2-location ECI data read buffer.
- Programmable predictive read capability.
- Interface is tuned for the following accesses:
  - DMA. WRAP4 (aligned, non-wrapping) of 64-bit read/write data.
  - IFU. WRAP4 (aligned, wrapping) of 128-bit program read data.
  - ECI. SINGLE of 64, 32, 16, and 8-bit read data.
  - Ethernet. WRAP4 (aligned, non-wrapping) of 32-bit read/write data.
- Performs optionally-enabled predictive read accesses on:

— IFU SINGLE and WRAP4 128-bit program read data.
— DMA WRAP4 64-bit read data.
— ECI SINGLE 64,32,16, or 8-bit data.
— Ethernet WRAP4 32-bit read data.

- Data read buffer coherency checking and resolution.
- Error detection for unsupported AHB data and program accesses.

10.2 Architecture

Figure 10-2 shows the interface between the crossbar switch and the DDR memory controller.

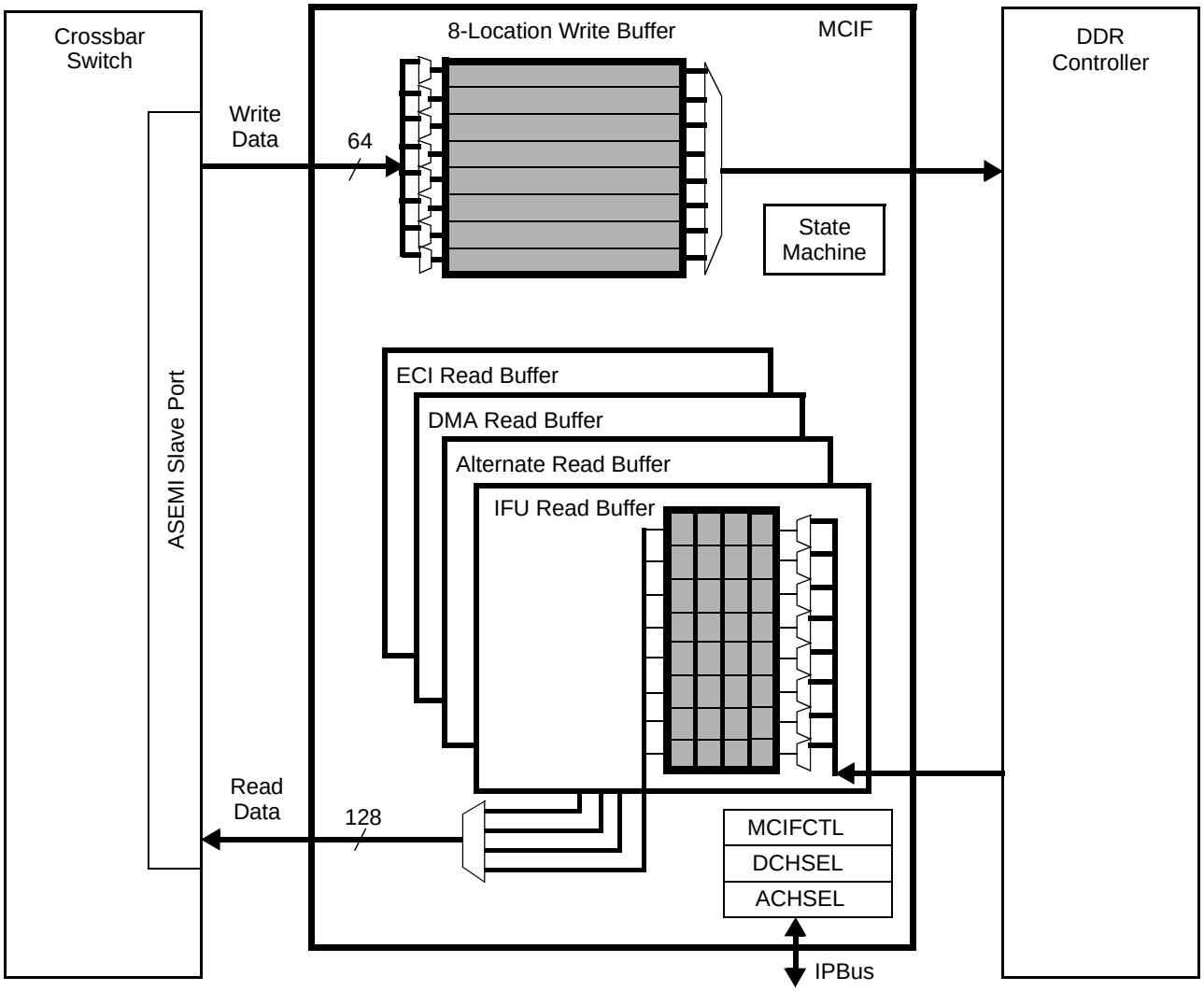


Figure 10-2. Memory Controller Interface

10.2.1 Write Buffer Characteristics

A 64-bit 8-location write buffer improves write access performance on ASEMI. This buffer is always enabled. It performs zero wait-state ASEMI write accesses when it is not full. It accepts eight 64-bit write accesses before stalling the ASEMI bus. The write buffer can accept up to eight

SINGLE accesses, two WRAP4 write accesses, or a combination of SINGLES and a WRAP4 before stalling the ASEMI bus. Each SINGLE or WRAP4 beat stored in the write buffer occupies a full buffer location and is serviced in the order it is accepted.

10.2.2 Read Prediction Characteristics

The MCIF can perform predictive reads for important program and data accesses. The following four dedicated read buffers allow read buffering and predictive read capability from different masters:

- IFU read buffer
- DMA read buffer
- ALT read buffer
- ECI read buffer

10.2.2.1 Program Predictive Reads

To optimize ASEMI accesses issued from the IFU, the MCIF can perform predictive reads and store the data in the 128-bit 4-location IFU read buffer. This read buffer accepts SINGLE and WRAP4 128-bit IFU accesses:

- Program predictive read:
  - Configurable enable that is hardware-controlled, always enabled, disabled (default).
  - Disabled when a DDR chip-select boundary can be crossed.
- Hardware-controlled predictive read enable mode:
  - Enabled on primary-set and pre-fetch accesses.
  - Disabled when a cache-line boundary (16×16 bytes) can be crossed.

10.2.2.2 Data Predictive Reads

To optimize ASEMI accesses from the ECI, DMA controller, and FEC, the MCIF can perform predictive reads and store the data in one of three data read buffers:

- ECI data read buffer:
  - 2 locations by 64 bits.
  - Configurable predictive read enable (disabled by default).
- DMA data read buffer:
  - 4 locations by 64 bits.
  - Non-configurable master selection (DMA).
  - Configurable DMA channel selection and enable (0–5 channels can be selected).
  - Configurable predictive read enable (disabled by default).
- Alternate data read buffer
  - 4 locations by 64-bits.

  - Configurable master selection (DMA or FEC).
  - Default master selection = FEC.
  - Configurable DMA channel selection and enable (optional when DMA is selected).
  - Configurable predictive read enable (disabled by default).
- Data predictive read that is disabled when a DDR chip-select boundary can be crossed.

10.2.2.3 Predictive Read Hardware Disable

A predictive read is not enabled if it could cross a DDR chip-select boundary and cause an undesirable NMI because of illegal MCIF accesses to illegal memory space. The DDR memory chip select regions are programmable via the DDR Chip Select Memory Bounds Registers (CSBRx). For 32-bit DDR configuration, the upper nine address bits are compared with the SAn and EAn fields; for 16-bit configuration the upper ten address bits are compared. A predictive read is disabled if it would cross a 2^{22} or 2^{21} address boundary for 32-bit and 16-bit DDR configurations, respectively.

10.2.3 Non-Optimized Accesses

For the following non-critical data and program accesses, optimization may not be enabled or is not supported:

- ECI SINGLE read accesses when a predictive read is disabled.
- DMA SINGLE read accesses.
- FEC SINGLE read accesses.
- IFU SINGLE read accesses when a predictive read is disabled.

10.2.4 Error Detection

Error detection for ASEMI accesses is accomplished via MCIF error detection logic, crossbar master bus monitors, and the DDR memory controller:

- MCIF performs ASEMI illegal access type detection and generates an ASEMI bus error in response. See **Table 10-1.**
- MCIF detects misaligned WRAP4 32-bit and 64-bit accesses and generates an ASEMI bus error in response.
- AMIF, AMDMA, AMENT, and AMEC bus monitors detect misaligned accesses that would affect ASEMI, and they generate NMIs.
- ASEMI accesses to non-selected DDR memory are detected by the DDR memory controller, which generates the DDR_ER NMI.

Table 10-1. ASEMI Illegal Access Type Detection

MASTER	HBURST	HSIZE	HWRITE	ASEMI Bus Error	Comment
AMIC	SINGLE	128-bit	R	N	Access accepted by MCIF.
	WRAP4	128-bit	R	N	Access accepted by MCIF.
	All other access types.			Y	Access rejected by MCIF resulting in an ASEMI bus error.
AMDMA	SINGLE	8-bit 16-bit 32-bit 64-bit	R/W	N	Access accepted by MCIF.
	WRAP4	64-bit	R/W	N	Access accepted by MCIF.
	All other access types.			Y	Access rejected by MCIF resulting in an ASEMI bus error.
AMENT	SINGLE	32-bit	R/W	N	Access accepted by MCIF.
	WRAP4	32-bit	R/W	N	Access accepted by MCIF.
	All other access types.			Y	Access rejected by MCIF resulting in an ASEMI bus error.
AMEC	SINGLE	8-bit 16-bit 32-bit 64-bit	R/W	N	Access accepted by MCIF.
	All other access types.			Y	Access rejected by MCIF resulting in an ASEMI bus error.

10.2.5 MCIF Reset

The MCIF has two reset sources: peripheral module reset and AHB disable. Peripheral module reset is the primary reset source for the MCIF and is described in **Chapter 13**, *Reset*. This reset forces all MCIF sequential elements to their reset state. AHB disable asserts when the ASEMI bus time-out monitor detects an error condition, and all MCIF sequential elements are reset, excluding the configuration registers described in **Section 10.4**, *MCIF Programming Model*, on page 10-7.

10.3 Programming the MCIF

The MCIF optimally serves DDR accesses when the MSC711x device is configured as described in the following procedure for read accesses. No MCIF configuration is needed for write accesses because the write buffer serves all write accesses.

1. Set MCIFCTL[IPRE] = 01 to enable IFU predictive reads.
2. If the ECI is used to stream data sequentially from the DDR, enable predictive reads for the ECI read buffer by setting MCIFCTL[EPRE] = 1.

3. To optimize DMA read accesses, use both the DMA and alternate read buffers as needed. These buffers should be configured to serve DMA channels that stream data from the DDR. Since each read buffer supports five DMA channels, you may find that using only the DMA read buffer is adequate.

   DMA accesses using the DMA read buffer:

   a. Enable DMA predictive reads (MCIFCTL[DPRE] = 1).
   b. Enable the DMA channel select operation (MCIFCTL[DCOE] = 1).
   c. Set DMA channel select fields in the DCHSEL register to correspond with DMA channels that stream data from DDR memory.
   d. Use a DMA transfer size of 32 bytes (64-bit WRAP4).

   DMA accesses using the alternate read buffer (as needed):

   a. Select the alternate read buffer (MCIFCTL[AMSEL] = 0001).
   b. Enable alternate predictive reads (MCIFCTL[APRE] = 1).
   c. Enable the DMA channel select operation (MCIFCTL[ACOE] = 1).
   d. Set DMA channel select fields in the ACHSEL register to correspond with DMA channels that stream data from DDR memory.
   e. Use a DMA transfer size of 32 bytes (64-bit WRAP4).

4. To optimize FEC read accesses, use the alternate read buffer. Give alternate read buffer service preference to the DMA controller if more than five DMA channels are used for DDR streaming. Otherwise, the alternate buffer should be enabled to serve the FEC.

   a. Select the alternate read buffer to serve the FEC (default) (MCIFCTL[AMSEL] = 0011).
   b. Enable alternate predictive reads (MCIFCTL[APRE] = 1).

For an application that uses code overlays, the source data is usually in DDR memory and the destination for the overlay code is in M2 memory. When an application performs code overlays with the source data in M1, M2, or DDR memory and the destination for the overlay code in DDR memory, you must ensure that there are no coherency issues. This case is uncommon but is documented here for completeness. To address this case, use the following sequence:

1. Leave the IFU predictive read turned on.
2. Jump to a section of code in DDR memory outside the overlay region that contains a sequence of 80 NOPs.
3. Execute these NOPs from this location.
4. Return to the desired location.

Jumping to a location outside the overlay space fills the read buffer with the new instructions, effectively flushing out the old read buffer contents.

10.4 MCIF Programming Model

The MCIF registers are listed as follows and discussed in this section:

- MCIF Control Register (MCIFCTL), **page 10-7**.
- DMA Read Buffer Channel Select Register (DCHSEL), **page 10-8**.
- Alternate Read Buffer Channel Select Register (ACHEL), **page 10-10**.
- MCIF Status Register (MCIFSTAT), **page 10-11**.

The value of the base address for this register file, MCIF_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4.

MCIFCTL MCIF Control Register MCIF_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—			IPRE		EPRE	DPRE	APRE	—							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0x0							

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—			DCOE	DMSEL				—			ACOE	AMSEL			
TYPE	R/W				R				R/W							
RESET	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1

MCIFCTL configures the MCIF module.

Table 10-2. MCIFCTL Bit Descriptions

Name	Reset	Description	Settings
— 31–29	0x0	Reserved. Write to zero for future compatibility.	
IPRE 28–27	0x0	**IFU Predictive Read Enable** Enables the MCIF to generate predictive reads on program accesses. If IFU predictive reads are enabled and the IFU is configured to have burst size = 1, the DDR address regions must be configured to be cacheable.	00 IFU predictive read disabled. 01 IFU predictive read always enabled. 10 IFU predictive read dynamically enabled by hardware. 11 Reserved.
EPRE 26	0x0	**ECI Predictive Read Enable** Enables the MCIF to generate predictive reads on ECI accesses.	0 ECI predictive read disabled. 1 ECI predictive read enabled.
DPRE 25	0	**DMA Predictive Read Enable** Enables the MCIF to generate predictive reads on DMA accesses.	0 DMA predictive read disabled. 1 DMA predictive read enabled.

Table 10-2. MCIFCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
APRE 24	0	**Alternate Predictive Read Enable** Enables the MCIF to generate predictive reads on alternate master accesses.	0 Alternate predictive read disabled. 1 Alternate predictive read enabled.
— 23–13	0x0	Reserved. Write to zero for future compatibility.	
DCOE 12	0x0	**DMA Channel Select Operation Enable** Enables DMA channel-based operation in which predictive reads are issued for the channel(s) selected by the DCHSEL register. When disabled, the MCIF issues predictive reads based on DMA accesses, regardless of the channel.	0 DMA channel select operation disabled. 1 DMA channel select operation enabled.
DMSEL 11–8	0x1	**DMA Read Buffer Master Select** Selects the crossbar master to service with the DMA read buffer.	This read-only field always reads as 0b0001, which corresponds to “DMA controller selected.”
— 7–5	0x0	Reserved. Write to zero for future compatibility.	
ACOE 4	0	Alternate Channel Select Operation Enable Enables DMA channel-based operation in which predictive reads are issued for the channel(s) selected by the DCHSEL register. When disabled, the MCIF issues predictive reads based on DMA accesses, regardless of the channel. For this bit to be used, AMSEL must be set to select the DMA as the alternate read buffer master.	0 Alternate channel select operation disabled. 1 Alternate channel select operation enabled.
AMSEL 3–0	0x3	**Alternate Read Buffer Master Select** Selects the crossbar master to service with the alternate read buffer.	0000 Reserved. 0001 DMA controller. 0010 Reserved. 0011 Fast Ethernet controller. Other Reserved.

DCHSEL DMA Read Buffer Channel Select Register MCIF_BASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DCHE					DCHD					DCHC					—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DCHB					DCHA					DCEE	DCDE	DCCE	DCBE	DCAE	—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DCHSEL selects specific DMA channels for service by the DMA read buffer when the DMA channel select operation is enabled (MCIFCTL[DCOE] = 1).

Table 10-3. DCHSEL Bit Descriptions

Name	Reset	Description	Settings
DCHE 31–27	0	**DMA Channel Select E** Selects a DMA channel to be serviced by the DMA read buffer. This selection is enabled by DCHSEL[DCEE]. DMA channel select operation is enabled by MCIFCTL[DCOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
DCHD 26–22	0	**DMA Channel Select D** Selects a DMA channel to be serviced by the DMA read buffer. This selection is enabled by DCHSEL[DCDE]. DMA channel select operation is enabled by MCIFCTL[DCOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
DCHC 21–17	0	**DMA Channel Select C** Selects a DMA channel to be serviced by the DMA read buffer. This selection is enabled by DCHSEL[DCCE]. DMA channel select operation is enabled by MCIFCTL[DCOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
— 16	0x0	Reserved. Write to zero for future compatibility.	
DCHB 15–11	0	**DMA Channel Select B** Selects a DMA channel to be serviced by the DMA read buffer. This selection is enabled by DCHSEL[DCBE]. DMA channel select operation is enabled by MCIFCTL[DCOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
DCHA 10–6	0	**DMA Channel Select A** Selects a DMA channel for service by the DMA read buffer. This selection is enabled by DCHSEL[DCAE]. DMA channel select operation is enabled by MCIFCTL[DCOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
DCEE 5	0	**DMA Channel Select E Enable** Enables the DCHSEL[DCHE] field to select a DMA channel to service.	0 Channel select E disabled. 1 Channel Select E enabled,
DCDE 4	0	**DMA Channel Select D Enable** Enables the DCHSEL[DCHD] field to select a DMA channel to service.	0 Channel Select D disabled, 1 Channel Select D enabled.
DCCE 3	0	**DMA Channel Select C Enable** Enables the DCHSEL[DCHC] field to select a DMA channel to service.	0 Channel Select C disabled. 1 Channel Select C enabled.
DCBE 2	0	**DMA Channel Select B Enable** Enables the DCHSEL[DCHB] field to select a DMA channel to service.	0 Channel Select B disabled. 1 Channel Select B enabled.

Table 10-3. DCHSEL Bit Descriptions (Continued)

Name	Reset	Description	Settings
DCAE 1	0	**DMA Channel Select A Enable** Enables DCHSEL[DCHA] field to select a DMA channel to service.	0 Channel Select A disabled. 1 Channel Select A enabled.
— 0	0x0	Reserved. Write to zero for future compatibility.	

ACHSEL Alternate Read Buffer Channel Select Register MCIF_BASE + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ACHE					ACHD					ACHC					—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ACHB					ACHA					ACEE	ACDE	ACCE	ACBE	ACAE	—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ACHSEL selects specific DMA channels for service by the alternate read buffer when the alternate channel select operation is enabled (MCIFCTL[ACOE] = 1), and the alternate read buffer master is configured for DMA operation (MCIFCTL[AMSEL] = DMA).

Table 10-4. ACHSEL Bit Descriptions

Name	Reset	Description	Settings
ACHE 31–27	0	**Alternate Channel Select E** Selects a DMA channel to be serviced by the alternate read buffer. This selection is enabled by ACHSEL[ACEE]. Alternate channel select operation is enabled by MCIFCTL[ACOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
ACHD 26–22	0	**Alternate Channel Select D** Selects a DMA channel to be serviced by the alternate read buffer. This selection is enabled by ACHSEL[ACDE]. Alternate channel select operation is enabled by MCIFCTL[ACOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
ACHC 21–17	0	**Alternate Channel Select C** Selects a DMA channel to be serviced by the alternate read buffer. This selection is enabled by ACHSEL[ACCE]. Alternate channel select operation is enabled by MCIFCTL[ACOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.

Table 10-4. ACHSEL Bit Descriptions

Name	Reset	Description	Settings
— 16	0x0	Reserved. Write to zero for future compatibility.	
ACHB 15–11	0	**Alternate Channel Select B** Selects a DMA channel to be serviced by the alternate read buffer. This selection is enabled by ACHSEL[ACBE]. Alternate channel select operation is enabled by MCIFCTL[ACOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
ACHA 10–6	0	**Alternate Channel Select A** Selects a DMA channel to be serviced by the alternate read buffer. This selection is enabled by ACHSEL[ACAE]. Alternate channel select operation is enabled by MCIFCTL[ACOE].	00000 DMA channel 0 selected. 00001 DMA channel 1 selected. . . 11111 DMA channel 31 selected.
ACEE 5	0	**Alternate Channel Select E Enable** Enables ACHSEL[ACHE] field to select a DMA channel to service.	0 Channel Select E disabled. 1 Channel Select E enabled.
ACDE 4	0	**Alternate Channel Select D Enable** Enables ACHSEL[ACHD] field to select a DMA channel to service.	0 Channel Select D disabled. 1 Channel Select D enabled.
ACCE 3	0	**Alternate Channel Select C Enable** Enables ACHSEL[ACHC] field to select a DMA channel to service.	0 Channel Select C disabled. 1 Channel Select C enabled.
ACBE 2	0	**Alternate Channel Select B Enable** Enables ACHSEL[ACHB] field to select a DMA channel to service.	0 Channel Select B disabled. 1 Channel Select B enabled.
ACAE 1	0	**Alternate Channel Select A Enable** Enables ACHSEL[ACHA] field to select a DMA channel to service.	0 Channel Select A disabled. 1 Channel Select A enabled.
— 0	0x0	Reserved. Write to zero for future compatibility.	

MCIFSTAT MCIF Status Register MCIF_BASE + 0x18

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	MCTLWD	DCHWD	ACHWD	—												
TYPE								R								
RESET	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE								R								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MCIFSTAT provides information on MCIF status.

Table 10-5. MCIFSTAT Bit Descriptions

Name	Reset	Description	Settings
MCTLWD 31	0x1	**MCIFCTL Register Write Access Done** Indicates that a write access to the MCIFCTL register is done (has taken affect). A write access to MCIFCTL will not take affect until the MCIF becomes idle.	0 MCIFCTL register was written but has not taken effect because the MCIF is not idle. 1 Write access to MCIFCTL register is done.
DCHWD 30	0x1	**DCHSEL Register Write Access Done** Indicates that a write access to the DCHSEL register is done (has taken affect). A write access to DCHSEL will not take affect until the MCIF becomes idle.	0 DCHSEL register was written but has not taken effect because the MCIF is not idle. 1 Write access to DCHSEL register is done.
ACHWD 29	0x1	**ACHSEL Register Write Access Done** Indicates that a write access to the ACHSEL register is done (has taken affect). A write access to ACHSEL will not take affect until the MCIF becomes idle.	0 ACHSEL register was written but has not taken effect because the MCIF is not idle. 1 Write access to ACHSEL register is done.
— 28–0	0x0	Reserved. Write to zero for future compatibility.	

Clocks and Power Management 11

This chapter describes the MSC711x clocking module, low-power modes of operation, and programming model.

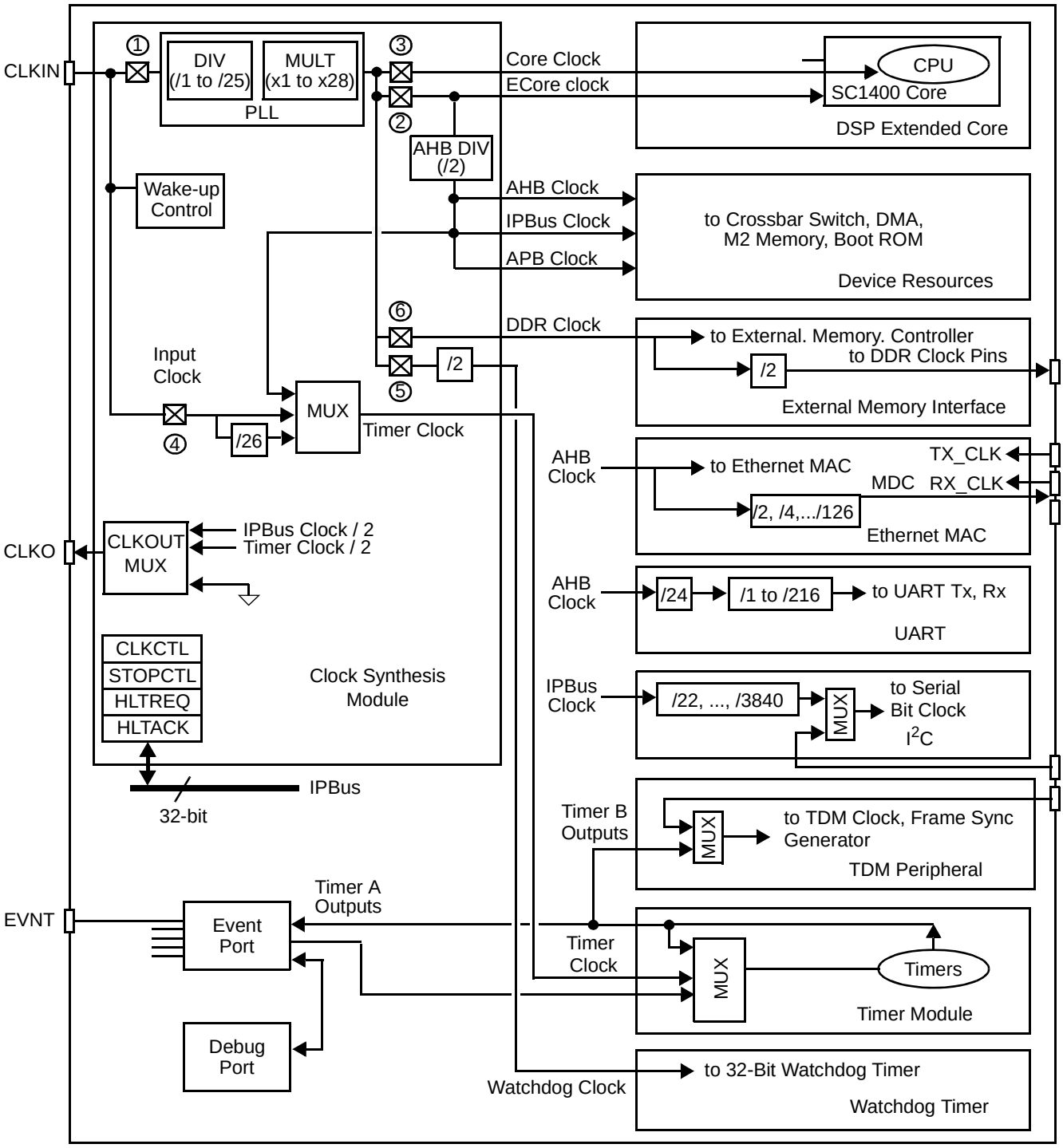
11.1 Timing System Architecture

The MSC711x timing system, shown in **Figure 11-1**, has the clock synthesis module at its core. This module is composed of the following blocks:

- Phase lock loop (PLL) with associated multipliers and dividers.
- Bus clock divider.
- Timer clock multiplex.
- Wake-up control to wake the MSC711x device out of its low-power modes.
- Control registers for programming the clock synthesis module. The SC1400 PCTL0 and PCTL1 registers are not used on the MSC711x architecture.

Together, these blocks generate the clock signals used for core and peripheral clocking:

- An external input clock provides a reference clock for the system.
- The core clock is obtained by dividing the input clock and multiplying the frequency in the PLL.
- The AHB clock is generated much like the core clock but with an additional division stage.
- The IPBus clock is generated from the AHB clock.
- The APB clock is generated from the AHB clock.
- The timer clock is derived from the external input clock or from the IPBus clock.



① Clocks can be disabled in Stop mode. Disables PLL, core clock, ECore clock, AHB clock, IPBus clock, and APB Clock.

② Clocks can be disabled at this point in Stop mode. Disables ECore clock, AHB clock, IPBus clock, and APB clock.

③ Clocks are disabled at this point in Wait and Stop modes. Disables the ECore clock.

④ Clocks can be disabled at this point in Stop mode. Disables the input clock used in timer clock generation.

⑤ Clocks can be disabled at this point in Stop mode. Disables the watchdog timer clock.

⑥ Clocks can be disabled at this point in Stop mode. Disables the DDR clock.

Figure 11-1. MSC711x Timing System

Table 11-1. MSC711x Clocking by Module

Block	Primary Clock for Data Transfers	Secondary Clock for Register Access	Baud-Rate Clock or Clock Generation by the Module	Comments
Extended Core Modules				
SC1400 Core	Core clock	—	N/A	—
Instruction Cache	ECore clock	AHB clock	N/A	AHB clock to access the array.
IFU	ECore clock	AHB clock	N/A	AHB clock to interface with the AMIC bus and access the IFU registers.
ECI	ECore clock	AHB clock	N/A	AHB clock to interface with the AMEC bus and access the ECI registers.
Data Transfer Modules				
DMA	AHB clock	IPBus clock	N/A	—
Crossbar Switch	AHB clock	IPBus clock	N/A	Primary clock for state machines, arbitration logic, and latching signals when the slave port is busy.
Memory Modules				
M1 SRAM	ECore clock	—	N/A	Values on the 64-bit ASM1 bus are written at the AHB clock frequency.
M2 SRAM	AHB clock	—	N/A	—
Boot ROM	AHB clock	—	N/A	—
Peripheral Modules				
External memory interface	AHB clock	IPBus clock	DDR clock	DDR clock generated by this module at 1:2, 1:4, or 1:8 ratio of the ECore clock.
Ethernet MAC	AHB clock	IPBus clock	TX_CLK: from pin RX_CLK: from pin MDC: from AHB clock	MDC baud clock generated internally from the AHB clock via a divider.
HDI16	AHB clock	APB clock	N/A	**Note:** The state machines for the HDI clock are clocked off ECore clock.
TDMs	AHB clock	APB clock	Timer B or TDM Pins	Baud clock received from external sources (TDM pins) or generated internally via Timer B outputs.
Interrupt Controller	AHB clock	AHB clock	N/A	Registers accessed from the ASAPB bus. Edge detection circuitry for non-maskable interrupts is clocked off the ECore clock.
I^2C	—	IPBus clock	IPBus clock	Serial bit clock received from the SCL pin or generated internally from the IPBus clock via a divider.
UART	—	APB clock	APB clock	Serial bit clock generated internally from the APB clock via a divider.

Table 11-1. MSC711x Clocking by Module (Continued)

Block	Primary Clock for Data Transfers	Secondary Clock for Register Access	Baud-Rate Clock or Clock Generation by the Module	Comments
Timer Modules (The feedback loop in the timer modules is used for cascaded counter operation.)				
Timer Module	—	IPBus clock	Timer clock or clocked by the event port	Timers are clocked by various sources (see **Figure 21-1**, *Timer Module Block Diagram, One of Four Channels,* on page 21-2)
Watchdog Timer	—	Watchdog clock	Watchdog clock	Watchdog timer is clocked by a gated version of the APB clock.
Clocking Modules				
PLL/clock	—	IPBus clock	Generates device clocks	Generates the following clocks: • Core clock • ECore clock • AHB clock • APB clock • IPBus clock • DDR clock • Timer clock • Watchdog clock
Event Port	—	IPBus clock		Can provide clocks to Timer Modules.
Note: A single source is used for the following clocks so that they are synchronous with each other and run at the same frequency: AHB clock (for AHB-Lite subsystem), APB clock, and IPBus clock.				

11.2 Clock Synthesis Module Operation

The clock synthesis module generates the following clocks, which are used throughout the MSC711x device:

- *Clocks at maximum frequency*: core clock, ECore clock, and DDR clock. The core clock provides clocking for the SC1400 core, and the ECore clock provides clocking for the extended core, including M1 memory, ICache, instruction fetch unit (IFU), and the extended core interface (ECI).
- *Clocks at half of maximum frequency*: AHB clock, IPBus clock, APB clock, and watchdog clock. For bus operation and register access, the AHB clock provides clocking for the AHB-Lite subsystem, including the DMA controller, crossbar switch, M2 memory, boot ROM, and the interrupt controller.
- *Other clocks*: timer clock. The timer clock provides clocking for the timer module. See **Section 11.4.4.10**, *Selecting the Input Clock as the Source for the Timer Modules,* on page 11-22 for restrictions when the input clock is selected as the source of this clock. The timer module can also be clocked from the EVNT[3–0] pins or other system events through event multiplexes within the system control block (see **Chapter 15**, *Event Port*). The watchdog clock provides clocking for the software watchdog timer.

Clock generation for internal peripherals is as follows:

- The IPBus clock for the IPBus peripherals.
- The APB clock for the APB peripherals.
- The DDR clock for the DDR memory controller.

When clocks are internally generated for the TDMs, the desired baud rate is generated by timer(s) in timer module B. All scaling of the frequency occurs within timer module B. The TDMs can also receive external clocks on their RCLKx and TCLKx pins. The correct baud rate for the I^2C is generated from the IPBus clock, with appropriate scaling within the I^2C module. The correct baud rate for the UART is generated from the APB clock, with appropriate scaling within the UART.

A multiplex is provided for selecting a clockout signal that is output on the EVNT1 pin. The sources for this clock can be the IPBus clock /2 or the timer clock /2. This clock can also be disabled, which is its default configuration out of reset. When the clockout feature is not disabled by the CLKO bits, it overrides all other functionality on the EVNT1 signal and provides the desired output clock. When it is disabled, the EVNT1 signal is controlled by the system control unit.

To enable the CLKO bits, perform the following steps:

1. Set the CLKCTL[CLK0] bits to select the desired source clock out (IPBus/2 or timer).
2. Set DEVCFG[PAS] = 1 to select the additional multiplexing function.
3. Set GPACTL[CTL17] = 1 for hardware control.

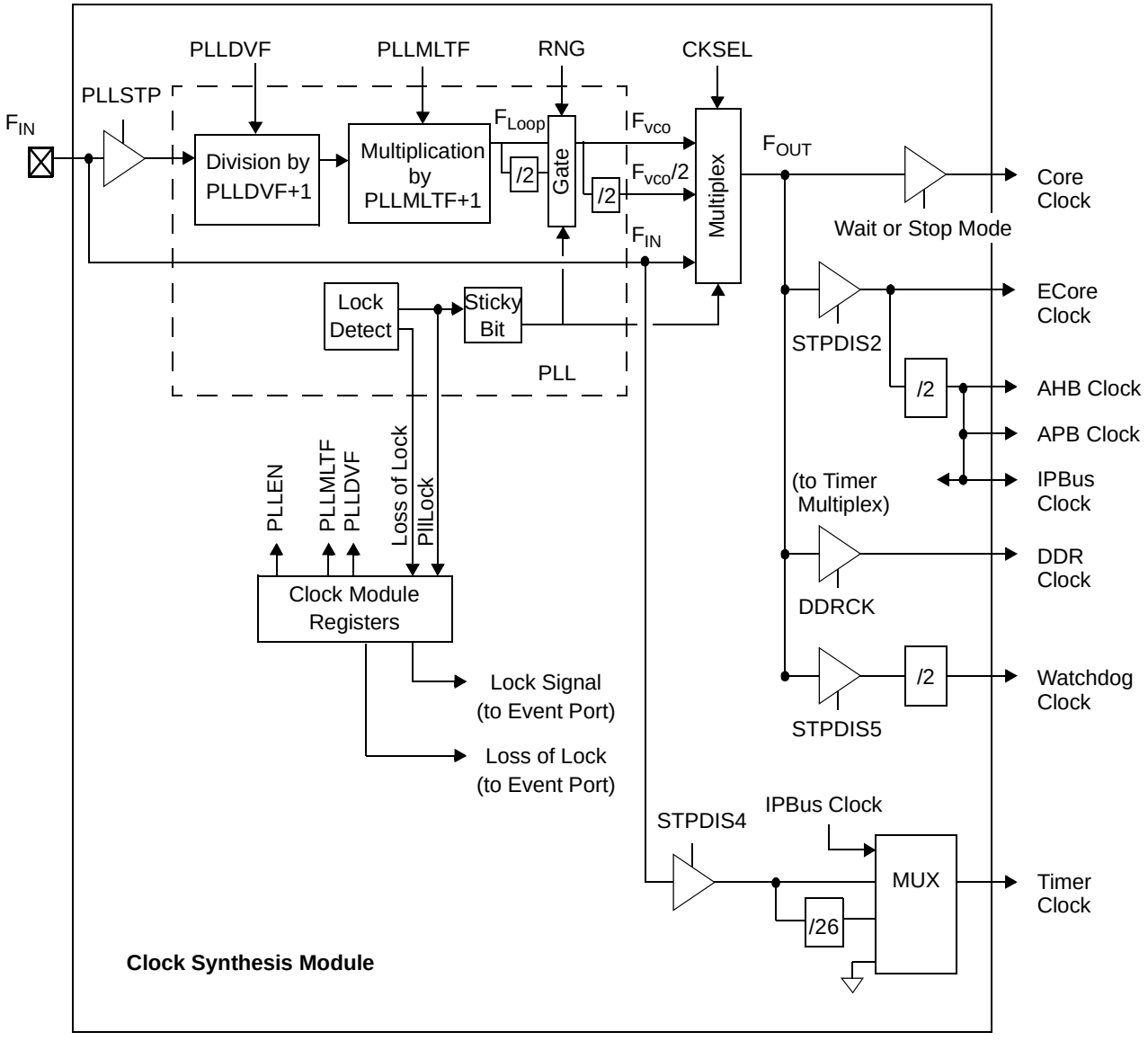
11.2.1 Generating the Clocks

MSC711x devices use the PLL for the primary clock generation. The CLKOUT signal, the internal core clock, AHB clock, IPBus clock, and APB clock are generated as a function of the F_{IN} and PLL clock out signals (see **Figure 11-2**). F_{IN} is clocked from an external clock source and is fed to the PLL that divides and multiplies its frequency according to the PLLDVF and the PLLMLT fields of the Clock Control Register (CLKCTL). The PLLDVF field is a software controllable input division factor that divides the reference clock frequency F_{IN} by a 6-bit value. The PLLMLTF field multiplies the frequency according to its 8-bit value in the CLKCTL register.

11.2.2 Configuring the Clocks

The CLKCTL register settings determine the PLL clockout frequency as well as the CLKIN, AHB clock, APB clock, and CORE clock frequency ratios. The following factors can be configured:

- PLL input division factor (PLLDVF)
- PLL multiplication factor (PLLMLTF)
- Clock select bits (CKSEL)
- Range bit (RNG)



PLLDVF: Input Division Factor

PLLMLTF: Multiplication Factor

Figure 11-2. Clock Generation

The equations for the different frequencies are as follows:

$F_{Loop} = F_{IN} * ((PLLMLTF+1)/(PLLDVF+1))$

F_{vco}	$= F_{Loop}/2$	RNG = 0	--low freq chip operation
	$= F_{Loop}$	RNG = 1	--high freq chip operation
F_{out}	$= F_{IN}$	CKSEL = 00	
	= 0	CKSEL = 01	--while waiting for 1st lock
	$= F_{vco}/2$	CKSEL = 01	--once the PLL has locked

```
                = 0             CKSEL = 11   --while waiting for 1st lock
                = Fvco          CKSEL = 11   --once the PLL has locked
FCore = Fout
```

$$F_{AHB} = F_{APB} = F_{IPBus} = F_{out} / 2$$

There are restrictions on the frequencies permitted at the input of the multiplication portion of the PLL. Specifically, the frequency at this node must fall within a particular range of frequencies. Similarly, there are restrictions on the minimum and maximum frequencies at the output of the multiplication block. Refer to the device data sheet for details. **Table 11-2** summarizes the equations for the frequency of the clocks.

Table 11-2. Summary of Device Frequency Equations

RNG	CKSEL	Frequency of Core and ECore Clock	AHB, APB, and IPBus Clock	Comments
0	00	= F_{IN}	Core Clock / 2	Uses bypass clock.
0	01	= F_{IN} * [(PLLMLTF+1) / (PLLDVF+1)] / 4	Core Clock / 2	See Note below.
0	11	= F_{IN} * [(PLLMLTF+1) / (PLLDVF+1)] / 2	Core Clock / 2	See Note below.
1	00	= F_{IN}	Core Clock / 2	Uses bypass clock.
1	01	= F_{IN} * [(PLLMLTF+1) / (PLLDVF+1)] / 2	Core Clock / 2	See note.
1	11	= F_{IN} * [(PLLMLTF+1) / (PLLDVF+1)]	Core Clock / 2	See note.
Note: Depends on PLL lock. See **Section 11.3.4.2**, *Bypass Clock,* on page 11-10 and **Section 11.3.4**, *Modifying the PLL Settings,* on page 11-10.				

11.2.3 Selecting Clock Frequencies

Two restrictions that must be met for correct usage of the PLL are as follows:

- The allowed range for the input frequency to the PLL, which is the output of the input divider, is 10–25 MHz.
- The allowed range for the output frequency of the PLL, which is the output of the multiplier, is 266–532 MHz.

The output frequencies are programmed via the PLLDVF, PLLMLTF, and RNG fields of the Clock Control Register (CLKCTL). The restriction on the input divider output results in the allowed ranged for the device input clock listed in **Table 11-3**.

Table 11-3. Allowed Division Factors and Frequency Ranges for CLKIN

PLLDVF Field	Input Divide Factor	Allowed Range at CLKIN Pin	Comments
0x00	1	10–25 MHz	Input division by 1
0x01	2	20–50 MHz	Input division by 2
0x02	3	30–75 MHz	Input division by 3
0x03	4	40–100 MHz	Input division by 4
0x04	5	50–100 MHz	Input division by 5

Table 11-3. Allowed Division Factors and Frequency Ranges for CLKIN

PLLDVF Field	Input Divide Factor	Allowed Range at CLKIN Pin	Comments
0x05	6	60–100 MHz	Input division by 6
0x06	7	70–100 MHz	Input division by 7
0x07	8	80–100 MHz	Input division by 8
0x08	9	90–100 MHz	Input division by 9
0x09	10	100 MHz	Input division by 10
Note: The maximum frequency allowed on the CLKIN pin is 100 MHz. As a result, only division factors from 1 to 10 are allowed.			

The restriction on the output of the multiplier results in the allowed ranges for the device input clock listed in **Table 11-4**.

Table 11-4. Allowed Multiplication Factors

Allowed Range at Output of V_{CO}	Minimum Permitted Multiplier	Maximum Permitted Multiplier
266 <= [Input Divided Clock * (PLLMLTF+1)] <= 532 MHz	266/input divided clock	532/input divided clock
Notes: **1.** This table results from the allowed range for F_{Loop}. **2.** The minimum and maximum multiplication factors depend on the frequency of the input divided clock.		

As **Table 11-5** shows, the frequency delivered to the SC1400 core, extended core, and peripherals depends on the value of the CLKCTL[RNG] bit (see **page 11-25**).

Table 11-5. F_{vco} Allowed Frequency Ranges

CLKCTRL[RNG]	Allowed Range of F_{vco}
1	266 <= F_{vco} <= 532 MHz
0	133 <= F_{vco} <= 266 MHz
Note: This table results from the allowed range for F_{vco}, which is F_{Loop} modified by CLKCTRL[RNG].	

Both the CLKCTL[RNG, CKSEL] bits determine the final permitted frequency range of the core clock, as shown in **Table 11-6**.

Table 11-6. Resulting Ranges Permitted for the Core Clock

CLKCTRL[CKSEL]	CLKCTRL[RNG]	Resulting Division Factor	Allowed Range of Core Clock	Comments
11	1	1	266 <= Core_Clk <= 300 MHz	Limited by maximum frequency of the core
11	0	2	133 <= Core_Clk <= 266 MHz	Limited by range of PLL
01	1	2	133 <= Core_Clk <= 266 MHz	Limited by range of PLL
01	0	4	66.5 <= Core_Clk <= 133 MHz	Limited by range of PLL
Note: This table results from the allowed range for F_{OUT}, which depends on clock selected via CLKCTRL[CKSEL].				

The permitted range of the core clock is also affected by the allowed frequency range of the DDR devices in the system, which are listed in **Table 11-7**.

Table 11-7. Core Clock Ranges When Using DDR

DDR Type	Allowed Frequency Range for DDR CK pin	Corresponding Range for the Core Clock	Comments
DDR 200 (PC-1600)	83–100 MHz	166 <= Core_Clk <= 200 MHz	Core limited to 2 x Max DDR frequency
DDR 266 (PC-2100)	83–133 MHz	166 <= Core_Clk <= 266 MHz	Core limited to 2 x Max DDR frequency
DDR 333 (PC-2600)	83–166 MHz	166 <= Core_Clk <= 333 MHz	Core limited to 2 x Max DDR frequency

11.3 Clock Selection

This section describes important considerations for selecting clocking frequencies. Refer to the data sheet for details on selecting the clocking frequency.

11.3.1 Resetting the Clock Synthesis Module

The clock synthesis module is reset during power-on reset, as follows:

1. The CLKCTL register is reset, which disables the PLL
2. Bypass clock is selected.
3. PLL clock is gated off.
4. The sticky bit for clock gating is reset, as shown in **Figure 11-2**.

See **Table 13-2**, *Reset Actions for Each Reset Source,* on page 13-2 to see how the remainder of the clock synthesis module is affected by reset.

11.3.2 Enabling the PLL

When the application sets up the PLL for initial use or if the PLL is disabled, it performs the following steps:

1. An SC1400 instruction writes values to the fields of the CLKCTL register (see **Table 11-8**, *CLKCTL Bit Descriptions*, on page 11-25):
   - The desired multiplication factor is written to PLLMLTF.
   - The desired division factor is written to PLLDVF.
   - A value of 00 is written to CKSEL (01 or 11 is also permitted).
   - A value of 1 is written to RSTRT and PLLEN.
2. The bypass clock drives the F_{out} clock so that SC1400 instructions can execute while waiting for lock.
3. To use the PLL clock for the device, the CLKCTL[CKSEL] field is written with a value of 01 or 11 (which can be done in step 1, if desired). When the PLL clock is used:

- If the PLL is already locked, F_{out} immediately switches to the PLL clock.
- If the PLL is *not* locked, F_{out} remains on the bypass clock until lock occurs. Then the clock source is automatically changes to the PLL clock.

4. When the PLL acquires lock, its LOCK status bit is set.

This behavior ensures that if the PLL is still acquiring lock, it is driven with the bypass clock and waits until lock is acquired. Then it automatically switches to the PLL clock.

11.3.3 PLL Lock Status

A status bit, CLKCTL[LCK], reflects the state of the PLL lock and can be polled during the wait for the PLL to lock. This signal is also sent to the event port where it can be used for event port triggering.

11.3.4 Modifying the PLL Settings

When the PLL is already running, there are two techniques for modifying the settings:

- PLL restart
- Bypass clock

When the PLL is first enabled or the PLL frequency is modified using one of these techniques, you should place the segment of code for modifying the PLL settings into internal memory. If this code executes from DDR memory, the DDR clocking frequency may change to a value outside its permitted range.

11.3.4.1 PLL Restart

The RSTRT bit must be set in any instruction that writes a new value to the PLLMLTF or PLLDVF fields. The PLL restart technique applies only when the PLL is selected as the current clock, CLKCTRL[CKSEL] = 01 or 11. The steps for a PLL restart are as follows:

1. An SC1400 instruction writes a new value to the PLLMLTF and/or PLLDVF field(s) and writes a value of 1 to the RSTRT bit.
2. The clock module automatically switches to the bypass clock while the PLL acquires lock with the new ratios.
3. When the PLL acquires lock, the circuit automatically switches to the PLL clock, using the source determined by CLKCTL[CKSEL].

11.3.4.2 Bypass Clock

An application runs off the bypass clock while the PLL acquires lock, as follows:

1. An application selects the bypass clock by executing an instruction to modify the CLKCTL[CKSEL] field.
2. A new SC1400 instruction writes a new value to the PLLMLTF and/or PLLDVF field(s) and writes a value of 1 to the RSTRT bit. This instruction must not be grouped with the instruction in the previous step. That is, it must execute after the first instruction.
3. The bypass clock drives the F_{out} clock so SC1400 instructions can still execute during the wait for lock.
4. An SC1400 instruction can be used to select the new PLL clock even before it has acquired lock. The bypass clock continues to drive F_{out}.
5. When the PLL acquires lock, the circuit automatically gates the PLL clock onto F_{out}, providing a system clock at the new frequency.

11.3.5 Disabling the PLL

The clock must be set to the bypass clock via the CLKCTL[CKSEL] field before the PLL is disabled using the CLKCTL[PLLEN] bit. To re-enable the PLL, follow the steps in **Section 11.3.2**, *Enabling the PLL*, on page 11-9.

11.3.6 Loss of Lock Handling

The MSC711x PLL is robust and remains locked except in cases of loss of lock due to a temporary board-level disturbance in the power supply or input reference clock. The MSC711x lock detection circuit detects a loss of lock. This information is latched in the LSLK sticky bit within the CLKCTL register, and a signal is sent to the event port, where it can be used for event port triggering. The stickiness of this bit ensures that the loss of lock condition remains captured even if the loss of lock occurred for a small number of clock cycles.

Figure 11-3 illustrates the operation of the lock and loss-of-lock status bits:

1. The PLL is enabled as described in **Section 11.3.2**, *Enabling the PLL*, on page 11-9.
2. The system is disturbed by a power bounce, a drift in the reference clock, or a heavy noise, causing a loss of PLL lock.
3. This event is captured in the Loss of Lock status bit. When this sticky bit is set, it remains set.
4. Minor disturbances quickly relock. Clocks are continually sent to the system; that is, the clock gating signal remains asserted. Large disturbances require longer to relock.
5. When the PLL is disabled, it loses lock, clearing the lock and loss-of-lock status bits. The gating circuit disables the PLL clock, or the PLL is disabled on when the system enters Stop mode.

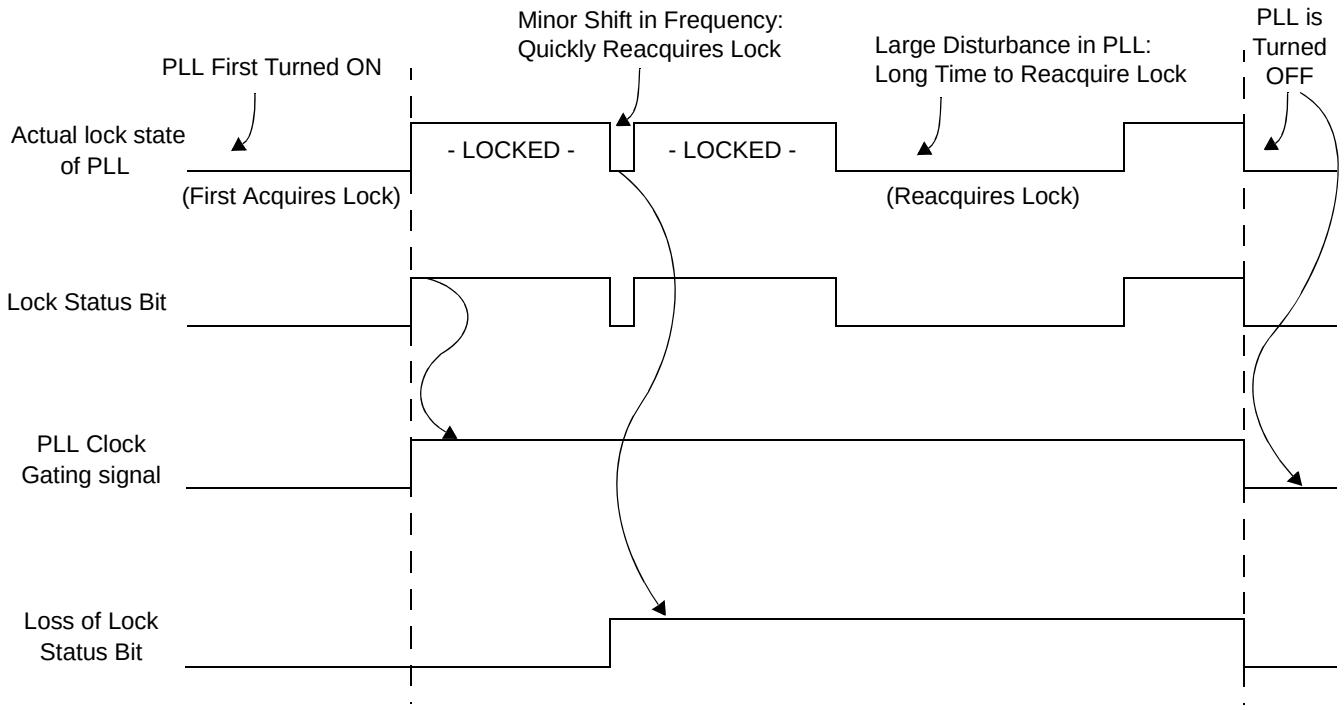


Figure 11-3. Demonstrating PLL Locking and Clock Gating

The system should be designed so that these disturbances do not occur and do not cause loss of lock. However, this example demonstrates how the MSC711x clock module responds to disturbances, such as might occur during initial board development.

11.4 Low-Power Operation

Effective low-power operation give complete user control over power reduction throughout the system. MSC711x devices allow individual power control over the clock synthesis module, extended core, AHB subsystem, and the peripheral subsystem, as shown in **Figure 11-4**.

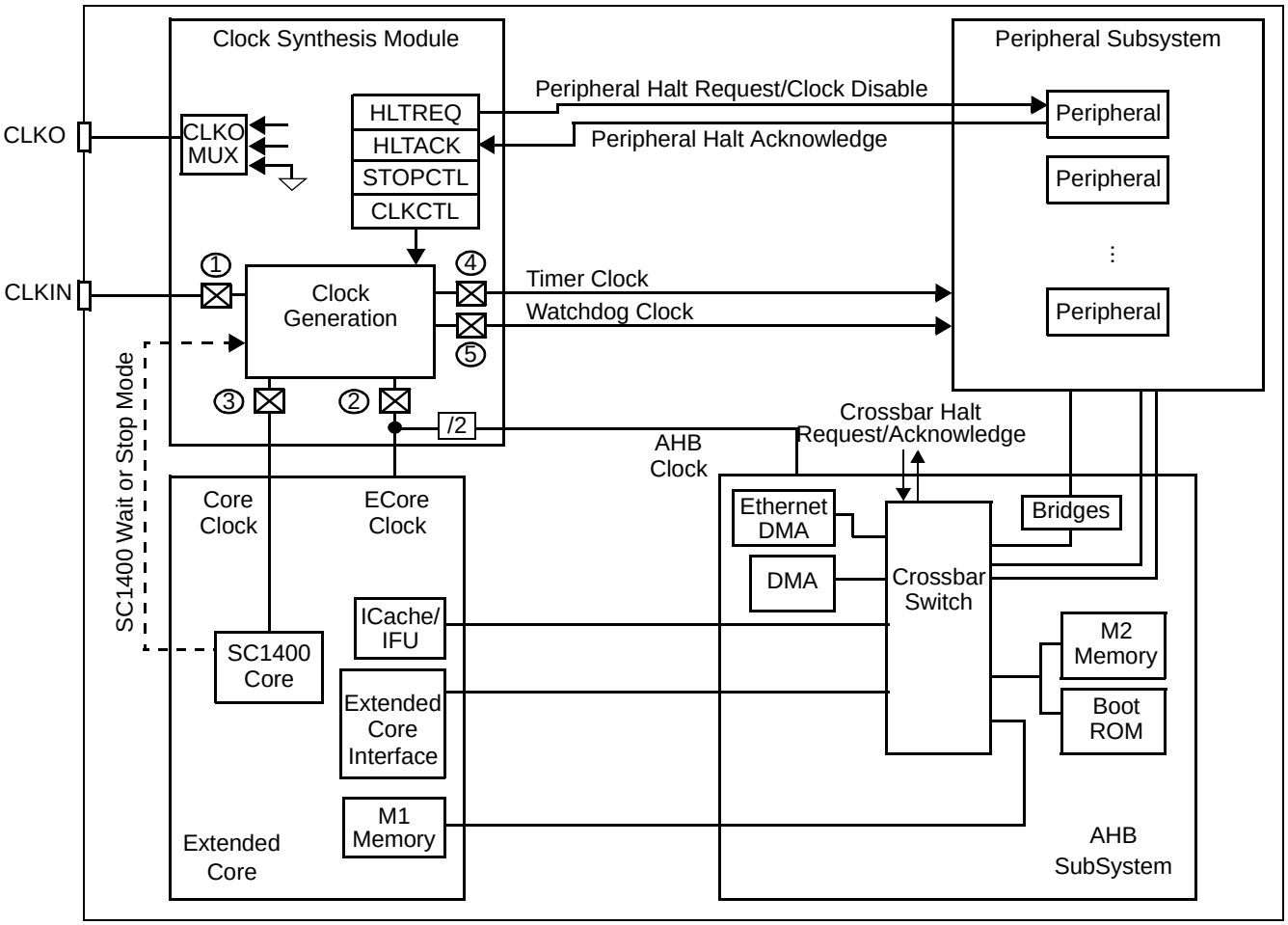


Figure 11-4. MSC711x Region-Based Low-Power Operation

The techniques for lowering the operating power of an MSC711x device include:

- Individually enable or shut down important MSC711x clocks (some clocks can be programmed to be shut down in the SC1400 Stop mode).
- Shut down the SC1400 core via the Wait or Stop modes.
- Use different available techniques to shut down the PLL.
- Freeze or shut down the AHB subsystem.
- Individually enable or shut down each peripheral.

11.4.1 Extended Core Low-Power Operation

The SC1400 core can be placed into either Stop or Wait mode, and the extended core can be placed into Stop mode. The extended core enters Wait mode when it issues the **wait** instruction. In Wait mode, the SC1400 core consumes minimal power because its clocks are frozen. The clocks of other modules inside the extended core do not stop, so the crossbar switch and interrupt controller are functional. The PLL is not affected. The SC1400 core exits Wait mode under the following conditions:

- Reception of a maskable interrupt request with a priority greater than the current core priority level.
- Reception of a non-maskable interrupt request, independent of the state of the SC1400 EMR[NMID] bit.
- Power-on reset, any hard reset, any soft reset.
- Any request to enter the SC1400 Debug mode.

The non-maskable and maskable interrupt request signals from the interrupt controller can wake the device from Wait mode. These signals are first synchronized in the clock controller when the device wakes up from Wait mode.

The SC1400 core and extended core enter Stop mode when the core issues the **stop** instruction. In Stop mode, the SC1400 core consumes minimal power because its clock is frozen. Stop mode is also used to shut down chip-level modules and subsystems. In Stop mode, the ECore clock is frozen and the system clocks behave as follows:

- Core clock is frozen.
- AHB, IPBus, APB and ECore clocks are *not* frozen unless selected by the STDIS2 bit.
- Timer clock is *not* frozen unless selected by the STDIS4 bit.
- Watchdog clock is *not* frozen unless selected by the STDIS5 bit.
- PLL is *not* frozen unless selected by the PLLSTP bits.
- DDR is *not* frozen unless selected by the DDRCK bits.

In Stop mode, the different interrupt request sources are handled as follows:

- Non-maskable chip-level requests are not masked and can optionally be programmed to bring the SC1400 core out of Stop mode (see **Section 11.4.5**, *Exit from Stop Mode*).
- Software can clear or mask the $\overline{\text{IRQ}}$ pins in the GPIO module before entry into Stop mode, and $\overline{\text{IRQ}}$ status can also be read (if $\overline{\text{IRQ}}$ requests are pending).
- Interrupt requests from device peripherals are handled in one of the following ways before entry into Stop mode:
  - A peripheral module is disabled, removing any pending interrupt requests.
  - Pending interrupt requests are removed by clearing interrupt enable bits in the peripheral.
  - Pending interrupt requests are removed by setting their associated IPLs within the interrupt controller to disabled.
  - Requests from peripherals are simply ignored in Stop mode.

When the **stop** instruction executes, there must be no changes to the STOPCTL register (**page 11-26**) in any of the three preceding instructions. If a write instruction is used to modify the STOPCTL register before entry into Stop mode, it must be a Write-Immediate No Freeze or, if not, the following conditions apply:

- A Write-Immediate must occur at least eight cycles before the **stop** instruction executes.
- Before a write to STOPCTL through the write buffer, the write buffer must be flushed at least eight cycles before the **stop** instruction executes.

11.4.2 Clock Synthesis Module Low Power Operation

The clock synthesis module generates clocking for an MSC711x device. In applications requiring minimum power consumption, you can use the following techniques to lower the power consumption via the clocks in the clock synthesis module:

- PLL operation:
  - PLL disabled.
  - In SC1400 Stop mode, power-down PLL digital circuitry (input divider not affected).
  - In SC1400 Stop mode, power-down entire PLL (input divider also powered down).
  - PLL normal operation continues in SC1400 Stop mode.
- Timer clock operation:
  - Timer clock disabled.
  - In SC1400 Stop mode, the timer clock is shut down.
  - Timer clock operation continues in SC1400 Stop mode, except when the IPBus clock is selected as the timer clock and STDIS2 is set. It is disabled via the IPBus clock.
- Watchdog clock operation:
  - Watchdog clock disabled.
  - In SC1400 Stop mode, the watchdog clock is shut down.

  - Watchdog clock normal operation continues in SC1400 Stop mode.
- DDR clock operation:
  - DDR clock disabled.
  - In SC1400 Stop mode, the DDR clock is shut down, with the result that the external DDR memory chips are not refreshed.
  - DDR clock normal operation continues in SC1400 Stop mode.
- Turn off the CLKO pin when it is not in use.

Normally Stop mode affects SC1400 core and extended core operation, but on MSC711x devices, the detection of entry into Stop mode is fed into the clock synthesis module so that selected chip-level regions can be programmed to shut down automatically in this mode and be re-enabled when the SC1400 exits Stop mode.

The PLL takes advantage of its automatic switching at PLL lock for fast Stop mode wake-up—unless it is programmed to be unaffected by Stop mode. However, when Stop mode powers down all or part of the PLL:

- If it is programmed for bypass clock in Stop mode, it remains with the bypass clock upon exit.
- If it is programmed for PLL clock upon entry into Stop, it exits Stop mode using the bypass clock.
- If the PLL acquires lock, it automatically switches from the bypass clock to the PLL clock upon acquiring lock.

11.4.3 AHB Subsystem Low-Power Operation

The AHB subsystem can continue to operate so that DMA activity can continue. The subsystem can also be frozen or powered down quickly. A freeze occurs independently of the SC1400 Stop mode and halts the crossbar switch though it does not halt the clock to the crossbar. A shutdown of the AHB subsystem used the SC1400 Stop mode and halts the crossbar switch and also the clock to the crossbar. Correctly halting the crossbar switch before entering Stop mode can be very important for data integrity in an application because it helps to ensure that the clocks are not gated off in the middle of a system-level transfer. The crossbar HLTREQ[XBRHRQ] bit is used to request a halt from the switch, and the switch in turn provides an acknowledge of its halt with the crossbar HLTACK[XBRSTA] bit.

11.4.3.1 Limited Halt of the Crossbar Switch

To halt the crossbar switch for low power consumption, use the following procedure:

1. Ensure that HLTREQ[ITCCD] is cleared so that clocks to the interrupt controller are enabled.
2. Flush the extended core write buffer by reading the Write Buffer Flush Register (WBFR).

3. Enable the AMEC and AMIC bus error detection units, which monitor the master buses for time-outs (see **Table 7-2**, *BERRCTL Bit Descriptions*, on page 7-10).
4. Disable the AMDMA and AMENT Bus Error Detection Units, which monitor the master buses for time-outs (see **Table 7-2**, *BERRCTL Bit Descriptions*, on page 7-10).
5. Set the ECI GPSCTL[XHRQ] bit to request a halt of the crossbar switch (see **Table 4-12**, *GPSCTL Bit Descriptions*, on page 4-43.).
6. Wait for an acknowledge by polling the crossbar switch GPSCTL[XHACK] bit. The crossbar asserts this bit when all slave ports have arbitrated for the halt and have completed all other activity so each slave port has halted.

To restart the crossbar switch:

1. Clear the ECI GPSCTL[XHRQ] bit to remove the halt request.
2. Re-enable the bus error detection units.

Note: If the interrupt vector table is in DDR, then the DDR must remain on when the crossbar is powered down. It may be desirable that the HLTREQ[EPCD] bit remain enabled when the crossbar is powered down.

11.4.3.2 Complete Halt of the Crossbar Switch

To halt the crossbar switch completely for lowest power consumption, use the following procedure:

1. Ensure that HLTREQ[ITCCD] is cleared so that clocks to the interrupt controller are enabled.
2. Flush the extended core write buffer by reading the Write Buffer Flush Register (WBFR).
3. Enable the AMEC and AMIC bus error detection units, which monitor the master buses for time-outs (see **Table 7-2**, *BERRCTL Bit Descriptions*, on page 7-10).
4. Disable the AMDMA and AMENT Bus Error Detection Units, which monitor the master buses for time-outs (see **Table 7-2**, *BERRCTL Bit Descriptions*, on page 7-10).
5. Set the ECI GPSCTL[XBRHRQ] bit to request a halt from the crossbar switch.
6. Wait for an acknowledge by polling the crossbar GPSCTL[XBRHAK] bit. The crossbar switch asserts this bit when all slave ports arbitrate for the halt and completes all other activity so each slave port halts.
7. Set the STOPCTL[STDIS2] bit to shut down the AHB, APB, ECore clock, and IPBus clocks in the SC1400 Stop mode for lowest power consumption.
8. Execute a **stop** instruction from the SC1400 core.

To restart the crossbar switch, use any of the valid mechanisms for exiting the SC1400 Stop mode, which re-enables the system clock to the module. Clear the ECI GPSCTL[XBRHRQ] bit to remove the halt request. Then re-enable the bus error detection units.

When halting the crossbar, take care to ensure that desired modules can still be accessed correctly. When the crossbar is halted, there is no access to peripherals through the APB or IPBus bridges or through the ASTH bus. In addition, there is no access to the M2 memory, boot ROM, or DDR controller. Therefore, the crossbar halt request and halt acknowledge bits are in the ECore, which is accessible even when the crossbar is halted, rather than in the clock synthesis module.

Also, M2 memory, the DDR memory controller are unavailable and ICache misses can no longer be serviced when the crossbar is halted. Therefore, code should be executed from M1 memory when the crossbar is halted.

11.4.4 Peripheral Subsystem Low Power Operation

For greatest power reduction, you can individually halt the MSC711x peripherals with the Halt Request Control Register. Once halted, these modules must not be accessed until they are re-enabled. The interrupt controller uses its halt request bit to stop its clocks in a sequence. It keeps pending interrupts and handles them, but it does not detect new interrupts. It signals a halt acknowledge only when there are no more enabled pending interrupts.

11.4.4.1 Complete Halt of the DDR Memory Controller

To halt the memory controller completely, use the following procedure:

1. Set up the DDR memory controller self refresh and dynamic power management as desired in the SCFG register.
2. Set the HLTREQ[DDRHRQ] bit to request a halt.
3. Wait for an acknowledge by polling the HLTACK[DDRHAK] bit. The memory controller asserts this bit when all accesses have completed.
4. Set the STOPCTL[DDRCK] bits to a value of 01 to shut down the DDR clock for lowest power consumption.

To restart the memory controller:

1. Set the STOPCTL[DDRCK] bits to 00 or 10 to re-enable the DDR clock to the module.
2. Clear the HLTREQ[DDRHRQ] bit to remove the halt request.

11.4.4.2 Halt of the DDR Memory Controller in Stop Mode Only

To halt the memory controller in Stop mode, use the following procedure:

1. Before enabling the DDR memory controller, set the STOPCTL[DDRCK] bits to 10 to shut down the DDR clock in Stop mode.
2. Set up the DDR memory controller as needed, including self refresh and dynamic power management.
3. Enable the DDR controller.
4. Set the HLTREQ[DDRHRQ] bit to request a halt from the memory controller.
5. Wait for an acknowledge by polling the HLTACK[DDRHAK] bit. The memory controller asserts this bit when all accesses have completed.
6. Execute a **stop** instruction from the SC1400 core.

To restart the memory controller and exit from Stop mode, use any valid mechanism for exiting the SC1400 Stop mode and automatically re-enable the clock to the module (see **Section 11.4.5**, *Exit from Stop Mode*, on page 11-22).

11.4.4.3 Complete Halt of the Ethernet MAC

To halt the Ethernet MAC completely, use the following procedure:

1. Set the Ethernet MAC TCTL[GTS] bit to request a graceful stop of transmit data.
2. Poll the Ethernet MAC IEVENT[GRA] bit until this bit is set. The Ethernet MAC asserts this bit when all transmits in progress have completed. Alternatively, use the Ethernet MAC summary interrupt, which requires that the IMASK[GRAEN] bit be set.
3. Shut down the MII bridge by clearing the MIIENR[MIIEN] bit.
4. Optionally, to empty values still in the ENET Rx buffer more rapidly, raise the priority of the AMENT master at all slave ports in the crossbar that allow transfers from the AMENT bus.
5. Use a GP timer or software delay loop to wait an appropriate number of clocks to ensure that all the values in the ENET Rx buffer (up to 128 32-bit values) successfully transfer out to their appropriate destinations. The value of this count depends on the crossbar arbitration technique (fixed-priority versus round-robin), the programmed priority of the AMENT master at each crossbar slave port, as well as accesses from other AHB masters to slave ports with higher priority than the AMENT bus. A safe value is 10 ms.
6. Clear the Ethernet MAC ECTL[ETHEN] bit to shut down the module. Reception immediately stops, and transmission stops after a bad CRC is appended to any frame being transmitted.

7. Optionally set the HLTREQ[ENETCD] bit to shut down the system clock to the module.

Restart the Ethernet MAC as follows:

1. Clear the HLTREQ[ENETCD] bit to re-enables the system clock to the module.
2. Set the Ethernet MAC ECTL[ETHEREN] bit to re-enables the module.
3. Clear the TCTL[GTS] bit to remove the graceful stop request.

11.4.4.4 Complete Halt of the HDI16

To halt the HDI16 completely, use the following procedure:

1. If the HDI16 is serviced by the MSC711x DMA controller rather than by polling or interrupts, optionally mask new DMA requests from the HDI16, HDI16 TX and HDI16 RX, in the DMA DMAERQ register.
2. Optionally, poll the HDI16 HSR[HTFE] and HSR[HRFNE] bits to ensure that the FIFOs are empty. Service the HDI16 as appropriate until these bits are cleared. Omitting this step can result in transmission or reception of invalid data.
3. Clear the HDI16 HPCR[HEN] bit to shut down the module.
4. Set the HLTREQ[HDICD] bit to shut down the system clock to the module for lowest power consumption.

To restart the HDI16, use the following procedure:

1. Clear the HLTREQ[HDICD] bit to re-enable the system clock to the module.
2. If the on-device DMA controller services the HDI16, re-enable requests in the DMAERQ register.
3. Set the HDI16 HPCR[HEN] bit to re-enable the module.

11.4.4.5 Complete Halt of a TDM

To halt the TDM completely, use the following procedure:

1. Follow the shutdown procedure in **Section 19.6**, *Software Programming Sequence*, on page 19-23. To verify that the TDM has shut down correctly, poll its status bits (TDMxRSR[RENS] and TDMxTSR[TENS]).
2. Set the HLTREQ[TDMxCD] bit to shut down the system clock to the module.

To restart the TDM, use the following procedure:

1. Clear the HLTREQ[TDMxCD] bit to re-enable the system clock to the module.
2. Follow the TDM start-up procedure in **Section 19.6**, *Software Programming Sequence*, on page 19-23.

11.4.4.6 Complete Halt of the UART

To halt the UART completely, use the following procedure:

1. Optionally, poll the UART' LSR[TEMT] bit to ensure that the UART is not transmitting data. Omitting this step can result in transmission of invalid data.
2. Set the HLTREQ[UARTCD] bit to shut down the system clock to the module. Asserting this bit during transmission or reception can result in invalid data.

To restart this block, clear the HLTREQ[UARTCD] bit to re-enable the system clock to the module.

11.4.4.7 Complete Halt of the I^2C

To halt the I^2C module completely, use the following procedure:

1. Optionally, poll the I^2C I2SR[ICF] to ensure that no transfer is in progress. Omitting this step can result in transmission or reception of invalid data.
2. Clear the I2CTLR[IEN] bit to shut down the module.
3. Set the HLTREQ[I2CCD] bit to shut down the system clock to the module.

To restart this the I2C module, use the following procedure:

1. Clear the HLTREQ[I2CCD] bit to re-enable the system clock to the module.
2. Set the I2CTLR[IEN] bit to re-enable the module.

11.4.4.8 Shutting Down One Timer in a Timer Module

To shut down one of the timers in a timer module, clear the TMRxCTL[CM] bits to 000.

11.4.4.9 Shutting Down a Timer Module

To halt the timer module completely, use the following procedure:

1. Clear the TMRxCTL[CM] bits to 000, for each timer in the module.
2. Set the HLTREQ[TMRxCD] bit for the appropriate module to shut down the system clock to the module.

To restart this the timer module:

1. Clear the HLTREQ[TMRxCD] bit for the appropriate module to re-enable the system clock to the module.
2. Re-enable the individual timers as appropriate.

11.4.4.10 Selecting the Input Clock as the Source for the Timer Modules

For very low power operation in Stop mode, the IPBus clock can be shut off via the STDIS2 bit. The timer modules can still operate from the device input clock or scaled input clock and use it to exit Stop mode after a predetermined delay. The procedure is as follows:

1. Complete any remaining accesses required to registers in any of the two timer modules and then set the input clock in the CLKCTL[TMUX] bits. The programming models of all timer modules are not accessible.
2. Set the desired behavior of the clock in Stop mode via the STOPCTL[STDIS4] bit.
3. Execute the SC1400 **stop** instruction when Stop mode desired.
4. Change the timer clock back to the IPBus clock for register access.

When the input or scaled input clock is selected as the timer clock, the timer module registers become inaccessible. When there is a switch from the input clock or scaled input clock to the IPBus clock as the source for the timer clock, accesses to the registers or functionality of the timer module are permitted only after four clock periods of the original clock source.

11.4.5 Exit from Stop Mode

Two sets of operations cause the MSC711x to exit from Stop mode.

- Basic Exit Operations
- STOPCTL Register-Enabled Exit Operations
  - Direct Exit Operations
  - Event Port Multiplexor 0 Exit Operations

11.4.5.1 Basic Exit Operations

The Basic Exit operations do not need to be enabled through the STOPCTL register. Each of the following Basic Exit operations cause the MSC711x to exit from Stop mode.

- Power-on reset, any hard reset, soft reset
- Debug request via a JTAG command
- Debug request via assertion on the DBREQ/EE0 pin, when programmed as DBREQ
- Software watchdog timer configured to generate a non-maskable interrupt on a time-out:
  - The watchdog timer must receive a clock when the MSC711x is in Stop mode.
  - When programmed for generating a reset, the watchdog timer also brings the MSC711x out of Stop mode.

— The watchdog time must not be configured to pause in Stop mode.

11.4.5.2 STOPCTL Register-Enabled Exit Operations

In addition to the Basic Exit operations, the following operations can be used to exit the MSC711x from Stop mode by programming the associated enable bits of the the STOPCTL register. Direct Exit operations cause exit from Stop mode independent of the Event Port, while Event Port Mulitiplexor 0 Exit operations require appropriate Event Port configuration.

11.4.5.2.1 Direct Exit Operations

- Assertion of the EVNT4 pin:

  Does not require HLTREQ[ITCCD] or STOPCTL[STPDIS2] to be cleared.
- Any chip-level non-maskable interrupt:
  - Requires that the ECore clock continue in Stop mode, which is accomplished by ensuring that both HLTREQ[EIRQHR] (see **page 11-28**) and STOPCTL[STDIS2] (see **page 11-26**) are cleared.
  - Requires clocking to the GPIO if desired on the $\overline{\text{NMI}}$ pin.
- Timer overflow detection (channel 0 in timer module A only). Timer module A must receive a clock in Stop mode.
- Timer compare detection (channel 1 in timer module A only). Timer module A must receive a clock in Stop mode.

11.4.5.2.2 Event Port Multiplexor 0 Exit Operations

When the Event Port Multiplexer 0 event is enabled to cause Stop mode exit via setting STOPCTL[WMX0], the following input sources can cause a Stop mode exit.

- *Any Ethernet MAC interrupts*. Requires the FEC to receive a running clock in Stop mode by ensuring both HLTREQ[ENETCD] and STOPCTL[STPDIS2] are cleared.
- *EVNT4 signal* . Requires EVNT4 synchronization clock to run in Stop mode by ensuring both HLTREQ[ITCCD] and STOPCTL[STPDIS2] are cleared.
- *EVNT3 signal*. Requires EVNT3 synchronization clock to run in Stop mode by ensuring both HLTREQ[ITCCD] and STOPCTL[STPDIS2] are cleared.
- *Timer A TOUT[3–0] signals.* Requires Timer module A to receive a running clock in Stop mode.
- *DMA sources*. Requires the DMA to receive a running clock in Stop mode by ensuring that STOPCTL[STPDIS2] is cleared. Note that if a hardware initiated DMA request (for example, from the HDI or TDM) is used then the request initiator must receive a running clock in Stop mode.
- *TDM0 or TDM1 Rx interrupts*. For TDM0 and TDM1 Rx interrupts, TDM0 and TDM1 are required to receive a running clock in Stop mode by ensuring that

HLTREQ[TDM0CD] and HLTREQ[TDM1CD], respectively, are cleared. Additionally, STOPCTL[STPDIS2] must be cleared.

- *NMICHP*. The arbitrated non-maskable interrupt received by the SC1400 core from the Interrupt Controller. Requires the Interrupt Controller to receive a running clock in Stop mode by ensuring both HLTREQ[ITCCD] and STOPCTL[STPDIS2] are cleared.

Note: When using Event Port Multiplexor 0 to exit Stop mode, the following rules apply for Event Port Multiplex 0:

- Combine bits must be set for an OR function.
- Enable bits must be set as always enabled.
- Invert capability can be used.
- Clocks must be enabled (STOPCTL[STPDIS2] cleared) for using the event port to exit Stop mode except for when a timer is selected as an input source. For this exception, the Timer module must receive a running clock in Stop mode.

11.5 Clock Programming Model

The MSC711x clock registers include clock configuration registers, stop configuration registers, and reset configuration registers. The value of the base address for this register file, CLK_BASE, is in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files,* on page 5-4. The clock configuration registers are as follows:

- Clock Control Register (CLKCTL), **page 11-24**.
- Stop Mode Control Register (STOPCTL), **page 11-26**.
- Halt Request Register (HLTREQ), **page 11-28**.
- Halt Acknowledge Status Register (HLTACK), **page 11-29**.

CLKCTL Clock Control Register CLK_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TMUX		—	RNG	CLKO		LCK	LSLK	PLLMLTF							
TYPE	R/W		R	R/W			R	R/W								
RESET	0	0	0	0	0	0	0	0	0x0							

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—		PLLDVF						PLLEN	RSTRT	CKSEL		—			
TYPE	R		R/W										R			
RESET	0	0	0x0						0	0	0	0	0	0	0	0

CLKCTL is the register for configuring the clock module. There are restrictions on the frequencies permitted at the input of the multiplication portion of the PLL. Specifically, the frequency at this node must fall within a particular range of frequencies. Restrictions on the minimum and maximum frequencies at the output of the multiplication block can affect the values chosen for the PLLDVF and PLLMLTF fields. Refer to the device data sheet for details.

Table 11-8. CLKCTL Bit Descriptions

Name	Reset	Description	Settings
TMUX 31–30	0x0	**Timer Clock Multiplex** Selects the clock source for the timer clock. The timer module registers are not accessible when the input clock or scaled input clock is selected. See **Section 11.4.4.10**, *Selecting the Input Clock as the Source for the Timer Modules,* on page 11-22. The 2^6 divider is enabled only when selected. Otherwise, it is disabled to reduce power.	00 Disabled. 01 Input clock. 10 Input clock / 2^6. 11 APB clock.
— 29	0x0	Reserved. Write to zero for future compatibility.	
RNG 28	0	**PLL Frequency Range.** Divides the output of the PLL loop by two. When RNG is used, take care to ensure that F_{LOOP} meets the range specified in the device data sheet.	0 $F_{VCO} = F_{LOOP} / 2$. 1 $F_{VCO} = F_{LOOP}$.
CLKO 27–26	0x0	**Clock Out MUX** Selects the clock source for clock out.	00 Disabled. 01 IPBus clock/2. 10 Timer clock/2. 11 Reserved.
LCK 25	0	**PLL Lock.** Status bit indicating the state of PLL lock.	0 PLL is out of lock. 1 PLL is in lock.
LSLK 24	0	**PLL Loss of Lock.** Sticky bit indicating whether the PLL has lost lock after successfully acquiring lock. Writing a 1 to this bit clears the bit. The value of this bit is an input to the event port.	0 PLL has not lost lock. 1 PLL has lost lock.
PLLMLTF 23–16	0x0	**PLL Multiplication Factor** Specifies the multiplication factor for the PLL. The multiplication factor is PLLMLTF + 1. See the note on allowed frequency ranges immediately following this table.	
— 15–14	0x0	Reserved. Write to zero for future compatibility.	
PLLDVF 13–8	0x0	**PLL Input Division Factor** Specifies the PLL division factor. The division takes place before the multiplication. The division factor is PLLDVF + 1.	
PLLEN 7	0	**PLL Enable** Enables or fully powers down the PLL circuitry.	0 PLL disabled. 1 PLL enabled.

Table 11-8. CLKCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
RSTRT 6	0	**PLL Restart** Restarts the PLL when the multiplication or division factor is modified. This bit must be written with a 1 when either of these fields is modified. It is written with a 0 in all other cases. This bit always reads as a 0.	0 No effect. 1 Restarts the PLL. Required when the multiplication or division factor changes.
CKSEL 5–4	0x0	**Core Clock Select** Selects the clock source for the ECore clock. When 00 is selected, the clock immediately switches to the bypass clock. When any other mode is selected, the clock controller uses the bypass clock until the PLL is locked. See **Section 11.3.4.2**, *Bypass Clock,* on page 11-10).	00 Input clock (bypass clock). 01 Fvco / 2 (PLL clock). 10 Reserved. 11 FVCO (PLL clock).
— 3–0	0x0	Reserved. Write to zero for future compatibility.	

STOPCTL Stop Mode Control Register CLK_BASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—											WMX0	WTA1	WTA0	WEV4	WNMI
TYPE	R											R/W				
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PLLST		DDRCK		—									STDIS5	STDIS4	STDIS2
TYPE	R/W				R									R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

STOPCTL programs clock module operation during SC1400 Stop mode. It can also disable the DDR clock.

Table 11-9. STOPCTL Bit Descriptions

Name	Reset	Description	Settings
— 31–21	0x0	Reserved. Write to zero for future compatibility.	
WMX0 19	0	**Wake Up on Event Multiplex 0** Enables wake-up on the event selected in Event Multiplex 0.	0 Disable Stop wake-up. 1 Enable Stop wake-up.
WTA1 19	0	**Wake Up on Channel 1 of Timer Module A** Enables wake-up on a compare detection for this channel.	0 Disable Stop wake-up. 1 Enable Stop wake-up.

Table 11-9. STOPCTL Bit Descriptions

Name	Reset	Description	Settings
WTA0 18	0	**Wake Up on Channel 0 of Timer Module A** Enables wake-up on an overflow detection for this channel.	0 Disable Stop wake-up. 1 Enable Stop wake-up.
WEV4 17	0	**Wake Up on EVNT4 Pin** For enabling wake-up on any assertion of this pin.	0 Disable Stop wake-up. 1 Enable Stop wake-up.
WNMI 16	0	**Wake Up on Non-Maskable Interrupt Request** For enabling wake-up on any system-level non-maskable interrupt. The non-maskable sources can be seen in the NMIPR register (see **Section 12.6**, *Interrupt Controller Programming Model,* on page 12-12)	0 Disable Stop wake-up. 1 Enable Stop wake-up.
PLLSTP[1] 15–14	0x0	**PLL Stop Modes** Selects how PLL is affected when the SC1400 core enters Stop mode.	00 PLL unaffected by Stop mode. 01 Power down PLL digital only in Stop mode (input divider remains active). 10 Power down entire PLL in Stop mode (including input divider). 11 Reserved.
DDRCK 13–12	0x0	**DDR Clock Modes** Selects operation of the DDR clock and how it is affected when the SC1400 core enters Stop mode. See **Section 11.4.4.1**, *Complete Halt of the DDR Memory Controller* and **Section 11.4.4.2**, *Halt of the DDR Memory Controller in Stop Mode Only*.	00 DDR clock enabled. 01 DDR clock disabled. 10 DDR clock disabled in Stop mode. 11 Reserved.
— 11–3	0x0	Reserved. Write to zero for future compatibility.	
STDIS5 2	0	**Stop Disable 5** Disables the watchdog clock in SC1400 Stop mode.	0 Watchdog clock runs in Stop mode. 1 Watchdog clock disabled in Stop mode.
STDIS4 1	0	**Stop Disable 4** Disables CLKIN to the timer multiplex in SC1400 Stop mode.	0 Clock runs in Stop mode. 1 Clock disabled in Stop mode.
STDIS2[1] 0	0	**Stop Disable 2** Disables the AHB, IPBus, ECore clock, and APB clocks in SC1400 Stop mode.	0 Clocks run in Stop mode. 1 Clocks disabled in Stop mode.

Note: There is no Stop Disable 3 because the Core clock is automatically gated off by the execution of a Stop or Wait instruction. There is no Stop Disable 1 because the function is handled by the PLLSTP bits.

HLTREQ Halt Request Register CLK_BASE + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—		EIRQHR	TDM2CD	TDM1CD	TDM0CD	ENETCD	EPCD	HDICD	TMBCD	TMACD	DDRHQ	ITCCD	—	I2CCD	UARTCD
TYPE	R		R/W													
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HLTREQ allows individual shutdown of peripherals using one of two different techniques, disabling the clock to the peripheral or using a peripheral's built-in request for halt capability. To power down a specific peripheral, set the appropriate bit in the HLTREQ register. You should clear this bit when normal operation is desired.

Table 11-10. HLTREQ Bit Descriptions

Name	Reset	Description	Settings
— 31–14	0	Reserved. Write to zero for future compatibility.	
EIRQHR 13	0	**External Interrupt Pin Halt Request** A global disable of all external $\overline{\text{IRQ}}$ pins in the GPIO block for power reduction.	0 External $\overline{\text{IRQ}}$ pins are enabled. 1 External $\overline{\text{IRQ}}$ pins are disabled.
TDM2CD 12	0	**TDM 2 Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
TDM1CD 11	0	**TDM 1 Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
TDM0CD 10	0	**TDM 0 Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
ENETCD 9	0	**Ethernet MAC Clock Disable** Gates clock to the module for power reduction. See **Section 11.4.4.3**, *Complete Halt of the Ethernet MAC,* on page 11-19	0 Clocks are enabled. 1 Clocks are disabled.
EPCD 8	0	**Event Port Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
HDICD 7	0	**HDI16 Clock Disable** Gates clock to the module for power reduction. When this bit is set, it disables both the fast and slow clocks to this block.	0 Clocks are enabled. 1 Clocks are disabled.
TMRBCD 6	0	**Timer Module B Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.

Table 11-10. HLTREQ Bit Descriptions (Continued)

Name	Reset	Description	Settings
TMRACD 5	0	**Timer Module A Clock Disable** Gates clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
DDRHRQ 13	0	**DDR Halt Request** Requests a DDR halt, which uses the halt capability of the DDR controller.	0 No request for halt. 1 Request halt from module.
ITCCD 3	0	**Interrupt Controller Clock Disable** Gates the clock to the module for power reduction. This bit can affect wake-up from Stop mode during a wake-up from NMI. See **Section 11.4.5**, *Exit from Stop Mode*, on page 11-22. **Note:** This bit must be cleared if you are using an EVNT signal as an input.	0 Clocks are enabled. 1 Clocks are disabled.
— 2	0	Reserved. Write to zero for future compatibility.	
I2CCD 1	0	**I2C Clock Disable** Gates the clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.
UARTCD 0	0	**UART Clock Disable** Gates the clock to the module for power reduction.	0 Clocks are enabled. 1 Clocks are disabled.

Note: See **Section 11.4.4**, *Peripheral Subsystem Low Power Operation*, on page 11-18 for correct usage of these bits.

HLTACK Halt Acknowledge Status Register CLK_BASE + 0x18

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—											DDRHAK	—			
TYPE	R											R/W	R			
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HLTACK provides a halt acknowledge from the individual peripherals. When a halt is requested of a peripheral and the peripheral is ready to shut down, the appropriate bit in the HLTACK register is set. When the halt or clock disable is removed from the individual peripheral, its corresponding acknowledge bit is cleared.

Table 11-11. HLTACK Bit Descriptions

Name	Reset	Description	Settings
— 31–5	0	Reserved. Read returns zero.	
DDRHAK 4	0	**DDR Halt Acknowledge** Status bit indicating if the module is currently halted.	0 Module is not currently halted. 1 Module is currently halted.
— 3–0	0	Reserved. Read returns zero.	

Note: See **Section 11.4.4**, *Peripheral Subsystem Low Power Operation*, on page 11-18 for correct usage of these bits.

Interrupt Processing 12

The MSC711x interrupt system is optimized for real-time interrupt processing for up to 120 interrupt input channels. Eight programmable priority levels and vectored interrupt servicing speeds up the processing. MSC711x interrupts can arrive from many different sources:

- Interrupt pins.
- Peripherals such as TDMs, timers, and the DMA controller; one peripheral can generate more than one interrupt request.
- Non-maskable error conditions such as bus time-outs and illegal accesses.

The MSC711x interrupt controller works with the SC1400 core to handle prioritized, nestable, vectored interrupts. Most sources are processed as level-sensitive, but edge triggering is used where appropriate.

The interrupt controller register file contains registers programmed by software to handle different priority levels for the different sources.

12.1 Interrupt Controller Architecture

The MSC711x interrupt controller collects maskable and non-maskable interrupt requests from all device resources (with the exception of SC1400 core interrupt requests), arbitrates among all, and consolidates them to two interrupt requests signals that are sent to the SC1400 Maskable Interrupt Request and Auto-NMI Request. **Figure 12-2** shows the a block diagram of the interrupt controller.

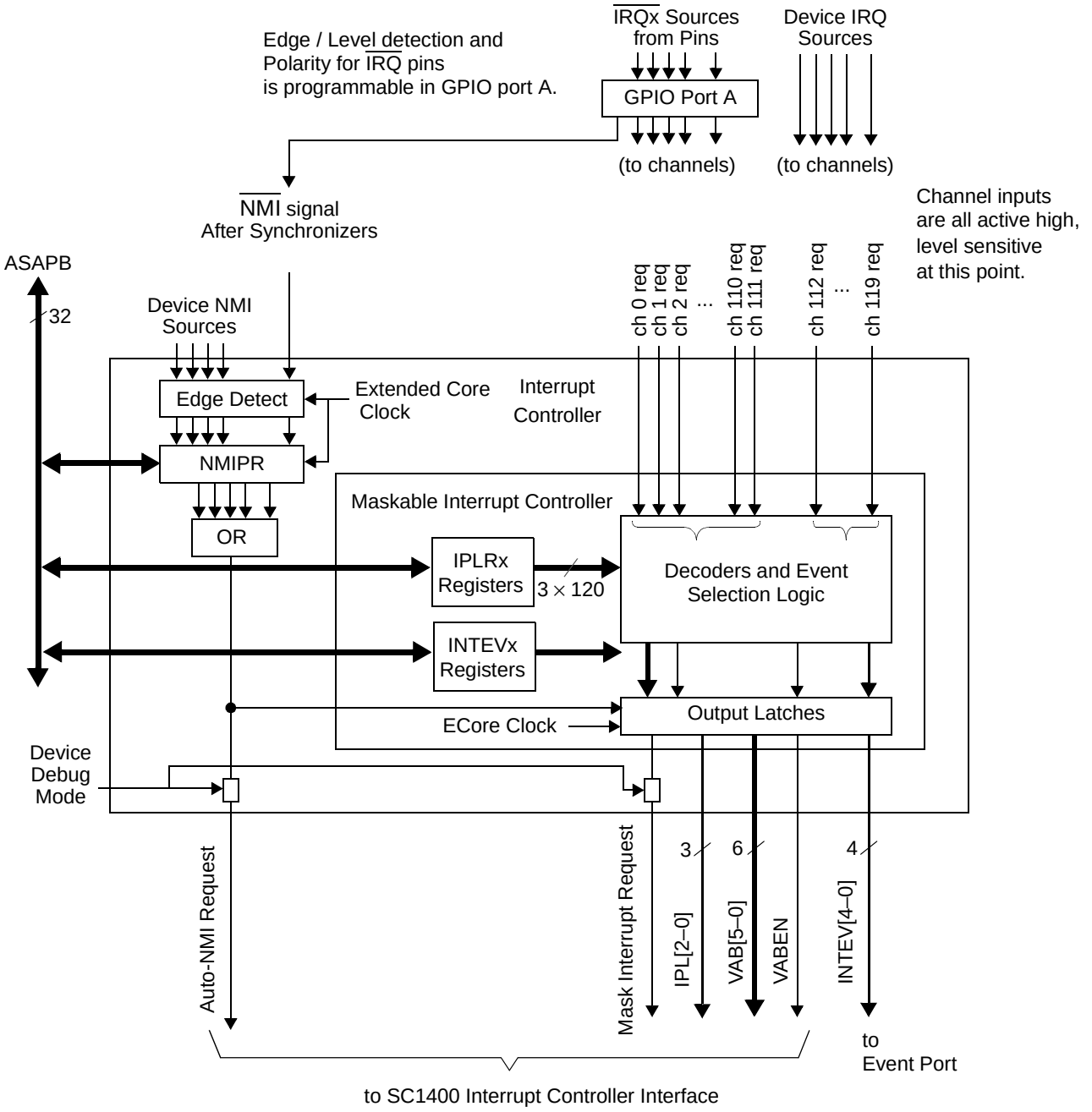


Figure 12-1. Interrupt Controller Block Diagram

12.1.1 IRQ Pins Preprocessed in GPIO Port A

The external interrupt request signals received on the IRQ pins are preprocessed in the GPIO port A interrupt logic. Interrupts can be detected as level-sensitive active high/low or edge-triggered on the falling/rising edge. Edge-triggered and level-sensitive signals are synchronized at port A via the ECore clock to reduce interrupt latency. Interrupts at these pins are individually enabled or disabled via control bits within the GPIO. The value of each interrupt pin can be read before or after this masking. The pre-processed signals are then sent to the Maskable Interrupt Controller.

For details, see **Section 24.4**, *Interrupts*, on page 24-11. A block diagram of the preprocessing logic is shown in **Figure 24-2**.

12.1.2 $\overline{\text{NMI}}$ Interrupts

The signal from the $\overline{\text{NMI}}$ pin is first passed through the logic in GPIO port A where it is synchronized. GPAIEN[18] must be set to enable this interrupt. You must then explicitly set up the $\overline{\text{NMI}}$ in the GPIO for level-sensitive (synchronized) operation. The output of the synchronizer is sent to the interrupt controller where it is edge detected, using the logic for all device-level non-maskable interrupt requests.

The $\overline{\text{NMI}}$ non-maskable interrupt request signal received on the $\overline{\text{NMI}}$ pin is edge triggered on the falling edge. It is synchronized via the core clock. This request is cleared by writing a value of 1 to the appropriate bit in the Non-Maskable Interrupt Pending Register (NMIPR) (**page 12-12**).

12.1.3 Operation in Debug Mode

When the SC1400 core enters Debug mode, maskable and non-maskable interrupt requests are optionally masked using MIPR[DDBG] (**page 12-15**) until the SC1400 core exits Debug mode.

12.2 Interrupt Arbitration

All maskable interrupt sources are programmable at any of the eight priority levels (**Table 12-1**). System critical maskable sources have their own dedicated interrupt vector. Other resources share an interrupt vector with one other source, and the least critical sources share a single vector.

The maskable interrupt sources are generated by the SC1400 core or the SC1400 OCE10 port, or they are generated by system-level resources that share the default autovector, Auto-NMI. The maskable sources can have their own dedicated interrupt vector (channels 0, 2, 4, ..., 60, 62), share an interrupt vector (Channels 64–111), or share the default autovector, Auto-Maskable (channels 112–119). The maskable sources are separated into different types because the SC1400 core supports only 64 interrupt vectors. Because there are more than 64 interrupt sources on an MSC711x device, some sources must share a vector. The MSC711x interrupt controller has 120 channels for maskable sources, which are then combined to fit within the 64 available vectors. The non-maskable sources are handled differently within the controller.

Table 12-1. Programming the Priorities of Maskable Interrupt Sources

IPL	Enabled?	Request Priority Level	Comments
000	No	—	Interrupt requests *disabled* from this interrupt source.
001	Yes	1	Interrupts enabled from this source at ***l****owest priority*.
010	Yes	2	Interrupts enabled from this source at next highest priority.
011	Yes	3	Interrupts enabled from this source at next highest priority.

Table 12-1. Programming the Priorities of Maskable Interrupt Sources (Continued)

IPL	Enabled?	Request Priority Level	Comments
100	Yes	4	Interrupts enabled from this source at next highest priority.
101	Yes	5	Interrupts enabled from this source at next highest priority.
110	Yes	6	Interrupts enabled from this source at next highest priority.
111	Yes	7	Interrupts enabled from this source at *highest maskable priority.*

Interrupt input channels are individually enabled or disabled by programming the maskable priority for each channel. All maskable interrupts can be simultaneously disabled with the SC1400 DI instruction, which sets the DI bit in the SC1400 core Status Register (SR). No interrupts are serviced after the DI instruction executes. As a result, the code following the DI instruction does not need to take into account any possible pipeline effects caused by interrupts. The EI instruction re-enables all unmasked interrupts and clears the DI bit in the SR.

Non-maskable interrupt sources are arbitrated with a higher priority than all maskable sources and are selected independently of the priority level in the SC1400 core Status Register. Therefore, the non-maskable exceptions are *not* affected by the DI instruction. Non-maskable sources are disabled when the non-maskable interrupt disable (NMID) bit is set in the Exception and Mode Register (EMR) within the SC1400 core.

All interrupt requests arrive at the interrupt controller already synchronized and are arbitrated there. Outputs from the interrupt controller are then latched and sent to the SC1400 core. Interrupts are arbitrated with the order of priorities shown in **Table 12-2**.

Table 12-2. Interrupt Arbitration Priorities

Maskable/Non-Maskable Sources	Rule
All non-maskable sources	There is arbitration between non-maskable sources only if more than one non-maskable request occurs simultaneously, in the following order of priorities: 1. TRAP 2. ILLEGAL 3. Debug port 4. Overflow 5. Device-level non-maskable sources
All maskable sources at IPL7	• If only one source requests an interrupt, it is selected. • If more than once source requests an interrupt, the source with the lowest interrupt input channel number is selected.
All maskable sources at IPL6	
All maskable sources at IPL5	
All maskable sources at IPL4	
All maskable sources at IPL3	
All maskable sources at IPL2	
All maskable sources at IPL1	

Only interrupts with a request IPL greater than the core IPL (in the SC1400 core Status Register) are serviced. Maskable sources programmed at IPL 0 are disabled.

The interrupt channel inputs are directly arbitrated without latching, each is compared against its associated programmed priority level, and the source with the highest arbitrated priority is selected. The results are then latched in the output latches, as follows:

- *Mask*. An interrupt request signal indicates to the SC1400 core that a maskable interrupt is pending.
- VAB provides the vector offset for the source, which is added to the base address (VBR) when the SC1400 core services the interrupt.
- VABEN indicates that the value on the VAB is valid.
- IPL provides the priority level of the highest-priority maskable interrupt that is pending.

12.3 Interrupt Vectors

The vector base address resides in the VBA register of the SC1400 core. The 64 vector addresses of the SC1400 core are allocated as follows:

- 56 for maskable sources (input channels 0 through 111). The interrupt controller drives the vector address (VAB field).
- 1 for maskable sources that use the SC1400 interrupt autovector capability (input channels 112 through 119 and the device-level NMI). The SC1400 core provides the address.
- 1 for device-level non-maskable sources (uses autovector for non-maskable interrupts).
- 6 for core-generated non-maskable sources.

Figure 12-2 shows the address that the SC1400 core drives onto the PAB.

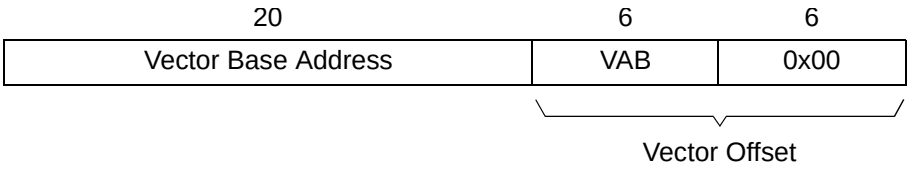


Figure 12-2. Forming the Interrupt Vector Address

The vector base address resides in the VBA register of the SC1400 core. The vector address (VAB field) is driven by the interrupt controller for channels 0–111. Separate autovectors are provided for the maskable and non-maskable channels. Each interrupt vector is allocated and separated by 2^6 = 64 bytes.

12.4 Interrupt Sources

Separate autovectors are provided for the maskable and non-maskable channels. Each interrupt vector is allocated and separated by 64 bytes. Non-maskable sources from the SC1400 core are arbitrated in the core and thus do not use any interrupt input channels. The SC1400 core provides the address offsets for these sources—except for the auto-NMI sources and auto-maskable sources. All non-maskable sources generated by device-level resources use the auto-NMI source. These are ORed together into a single NMI channel, and all share a common interrupt vector provided by the SC1400 core with an offset of 0x180. The auto-maskable sources are arbitrated by the interrupt controller and use channels 112 through 119. These sources share a common interrupt vector provided by the SC1400 with an offset of 0x1C0.

Interrupt input channels 1, 3, 5, 7, ..., 59, 61, and 63 are not used because each group of two channels shares one interrupt vector. For higher-priority interrupt sources, only one of these two channels is used so that the interrupt source has its own dedicated vector. As a result, many of the 120 available interrupt channels are not used on MSC711x.

Table 12-3 lists the non-maskable interrupt sources from the core and **Table 12-4** lists the chip-level non-maskable interrupt sources.

Table 12-3. MSC711x Non-Maskable Interrupt Sources from the SC1400 Core

Interrupt Input Channel	Exception Address Offset	Source	Description
Non-Maskable Sources from the SC1400 Core			
—	0x000	TRAP	TRAP Instruction
—	0x040	—	Reserved
—	0x080	ILLEGAL	Illegal execution set, illegal instruction
—	0x0C0	Debug port	Debug exceptions generated by the OCE10 port
—	0x100	Overflow	DALU Overflow
—	0x140	—	Reserved
—	0x180	Auto-NMI	Default vector for NMI sources (autovector). See non-maskable sources from device-level resources in the discussion that follows.
—	0x1C0	Auto-maskable	Default vector for maskable interrupt sources (autovector). See maskable sources from device-level resources (**Table 12-5**).

The non-maskable interrupt sources from the SC1400 core are defined in the documentation for the SC1400 core. Note that the DALU overflow non-maskable interrupt can be selectively enabled with the SC1400 core SR[OVE] bit. All non-maskable device-level requests are detected with edge triggering and are latched in the Non-Maskable Interrupt Pending register. They are then ORed together and sent as one signal to the SC1400 interrupt controller. This signal overrides the operation of the Maskable Interrupt Controller. The detection of a device-level non-maskable interrupt also sets DEVCFG[CNMI], which elevates the priority of the SC1400

core during instruction accesses, ensuring that it can service the NMI. Detection of a device-level non-maskable interrupt also clears the GPSCTL[XHRQ] bit in the extended core (**page 4-43**), forcing the crossbar switch out of a halt state.

Edge-triggered interrupt requests are reset by writing a 1 to the bit associated with a particular source, indicating to the interrupt controller that the SC1400 core has acknowledged the corresponding edge-triggered source. The edge detection circuitry is then re-enabled for this source.

Table 12-4. MSC711x Device-Level Non-Maskable Interrupt Sources

Interrupt Input Channel	Exception Address Offset	Source	Description
Non-Maskable Sources from Device-Level Resources			
NMI_ch	0x180	NMI	$\overline{\text{NMI}}$ signal
NMI_ch	0x180	WDTEX	Watchdog timer expiration
NMI_ch	0x180	HDI_NMI	HDI16 Host NMI
NMI_ch	0x180	MISAL_P	Extended Core misaligned program access
NMI_ch	0x180	MISAL_D	Extended Core misaligned data access
NMI_ch	0x180	MISAL_AMEC	Misaligned access on AMEC bus
NMI_ch	0x180	MISAL_AMIC	Misaligned access on AMIC bus
NMI_ch	0x180	MISAL_AMDMA	Misaligned access on AMDMA bus
NMI_ch	0x180	MISAL_AMENT	Misaligned access on AMENT bus
NMI_ch	0x180	BE_AMEC	Bus error detected on AMEC bus
NMI_ch	0x180	BE_AMIC	Bus error detected on AMIC bus
NMI_ch	0x180	BE_AMDMA	Bus error detected on AMDMA bus
NMI_ch	0x180	BE_AMENT	Bus error detected on AMENT bus
NMI_ch	0x180	BT_ASM1	Bus time-out on ASM1 bus
NMI_ch	0x180	BT_ASM2	Bus time-out on ASM2 bus
NMI_ch	0x180	BT_ASEMI	Bus time-out on ASEMI bus
NMI_ch	0x180	BT_ASTH	Bus time-out on ASTH bus
NMI_ch	0x180	BT_ASAPB	Bus time-out on ASAPB bus
NMI_ch	0x180	BT_ASSB	Bus time-out on ASSB bus
NMI_ch	0x180	AORP_AMIC	Program address out of range on AMIC bus
NMI_ch	0x180	AORX_AMEC	Data address out of range on AMEC bus
NMI_ch	0x180	AORX_AMDMA	DMA address out of range on AMDMA bus
NMI_ch	0x180	AORX_AMENT	Ethernet MAC address out of range on AMENT bus (only on devices with an Ethernet MAC)
NMI_ch	0x180	AORP_E	Program address out of range, access at system level
NMI_ch	0x180	AORX_E	Data address out of range, access at system level

Table 12-4. MSC711x Device-Level Non-Maskable Interrupt Sources (Continued)

Interrupt Input Channel	Exception Address Offset	Source	Description
NMI_ch	0x180	ROM_WR	Write to ROM
NMI_ch	0x180	ISZ_PF	Access of illegal size to peripherals
NMI_ch	0x180	DDR_ER	DDR error interrupt
NMI_ch	0x180	CADNMI	Core address detection (non-maskable)
NMI_ch	0x180	PADNMI	Peripheral address detection (non-maskable)

All internal maskable interrupt requests are tied to the 120 channels of the maskable interrupt controller, arriving as active high, level-sensitive values (although the pins may have already been detected as edge triggered within the GPIO), and are connected to the 120 available input channels as mapped in **Table 12-5**.

Table 12-5. MSC711x Maskable Interrupt Sources

interrupt input channel	Exception Address Offset	Source	Description
Maskable Sources — from Chip Level Resources			
0	0x200	Tx0	TDM 0 transmit
2	0x240	Rx0	TDM 0 receive
4	0x280	Tx1	TDM 1 transmit
6	0x2C0	Rx1	TDM 1 receive
8–14	0x300–0x3C0	—	Reserved
16	0x400	IRQ0	$\overline{\text{IRQ0}}$ signal
18	0x440	IRQ1	$\overline{\text{IRQ1}}$ signal
20	0x480	IRQ2	$\overline{\text{IRQ2}}$ signal
22	0x4C0	IRQ3	$\overline{\text{IRQ3}}$ signal
24	0x500	DMA0	DMA channel 0
26	0x540	DMA1	DMA channel 1
28	0x580	DMA2	DMA channel 2
30	0x5C0	DMA3	DMA channel 3
32	0x600	DMA4	DMA channel 4
34	0x640	DMA5	DMA channel 5
36	0x680	DMA6	DMA channel 6
38	0x6C0	DMA7	DMA channel 7
40	0x700	TMR_A0	Timer A interrupt, channel 0

Table 12-5. MSC711x Maskable Interrupt Sources (Continued)

interrupt input channel	Exception Address Offset	Source	Description
42	0x740	TMR_A1	Timer A interrupt, channel 1
44	0x780	TMR_A2	Timer A interrupt, channel 2
46	0x7C0	TMR_A3	Timer A interrupt, channel 3
48–62	0x800–0x9C0	—	Reserved
64	0xA00	ENTRxF	Ethernet receive frame (only on devices with an Ethernet MAC)
65	0xA00	ENTTxF	Ethernet transmit frame (only on devices with an Ethernet MAC)
66	0xA40	ENTSMRY	Ethernet summary interrupt (only on devices with an Ethernet MAC)
67	0xA40	HDICMD	HDI16 host command vector
68	0xA80	HDIRxF	HDI16 receive data FIFO full
69	0xA80	HDIRxNE	HDI16 receive data not empty
70	0xAC0	HDITxE	HDI16 transmit data FIFO empty
71	0xAC0	HDITxNF	HDI16 transmit data FIFO not full
72	0xB00	Tx2	TDM 2 transmit (only on devices with TDM2)
73	0xB00	Rx2	TDM 2 receive (only on devices with TDM2)
74	0xB40	TDM0ERR	TDM 0 error interrupts
75	0xB40	TDM12ERR	TDM 1, 2 error interrupts
76	0xB80	EVINT0	Event port request 0
77	0xB80	IRQ4	$\overline{\text{IRQ4}}$ signal
78	0xBC0	EVINT1	Event port request 1
79	0xBC0	IRQ5	$\overline{\text{IRQ5}}$ signal
80	0xC00	IRQ6	$\overline{\text{IRQ6}}$ signal
81	0xC00	IRQ7	$\overline{\text{IRQ7}}$ signal
82	0xC40	$\overline{\text{IRQ8}}$, $\overline{\text{IRQ12}}$	$\overline{\text{IRQ8}}$ signal or $\overline{\text{IRQ12}}$ signal
83	0xC40	$\overline{\text{IRQ9}}$, $\overline{\text{IRQ13}}$	$\overline{\text{IRQ9}}$ signal or $\overline{\text{IRQ13}}$ signal
84	0xC80	$\overline{\text{IRQ10}}$, $\overline{\text{IRQ14}}$	$\overline{\text{IRQ10}}$ signal or $\overline{\text{IRQ14}}$ signal
85	0xC80	$\overline{\text{IRQ11}}$, $\overline{\text{IRQ15}}$	$\overline{\text{IRQ11}}$ signal or $\overline{\text{IRQ15}}$ signal
86	0xCC0	CADINT0	Core address detection 0
87	0xCC0	—	Reserved
88	0xD00	CADINT1	Core address detection 1
89	0xD00	—	Reserved
90	0xD40	DMA8	DMA channel 8
91	0xD40	DMA9	DMA channel 9

Table 12-5. MSC711x Maskable Interrupt Sources (Continued)

interrupt input channel	Exception Address Offset	Source	Description
92	0xD80	DMA10	DMA channel 10
93	0xD80	DMA11	DMA channel 11
94	0xDC0	DMA12	DMA channel 12
95	0xDC0	DMA13	DMA channel 13
96	0xE00	DMA14	DMA channel 14
97	0xE00	DMA15	DMA channel 15
98	0xE40	TMR_B0	Timer B interrupt, channel 0
99	0xE40	TMR_B1	Timer B interrupt, channel 1
100	0xE80	TMR_B2	Timer B interrupt, channel 2
101	0xE80	TMR_B3	Timer B interrupt, channel 3
102–109	0xEC0–0xF80	—	Reserved
110	0xFC0	I2C	I^2C interrupt
111	0xFC0	UART	UART Tx and Rx interrupt
112	0x1C0	—	Reserved. Uses SC1400 core autovector capability.
113	0x1C0	DMAREM	All Remaining DMA channels (16 through 31) use SC1400 core autovector capability
114	0x1C0	DMA_ERROR	DMA error. Uses SC1400 core autovector capability.
115	0x1C0	—	Reserved. Uses SC1400 core autovector capability.
116	0x1C0	EC_DUALWR	Extended core dual write, uses SC1400 core autovector capability
117	0x1C0	—	Reserved. Uses SC1400 core autovector capability.
118	0x1C0	IRQPINS	For $\overline{\text{IRQ16}}$ through $\overline{\text{IRQ26}}$ pins, uses SC1400 core autovector capability
119	0x1C0	—	Reserved. Uses SC1400 core autovector capability.

Note: Although two interrupt sources often share an interrupt vector, it is useful to place these sources on different interrupt channels because each channel can be individually disabled or enabled with its own priority level.

12.5 Interrupt Event Selection

The interrupt controller can send user-selected interrupt request signals to the event port for triggering. The interrupt event selection logic is shown in **Figure 12-3**. Interrupt event selection is performed as follows:

1. The user selects the desired maskable and non-maskable interrupt requests for event port triggering by programming the INTEV and INTEVNM registers.
2. When interrupt requests arrive, they are enabled via the corresponding register bit.
3. Each group of requests is ORed together and latched for triggering the event port.
4. The INTEV trigger, which must also be enabled in the event port EVINx register(s), performs the action programmed in the event port.

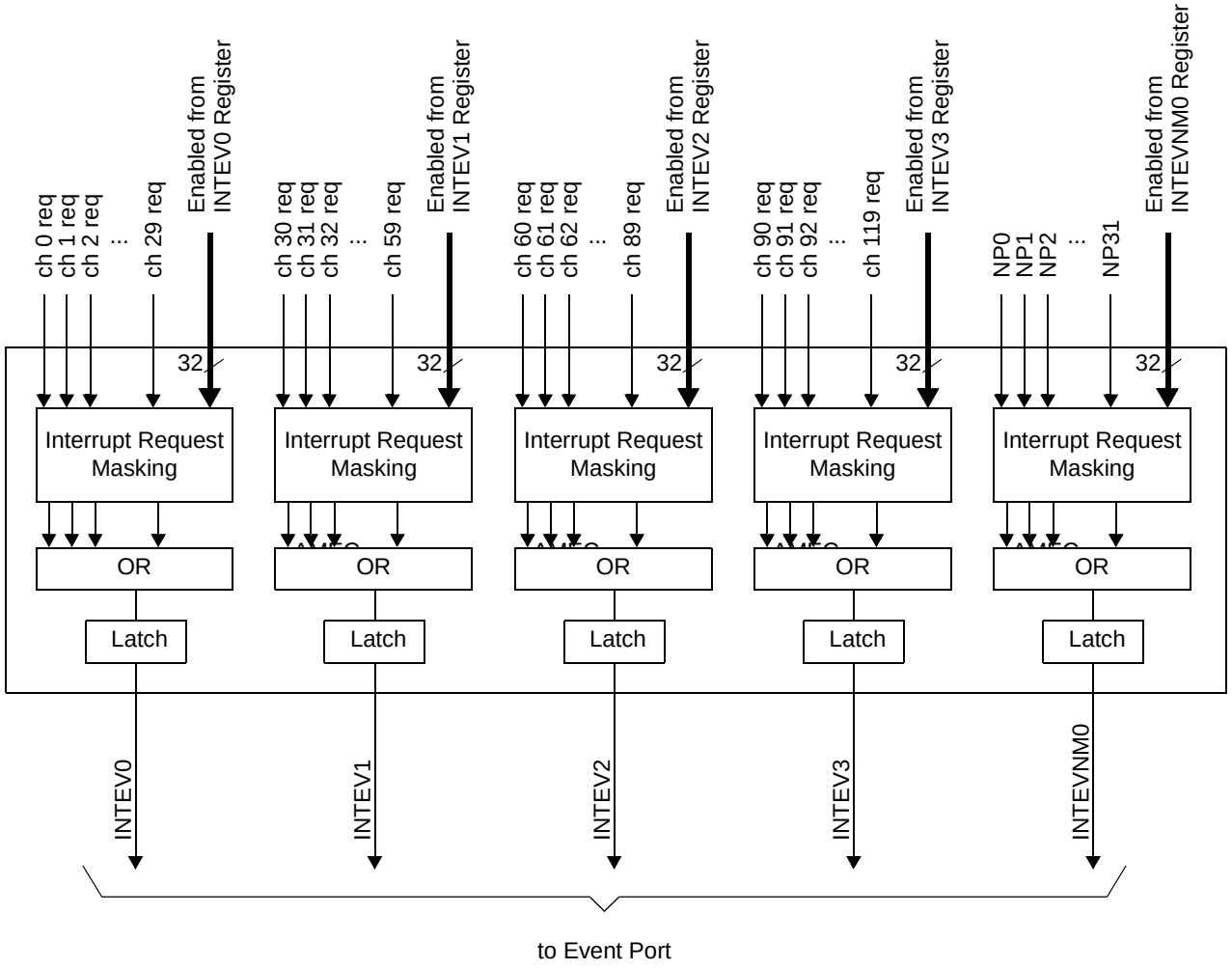


Figure 12-3. Interrupt Event Selection Logic

12.6 Interrupt Controller Programming Model

The interrupt controller registers reside in the ASAPB address space. These registers individually set the priority level for each of the 120 available interrupt channels. **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4 lists the value of the base address for the ICTL_BASE register file and the value of the base address for the IPL_BASE register file. The Vector Base Address Register (VBA) is discussed in the documentation for the SC1400 core and only briefly covered here. In addition, the interrupt controller registers are as follows:

- Non-Maskable Interrupt Pending Register (NMIPR), **page 12-12**.
- Maskable Interrupt Pending Register (MIPR), **page 12-15**.
- Interrupt Priority-Level Registers (IPLRx), **page 12-16**.
- Interrupt Event Selection Register (INTEVx),
- Non-maskable Interrupt Event Selection Register (INTEVNMx),

VBA Vector Base Address Register

VBA specifies the base address for the interrupt vector table. This register is located within the SC1400 core, not in the interrupt controller. At reset, the value of the 20-bit wide VBA Register is set to zero. The offset for each exception vector is predefined. There are 64 possible exception vector locations. The spacing between two exception vectors is 64 bytes (four full execution sets).

NMIPR Non-Maskable Interrupt Pending Register ICTL_BASE + 0x000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	NP30	NP29	NP28	—	NP26	NP25	NP24	NP23	NP22	NP21	NP20	NP19	NP18	NP17	NP16
TYPE	R			R/W												
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NP15	NP14	NP13	NP12	NP11	NP10	NP9	NP8	NP7	NP6	NP5	NP4	NP3	NP2	NP1	NP0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NMIPR resets edge-triggered interrupts from all pending device-level non-maskable interrupt sources:

- Each bit is set when its associated non-maskable interrupt request arrives.
- Each bit can only be cleared by writing a value of 1 to the bit.
- These bits cannot be set by writing to the register.

Table 12-6. NMIPR Bit Descriptions

Name	Reset	Description	Settings
— 31	0	Reserved. Write to zero for future compatibility.	
NP30 30	0	**Status of NMI Input 30, Core Address Detection (Non-Maskable)** Address out-of-range detected on one of the SC1400 buses.	0 No interrupt pending. 1 Interrupt pending.
NP29 29	0	**Status of NMI Input 30, Peripheral Address Detection (Non-Maskable)** Address out-of-range detected on one of the AHB-Lite master buses.	0 No interrupt pending. 1 Interrupt pending.
NP28 28	0	**Status of NMI Input 28** Misaligned data access.	0 No interrupt pending. 1 Interrupt Pending.
NP27 27	0	Reserved. Write to zero for future compatibility.	
NP26 26	0	**Status of NMI Input 26, DDR Error** Flags memory select errors from the DDR controller.	0 No interrupt pending. 1 Interrupt pending.
NP25 25	0	**Status of NMI Input 25** Misaligned P access.	0 No interrupt pending. 1 Interrupt pending.
NP24 24	0	**Status of NMI Input 24** Misaligned AMEC bus access.	0 No interrupt pending. 1 Interrupt pending.
NP23 23	0	**Status of NMI Input 23** Misaligned AMIC bus access.	0 No interrupt pending. 1 Interrupt pending.
NP22 22	0	**Status of NMI Input 22** Misaligned AMDMA bus access.	0 No interrupt pending. 1 Interrupt pending.
NP21 21	0	**Status of NMI Input 21** Misaligned AMENT bus access.	0 No interrupt pending. 1 Interrupt pending.
NP20 20	0	**Status of NMI Input 20** Illegal 64-bit access to peripheral on APB or IPBus.	0 No interrupt pending. 1 Interrupt pending.
NP19 19	0	**Status of NMI Input 19** Address out of range on a program access within the extended core address space.	0 No interrupt pending. 1 Interrupt pending.
NP18 18	0	**Status of NMI Input 18** Address out of range on a data access (on XA or XB) within the extended core address space.	0 No interrupt pending. 1 Interrupt pending.
NP17 17	0	**Status of NMI Input 17** NMI signal.	0 No interrupt pending. 1 Interrupt pending.
NP16 16	0	**Status of NMI Input 16** HDI16 NMI.	0 No interrupt pending. 1 Interrupt pending.
NP15 15	0	**Status of NMI Input 15** Watchdog time-out.	0 No interrupt pending. 1 Interrupt pending.
NP14 14	0	**Status of NMI Input 14** Bus error: AMENT.	0 No interrupt pending. 1 Interrupt pending.

Table 12-6. NMIPR Bit Descriptions (Continued)

Name	Reset	Description	Settings
NP13 13	0	**Status of NMI Input 13** Bus error: AMEC.	0 No interrupt pending. 1 Interrupt pending.
NP12 12	0	**Status of NMI Input 12** Bus error: AMIC.	0 No interrupt pending. 1 Interrupt pending.
NP11 11	0	**Status of NMI Input 11** Bus error: AMDMA.	0 No interrupt pending. 1 Interrupt pending.
NP10 10	0	**Status of NMI Input 10** Bus time-out: ASMI.	0 No interrupt pending. 1 Interrupt pending.
NP9 9	0	**Status of NMI Input 9** Bus time-out: ASM2.	0 No interrupt pending. 1 Interrupt pending.
NP8 8	0	**Status of NMI Input 8** Bus time-out: ASEMI.	0 No interrupt pending. 1 Interrupt pending.
NP7 7	0	**Status of NMI Input 7** Bus time-out: ASTH.	0 No interrupt pending. 1 Interrupt pending.
NP6 6	0	**Status of NMI Input 6** Bus time-out: ASAPB.	0 No interrupt pending. 1 Interrupt pending.
NP5 5	0	**Status of NMI Input 5** Bus time-out: ASSB.	0 No interrupt pending. 1 Interrupt pending.
NP4 4	0	**Status of NMI Input 4** Address out of range on IFU access AMIC bus.	0 No interrupt pending. 1 Interrupt pending.
NP3 3	0	**Status of NMI Input 3** Address out of range on Ethernet DMA access on AMENT bus.	0 No interrupt pending. 1 Interrupt pending.
NP2 2	0	**Status of NMI Input 2** Address out of range on SC1400 data access on AMEC bus.	0 No interrupt pending. 1 Interrupt pending.
NP1 1	0	**Status of NMI Input 1** Address out of range on DMA access on AMDMA bus.	0 No interrupt pending. 1 Interrupt pending.
NP0 0	0	**Status of NMI Input 0** Write to the boot ROM.	0 No interrupt pending. 1 Interrupt pending.

MIPR Maskable Interrupt Pending Register ICTL_BASE + 0x008

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DDBG	—														
TYPE	R/W	R														

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE	R															

MIPR masks or enables interrupt sources in Debug mode.

Table 12-7. MIPR Bit Descriptions

Name	Reset	Description	Settings
DDBG 31	0	**Disable Debug** Determines whether the maskable interrupt request and non-maskable interrupt request signals are recognized or disabled in debug mode.	0 While the device is in Device Debug mode, interrupts are disabled; that is, requests are masked. 1 Device Debug mode does not mask interrupt requests.
— 30–0	0x0	Reserved. Write to zero for future compatibility.	

The IPLRx registers, discussed in the remainder of this chapter, individually program the priority level of each interrupt channel. Each 3-bit field has an associated write disable. When this write disable bit is cleared to zero, writes to the corresponding 3-bit field are enabled. An application can easily change the priority level of a single interrupt source without the need to read the register, modify it, and then write the new value back. For the interrupt source with the IPL to be modified, this bit is cleared to zero, whereas the WD bits for all other fields in the register are set to a value of one. The WD bit for each source always reads as a zero.

IPLR0 Interrupt Priority Level Register 0 IPL_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD7	IPL7			WD6	IPL6			WD5	IPL5			WD4	IPL4		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD3	IPL3			WD2	IPL2			WD1	IPL1			WD0	IPL0		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR0 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-8. IPLR0 Bit Descriptions

Name	Reset	Description	Settings
WD7 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL7 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD6 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL6 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD5 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL5 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD4 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL4 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD3 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL3 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD2 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL2 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4

Table 12-8. IPLR0 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD1 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL1 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD0 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL0 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4

IPLR1 Interrupt Priority Level Register 1 IPL_BASE + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD15	IPL15			WD14	IPL14			WD13	IPL13			WD12	IPL12		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD11	IPL11			WD10	IPL10			WD9	IPL9			WD8	IPL8		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR1 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources,* on page 12-3.

Table 12-9. IPLR1 Bit Descriptions

Name	Reset	Description	Settings
WD15 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL15 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD14 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL14 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD13 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL13 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4

Table 12-9. IPLR1 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD12 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL12 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD11 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL11 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD10 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL10 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL9 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4
WD8 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL8 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-2**, *Interrupt Arbitration Priorities,* on page 12-4

IPLR2 Interrupt Priority Level Register 2 IPL_BASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD23	IPL23			WD22	IPL22			WD21	IPL21			WD20	IPL20		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD19	IPL19			WD18	IPL18			WD17	IPL17			WD16	IPL16		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR2 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-10. IPLR2 Bit Descriptions

Name	Reset	Description	Settings
WD23 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
— 30–28	0x0	Reserved. Write to zero for future compatibility.	
WD22 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL22 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	
WD21 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
— 22–20	0x0	Reserved. Write to zero for future compatibility.	
WD20 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL20 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	
WD19 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
— 14–12	0x0	Reserved. Write to zero for future compatibility.	
WD18 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL18 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	
WD17 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
— 6–4	0x0	Reserved. Write to zero for future compatibility.	
WD16 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL16 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	

IPLR3 Interrupt Priority Level Register 3 IPL_BASE + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD31	IPL31			WD30	IPL30			WD29	IPL29			WD28	IPL28		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD27	IPL27			WD26	IPL26			WD25	IPL25			WD24	IPL24		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR3 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-11. IPLR3 Bit Descriptions

Name	Reset	Description	Settings
WD31 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL31 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD30 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL30 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD29 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL29 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD28 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL28 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD27 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL27 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD26 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL26 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-11. IPLR3 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD25 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL25 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD24 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL24 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR4 Interrupt Priority Level Register 4 IPL_BASE + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD39	IPL39			WD38	IPL38			WD37	IPL37			WD36	IPL36		
TYPE	W	R/W			W	R/W			W	R/R			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD35	IPL35			WD34	IPL34			WD33	IPL33			WD32	IPL32		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR4 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-12. IPLR4 Bit Descriptions

Name	Reset	Description	Settings
WD39 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL39 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD38 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL38 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD37 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL37 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-12. IPLR4 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD36 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL36 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD35 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL35 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD34 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL34 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD33 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL33 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD32 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL32 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR5 Interrupt Priority Level Register 5 IPL_BASE + 0x14

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD47	IPL47			WD46	IPL46			WD45	IPL45			WD44	IPL44		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD43	IPL43			WD42	IPL42			WD41	IPL41			WD40	IPL40		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR5 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-13. IPLR5 Bit Descriptions

Name	Reset	Description	Settings
WD47 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL47 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD46 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL46 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD45 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL45 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD44 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL44 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD43 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL43 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD42 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL42 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD41 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL41 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD40 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL40 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR6 Interrupt Priority Level Register 6 IPL_BASE + 0x18

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD55	IPL55			WD54	IPL54			WD53	IPL53			WD52	IPL52		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD51	IPL51			WD50	IPL50			WD49	IPL49			WD48	IPL48		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR6 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-14. IPLR6 Bit Descriptions

Name	Reset	Description	Settings
WD55 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL55 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD54 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL54 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD53 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL53 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD52 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL52 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD51 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL51 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD50 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL50 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-14. IPLR6 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD49 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL49 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD48 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL48 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR7 Interrupt Priority Level Register 7 IPL_BASE + 0x1C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD63	IPL63			WD62	IPL62			WD61	IPL61			WD60	IPL60		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD59	IPL59			WD58	IPL58			WD57	IPL57			WD56	IPL56		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR7 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-15. IPLR7 Bit Descriptions

Name	Reset	Description	Settings
WD63 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL63 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD62 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL62 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD61 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL61 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-15. IPLR7 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD60 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL60 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD59 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL59 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD58 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL58 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD57 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL57 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD56 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL56 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR8 Interrupt Priority Level Register 8 IPL_BASE + 0x20

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD71	IPL71			WD70	IPL70			WD69	IPL69			WD68	IPL68		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD67	IPL67			WD66	IPL66			WD65	IPL65			WD64	IPL64		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR8 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-16. IPLR8 Bit Descriptions

Name	Reset	Description	Settings
WD71 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL71 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD70 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL70 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD69 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL69 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD68 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL68 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD67 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL67 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD66 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL66 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD65 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL65 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD64 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL64 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR9 Interrupt Priority Level Register 9 IPL_BASE + 0x24

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD79	IPL79			WD78	IPL78			WD77	IPL77			WD76	IPL76		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD75	IPL75			WD74	IPL74			WD73	IPL73			WD72	IPL72		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR9 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-17. IPLR9 Bit Descriptions

Name	Reset	Description	Settings
WD79 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL79 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD78 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL78 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD77 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL77 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD76 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL76 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD75 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL75 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD74 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL74 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-17. IPLR9 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD73 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL73 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD72 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL72 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR10 Interrupt Priority Level Register 10 IPL_BASE + 0x28

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD87	IPL87			WD86	IPL86			WD85	IPL85			WD84	IPL84		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD83	IPL83			WD82	IPL82			WD81	IPL81			WD80	IPL80		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR10 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-18. IPLR10 Bit Descriptions

Name	Reset	Description	Settings
WD87 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL87 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD86 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL86 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD85 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL85 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-18. IPLR10 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD84 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL84 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD83 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL83 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD82 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL82 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD81 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL81 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD80 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL80 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR11 Interrupt Priority Level Register 11 IPL_BASE + 0x2C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD95	IPL95			WD94	IPL94			WD93	IPL93			WD92	IPL92		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD91	IPL91			WD90	IPL90			WD89	IPL89			WD88	IPL88		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR11 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-19. IPL11 Bit Descriptions

Name	Reset	Description	Settings
WD95 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL95 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD94 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL94 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD93 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL93 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD92 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL92 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD91 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL91 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD90 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL90 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD89 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL89 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD88 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL88 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR12 Interrupt Priority Level Register 12 IPL_BASE + 0x30

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD103	IPL103			WD102	IPL102			WD101	IPL101			WD100	IPL100		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD99	IPL99			WD98	IPL98			WD97	IPL97			WD96	IPL96		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR12 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-20. IPLR12 Bit Descriptions

Name	Reset	Description	Settings
WD103 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL103 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD102 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL102 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD101 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL101 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD100 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL100 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD99 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL99 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD98 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL98 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-20. IPLR12 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD97 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL97 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD96 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL96 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR13 Interrupt Priority Level Register 13 IPL_BASE + 0x34

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD111	IPL111			WD110	IPL110			WD109	IPL109			WD108	IPL108		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD107	IPL107			WD106	IPL106			WD105	IPL105			WD104	IPL104		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR13 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-21. IPLR13 Bit Descriptions

Name	Reset	Description	Settings
WD111 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL111 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD110 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL110 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD109 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL109 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

Table 12-21. IPLR13 Bit Descriptions (Continued)

Name	Reset	Description	Settings
WD108 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL108 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD107 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL107 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD106 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL106 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD105 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL105 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD104 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL104 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

IPLR14 Interrupt Priority Level Register 14 IPL_BASE + 0x38

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	WD119	IPL119			WD118	IPL118			WD117	IPL117			WD116	IPL116		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	WD115	IPL115			WD114	IPL114			WD113	IPL113			WD112	IPL112		
TYPE	W	R/W			W	R/W			W	R/W			W	R/W		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPLR14 sets the interrupt priority levels for eight different maskable interrupt channels. See **Table 12-1**, *Programming the Priorities of Maskable Interrupt Sources*, on page 12-3.

Table 12-22. IPLR14 Bit Descriptions

Name	Reset	Description	Settings
WD119 31	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL119 30–28	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD118 27	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL118 26–24	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD117 23	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL117 22–20	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD116 19	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL116 18–16	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD115 15	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL115 14–12	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD114 11	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL114 10–8	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD113 7	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL113 6–4	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.
WD112 3	0	**Write Disable** Write disable for the associated channel.	0 Write access enabled. 1 Write access disabled.
IPL112 2–0	0x0	**Maskable Priority Level** Specifies the priority level for this interrupt input channel.	See **Table 12-1** on page 12-3.

INTEV0 Interrupt Event Register 0 ICTL_BASE + 0x40

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	EN29	EN28	EN27	EN26	EN25	EN24	EN23	EN22	EN21	EN20	EN19	EN18	EN17	EN16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTEV0 enables interrupt requests to be ORed together and create a trigger for the event port.

Table 12-23. INTEV0 Bit Descriptions

Name	Reset	Description	Settings
— 31–30	0x0	Reserved. Write to zero for future compatibility.	
EN[29–0] 29–0	0	**Interrupt Select Enable** Enables an individual interrupt channel (channels 29-0) for ORing into event port input.	0 Disabled for interrupt channel i. 1 Enabled for interrupt channel i.

Note: Channels that currently do not have an associated interrupt request should always be disabled (see **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8).

INTEV1 Interrupt Event Register 1 ICTL_BASE + 0x44

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	EN59	EN58	EN57	EN56	EN55	EN54	EN53	EN52	EN51	EN50	EN49	EN48	EN47	EN46
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EN45	EN44	EN43	EN42	EN41	EN40	EN39	EN38	EN37	EN36	EN35	EN34	EN33	EN32	EN31	EN30
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTEV1 enables interrupt requests to be ORed together and create a trigger for the event port.

Table 12-24. INTEV1 Bit Descriptions

Name	Reset	Description	Settings
— 31–30	0x0	Reserved. Write to zero for future compatibility.	
EN[59–30] 29–0	0	**Interrupt Select Enable** Enables an individual interrupt channel (channels 59–30) for ORing into event port input.	0 Disabled for Interrupt Channel i. 1 Enabled for Interrupt Channel i.

Note: Channels that currently do not have an associated interrupt request should always be disabled (see **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8).

INTEV2 Interrupt Event Register 2 ICTL_BASE + 0x48

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTEV2 enables interrupt requests to be ORed together and create a trigger for the event port.

Table 12-25. INTEV2 Bit Descriptions

Name	Reset	Description	Settings
— 31–30	0x0	Reserved. Write to zero for future compatibility.	
ENi 29–0	0	**Interrupt Select Enable** Enables an individual interrupt channel (channels 89–60) for ORing into event port input.	0 Disabled for Interrupt Channel i. 1 Enabled for Interrupt Channel i.

Note: Channels that currently do not have an associated interrupt request should always be disabled (see **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8).

INTEV3 Interrupt Event Register 3 ICTL_BASE + 0x4C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi	ENi
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTEV3 enables interrupt requests to be ORed together and create a trigger for the event port.

Table 12-26. INTEV3 Bit Descriptions

Name	Reset	Description	Settings
— 31–30	0x0	Reserved. Write to zero for future compatibility.	
ENi 29–0	0	**Interrupt Select Enable** Enables an individual interrupt channel (channels 119-90) for ORing into event port input.	0 Disabled for Interrupt Channel i. 1 Enabled for Interrupt Channel i.

Note: Channels that currently do not have an associated interrupt request should always be disabled (see **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8).

INTEVNM0 Interrupt Event Non-Maskable Register 0 ICTL_BASE + 0x60

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	NP31	NP30	NP29	NP28	NP27	NP26	NP25	NP24	NP23	NP22	NP21	NP20	NP19	NP18	NP17	NP16
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NP15	NP14	NP13	NP12	NP11	NP10	NP9	NP8	NP7	NP6	NP5	NP4	NP3	NP2	NP1	NP0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INTEVNM0 enables interrupt requests to be ORed together and create a trigger for the event port.

Table 12-27. INTEVNM0 Bit Descriptions

Name	Reset	Description	Settings
NPi 31–0	0	**Non-Maskable Interrupt Select Enable** Enables an individual interrupt channel (NP0-28) for ORing into event port input.	0 Disabled for Interrupt Channel i. 1 Enabled for Interrupt Channel i.

Channels that currently do not have an associated interrupt request should always be disabled (see **Table 12-5**, *MSC711x Maskable Interrupt Sources*, on page 12-8).

Reset

13

All MSC711x reset sources are fed into the reset controller, which takes different actions depending on the source of the reset. **Table 13-1** describes the reset sources.

Table 13-1. Reset Sources

Name	Pin Direction	Description
Power-on reset (PORESET)	Input	Initiates the power-on reset flow that resets the MSC711x devices and configures various attributes of the MSC711x. On PORESET, the entire MSC711x device is reset. PLL and clock synthesis states are reset, HRESET is driven, the SC1400 extended core is reset, and system configuration is sampled. The configuration pins are configured only at the deassertion of PORESET.
External hard reset (HRESET)	I/O	Initiates the hard reset flow that configures various attributes of the MSC711x device. HRESET is an open-drain output. Upon hard reset, HRESET is driven.
Software watchdog reset		When the MSC711x watchdog count reaches zero, a software watchdog reset is signalled. The enabled software watchdog event then generates an internal hard reset sequence. This reset also asserts the HRESET signal.
Bus monitor reset		When the MSC711x bus monitor count reaches zero, a bus monitor hard reset is asserted. The enabled bus monitor event then generates an internal hard reset sequence. This reset also asserts HRESET.
JTAG Commands: EXTEST, CLAMP, or HIGHZ		When one of the EXTEST, CLAMP, or HIGHZ JTAG commands executes, JTAG logic asserts the JTAG reset signal and an internal soft reset sequence is generated.

Table 13-2 summarizes the reset actions that occur as a result of the different reset sources.

Table 13-2. Reset Actions for Each Reset Source

Reset Action	Reset Source		
	External Power-On Reset	**External Hard Reset[1], Software Watchdog, Bus Monitor**	**Soft Reset JTAG Commands: EXTEST, CLAMP, or HIGHZ**
Configuration signals sampled (See **Section 13.1**)	Yes	No	No
PLL and clock synthesis states Reset	Yes	No	No
$\overline{\text{HRESET}}$ driven	Yes	Yes	No
Software watchdog and bus time-out monitor registers	Yes	Yes	No
Clock synthesis module STOPCTRL, HLTREQ, and HLTACK registers are reset	Yes	Yes	Yes
Extended core reset	Yes	Yes	Yes
Peripheral modules reset. Does not include the extended core and PLL/clocking units.	Yes	Yes	Yes
Note: A hard reset can be caused via the External Hard Reset pin ($\overline{\text{HRESET}}$), a watchdog time-out, or detection by any of the bus time-out monitors.			

13.1 Reset Sources

This section covers power-on reset, hard reset, and soft reset.

13.1.1 Power-On Reset

Asserting $\overline{\text{PORESET}}$ initiates the power-on reset flow. $\overline{\text{PORESET}}$ must be asserted externally for at least 16 input clock cycles after MSC711x external power reaches at least 2/3 V_{CC}. **Table 13-3** shows the MSC711x configuration signals. These signals are sampled on the third rising edge of the input clock after the deassertion (rising edge) of $\overline{\text{PORESET}}$, and they determine different MSC711x configuration features.

Table 13-3. External Configuration Signals Sampled on Power-On Reset

Signal	Description	Settings
BM[3–0]	**Boot Mode** Input lines sampled at the rising edge of PORESET and loaded into the RSR. These lines determine the MSC711x boot mode. The value of these pins is latched into the RSR register.	See definition in **Section 14-2**, *Boot Mode Source Selection,* on page 14-5.
SWTE	**Software Watchdog Timer Enable** Input line sampled at the rising edge of PORESET. This bit can override the software watchdog timer functionality. Its value is loaded into the DEVCFG register when sampled. The value of this pin is latched into the DEVCFG register.	0 Watchdog timer operation disabled. 1 Watchdog timer operation enabled.
HDSP	**Host Data Strobe Polarity** Input line sampled at the rising edge of PORESET. This bit configures the strobes on the HD16 port for positive or negative polarity. Immediately out of power-on reset, HPCR[HDSP] reflects the sampled value of this pin. However, this bit can be modified on future accesses.	0 Active low. 1 Active high.
H8BIT	**Host 8-Bit Mode** Input line sampled at the rising edge of PORESET. This bit configures the HDI16 port for either 8 or 16-bit modes of operation. Immediately out of power-on reset, HPCR[H8BIT] reflects the sampled value of this pin. However, this bit can be modified on future accesses.	0 16-bit operation. 1 8-bit operation.

On the third rising edge of the input clock after the deassertion (rising edge) of PORESET, one additional pin is sampled, DBREQ/EE0. The SC1400 core can be immediately placed into Debug mode if the DBREQ pin is asserted when reset is exited. Therefore, you can place the device into Debug mode immediately out of reset, if desired. This pin is not captured in the Reset Status Register (RSR).

13.1.2 Hard Reset

A hard reset sequence is initiated either externally when HRESET is asserted or internally from a software watchdog timer reset or a bus time-out monitor reset. The source of the reset controls the direction of the pin. Normally, a hard reset is configured as an input for external resets via the HRESET signal. However, this open-drain signal is driven by the MSC711x reset controller when an internal resource initiates a hard reset sequence. In both cases, the MSC711x continuously asserts HRESET throughout the hard reset sequence. After the MSC711x asserts HRESET for 521 bus clock cycles, it releases this signal and exits the hard reset sequence. An external pull-up resistor should deassert the signal. Then there is a 16 input clock cycle wait before testing for an external hard reset begins.

13.1.3 Soft Reset

A soft reset sequence is initiated externally when the MSC711x detects a cause to start the soft reset sequence (JTAG commands: EXTEST, CLAMP, or HIGHZ). While a soft reset is in progress, internal hardware is reset, but $\overline{\text{HRESET}}$ is not asserted.

Note: The main difference between hard reset and soft reset is that the $\overline{\text{HRESET}}$ pin is not driven during soft reset.

13.2 Reset Timing

Figure 13-1 shows the timing for power-on reset on a MSC711x device. During power-on reset:

- The external $\overline{\text{PORESET}}$ signal must be asserted for a minimum of 16 cycles of CLKIN.
- The MSC711x device drives the $\overline{\text{HRESET}}$ signal.
- The following modules in the MSC711x device are reset:
  - Clock synthesis module is reset and the PLL is disabled.
  - SWT is reset as disabled.
  - Other MSC711x peripherals are reset.
  - Bus time-out monitors are reset.
  - The extended core is reset.
- The device is clocked with the bypass clock.
- On the third rising edge of the input clock after $\overline{\text{PORESET}}$ is deasserted, the pins listed in **Table 13-3** are sampled.
- The $\overline{\text{HRESET}}$ signal is driven for an additional 521 clock cycles (CLKIN cycles).
- The device exits the reset state.
- The device does not sample the $\overline{\text{HRESET}}$ pin for the next 16 clock cycles (CLKIN cycles).

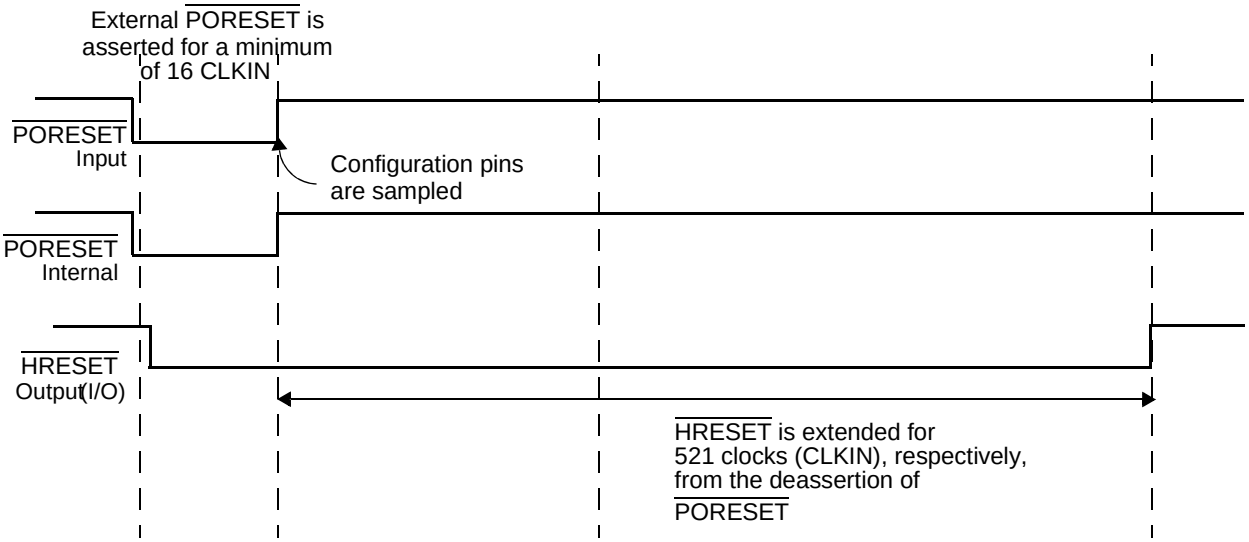


Figure 13-1. MSC711x Power-On Reset Timing

Figure 13-2 shows the timing for a hard reset on a MSC711x device. During a hard reset:

- An external device asserts the $\overline{\text{HRESET}}$ signal (power-on reset is not asserted).
- Modules within the MSC711x device are reset:
  - The SWT is reset as disabled.
  - The other MSC711x peripherals are reset.
  - The bus time-out monitors are reset.
  - The extended core is reset.
- The device can be clocked with either the PLL or the bypass clock.
- No pins are sampled.
- The $\overline{\text{HRESET}}$ signal remains driven for an additional 521 bus clock cycles.
- The device exits the reset state.
- The device does not sample the $\overline{\text{HRESET}}$ pin for the next 16 bus clock cycles.

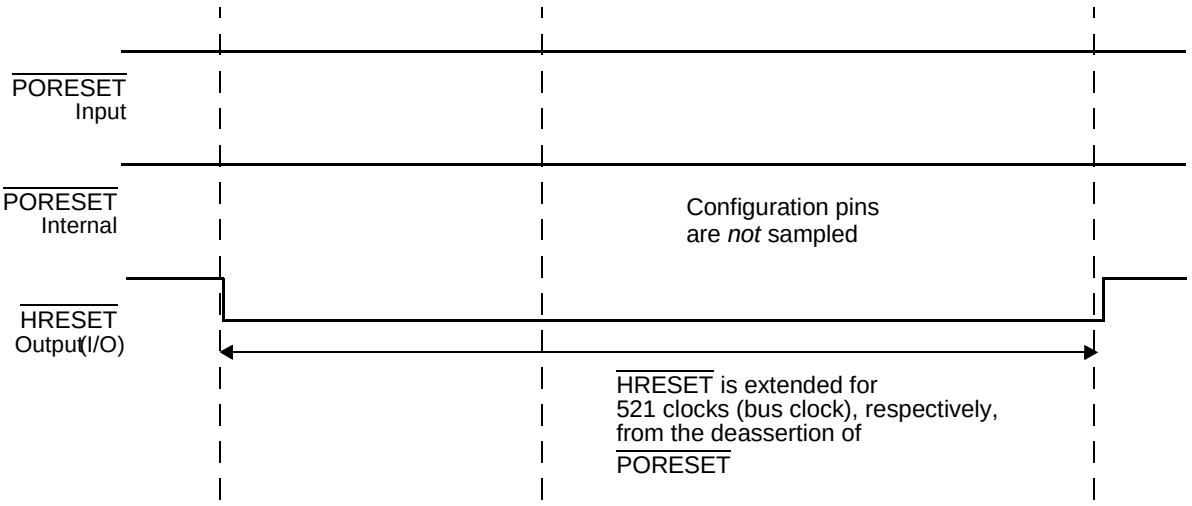


Figure 13-2. MSC711x Hard Reset Timing

13.3 Exiting Reset and Booting the Device

MSC711x devices begin program execution at an appropriate reset vector within the boot ROM when exiting the reset processing state. Each reset vector has 64 bytes allocated for it. **Figure 13-3** shows the address that the SC1400 core drives onto the PAB when it exits reset. The 23-bit field providing the boot ROM base can be determined from the upper 23-bits of the boot ROM starting address (see **Table 5-3**, *Summary — MSC711x Memory Map,* on page 5-30).

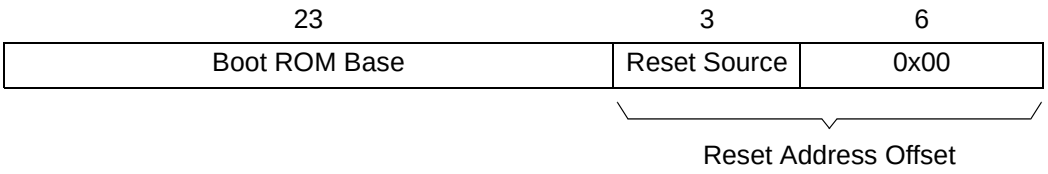


Figure 13-3. Interrupt Vector Address

The reset vector selected depends on the source of the reset and is one of those listed in **Table 13-4**.

Table 13-4. MSC711x Reset Sources

Reset Source	Priority	Reset Address Offset	Description
Power-on reset	Highest - 1	0x000	Power-on reset signal asserted.
Hard reset	2	0x040	External source asserts the external hard reset signal.
Software watchdog reset	3	0x080	Software watchdog timer expiration.
Bus time-out monitor reset	4	0x0C0	Bus time-out on any AHB-Lite bus.
JTAG command reset	Lowest - 5	0x100	Upon executing a JTAG command that causes an internal reset.
—	—	0x140	Reserved
—	—	0x180	Reserved
—	—	0x1C0	Reserved
Note: Addresses 0x1C0–0x1FF are reserved for the last reserved reset vector.			

If multiple resets occur simultaneously, the highest-priority address as shown in **Table 13-4** is the vector used.

13.4 Reset Programming Model

This section describes the Reset Status Register (RSR). The value of the base address for this register file, CLK_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4.

RSR Reset Status Register CLK_BASE + 0x40

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BM3	BM2	BM1	BM0	—						JTRS	—	SWRS	BMRS	—	EHRS
TYPE	R/W															
RESET	0	0	**	**	0	0	0	0	0	0	*	0	*	*	1	*

* These bits are set appropriately based on the type of reset.

** These bits are sampled from MSC711x pins on the deassertion of power-on reset and loaded here.

RSR records the values of pins sampled at reset as well as reset events. For example, when a software watchdog expiration generates a hard reset, RSR[SWRS] is set. All bits are cleared by writing a 1 (writing zero has no effect).

Table 13-5. RSR Bit Descriptions

Name	Reset	Description	Settings
— 31–16	0	Reserved. Write to zero for future compatibility.	
BM3 15	0	**BM3 Pin Status** Contains the value of the boot mode pin when it is sampled on the deassertion of power-on reset.	0 Signal deasserted. 1 Signal asserted.
BM2 14		**BM2 Pin Status** Contains the value of the boot mode pin when it is sampled on the deassertion of power-on reset.	0 Signal deasserted. 1 Signal asserted.
BM1 13	**[1]	**BM1 Pin Status** Contains the value of the boot mode pin when it is sampled at the deassertion of power-on reset.	0 Signal deasserted. 1 Signal asserted.
BM0 12	**[1]	**BM0 Pin Status** Contains the value of the BM0 signal when it is sampled at the deassertion of power-on reset.	0 Signal deasserted. 1 Signal asserted.
— 11–6	0	Reserved. Write to zero for future compatibility.	
JTRS 5	*[2]	**JTAG Reset Status** When a host reset command is written, through JTAG logic (JTAG reset request), JTRS is set and remains set until software clears it.	0 No host reset command through JTAG. 1 Host reset command through JTAG.
— 4	0	Reserved. Write to zero for future compatibility.	

Table 13-5. RSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
SWRS 3	*[2]	**Software Watchdog Reset Status** When a software watchdog expire event (which causes a reset) is detected, the SWRS bit is set and remains set until the software clears it.	0 No software watchdog reset event. 1 Software watchdog reset event.
BMRS 2	*[2]	**Bus Monitor Reset Status** When a bus monitor expire event (which causes a reset) is detected, BMRS is set and remains set until the software clears it.	0 No bus monitor reset event. 1 Bus monitor reset event.
— 1	0	Reserved. Write to zero for future compatibility.	
EHRS 0	*[2]	**External hard reset status** When an external hard reset event is detected, EHRS is set and it remains set until software clears it. If the HRESET pin is asserted in response to a software watchdog time-out or bus monitor event, this bit is not asserted.	0 No external hard reset event. 1 External hard reset event.

Notes: **1.** At power-on reset, the BM bits are loaded from pins, and the JTRS, SWRS, BMRS, and EHRS are all reset to zero.

2. On other resets, the BM bits are not affected, and the JTRS, SWRS, BMRS, and EHRS are set to values that reflect the type of the reset. If multiple resets occur simultaneously, the corresponding status bits of all recognized sources are set.

Boot Program 14

The boot program, which resides in the internal ROM, initializes the MSC711x device after it completes a reset sequence. MSC711x devices can boot through the ports listed in **Table 14-1**.

Table 14-1. Available Boot Sources

Boot Port	PLL	Comments
HDI16	Enabled or Disabled	Can boot with either HDI8 or HDI16 functionality. PLL operation is determined by the BM[3–0] pins.
I^2C	Disabled	—
SPI	Enabled or Disabled	SPI is implemented in boot ROM software via one of two different sets of GPIO pins: • BM3, BM2, HA3, HCS2 (PLL enabled or disabled) • UTXD, URXD, SCA, SCL (PLL disabled) PLL operation is specified by the BM[3–0] signal pins.

This chapter describes the boot process. It begins with booting basics, including the default values programmed by the boot program and interrupt handling during the boot process. Then it considers different ways to boot the device.

14.1 Boot Basics

Immediately upon exiting power-on reset or a hard reset, the SC1400 core begins executing the boot program, which performs the following tasks:

1. Start executing instructions from the boot ROM. Instructions execute with the instruction cache (ICache) disabled.
2. To determine which port is used for booting, examine the BM[3–0] bits in the Reset Status Register (discussed on **page 13-7**).
3. To determine the clocking used when the boot code runs, examine the BM[3–0] bits. If the PLL is used, appropriately set up the PLL in the clock synthesis module. The boot program waits for lock before continuing.
4. Begin loading the user's program through the appropriate port. The user's data must be formatted according to the boot data record for that port (shown later in this chapter).
5. When the last record is processed, jump to the address specified by the user in one of the boot records.

Figure 14-1 shows a basic flowchart of the boot program.

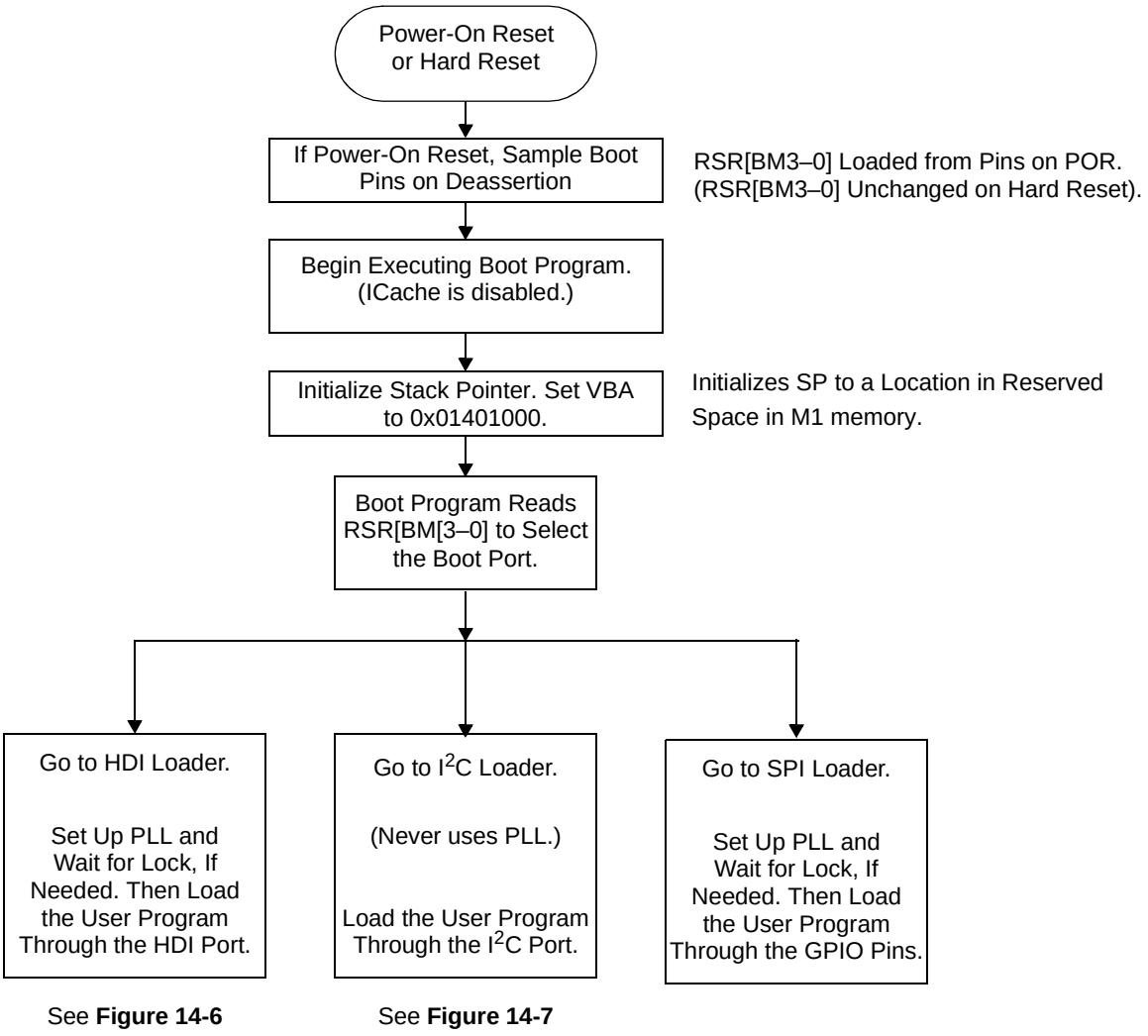
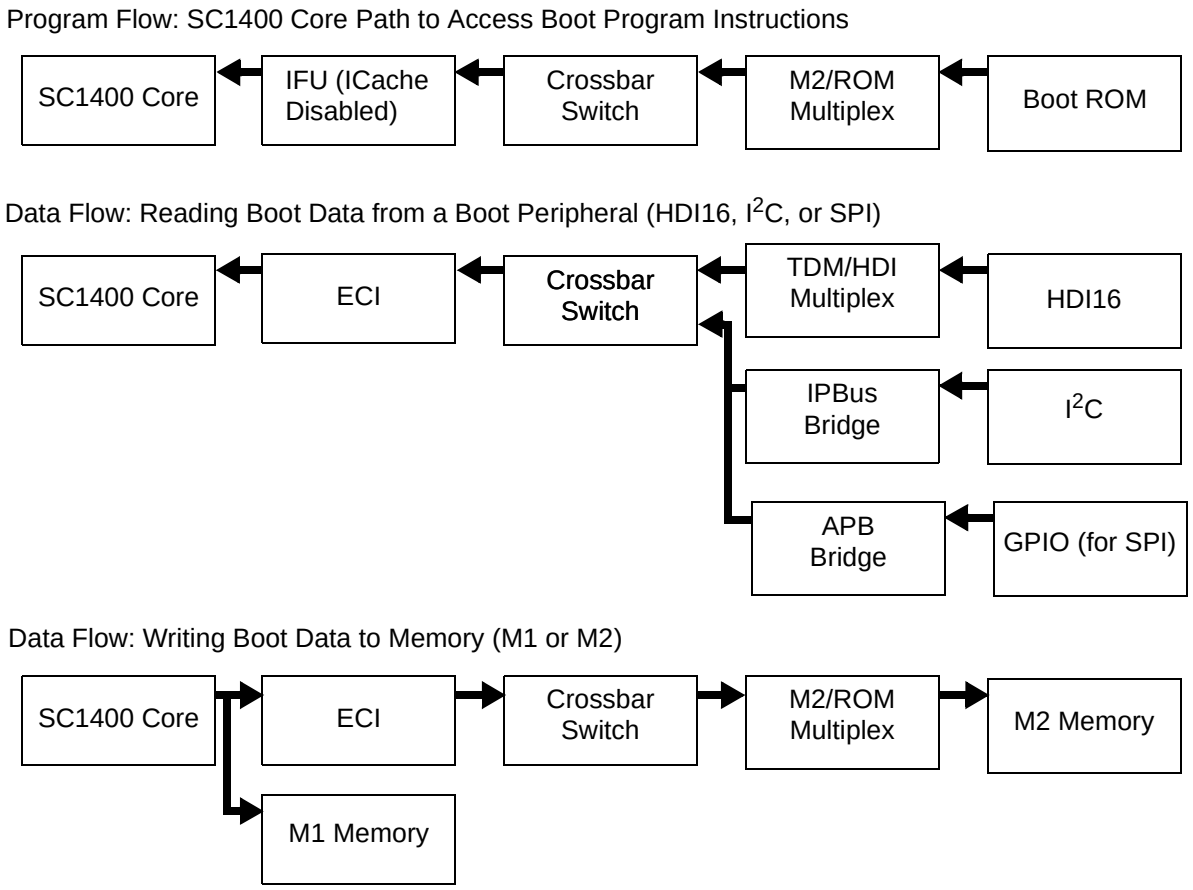


Figure 14-1. Boot Program Flow Diagram

When the SC1400 core begins executing instructions from the boot ROM, it fetches them through the instruction fetch unit and the crossbar switch. The ICache is disabled. **Figure 14-2** shows the flow for accessing the boot ROM instructions.

The data booted into the MSC711x device (that is, the data loaded into the MSC711x memories) is stored in a record format. The SC1400 core reads the data from the designated peripheral and then writes data from the data records to the address location specified by the data record. **Figure 14-2** shows the paths by which the boot program reads data from a peripheral and writes the data to memories.



Note: Writing boot data to the DDR memory is not supported. However, a user boot program can correctly configure the DDR port and then load boot data directly to DDR memory.

Figure 14-2. Boot Program and Data Flow Program

The source program to be loaded into the device can be organized into several blocks, each of which is placed into a boot data record (see **Figure 14-3**). Each block can be either a data block or an instruction block, and it is loaded to a specified destination. A checksum method ensures correct data loading. The boot records are loaded through the peripheral and processed by the boot program. The boot data record is different for each peripheral, but in general a boot record contains the following information:

- Size of the boot data record
- Address where the data in the boot data record is to be stored
- Boot data contained within the boot data record
- Optional checksum to verify correct loading of data

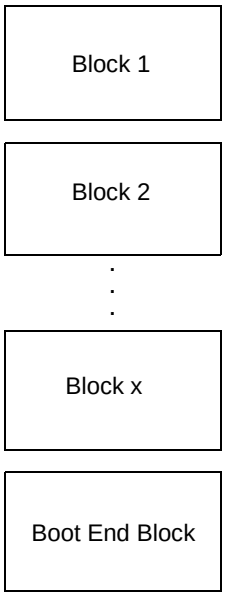


Figure 14-3. Boot Code Stream Structure

14.1.1 Boot Procedure

When an MSC711x device exits reset, there is no user code yet for it to execute. Booting loads the data, usually executable code, from an external source (device or memory) into the memory areas in the MSC711x memory map. Control then transfers from the MSC711x boot program to the newly-loaded user application code.

When an MSC711x device boots, a small user boot program is usually loaded first through one of the valid boot ports into the M1 memory. Code from this user boot program loads a much larger user program into the other memory areas of the MSC711x memory map. Loading a user boot program provides more flexibility for booting large programs into MSC711x because a user boot program can be better configured to address the needs of a particular system. Also, a user boot program can speed up boot time, first enabling the ICache and then loading the boot data. The steps in loading a user boot program are summarized as follows:

1. Reset the MSC711x device.
2. Exit from reset into the MSC711x boot program.
3. The boot program loads a user boot program.
4. Jump to the user boot program when it is loaded.
5. Use the user boot program to load the user application code.
6. Jump to this application code when it is loaded.

A user boot program is useful if you want to boot from a peripheral that is not a valid boot source. You would first boot from a valid boot source, loading your user boot program to configure the MSC711x peripheral for user booting. Program control switches to this new program, and the

application code for the program is booted through this newly selected peripheral. For example, to boot over the Ethernet (FEC), first boot from the I^2C module, get the MAC address from the user boot program, and then boot over the Ethernet. Although it is useful to load a user boot program, it is not necessary. The application code can be directly loaded from one of the boot sources available on an MSC711x device.

14.1.2 Boot Modes

The boot operating mode is set by the BM pins, which are sampled on the rising edge of $\overline{\text{PORESET}}$. **Table 14-2** shows the different booting options and the frequency ranges for the PLL. Consult **Section 11.2.2**, *Configuring the Clocks*, on page 11-5 to determine how these multiplications are calculated for the PLL.

Table 14-2. Boot Mode Source Selection

BM[3–0]	Boot Port	Input Clock Frequency	Clock Divide	PLL	CKSEL	RNG Bit	Core Clock Frequency	Comments
HDI Boot Modes								
0000	HDI16	< F_{max}	N/A	N/A	00	0	< F_{max}	Not clocked by the PLL. Can boot as 8- or 16-bit HDI.
0101	HDI16	22.2-25 MHz	1	12	11	1	266–300 MHz	Can boot as 8- or 16-bit HDI.
0010	HDI16	25-33.3 MHz	2	32	01	1	200–266 MHz	
0111	HDI16	33-66 MHz	3	12	11	1	132–264 MHz	
0100	HDI16	44.3-50 MHz	2	12	11	1	266–300 MHz	
SPI Boot Modes - Using HA3, HCS2, BM3, BM2 Pins								
1000	SPI (SW)	< F_{max}	N/A	N/A	00	0	< F_{max}	The boot program automatically determines whether EEPROM or Flash memory.
1001	SPI (SW)	15.6-25 MHz	1	17	11	0	133–212.5 MHz	
1010	SPI (SW)	33-50 MHz	2	16	11	0	132–200 MHz	
1011	SPI (SW)	44.3-75 MHz	3	18	11	0	133–225 MHz	
SPI Boot Modes - Using URXD, UTXD, SCL, SDA Pins								
1100	SPI (SW)	< F_{max}	N/A	N/A	00	0	< F_{max}	Boots through different set of pins.
I2C Boot Modes								
0001	I^2C	< 100 MHz	N/A	N/A	00	0	< 100 MHz	Not clocked by the PLL. I^2C is limited to a maximum bit rate of 400 Kbps. With a clock divider of 128, this limits the maximum input clock frequency to 100 MHz.
Reserved								
0011	Reserved	—	—	—	—	—	—	—

Table 14-2. Boot Mode Source Selection (Continued)

BM[3–0]	Boot Port	Input Clock Frequency	Clock Divide	PLL	CKSEL	RNG Bit	Core Clock Frequency	Comments
1111	Reserved	—	—	—	—	—	—	—
0110	Reserved	—	—	—	—	—	—	—
1101	Reserved	—	—	—	—	—	—	—
1110	Reserved	—	—	—	—	—	—	—

Notes: **1.** The clock divider determines the value used in the clock module CLKCTRL[PLLDVF] field.
2. The clock multiplier determines the value used in the clock module CLKCTRL[PLLMLTF] field.
3. F_{max} is determined by the maximum frequency of the peripheral and of the SC1400 core as specified in the data sheet.

Table 14-3 shows examples of these boot modes for several important CLKIN frequencies.

Table 14-3. Booting — Examples of Different CLKIN Frequencies

CLKIN Frequency	Boot Port	BM[3–0]	Clock Divider	PLL Multiplier	Post Division	Core Clock Frequency	Comments
HDI Boot Modes							
< F_{max}	HDI16	0000	N/A	N/A	N/A	CLKIN	—
25 MHz	HDI16	0010	2	32	2	200 MHz	—
25 MHz	HDI16	0101	1	12	1	300 MHz	—
33 MHz	HDI16	0111	3	12	1	132 MHz	—
33 MHz	HDI16	0010	2	32	2	264 MHz	—
50 MHz	HDI16	0111	3	12	1	200 MHz	—
50 MHz	HDI16	0100	2	12	1	300 MHz	—
66 MHz	HDI16	0111	3	12	1	264 MHz	—
SPI Boot Modes - Using HA3, HCS2, BM3, BM2 Pins							
< F_{max}	SPI (SW)	1000	N/A	N/A	N/A	CLKIN	—
16.384 MHz	SPI (SW)	1001	1	17	2	139.3 MHz	—
25 MHz	SPI (SW)	1001	1	17	2	212.5 MHz	—
33 MHz	SPI (SW)	1010	2	16	2	132 MHz	—
50 MHz	SPI (SW)	1011	3	18	2	150 MHz	—
50 MHz	SPI (SW)	1010	2	16	2	200 MHz	—
66 MHz	SPI (SW)	1011	3	18	2	198 MHz	—
SPI Boot Modes - Using URXD, UTXD, SCL, SDA Pins							
< F_{max}	SPI (SW)	1000	N/A	N/A	N/A	CLKIN	Limited by the maximum frequency of the SPI implementation.
I2C Boot Modes							
< F_{max}	I^2C	0001	N/A	N/A	N/A	CLKIN	Limited by the maximum frequency of the I^2C.

14.2 Boot Program Operation

This section describes the correct way to boot an MSC711x device in different scenarios.

14.2.1 Boot from Power-On Reset

The MSC711x device is configured as follows out of power-on reset when the boot program executes:

- The ICache is disabled.
- Clocking is determined by the boot mode.
- The software watchdog timer is disabled.
- Execution begins with the power-on reset vector in the boot ROM (see **Section 13.1.1**, *Power-On Reset*, on page 13-2).
- Maskable interrupts are disabled.
- Non-maskable interrupts are active.

See **Table 13-2**, *Reset Actions for Each Reset Source*, on page 13-2 for a summary of the state of the device after power-on reset.

14.2.2 Boot from Hard Reset

The MSC711x device is configured as follows out of hard reset when the boot program executes:

- The ICache is disabled.
- The device is clocked the same way it was clocked before the hard reset.
- The software watchdog timer is disabled.
- Execution begins with the hard reset vector in the boot ROM (see **Section 13.1.2**, *Hard Reset*, on page 13-3).
- Maskable interrupts are disabled.
- Non-maskable interrupts are enabled.

For a summary of the state of the device after hard reset, see **Table 13-2**, *Reset Actions for Each Reset Source*, on page 13-2.

14.2.3 Bootstrapping and the Watchdog Timer

Because the software watchdog timer is disabled during power-on reset or hard reset, it is not active during the bootloading process. For details on the software watchdog, refer to **Section 7.3**, *Software Watchdog Timer*, on page 7-4. The watchdog timer is *not* disabled by soft reset, so take care to ensure that the watchdog does not expire.

14.2.4 Writing Boot Data to External DDR Memory Not Supported

For the boot program to write data to the DDR, the DDR memory controller must first be initialized. This is not supported on the MSC711x device. Therefore, to write boot data to DDR memory, you must create a user boot program as described in **Section 14.1.1**, *Boot Procedure*, on page 14-4 to initialize the DDR memory controller for specific DDR SDRAM devices, including all timing characteristics, and so on. Then the user boot program can write boot data to DDR memory.

14.2.5 Reserved M1 Memory for Bootstrap Program

Bootstrapping uses a small portion of M1 memory, as shown in **Table 14-4**. These locations are reserved, and you must never write data to them.

Table 14-4. M1 Memory Use During Boot

Device	M1 Memory Size	Reserved Locations Used During Boot	Comments
MSC7110	64 KB	0x0000FE00–0x0000FFFF	Last 512 bytes.
MSC7112, MSC7113, MSC7115, MSC7116	192 KB	0x0002FE00–0x0002FFFF	
MSC7118, MSC7119	256 KB	0x0003FE00–0x0003FFFF	

During bootstrapping, some reserved M1 memory locations in **Table 14-4** are used to store boot variables that reflect the state of the device. **Table 14-5** shows these boot variables and their corresponding locations in M1 memory.

Table 14-5. Boot Program Variables

Boot Variable	Variable Size	Address of Boot Variables in Reserved Area			Comments
		MSC7110	MSC7112, MSC7113, MSC7115, MSC7116	MSC7118, MSC7119	
NMITYPE NMI Type	8 bits	0x0000FF66	$0002FF66	0x0003FF66	Indicates what type of NMI occurred: –1 None. 0 Trap. 1 Reserved. 2 ILLEGAL. 3 DEBUG. 4 Reserved. 5 Auto-NMI. 6 Reserved. For auto-NMI, see also the CNMIPR variable in this table.

Table 14-5. Boot Program Variables (Continued)

Boot Variable	Variable Size	Address of Boot Variables in Reserved Area			Comments
		MSC7110	MSC7112, MSC7113, MSC7115, MSC7116	MSC7118, MSC7119	
RSTSRC Reset Source	8 bits	0x0000FF67	0x0002FF67	0x0003FF67	Indicates the type of reset: 0 Power-on reset. 1 External hard reset. 2 Software watchdog timer. 3 Bus time-out reset. 4 JTAG Internal reset.
CNMIPR Captured NMIPR	32 bits	0x0000FF7C	0x0002FF7C	0x0003FF7C	This variable contains a value only if a non-maskable auto-NMI occurs. Then this register captures the value of the NMIPR register in the auto-NMI interrupt service routine. Useful for debugging if a non-maskable interrupt occurs during booting. Shows the source of the interrupt.

14.2.6 Interrupt Handling During Booting

The boot program executes in response to a power-on reset or hard reset, so all maskable interrupts are disabled and only non-maskable interrupts are recognized. The MSC711x boot program initializes the interrupt handler table base address (VBA register of the SC1400 core) at its first instruction execution. Until this base address is initialized, no non-maskable interrupt (NMI) is allowed. The vector base address is initialized to the value of the boot ROM base address + 0x1000.

If a non-maskable interrupt occurs, the service routine captures its type in the NMITYPE boot program variable and then returns to the original code. However, if an auto-NMI is received, it is not only marked as an auto-NMI but also the value of the NMIPR is saved in the CNMIPR boot memory variable. See **Table 14-6**.

Table 14-6. Boot Program Recording of Non-Maskable Interrupts

Non-Maskable Interrupt Source	NMITYPE	CNMIPR	Comments
None	–1	Undefined	Normal operation.
TRAP	0	Undefined	Trap instruction executed.
ILLEGAL	2	Undefined	Illegal instruction or execution set executed.
DEBUG	3	Undefined	Debug exception.
OVERFLOW	4	Undefined	Overflow exception.
AUTO-NMI	6	Value of the NMIPR register in the interrupt handler.	Auto-NMI exception. This includes all exceptions generated within the device but outside the SC1400 core. Examples include the $\overline{\text{NMI}}$ pin and the bus time-outs.

14.3 Booting from an External Host through the HDI16

For booting through the HDI16 interface, the boot program configures the HDI16 as follows:

- Operates in non-DMA mode.
- Operates in Polled mode on the device side.
- Operates in Polled mode on the external host side.
- The external host must write four 16-bit values at a time from the boot data record:
  - TX0 contains first word.
  - TX1 contains second word.
  - TX2 contains third word.
  - TX3 contains fourth word.

TX0 is the most significant half word, and TX3 is the least significant half word. For booting from a power-on reset, the HDI16 is additionally configurable as follows:

- 8 or 16-bit mode as specified by the device H8BIT signal.
- Data strobe as specified by the device HDDS and HDSP pins (see **Section 20.5**, *Configuring the Host Interface Pins (External Host Side)*, on page 20-6).

This configuration applies to boot from power-on reset because these pins are sampled only on the deassertion of power-on reset. During a boot from hard reset, these signals remain configured as they were during the last power-on reset.

14.3.1 Host Flags

The HDI16 host flags facilitate communication between the MSC711x boot program and a host processor during booting:

- *ICR[HF3] is set by the host* to indicate that the checksums should be compared. This function is optional and is user-programmable (see **page 20-39**).
- *HCR[HF4] is set by the SC1400 core* to indicate that all blocks are loaded (see **page 20-28**).
- *HCR[HF7] is set by the SC1400 core* to indicate an error during loading since the checksums do not match. This bit is sticky, so if it is set for one block, it remains set until the end of the code transfer. The host can ascertain whether errors occurred during code transfer by reading the ISR[HF7] bit.

The use of these bits to provide status and error handling is described in **Section 14.3.6**, *Error Handling on Completion*.

14.3.2 Host Tasks During HDI16 Boot

When a host boots an MSC711x device, it waits for the MSC711x boot program to finish its default initialization and then initializes the device by loading code and data to the internal memory. The host polls a valid bit through the HDI16 host flags. A second valid bit is set when the MSC711x boot code finishes the default initialization and the host can access the internal resources, including internal memory. When the host finishes initialization, it should notify the MSC711x by asserting a host port bit for the SC1400 core to read. In summary, the user boot program running on the host typically operates as follows:

1. Waits for the assertion of the valid bit within the HDI16.
2. Loads code and data to internal RAM.
3. Signals the SC1400 core to initiate a jump to the address provided by the final record.

14.3.3 External Host-Side Boot Load Flow

The host performs these steps when an MSC711x device is booted through the host interface:

1. Writes the boot code in blocks conforming to the format described in **Section 14.3.5**, *HDI16 Boot Data Records*, on page 14-15.
2. Specifies pointers for the ISR, ICR, and Tx registers.
3. Resets the MSC711x device.
4. Waits for a predetermined number of clocks to ensure that the HDI16 can receive data. Then the data is transferred to the Tx register 64 bits at a time. After each transfer, the host polls the ISR[TXDE] bit, receives the data, and transfers the next 64 bits.

The MSC711x device sets the ISR[HF4] bit when all bits are transferred. If checksum calculation and verification is required, the host sets the ICR[HF3] bit. When the checksum function is used, the host checks the ISR[HF7] bit to detect checksum errors at the end of the code transfer. **Figure 14-4** presents a flow diagram of the host-side boot flow.

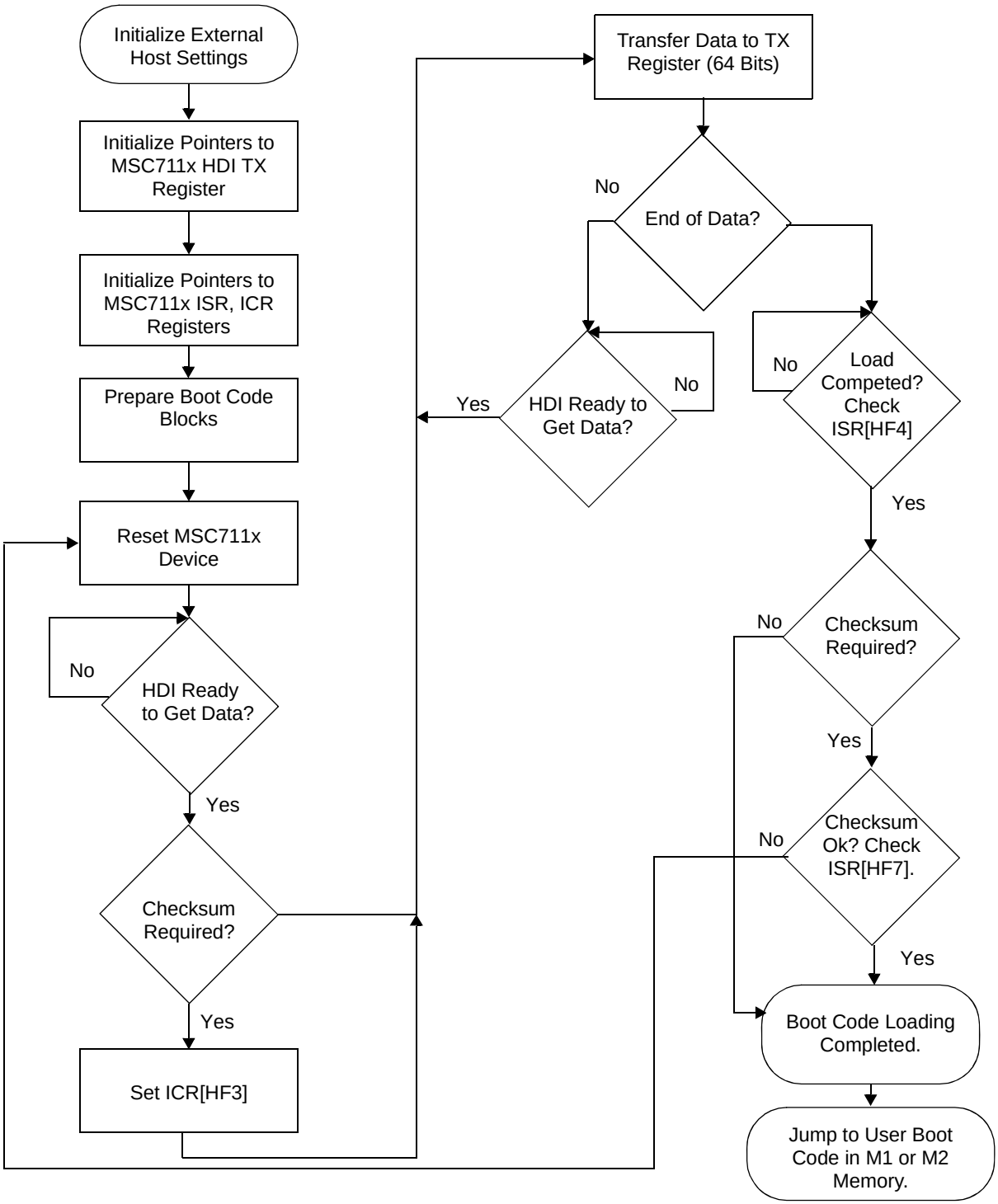


Figure 14-4. External Host Procedure During HDI16 Boot

14.3.4 Host Interface Boot Procedure

The boot program first initializes the HDI port:

1. The bootloader routine enables the HDI16 port by setting the HPCR[HEN] bit and checks the value of HPCR[H8BIT] to determine which port size the host is using.
2. The external host initializes the HDI16 when it is ready to start the boot source code transfer by setting the ICR[INIT] bit.

Figure 14-5 shows the flow for checking the host port size.

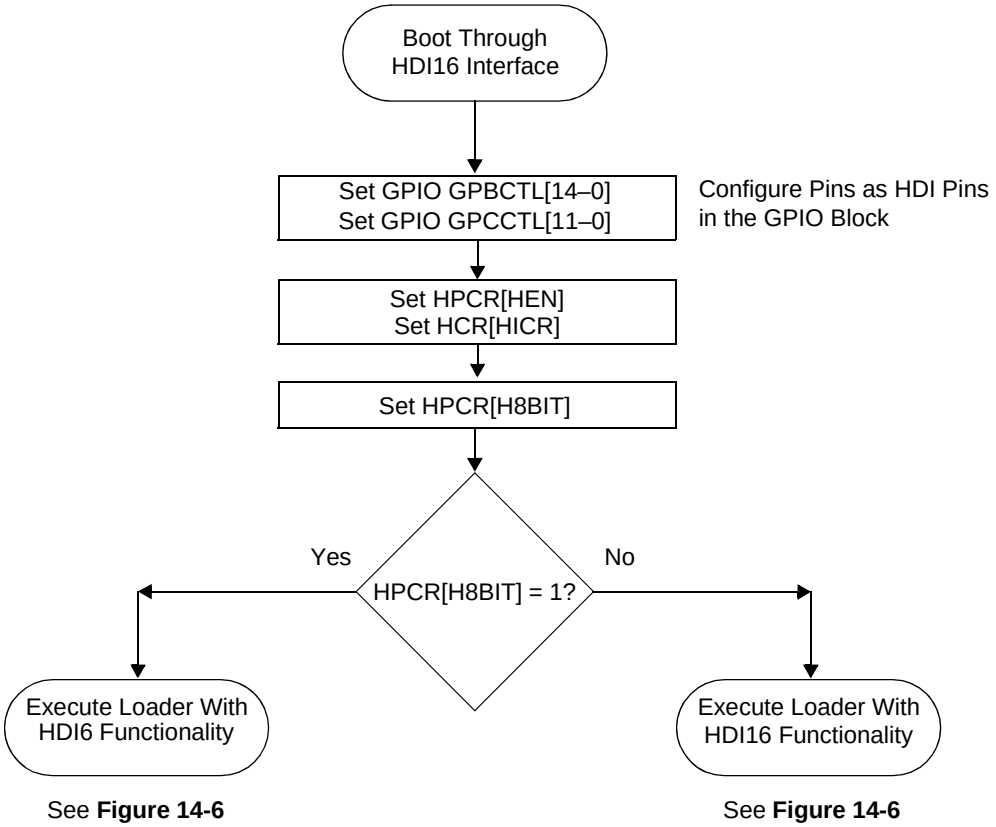


Figure 14-5. Host Port Loader

After the HDI port is initialized, the boot program loads the boot data records, stores the boot data at a given address, and performs error checking via checksums. **Figure 14-6** shows the flow of the host interface load procedure for both HDI8 and HDI16 functionality.

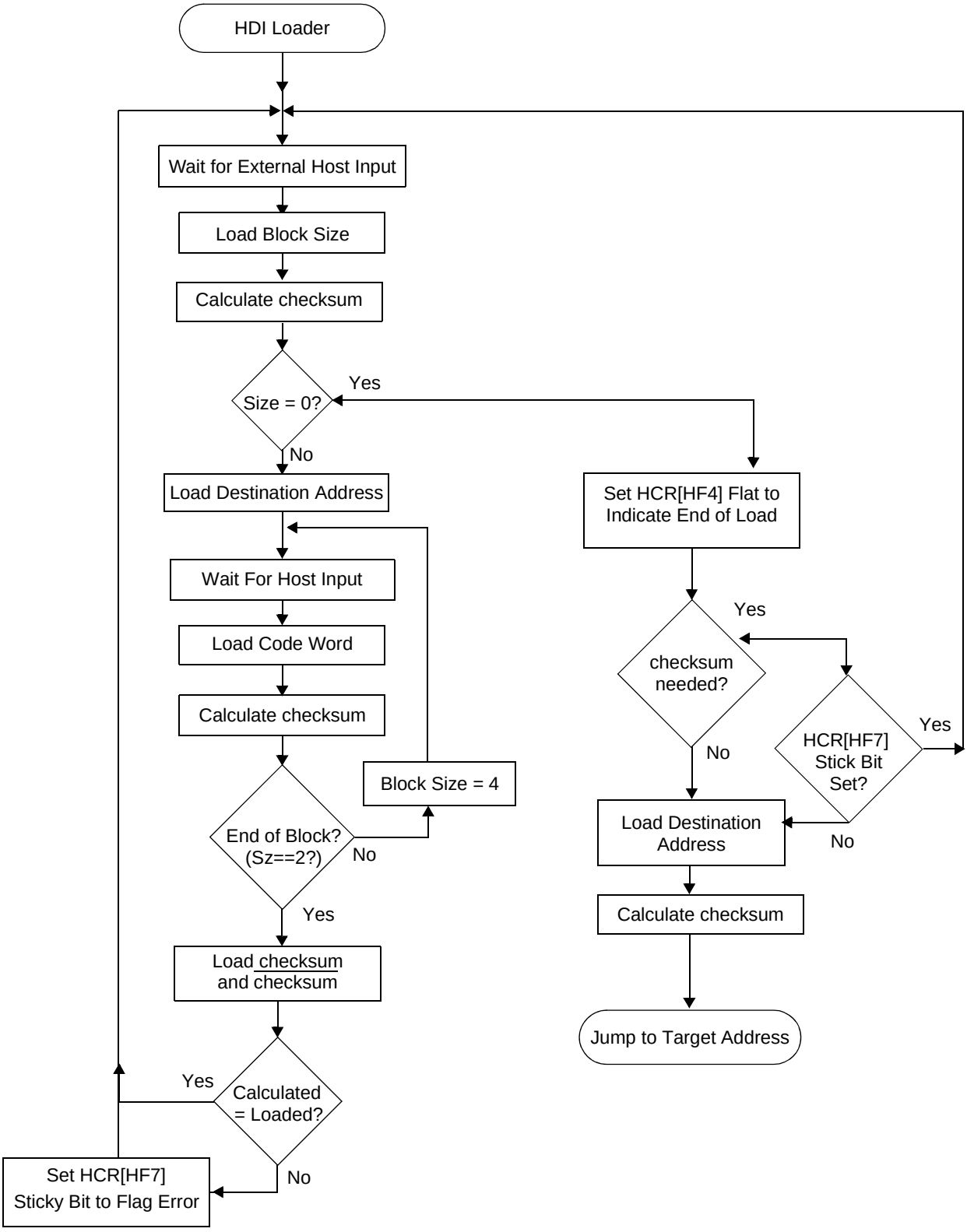


Figure 14-6. Bootloading Through HDI With 8- or 16-Bit Data Bus

14.3.5 HDI16 Boot Data Records

Each HDI16 boot data record contains the fields listed in **Table 14-7**. You must organize the boot data into records beforehand so that the boot program receives the data correctly through the HDI16 port.

Table 14-7. HDI Boot Record Fields (in order received by HDI)

Field Name	Size	Description
Block size	32 Bits	Contains the number of 16-bit boot data entries, N, within this record. The last two words are reserved for the final checksum values. The block size N must satisfy the following equation (where M is an integer): N = 4*M+2. This rule applies because the boot program always receives data from the HDI in 64-bit quantities. **Note:** The minimum number of data entries in a record is 2 (for the case where M = 0).
Load address	32 Bits	Specifies where the boot data is to be loaded in its destination memory. This value must be aligned on a 16-byte boundary.
Boot data entries	N x 16-bits	Contains the actual data values that are loaded into the destination memory. The number of values loaded, N, is the block size. These values usually contain the desired user program that is booted into the MSC711x device. **Note:** This data must be organized in Big-Endian format. That is, the most significant portion is loaded at the lower-order address.
$\overline{\text{Checksum}}$	16 Bits	Contains the expected value for the one's complement of the checksum.
Checksum	16 Bits	Contains the expected checksum for the boot record to be compared against the checksum calculated as described in **Section 14.3.6**, *Error Handling on Completion,* on page 14-17. The block size and load address are used in the checksum calculation.
Checksum comparison enable: • Enabled by the ICR[HF3] host flag. • See **Section 14.3.6**, *Error Handling on Completion,* on page 14-17.		

The record structure does not indicate the total number of records. Instead, the end of the data records is indicated by a special final record with a 32-bit block size of 0x00000000. **Table 14-8** shows the structure of this final record.

Table 14-8. Structure of the Final Record

Word	Description
1	0x0000
2	0x0000
3	Target Address — jump to this location when boot completes (most significant 16-bits)
4	Target Address — jump to this location when boot completes (least significant 16-bits)
5	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ including address
6	Checksum—XOR including address

Table 14-8. Structure of the Final Record (Continued)

Word	Description
7	0x0000
8	0x0000

14.3.5.1 HDI16 Boot Data Example

Table 14-9 shows three records containing boot data followed by a final record to indicate the end of the HDI16 boot data records. An application can contain any number of records as long as there is at least one record. The example in **Table 14-9** shows a generalized size for record 1, a block size of 2 for record 2, and a block size of 6 for record 3. The final record's target address indicates the address at which the program execution continues when the boot process completes. This address must be aligned on a 16-byte boundary.

Table 14-9. Record Structure of Boot Data Received Through HDI16 Port

Record	Word[1]	Description
1	1	Size of record 1 (most significant 16-bits)
	2	Size of record 1 (least significant 16-bits)
	3	Load address where data from record 1 is to be loaded (most significant 16-bits)
	4	Load address where data from record 1 is to be loaded (least significant 16-bits)
	5	Boot data: First 16-bit word
	...	**...**
	n	Boot data: Last 16-bit word
	n+1	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for record 1
	n+2	Checksum—XOR for record 1
2	1	Size of record 2 (most significant 16-bits)
	2	Size of record 2 (least significant 16-bits)
	3	Load address where data from record 2 is to be loaded (most significant 16-bits)
	4	Load address where data from record 2 is to be loaded (least significant 16-bits)
	5	Boot data: first 16-bit word
	6	Boot data: second 16-bit word
	7	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for record 2
	8	Checksum—XOR for record 2

Table 14-9. Record Structure of Boot Data Received Through HDI16 Port (Continued)

Record	Word[1]	Description
3	1	Size of record 3 (most significant 16-bits)
	2	Size of record 3 (least significant 16-bits)
	3	Load address where data from record 3 is to be loaded (most significant 16-bits)
	4	Load address where data from record 3 is to be loaded (least significant 16-bits)
	5	Boot data: First 16-bit word
	6	Boot data: Second 16-bit word
	7	Boot data: Third 16-bit word
	8	Boot data: Fourth 16-bit word
	9	Boot data: Fifth 16-bit word
	10	Boot data: Sixth 16-bit word
	11	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for record 3
	12	Checksum—XOR for record 3
Final Record	1	0x0000
	2	0x0000
	3	Target address — jump to this location when boot completes (most significant 16-bits)
	4	Target Address — jump to this location when boot completes (least significant 16-bits)
	5	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for last record
	6	Checksum—XOR for last record
	7	0x0000
	8	0x0000

Notes: **1.** Each word represents 16 bits.

When more than one code block is included in the source program data stream, word n + 5 contains the address of the second block, as shown in **Table 14-9**. The sequence repeats for subsequent blocks until the final block in the data stream arrives.

14.3.6 Error Handling on Completion

The SC1400 core calculates the checksum using data received for each block. The checksum is calculated by XORing the current word bit by bit with the result of XORing previous words. The value of bit *i* of the current result is equal to XORing bit *i* of the current word with bit *i* of the previous result. After the entire block is loaded, the calculated checksum is compared with the loaded checksum. When all blocks are loaded, the SC1400 core sets HCR[HF4] to notify the host. If the calculated checksum is not equal to the received checksum, the SC1400 core sets the HCR[HF7] bit. This bit is sticky. The ICR[HF3] bit indicates whether the result of the checksum check should be ignored.

The check procedure is as follows:

- If ICR[HF3] is set, HSR[HF3] is set and the SC1400 core checks the status of the HCR[HF7] sticky bit when the end of the block is reached:
  - If the HF7 bit is cleared, the loaded code is correct and the core jumps to the target address in the final boot record to execute the loaded code.
  - If the HF7 bit is set, the code is corrupted and should be reloaded. The core returns to the beginning of the routine and waits for the host to reload the boot routine. The flags are reset after the HDI16 reads the FIFO for the first time after reloading.
- If ICR[HF3] is cleared, the checksum check is not needed, and the SC1400 core ignores HCR[HF7] and jumps to the target address in the final boot record to execute the code.

14.3.7 Broadcast Boot Facility

The MSC711x broadcast boot facility is useful in a system when the host must boot several devices in the same way. The broadcast chip-select pin is ORed with the chip-select pin. To broadcast the commands to all devices, the host asserts the broadcast chip-select pin simultaneously for all MSC711x devices. To access a single MSC711x, the relevant chip-select pin is asserted. Refer to **Section 20.5.1**, *Host Port Chip Select Capability*, on page 20-8.

14.4 Booting From an I^{2}C Device

When an MSC711x device is configured to boot from the I^{2}C port, the boot program configures GPIO pins as I^{2}C pins. Then the MSC711x device initiates accesses to the I^{2}C module, downloading data to the MSC711x device. The I^{2}C port is configured as follows:

- PLL is disabled and bypassed so that the I^{2}C module is clocked with the IPBus clock.
- I^{2}C interface operates in master mode and polling is used.
- EPROM operates in slave mode.
- Clock divider is set to 128.
- Address of slave during boot is 0xA0.

The IPBus clock is internally divided to generate the bit clock, as follows:

- CLKIN must be <= 100 MHz
- PLL is bypassed.
- IPBus clock = CLKIN/2 <= 50 MHz.
- I^{2}C bit clock:
  - <= IPBus clock/I^{2}C clock divider
  - <= 50 MHz (max)/128
  - <= 390.6 KHz

This satisfies the maximum clock rate requirement of 400 kbps for the I^{2}C interface.

14.4.1 I2C Boot Procedure

When an MSC711x device boots through the I2C port, the boot data must be contiguous. That is, all I2C boot data records must be organized sequentially. The next record address feature is not supported. **Figure 14-7** shows the procedure used by the I2C boot code.

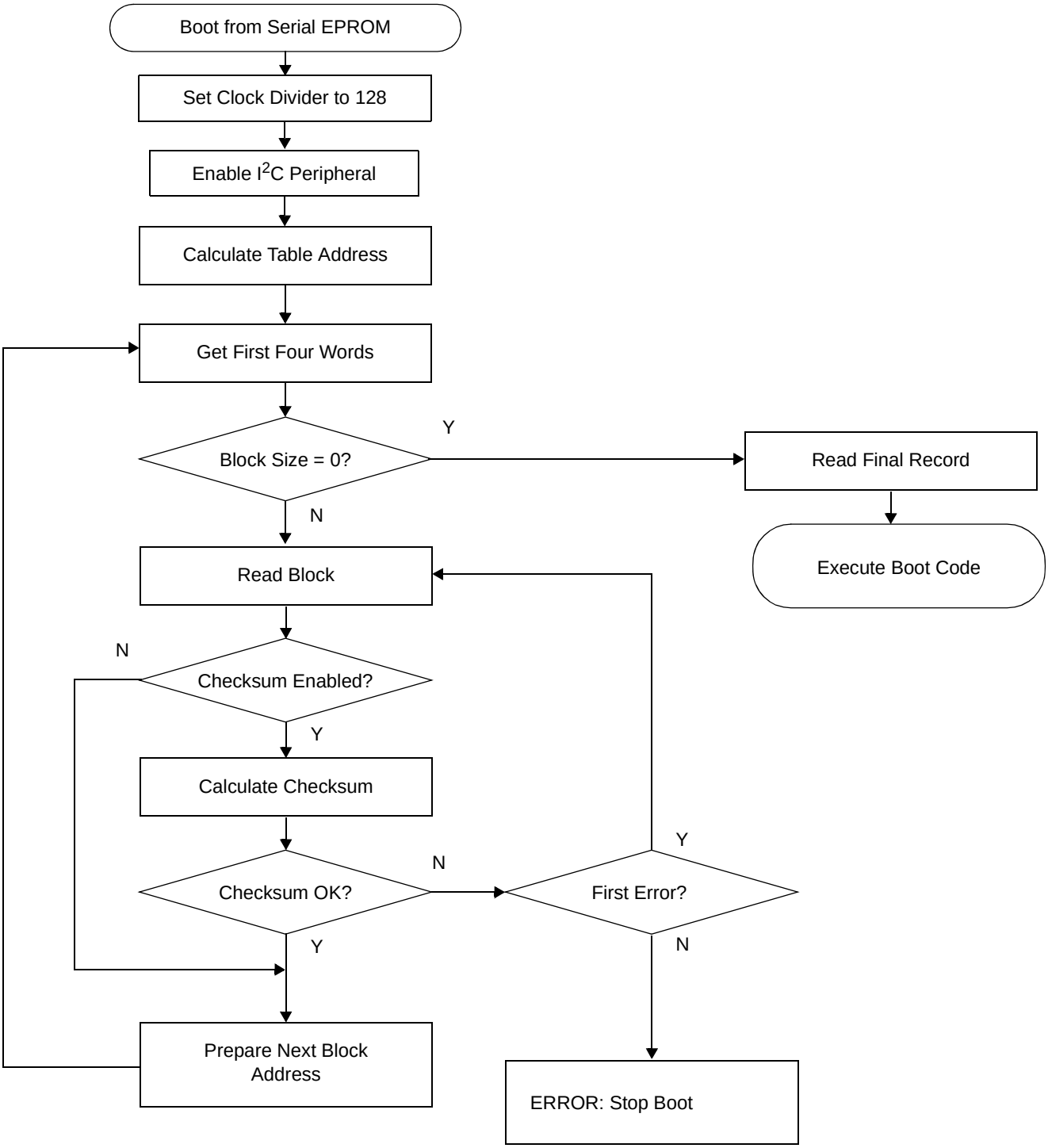


Figure 14-7. I2C Boot Load Procedure

14.4.2 I²C Boot Data Records

Each I²C boot data record contains the fields listed in **Table 14-10**. You must organize the boot data into records beforehand so that the boot program receives the data correctly through the I²C port.

Table 14-10. I²C Boot Record Fields In The Order Received by I²C Module

Field Name	Size	Description
Block size	16 Bits	The lowest 15 bits of this 16-bit value contains the number of bytes in the boot data entries, N, within this record. The block size includes four additional bytes used by the expected checksum fields. The block size must always be a multiple of 2. The MSB of this field is used to enable the checksum comparison. If this bit is set, the comparison is performed. **Note:** The minimum number of data entries in a record is 1. A record size of zero indicates the final record. The maximum block size is 32 KB.
Next record address	16 Bits	This field must always be set to 0x0000. The blocks must all be placed in sequential order. That is, the next record address feature is not supported.
Load address	32 Bits	Specifies where the boot data is to be loaded in destination memory. This value must be aligned on a 16-bit boundary.
Boot data entries	N × 16-bits	Contains the actual data values that are loaded into the destination memory beginning at the load address. These values usually contain the desired user program that is booted into the MSC711x device. **Note:** This data must be organized into big-endian format. That is, the most significant portion is loaded at the lower-order address.
checksum	16 Bits	Contains the expected checksum for this boot record, which is to be compared with the checksum calculated. The expected checksum is a bit-wise XOR of each boot data entry with the result of the XOR of the previous entries in the block. **Note:** The block size and load address are also included in the checksum calculation. Checksum comparison is disabled by clearing the record size field MSB.
$\overline{\text{checksum}}$	16 Bits	Contains the expected value for the one's complement of the checksum.
Note: Checksum comparison is enabled by the MSB of the 16-bit record size entry.		

When booting from the I²C port, the first boot record is read from address 0x00.

The record structure does not indicate the total number of records. Instead, the end of the data records is indicated by a special final record with a 16-bit block size of 0x0000. **Table 14-11** shows the structure of this final record.

Table 14-11. Structure of the Final Record

Word	Description
1	0x0000
2	0x0000
3	Target Address — jump to this location when booting completes (most significant 16-bits)
4	Target Address — jump to this location when booting completes (least significant 16-bits)

Table 14-11. Structure of the Final Record (Continued)

Word	Description
5	0x0000
6	0x0000
7	checksum—XOR for final record
8	$\overline{\text{checksum}}$—$\overline{\text{XOR}}$ for final record

14.4.2.1 I2C Boot Data Example

Table 14-12 shows three records containing boot data followed by a final record to indicate the end of the I2C boot data records. An application can contain any number of records as long as there is at least one record. The example in **Table 14-12** shows a generalized size for record 1, a block size of 2 for record 2, and a block size of 6 for record 3. The final record's target address indicates the address at which the program execution continues when booting completes. This address must be aligned on a 16-byte boundary.

Table 14-12. Record Structure of Boot Data Received Through the I2C Port

Record	Word[1]	Description
1	1	Size of record 1 (size in lowest 15 bits, MSB is the checksum enable)
	2	0x0000
	3	Load address where data from record 1 is to be loaded (most significant 16-bits)
	4	Load address where data from record 1 is to be loaded (least significant 16-bits)
	5	Boot data: First 16-bit word
	...	**...**
	n	Boot data: Last 16-bit word
	n+1	checksum—XOR for record 1
	n+2	$\overline{\text{checksum}}$—$\overline{\text{XOR}}$ for record 1
2	1	Size of record 2 (size in lowest 15 bits, MSB is the checksum enable)
	2	0x0000
	3	Load address where data from record 2 is to be loaded (most significant 16-bits)
	4	Load address where data from record 2 is to be loaded (least significant 16-bits)
	5	Boot data: first 16-bit word
	6	Boot data: second 16-bit word
	7	Checksum—XOR for record 2
	8	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for record 2

Table 14-12. Record Structure of Boot Data Received Through the I^2C Port (Continued)

Record	Word[1]	Description
3	1	Size of Record #3 (size in lowest 15-bits, MSB is the Checksum Enable)
	2	0x0000
	3	Load address where data from record 3 is to be loaded (most significant 16-bits)
	4	Load address where data from record 3 is to be loaded (least significant 16-bits)
	5	Boot data: first 16-bit word
	6	Boot data: second 16-bit word
	7	Boot data: third 16-bit word
	8	Boot data: fourth 16-bit word
	9	Boot data: fifth 16-bit word
	10	Boot data: sixth 16-bit word
	11	Checksum—XOR for record 3
	12	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for record 3
Final record	1	0x0000
	2	0x0000
	3	Target address — jump to this location when booting completes (most significant 16-bits)
	4	Target Address — jump to this location when booting completes (least significant 16-bits)
	5	0x0000
	6	0x0000
	7	checksum—XOR for last record
	8	$\overline{\text{checksum}}$—$\overline{\text{XOR}}$ for last record
Note: Each word represents 16 bits.		

When more than one block is included in the source program data stream, word n + 5 contains the address of the second block, as shown in **Table 14-12**. The sequence repeats for subsequent blocks until the final block in the data stream arrives.

14.4.3 Error Handling on Completion

The SC1400 core calculates the checksum using data received for each block. The checksum is calculated by XORing the current word bit by bit with the result of XORing previous words. The value of bit *i* of the current result is equal to XORing bit *i* of the current word with bit *i* of the previous result. After the entire block is loaded, the calculated checksum is compared with the loaded checksum. If a checksum comparison is enabled and a checksum error occurs:

- On the first failure within a record, reload the record.
- On the second failure within the same record, proceed as follows:
  - Program the BM1/GPIO/EVNT3 pin as a general-purpose output pin rather than a general-purpose input.
  - Toggle this pin in an infinite loop.

When no error occurs, the BM1/GPIO/EVNT3 pin is at a stable value because an external pull-up/pull-down resistor is required for this pin's default BM1 functionality.

14.4.4 Example Source Program

Figure 14-7 shows an example source program in the serial EPROM. The opcodes at the left of the instructions can either be generated by an S-record utility or extracted from a program listing file. The code moves the data 0xAAAA1111, 0xBBBB1111, 0xCCCC1111, and 0xDDDD1111 to addresses 0x8000, 0x8004, 0x8008, and 0x800C, respectively. During the boot process, the source program words are downloaded from the serial EPROM to the MSC711x device, starting at memory location 0x00001100.

The source program is divided into three source blocks. The first program block is 0x18 or 24 bytes long, and checksum comparison is enabled. The block size includes the source program and the two checksum values, so the first word is 0x8018. The second word is 0x0030, which indicates the address of the next block in the EPROM. This block is loaded at address 0x00001100 in the MSC711x, and this address is specified in the third and fourth words of the source block. Twenty bytes of the source program are followed by the checksum values 0x7532 and 0x8CAD for a total block size of 24 bytes.

The second source program block is also 24 bytes long, and checksum comparison is enabled as indicated in the first word with a value of 0x8018. The next block in the EPROM is located at address 0x0050. This block is loaded at address 0x00001114 in the MSC711x. Twenty bytes of source program are followed by the checksum values 0x6233 and 0x9DCC for a total block size of 24 bytes.

The third source program block is 0x30 or 48 bytes long, and checksum comparison is also enabled. The next block in the EPROM is located at address 0x0088. This block is loaded at address 0x00001128 in the MSC711x. The 44 bytes of source program are followed by the checksum values 0x8E35 and 0x71CA for a total block size of 48 bytes.

The end block specifies that no more source program blocks are to be sent. The first two words are zero to indicate that this block is the last one. After the program is downloaded, the MSC711x jumps to address 0x00001100 to start program execution The next two words also have a value of zero. Finally, the last two words are the checksum values.

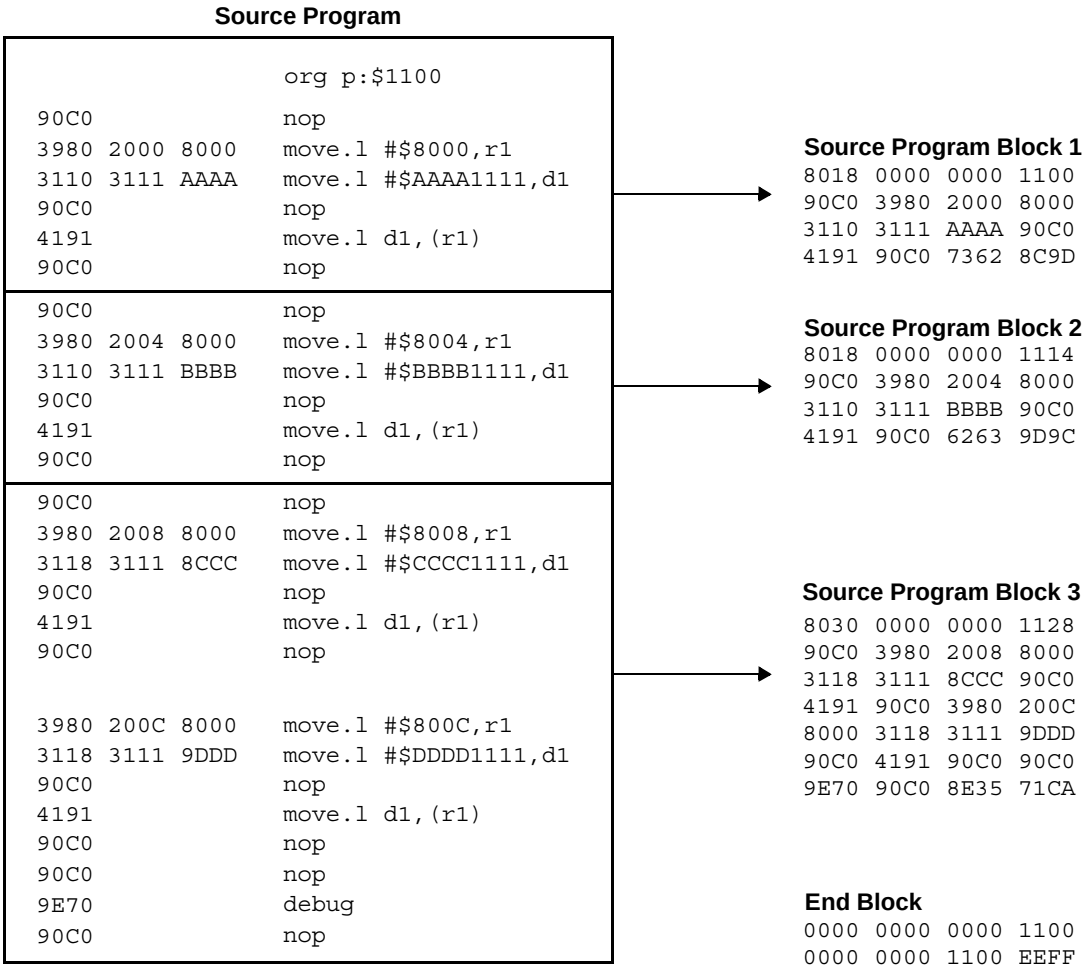


Figure 14-8. Example Source Program Blocks

14.4.5 Writing to an EPROM Over the I²C Port

Writing to the serial EPROM requires additional addressing information that must be sent with the data. In the example discussed here, it is the EPROM device address, followed by two 8-bit addresses and the data words. **Figure 14-9** shows an example connection between an MSC711x device and a serial EPROM

The write operation is shown in **Figure 14-10**. A start condition indicated by a logic 1 on the SDA line enables the EPROM for a read or write operation. Next is the 8-bit device address that consists of a mandatory 1, 0, 1, 0 for the most significant part, followed by the device address bits and the read/write bit for the least significant part. The address bits allow multiple EPROM devices to be connected on the same bus. These bits are compared with their corresponding hardwired input pins. If the EPROM address pins A_2, A_1, and A_0 are hardwired to logic 1, the address bits must also be set to logic 1. A read operation is initiated if the least significant bit of the device address is set while a write operation is initiated if this bit is cleared. Since the MSC711x device is writing to the EPROM to program the device, the R/W bit is cleared. The EPROM returns an acknowledgement of logic 0 when the device address is compared.

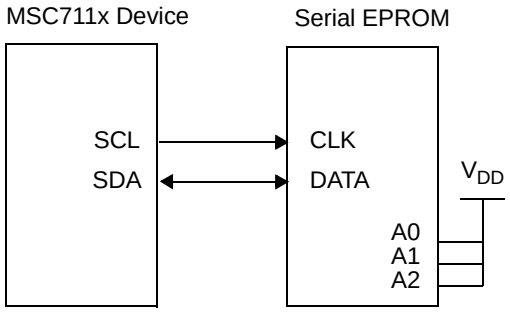


Figure 14-9. MSC711x to EPROM Connection

Following the device address, a write operation requires two 8-bit addresses. The first 8-bit address is the most significant part of the EPROM address, and the second is the least significant part. These two bytes are concatenated to indicate the address in the EPROM where the data is stored. The EPROM outputs a logic 0 on the SDA line after the first and second byte addresses. The data byte follows the first and second byte addresses. The EPROM acknowledges receipt of the data byte with a logic 0 on the SDA line. The EPROM address automatically increments after each data byte is received. Up to 63 more data words can be written to the EPROM. If more than 64 data bytes are transmitted, the data byte address rolls over, and previous data is overwritten. Therefore, to write more than 64 bytes, the device address and the first and second byte addresses must be reinitialized. The write sequence terminates with a stop condition, which is a logic 1 on the SDA line after the MSC711x device receives an acknowledgement that the EPROM received the last data byte.

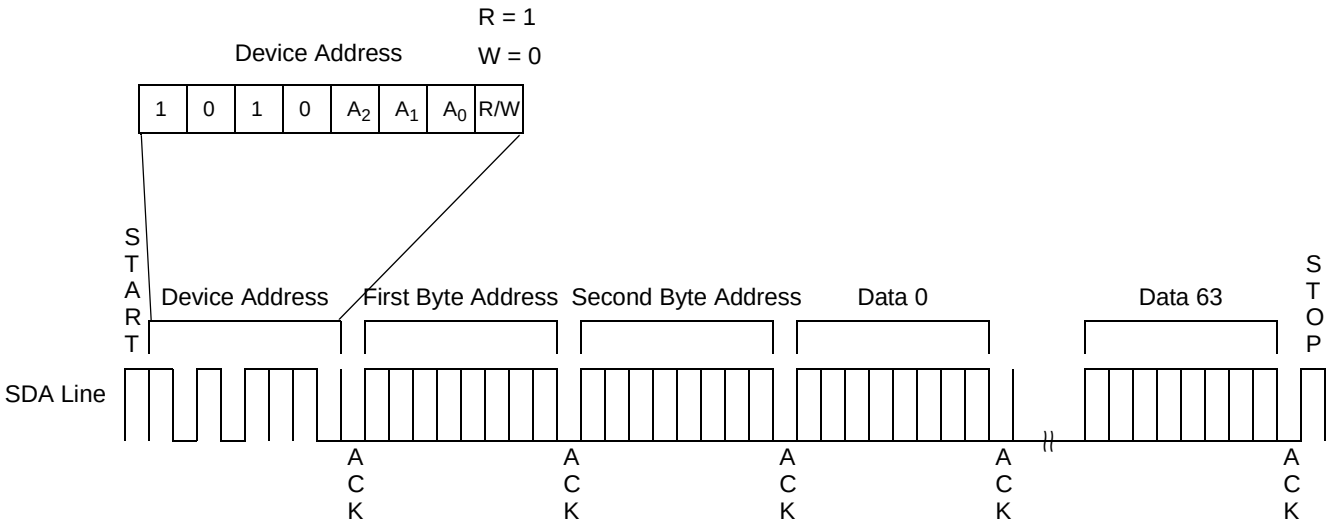


Figure 14-10. Write Operation

14.5 Booting from an SPI-Based Serial Flash or EEPROM

This section describes bootstrapping from serial peripheral interface (SPI) devices using GPIO pins and software routines in the boot ROM. When the SPI routines run in the boot ROM, the MSC711x is always configured as the SPI master. Booting through the SPI is supported for serial EEPROM devices and serial Flash devices. When a READ_ID instruction is issued to the serial memory device and the device returns a value of 0x00 or 0xFF, the routines for accessing a serial EEPROM are used, at a maximum frequency of 4 Mbps. Otherwise, the routines for accessing a serial Flash are used, and they can run at faster speeds. Booting is performed through one of two sets of pins:

- Main set: BM[2–3], HA3, and HCS2, which allow use of the PLL.
- Alternate set: UTXD, URXD, SDA, and SCL, which cannot be used with the PLL.

In either configuration, an error during SPI boot is flagged on the EVNT3 pin.

Note: During an SPI boot operation, the first 64 bytes of serial memory are reserved. The first block of boot data resides in address 0x40.

14.5.1 Main Set Pin Configuration

When booting through the SPI, the boot program configures several MSC711x pins as GPIO. See **Table 14-13**.

Table 14-13. Main Pin Set for SPI Boot Functionality

SPI Functionality	Pin	Input/Output	Comments
MOSI	BM3/GPIO	Output	—
MISO	$\overline{\text{HCS2}}$/GPIO	Input	—
SPICLK	BM2/GPIO	Output	—
SEL	HA3/GPIO	Output	Connects to the $\overline{\text{SS}}$ pin of the serial ROM device.
—	EVNT3/GPIO	Output	Indicates whether an error occurred during the boot process. This functionality is enabled/disabled in the first boot record.

If the SPI boot completes without error, the boot program reconfigures the pins as shown in **Table 14-14** before jumping to the specified user address.

Table 14-14. Pin Functionality Upon Completion of a Successful Boot

Pin	After Boot	Input/Output
BM3/GPIO	GPIO	Input
$\overline{\text{HCS2}}$/GPIO	$\overline{\text{HCS2}}$	Input
BM2/GPIO	GPIO	Input
HA3/GPIO	GPIO	Input
EVNT3/GPIO	GPIO	Input

14.5.2 Alternate Set Pin Configuration

One boot mode boots the MSC711x device from an SPI memory device through an alternate set of pins, as shown in **Table 14-15**.

Table 14-15. Pins Used for Software SPI Functionality — Alternate Pin Set

SPI Functionality	Pin	Input/Output	Comments
MOSI	UTXD/GPIO	Output	—
SEL	SDA/GPIO	Output	Connects to the $\overline{SS}$ pin of the serial ROM device.
SPICLK	URXD/GPIO	Output	—
MISO	SCL/GPIO	Input	—
—	EVNT3/GPIO	Output	Indicates whether an error occurred during the boot process. This functionality is enabled/disabled in the first boot record.

If the SPI boot completes without error, the boot program reconfigures the pins as shown in **Table 14-16** before jumping to the specified user address.

Table 14-16. Pin Functionality Upon Completion of a Successful Boot

Pin	After Boot	Input/Output	Comments
UTXD/GPIO	GPIO	Input	—
SDA/GPIO	GPIO	Input	—
URXD/GPIO	GPIO	Input	—
SCL/GPIO	GPIO	Input	—
EVNT3/GPIO	GPIO	Input	—

14.5.3 SPI Boot Loader Procedure

As **Figure 14-11** shows, the procedure for SPI booting is similar to the I^2C procedure.

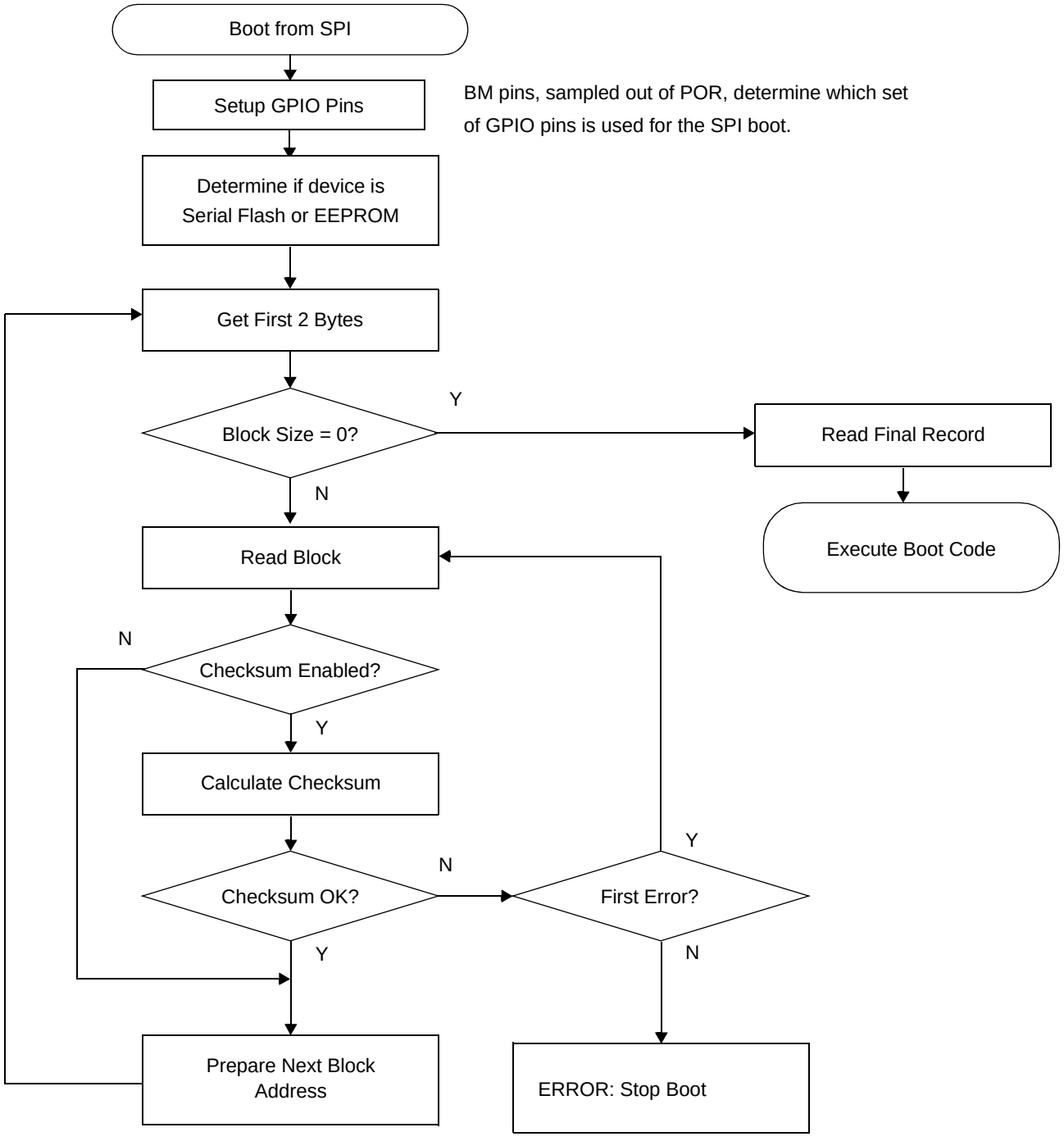


Figure 14-11. SPI Boot Load Procedure

14.5.4 SPI Boot Data Records

The source program that is downloaded from the serial Flash/EEPROM to the MSC711x device must be organized as a set of records. Each SPI boot data record contains the following information:

- Size of the boot data record (16-bits):
  - Checksum comparison enable in bit 15.
  - Lowest 15 bits specify the number of bytes in the boot data record.
- Next record address that specifies the address of the next boot record (32 bits).
- Load address where the data in the boot data record is to be stored (32 bits).
- Boot data entries containing the data to load into the device (N × 16-bits).
- Checksums to verify correct loading of data (two 16-bit checksums).

Organizing the data into this record structure ensures that the boot program knows which 16-bit values contain block size, destination address, etc. **Table 14-17** shows how boot data must be organized into records beforehand by the user to be received correctly by the boot program through the SPI port. The fields of the boot record are described in detail in **Table 14-17**:

Table 14-17. SPI Boot Record Fields (in order received by SPI)

Field Name	Size	Description
Block size	16 bits	The lowest 15 bits of this 16-bit value contain the number of bytes in the boot data entries, N, within this record. The block size includes the number of bytes in the Boot data entries, N, within this record as well as the four additional bytes used by the expected checksum fields. The MSB of this field enables the Checksum comparison. If this bit is set, the comparison is performed. The minimum number of data entries in a record is 2 (where N = 0). The Maximum block size is 32 KB.
Next record address	32 bits	If set to 0x0000, the next boot record follows sequentially. Otherwise, this field provides the address of the next record.
Load address	32 bits	Specifies where the boot data is to be loaded in destination memory. This value must be aligned on a 32-bit boundary.
Boot data entries	N × 8 bits	Contains the actual data values that are loaded into the destination memory. The number of values loaded, N, is the block size. These values usually contain the user program that is booted into the MSC711x device. This data must be organized in big-endian format. That is, the most significant portion is loaded at the lower-order address.
Checksum	16 bits	Contains the expected checksum for this boot record to be compared against the calculated checksum. The expected checksum is a bit-wise XOR of each boot data entry with the result of the XOR of the previous entries in the block. The block size and load address are used in the checksum calculation. Checksum comparison is disabled when the MSB of the record size field is cleared.
$\overline{\text{Checksum}}$	16 bits	Contains the expected value for the one's complement of the Checksum (see above).
Note: Checksum comparison is enabled by the MSB of the 16-bit record size entry.		

When booting from the SPI port, the first boot record is read from address 0x40.

14.5.4.1 Format of the Last Boot Record

The total number of records is never indicated by the record structure. Instead, the end of the data stream is indicated by a final record with a 16-bit block size of 0x0000. This special final record is structured as shown in **Table 14-18**.

Table 14-18. Structure of the Final Record

Word	Description
1	0x0000
2	0x0000
3	Target Address — jump to this location upon completing boot (most significant 16-bits)
4	Target Address — jump to this location upon completing boot (least significant 16-bits)
5	0x0000
6	0x0000
7	Checksum—XOR including address
8	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ including address

14.5.4.2 SPI Boot Data Example

This example in **Table 14-19** shows three records containing boot data followed by a final record to indicate the end of the SPI boot data records. An application must contain at least one record and can contain any number of records (1, 2, 3, ...). The example in **Table 14-19** shows a generalized size of $(n - 5) \times 2$ bytes for Record 1, a block size of 4 bytes for Record 2, and a block size of 12 bytes for Record 3. The final record's target address is the address at which program execution continues when booting completes. This address must be aligned on a 16-byte boundary.

Table 14-19. Record Structure of Boot Data Received Through Software SPI

Record	Word[1]	Description
1	1	Size of Record 1 (size in lowest 15-bits, MSB is the checksum enable)
	2	Next block address (most significant 16 bits)
	3	Next block address (least significant 16 bits)
	4	Load address where data from Record 1 is to be loaded (most significant 16 bits)
	5	Load address where data from Record 1 is to be loaded (least significant 16 bits)
	6	Boot data: First 16-bit word
	...	...
	n	Boot Data: Last 16-bit word
	n+1	Checksum—XOR for Record 1
	n+2	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for Record 1

Table 14-19. Record Structure of Boot Data Received Through Software SPI (Continued)

Record	Word[1]	Description
2	1	Size of Record 2 (size in lowest 15 bits, MSB is the checksum enable)
	2	Next block address (most significant 16 bits)
	3	Next block address (least significant 16 bits)
	4	Load address where data from Record 2 is to be loaded (most significant 16 bits)
	5	Load address where data from Record 2 is to be loaded (least significant 16 bits)
	6	Boot data: first 16-bit word
	7	Boot data: second 16-bit word
	8	Checksum—XOR for Record 2
	9	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for Record 2
3	1	Size of Record 3 (size in lowest 15 bits, MSB is the checksum enable)
	2	Next block address (most significant 16 bits)
	3	Next block address (least significant 16 bits)
	4	Load address where data from Record 3 is to be loaded (most significant 16 bits)
	5	Load address where data from Record 3 is to be loaded (least significant 16 bits)
	6	Boot data: first 16-bit word
	7	Boot data: second 16-bit word
	8	Boot data: third 16-bit word
	9	Boot data: fourth 16-bit word
	10	Boot data: fifth 16-bit word
	11	Boot data: sixth 16-bit word
	12	Checksum—XOR for Record 3
	13	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for Record 3
Final Record	1	0x0000
	2	0x0000
	3	Target Address — jump to this location upon completing boot (most significant 16-bits)
	4	Target Address — jump to this location upon completing boot (least significant 16-bits)
	5	0x0000
	6	0x0000
	7	Checksum—XOR for last record
	8	$\overline{\text{Checksum}}$—$\overline{\text{XOR}}$ for last record

Note: Each word represents 16 bits.

Note: This boot record format is different than the format for the SPI and HDI16 ports.

The bootloader routine expects at least one code block. When more than one block is included in the source program data stream, words 2 and 3 contain the address of the second block, as shown in **Table 14-9**. The sequence repeats for subsequent blocks until the final block in the data stream.

14.5.5 SPI Boot Error Handling

If a checksum comparison is not enabled, the entire record is processed as if no error has occurred. If checksum comparison is enabled and a checksum error occurs:

- On the first failure within a record, reload the record.
- On the second failure within the same record:
  - Program the BM1/GPIO/EVNT3 pin as a general-purpose output pin rather than as a general-purpose input.
  - Toggle this pin in an infinite loop

When no error occurs, the BM1/GPIO/EVNT3 pin is at a stable value because of an external pull-up/pull-down resistor required for this pin's BM1 functionality.

14.5.6 User Access to SPI Routines

After booting completes, an application can call the SPI emulation routines within the boot ROM. The MSC711x device must operate as the SPI master. Both serial Flash and serial EEPROM devices are accessible in this mode.

Event Port 15

The event port manages different levels of events on an MSC711x device, including SC1400 core-level, device-level, and external events. The event port interacts with the debug port breakpoint unit in the OCE10 on-chip emulator and the internal timers to manage external events on the EVNT pins or internal events such as DMA activity or ICache misses. The event port performs the following tasks:

- Selects events to output on the EVNT pins:
  - Directly output event onto a pin.
  - Set a value on a pin.
  - Toggle the value on a pin.
  - Set/Reset operation on a pin.
- Selects events to send to the debug port to:
  - Enable an SC1400 breakpoint unit.
  - Generate an emulator event.
- Selects events to send to the timer module for:
  - Event counting.
  - Gated counting.
  - Delaying an event for further use by the event port.
- Uses the EVNTx pins to:
  - Clock the timer module.
  - Output a clock from timer module A.
- Triggers on a preprogrammed sequence of events.
- Selects events for generating interrupts or DMA requests.

As **Figure 15-1** shows, inputs are gathered from various places, combined as programmed, and result in a desired action such as an interrupt, DMA transfer, or halt of the core.

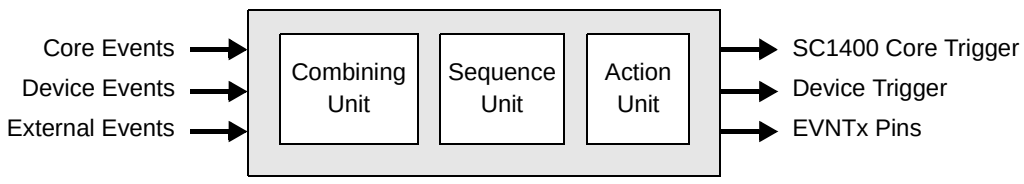


Figure 15-1. Event Port Operation

Following are examples of event port capabilities:

- Timer operations using the timer module described in **Chapter 21**, *Timers Module*:
  - Count transitions on an EVNT pin.
  - Measure the width of a pulse on an EVNT pin.
  - Drive a counter output onto an EVNT pin.
  - Count the number of ICache misses.
  - Count the time spent servicing ICache misses.
  - Set the EVNT4 pin when a timer expires or a TDM Rx interrupt is received.
  - Use one-shot capabilities on an internal chip signal or EVNT pin.
  - Measure the time required to lock the PLL.
- Peripheral interaction:
  - Toggle an EVNT pin on a TDM Tx interrupt or a DMA transfer on a particular channel.
- DMA interaction:
  - Assert a signal when DMA service is requested and deassert the signal when DMA service completes.
  - Count the number of cycles that a DMA channel is preempted.
  - Count the number of cycles from a DMA channel service request until the requested service completes.
  - Trigger if the number of cycles that a DMA channel is preempted above a threshold.
  - Assert the EVNT3 pin and then the EVNT2 pin to initiate a DMA transfer.
  - Generate an event port interrupt when a DMA transfer on channels 3 or 17 completes.
  - Measure the time between the completion of DMA channel 2 and the start of channel 6.
- Debug unit (emulator) interaction:
  - Use an EVNT pin to enable a breakpoint unit.
  - Enter Debug mode when a DMA data transfer completes.
  - Delay an input event using a timer and pass the delayed event to the debug unit breakpoint logic. This action requires two event multiplexers.
  - Use one event to enable trace buffer capture within the emulator and a second event to disable it.
  - Set an EVNT pin upon an HDIRxF interrupt and clear it upon an emulator breakpoint.
- Operations with an external host:
  - Initiate a DMA transfer by writing a value of 1 to a particular bit.
  - Assert an EVNT pin and write a value of 1 to a bit to halt the SC1400 core.
- Miscellaneous operations:
  - When the rising edge of the EVNT2 pin is detected, set a sticky bit for software to read.
  - Generate an interrupt or halt the SC1400 core if the PLL loses lock.
  - Switch the crossbar to its alternate priority register set if the PLL loses lock.
  - Wake up the SC1400 core from Stop mode when a DMA data transfer completes.
  - Wake up the SC1400 core from Stop mode when a TDM Rx interrupt occurs.

15.1 Event Port Architecture

The event port consists of eight event multiplexers that connect in a cascade, as shown in **Figure 15-2**. Each multiplexer can work independently, with up to eight independent event detections and actions, or the multiplexers can trigger in a sequence, each receiving a trigger from the preceding multiplexer and passing a trigger to the next one in the cascade.

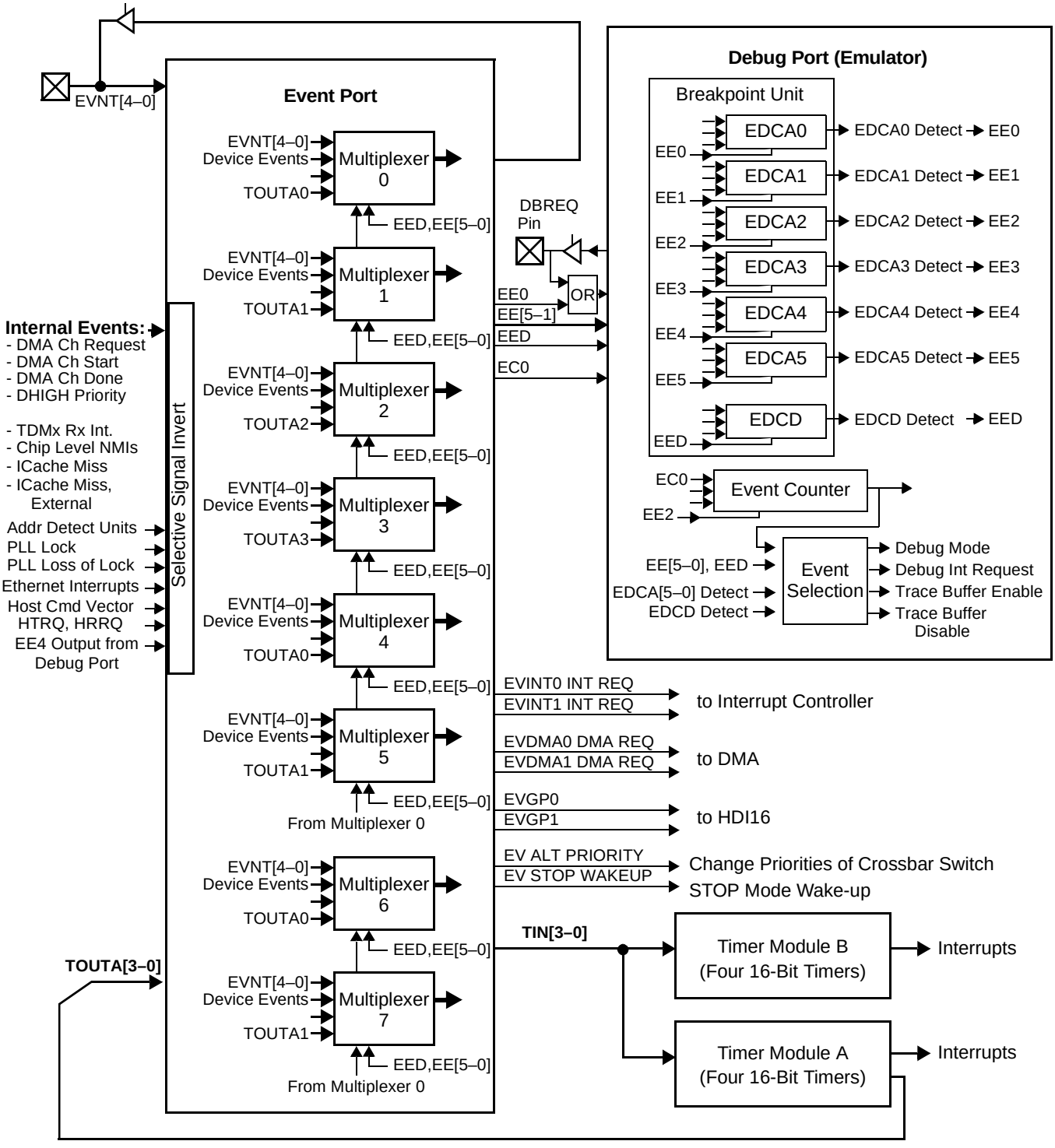


Figure 15-2. System-Level View of the Event Port

MSC711x Reference Manual, Rev. 1

The event multiplexers manage interaction between device-level timers, the SC1400 debug unit, and the EVNTx pins—all of which can send inputs to or receive outputs from the multiplexers. This section describes how the event port works at a system level, interacting with the debug port (emulator), EVNT pins, timer modules, interrupt controller, DMA controller, and the HDI16 host port. **Figure 15-3** shows the structure of one event multiplexer, which receives the signals of device-level events from various MSC711x modules and combines these signals as programmed in the application: performing OR, AND, set, invert, toggle, or set-reset operations on the selected signals. The event multiplexer performs a selectable event port action using a trigger:

- Drives one of the EVNTx pins.
- Generates an event port interrupt request.
- Passes the trigger directly to the timer module.
- Enables an emulator detection module.
- Performs an emulator action.

The event multiplexers can be configured so that an external host can trigger them via the HDI16 Command Vector Register (CVR) (see **page 20-46**). CVR[EVIN1] provides the first trigger, and CVR[EVIN0] provides a second. Therefore, an external host can trigger events such as DMA data transfers in an MSC711x device.

Each event multiplexer has an associated EVCTL[EMUX] bit (**page 15-27**) to provide status on whether an event multiplexer has triggered or forced a trigger to occur. The EMUX bits are described in **Section 15.8**, *Event Port Programming Model*, on page 15-26.

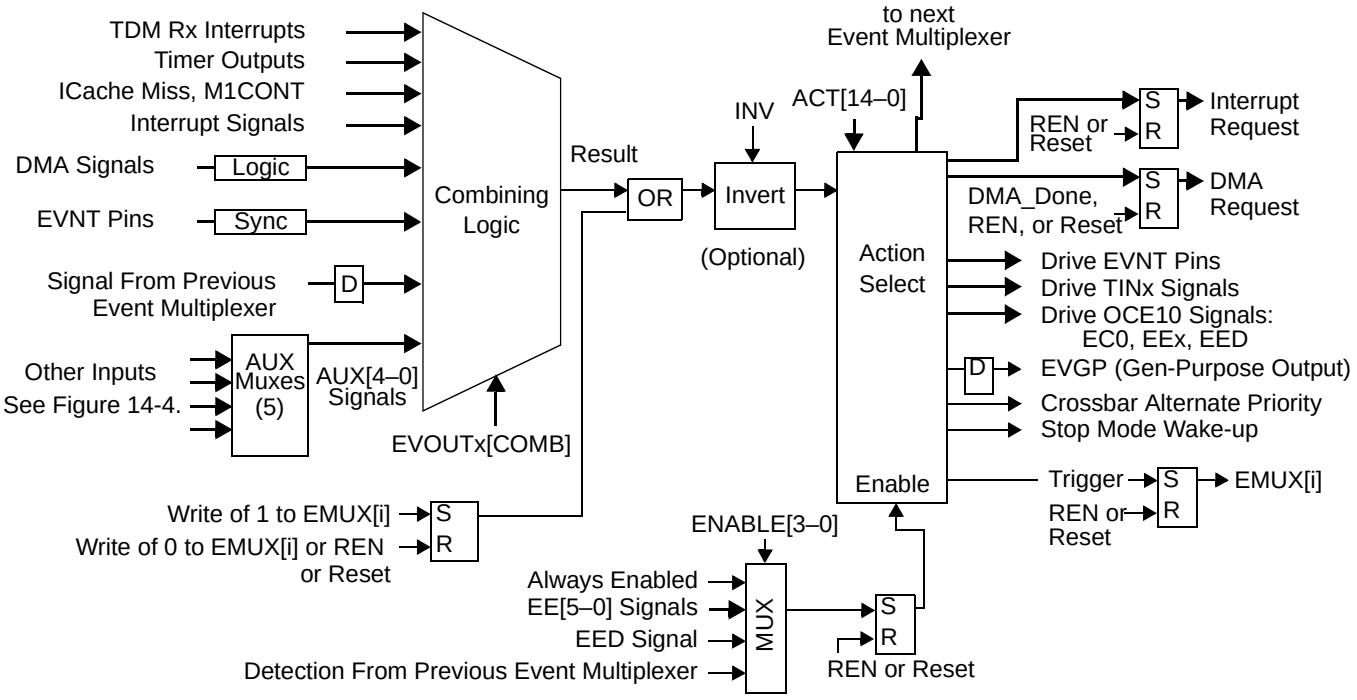


Figure 15-3. Block Diagram of One Event Multiplexer

Table 15-1, **Table 15-2**, and **Table 15-3** summarize the input, output, and enable signals associated with each event multiplexer.

Table 15-1. Inputs Associated with Each Event Multiplexer

Event Multiplexer	Event Multiplexer Inputs					
	EVNT Pins	DMA Channels	Timer Outputs (General Case)	Timer Outputs (Direct Connect)	Previous Event Multiplexer	Other Signals
Multiplexer 0	All	0–31	TOUT[3–0] from timer module A	TOUT0 from timer module A	Multiplexer 1	See notes.
Multiplexer 1	All	0–31	TOUT[3–0] from timer module A	TOUT1 from timer module A	Multiplexer 2	See notes.
Multiplexer 2	All	0–31	TOUT[3–0] from timer module A	TOUT2 from timer module A	Multiplexer 3	See notes.
Multiplexer 3	All	0–31	TOUT[3–0] from timer module A	TOUT3 from timer module A	Multiplexer 4	See notes.
Multiplexer 4	All	0–31	TOUT[3–0] from timer module A	TOUT0 from timer module A	Multiplexer 5	See notes.
Multiplexer 5	All	0–31	TOUT[3–0] from timer module A	TOUT1 from timer module A	Multiplexer 6	See notes.
Multiplexer 6	All	0–31	TOUT[3–0] from timer module A	TOUT2 from timer module A	Multiplexer 7	See notes.
Multiplexer 7	All	0–31	TOUT[3–0] from timer module A	TOUT3 from timer module A	Multiplexer 0	See notes.

Notes: **1.** Other Inputs to the Event Multiplexers include:
- Request to initiate a DMA transfer on a selected DMA channel.
- Start of a selected DMA channel.
- Completion of a selected DMA channel.
- Preemption during execution of a selected DMA channel.
- EE4 signal from the OCE10 emulator.
- M1 contention signal.
- Cache miss signal; Cache miss to external memory signal.
- Non-maskable request generated from a source outside the SC1400 core.
- Interrupt signals selectable in the interrupt controller.
- Interrupt signals from TDM0, TDM1, and Ethernet MAC.
- Address detection signals.
- Bits 15 and 14 of the HDI16 Host Command Vector Register (HCVR).

2. When any EVNT pin is used as an input, the signal first passes through a synchronizer, introducing a delay of up to two core clocks. When a previous event multiplexer is used as an event multiplexer input, the signal first passes through a latch, delaying the signal by one AHB clock.

3. When the output of a previous event multiplexer is used as an event multiplexer input, the output first passes through a latch, delaying the signal by one AHB clock.

Table 15-2. Outputs Associated with Each Event Multiplexer

Event MUX	Event MUX Outputs						
	EVNT Pins	Event Port Interrupts	Event Port DMA Transfers	Event Port General-Purpose Outputs	Timer Inputs (General Case)	Timer Inputs (Direct Connect)	OCE10 Input Signals
Multiplexer 0	All	All	All	All	TIN[3–0]	TIN0 to both timer modules	EE[5–0], EC0
Multiplexer 1	All	All	All	All	TIN[3–0]	TIN1 to both timer modules	EE[5–0], EED

Table 15-2. Outputs Associated with Each Event Multiplexer (Continued)

Event MUX	Event MUX Outputs						
	EVNT Pins	Event Port Interrupts	Event Port DMA Transfers	Event Port General-Purpose Outputs	Timer Inputs (General Case)	Timer Inputs (Direct Connect)	OCE10 Input Signals
Multiplexer 2	All	All	All	All	TIN[3–0]	TIN2 to both timer modules	EE[5–0],EC0
Multiplexer 3	All	All	All	All	TIN[3–0]	TIN3 to both timer modules	EE[5–0], EED
Multiplexer 4	All	All	All	All	TIN[3–0]	TIN0 to both timer modules	EE[5–0],EC0
Multiplexer 5	All	All	All	All	TIN[3–0]	TIN1 to both timer modules	EE[5–0], EED
Multiplexer 6	All	All	All	All	TIN[3–0]	TIN2 to both timer modules	EE[5–0], EC0
Multiplexer 7	All	All	All	All	TIN[3–0]	TIN3 to both timer modules	EE[5–0], EED

Notes: **1.** Other outputs from the event multiplexers include:
– Wake up the SC1400 core from Stop mode (multiplexer 0 only).
– Switch the crossbar to alternate priority.
– Outputs are readable by an external host in bits 15 and 14 of the HDI16 ISR via the general-purpose outputs.

Table 15-3. Enable Signals Associated with Each Event Multiplexer

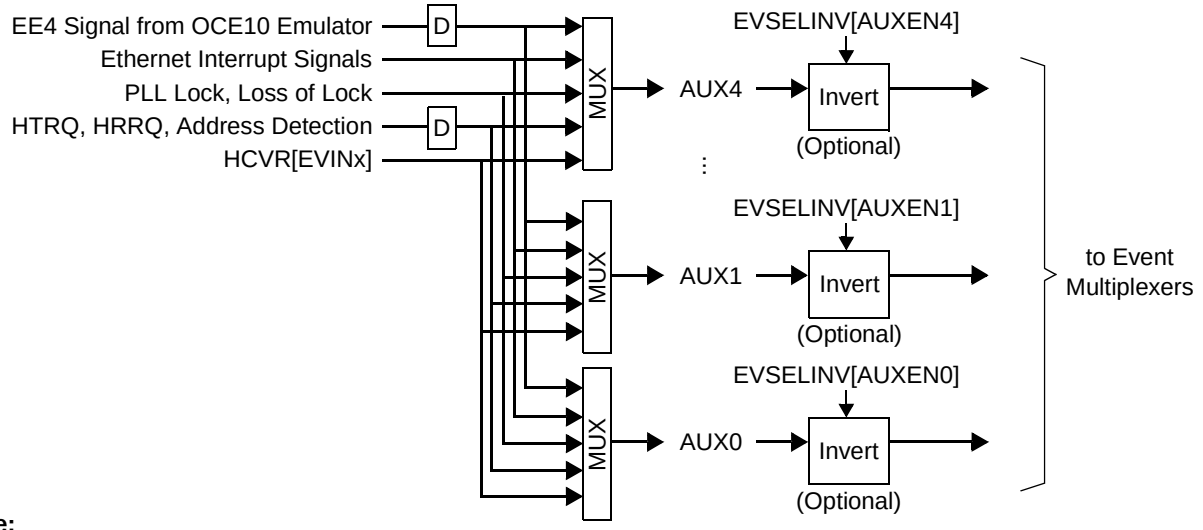
Event Multiplexer	Event Multiplexer Enables			Comments
	from Multiplexers	Emulator Outputs	Qualified Emulator Outputs	
Multiplexer 0	Multiplexer 1	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 1	Multiplexer 2	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 2	Multiplexer 3	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 3	Multiplexer 4	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 4	Multiplexer 5	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 5	Multiplexer 6	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 6	Multiplexer 7	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	—
Multiplexer 7	Multiplexer 0	EE0, EE1, EE2, EE3, EE4, EE5	EE[0–5] && with EED	When sequencing from an event multiplexer, receives its enable from event multiplexer 0

15.2 Multiplexer Inputs

Inputs enter the event port directly into the event multiplexer, or they are preprocessed as a set of auxiliary multiplexers and then sent to the event multiplexer. When signals from the device first enter the event port, they can optionally be inverted before the event port uses them. This optional inversion is individually selectable for each input via the EVSELINV register (see **Table 15-8**, *EVSELINV Bit Descriptions*, on page 15-30. This section covers topics pertaining to the multiplexer inputs, including restrictions on usage.

15.2.1 Auxiliary Input Operation

The EVCTL[AUX] bits provide two additional user-configurable inputs to all eight event multiplexers, and these inputs are selectable from several different sources. **Figure 15-4** shows the generation of the AUX[4–0] signals shared by all the event multiplexers. The selection is programmed in the EVCTL register.



Note:

1. EE4 signal is clocked with EIRQ clock
2. HTRQ, HRRQ, ADU signals are clocked with IPBus clock

Figure 15-4. Generating the AUX[4–0] Input Signals

The PLL lock signal comes from the CLKCTL[LCK] bit, and the loss of lock signal is sent to the event port. When the EE4 signal is used as an input, the HLTREQ[ITCCD] bit must be cleared.

15.2.2 Direct Connection Modes

For highest frequency operation, the event multiplexers allow direct connections between a peripheral such a timer and event pins, bypassing much of an event multiplexer's logic, as follows:

- Directly connect an event pin, EVNT[0–4], to a timer input, TINx.

- Directly connect a timer output, TOUTx, to an event pin, EVNTx.

When timer modules and the EVNTx pins are directly connected, each connection uses one event multiplexer. There are restrictions on which signals can be connected to which pins, and which multiplexer is used for the connection. **Table 15-4** and **Table 15-5** summarize the allowed combinations.

Table 15-4. EVNTx Pin → TINx Signal

Input Pin (Source)	Timer Signal (Destination)	Must be programmed in:
EVNT0	TIN0	Event multiplexer 0 or 4
EVNT1	TIN1	Event multiplexer 1 or 5
EVNT2	TIN2	Event multiplexer 2 or 6
EVNT3	TIN3	Event multiplexer 3 or 7
Note: Applies to timer modules A and B.		

Table 15-5. TOUTAx Signal → EVNTx Pin

Timer Signal (Source)	Output Pin (Destination)	Must be programmed in:
TOUT0	EVNT0	Event multiplexer 0 or 4
TOUT1	EVNT1	Event multiplexer 1 or 5
TOUT2	EVNT2	Event multiplexer 2 or 6
TOUT3	EVNT3	Event multiplexer 3 or 7
Note: Applies to timer module A only.		

The direct connect modes are programmed via the EVOUTx[DEVNT] bits (**page 15-34**), and they take precedence over all other programming for an event multiplexer's input sources and output actions. If an event multiplexer is programmed for one of the direct connect modes, no other input sources are selected/used and no other output actions are selected/performed. For information on connecting the EVNTx signals to the debug port EE0, EED, and EC0 signals, see **Section 15-6**, *Combining Logic in an Event Multiplexer*, on page 15-11.

When used as an input to the event port, each EVNT pin is synchronized with a two-stage synchronizer clocked off EIRQ clock, which runs at the frequency of the core clock. This synchronizer can introduce a delay on these pins of up to two core clocks. Also, when the EVNT pins are used as inputs, the HLTREQ[ITCCD] bit must be reset (**page 11-28**).

15.2.3 DMA Input Source Selection

Each event multiplexer can specify one DMA input source to monitor use of the EVINx[DMACH] bits (**page 15-32**). When a DMA channel is selected, it is also necessary to

specify, via the EVINx[DMATYP] bits (**page 15-32**), which type of DMA event is selected as an input source:

- DMA channel request, which is asserted when the selected channel requests service from the DMA controller.
- DMA channel start, which is asserted when the selected channel begins operation in the DMA controller.
- DMA channel completed, which is asserted when the selected DMA channel finishes the data transfer.
- DMA channel deprioritization, which is asserted when the selected DMA channel loses priority. This signal indicates the amount of time the channel is preempted.

In all cases, the DMAEN bit must be set in the corresponding event multiplexer. For example, in **Figure 15-5**, the desired channel is DMA channel 3, and DMA channels 1 and 0 have a higher priority than channel 3:

1. DMA channel 0 is transferring data.
2. Channel 3 issues a DMA channel request (EVINx[DMATYP] = 000).
3. Channel 0 completes (DMATYP = 010), and channel 3 begins (DMATYP = 001).
4. Channel 1 preempts channel 3 (DMATYP = 011).
5. Channel 1 completes (DMATYP = 010) and channel 3 resumes (DMATYP = 001).
6. Channel 1 preempts channel 3 (DMATYP = 011).
7. Channel 1 completes (DMATYP = 010) and channel 3 resumes (DMATYP = 001).
8. Channel 3 completes (DMATYP = 010).

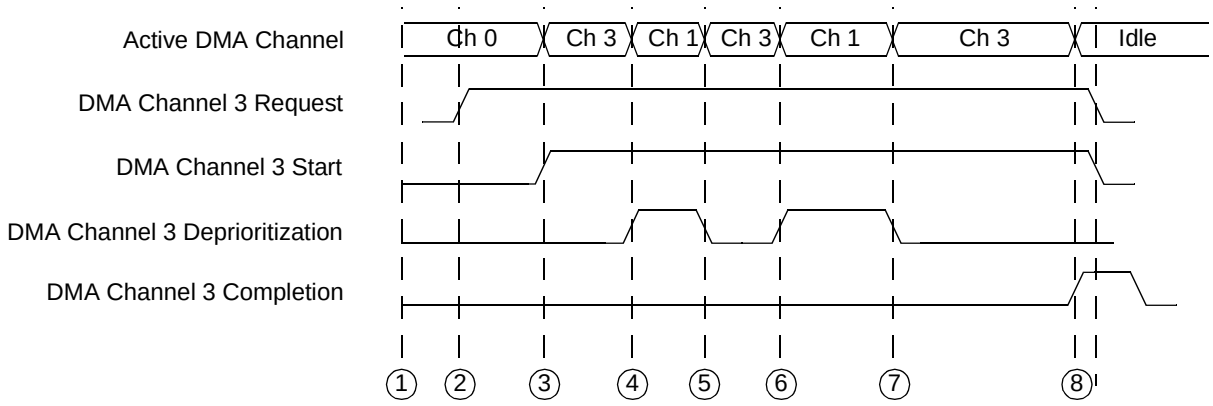


Figure 15-5. DMA Event Types

The pulses generated for channel request and channel start are asserted when the event is detected and deasserted one IPBus clock after channel completion is detected. The channel completion pulse is two IPBus clocks wide. It is often useful to measure the duration of the DMA transfer from beginning to end by sending the channel request or channel start waveform to a timer. It can

also be useful to measure the time the channel is preempted by sending the DMA channel deprioritization signal to a timer.

The different DMA event types (channel request, start, completed, and deprioritization) are available for both hardware and software-initiated DMA channels. For hardware-initiated channels, the event multiplexer uses signals from the requesting source: DREQ, DACK, and DDONE. However, for software-initiated channels, no signals indicate the request, start, and completion of the channel. Therefore, you must explicitly signal to an event multiplexer when the *request* for the channel occurs (the DMA channel start and DMA channel completion conditions are automatically determined by the event port logic):

- Program the desired DMA channel in the EVINx[DMACH] bits.
- Write a 1 to the EVINx[SWDRQ] bit, which signals a software-initiated request for a DMA channel to the event port.

The SWDRQ bit should be set when the application notifies the software-initiated channel to begin. Either the SC1400 instruction that sets this bit can execute immediately after the instruction that initiates the DMA channel, or these two instructions can be paired in a single SC1400 execution set so that they execute in parallel. The former is the recommended technique. If the first technique is used, the two instructions should be preceded by a DI instruction and followed by an EI instruction, as shown here:

```
; Use this code for signalling the event port
; when starting a software-initiated DMA channel.
;   (assumes that the event port has already programmed the DMAEN, DMACH,
;   and DMATYP bits for one of its event multiplexers)

    di                      ; Disable maskable interrupts.
    move.l    $xxxx,<DMA_REG>; Start a software-initiated DMA channel.
    bmset     $0040,EVINx   ; Set the EVINx[SWDRQ] bit (bit 22 of EVINx)
    ei                      ; Re-enable maskable interrupts
```

Setting the SWDRQ bit not only internally signals the DMA channel request, but also internally generates the DMA channel start and DMA channel completion conditions.

15.3 Event Multiplexer Combining Logic

The combining logic takes many different sources and allows them to be combined in different ways for advanced system-level triggering. All input sources to an event multiplexer are combined as programmed by the user. **Figure 15-6** shows the combining logic within each event multiplexer.

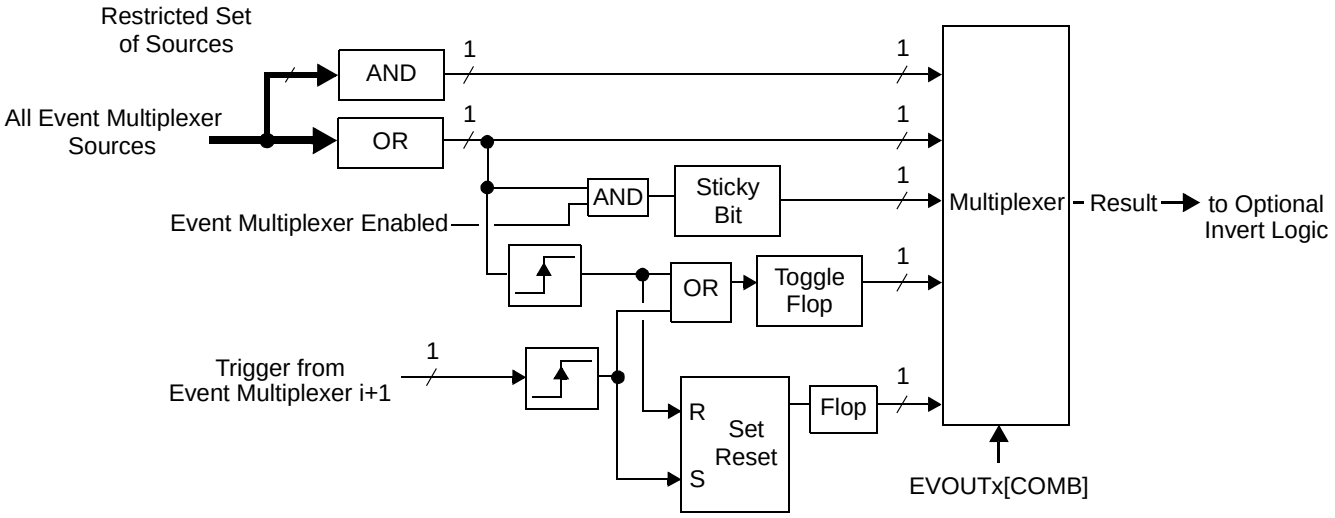


Figure 15-6. Combining Logic in an Event Multiplexer

Following are different ways to configure the combining logic:

- OR together all enabled input sources to provide a level-sensitive trigger for an event multiplexer.
- AND together any enabled input sources to provide a level-sensitive trigger for an event multiplexer. Restrictions on the sources allowed for the ANDing are covered in **Section 15.3.1**, *Restrictions on Combining via ANDing*.
- XOR together all enabled input sources to provide a level-sensitive trigger for an event multiplexer.
- Set operation. After the event multiplexer is enabled, an assertion on any enabled input source sets the sticky bit, which remains asserted until it is explicitly cleared.
- Set-reset operation:
  - — Sets the trigger on the rising edge of a trigger from MUX i+1.
  - — Resets the trigger on the rising edge of the ORing of all enabled input sources for MUX i.

  Set-reset mode can be used in sequences, but the event multiplexer that is programmed for set-reset combining (that is, the multiplexer that generates the reset) must be enabled by the previous multiplexer in the sequence (the multiplexer that generates the set). This is accomplished by EVOUTx[ENABLE] = 1110.
- Toggle operation: Toggle the trigger for MUX i on the rising edge of either of the following:
  - — ORing of all enabled input sources within this event multiplexer, MUX i.
  - — Any trigger from MUX i+1.

  The toggle mode cannot be used in sequences.

The rising edge detectors, sticky bit, toggle flop, and flop after the set-reset logic are clocked with the IPBus clock and cleared at reset or when the EVOUTx[REN] bit is set.

15.3.1 Restrictions on Combining via ANDing

Only a subset of signals is permitted for the ANDing operation. The EVOUTx[COMB] bits can be programmed for an ANDing operation only when all input signals used in the AND operation belong to the following subset:

- TOUT[3–0]
- AUX[4–0]
- TDM0 Rx, TDM1 Rx
- NMICHP
- EVNT[0–4]
- Selected DMA event (see **Section 15.2.3**, *DMA Input Source Selection*, on page 15-8)
- DHIGH signal from the DMA controller
- INTSV signal from the SC1400 core
- Interrupt signals from the interrupt controller

When any of these signals are used in the ANDing operation, they must be enabled appropriately via their corresponding bits in the EVINx and EVOUTx registers.

15.3.2 Set

The event port can be configured so that a set of conditions performs a set operation. Combined with the inversion capability, a set operation allows the event port to create waveforms similar to those shown in **Figure 15-7**, which can be sent to the timers, pins, and so on.

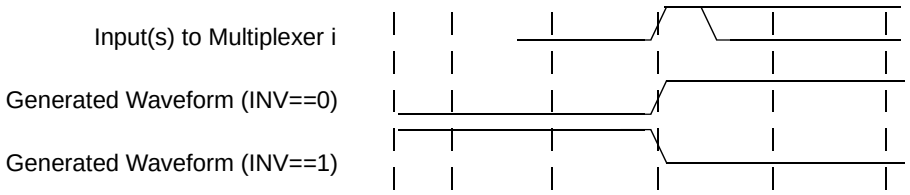


Figure 15-7. Set Function Using One Event Multiplexer

A set operation is programmed as follows:

1. Program the event multiplexer to detect the set:
   a. Set the EVOUTx[REN] bit (**page 15-34**) to a value of 1 to reset the multiplexer and prevent accidental triggering.
   b. Enable the desired inputs and auxiliary inputs for the set operation.
   c. Program the combine logic for a set operation (EVOUTx[COMB] = 010). If multiple input sources are enabled, they are ORed to provide the set condition.
   d. Program the EVOUTx[ENABLE] bits as desired. If a simple set function is desired, program it as always enabled. If the set operation is enabled by another event on the device, set the enable bits to enable the event multiplexer accordingly.

2. Clear the REN bit on event MUX i.

The set logic is now operational. The generated waveform in **Figure 15-7**, either high true or inverted, comes from event MUX i. The value of the event MUX i INV bit determines whether it is inverted.

15.3.3 Set-Reset

The event port can be configured so that one set of conditions performs a set operation and a second set of conditions performs a reset of a triggering signal. Combined with the inversion capability, set-reset allows the event port to create waveforms similar to those shown in **Figure 15-8**, which can be sent to the timers, pins, and so on.

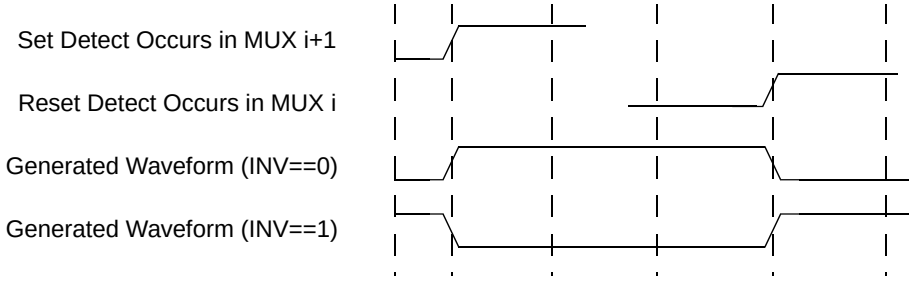


Figure 15-8. Set-Reset Waveforms (Uses two Event Multiplexers)

A set-reset operation requires two adjacent event multiplexers. Event MUX i+1 detects the set operation, and event MUX i detects the reset operation:

1. Event MUX i+1 detects the set operation and is programmed as follows:
   a. Set the EVOUTx[REN] bit (**page 15-34**) to a value of 1 to reset the multiplexer and prevent accidental triggering.
   b. Enable the desired inputs and auxiliary inputs for the set operation.
   c. The combine logic is programmed for OR, AND, or SET as desired. It cannot be programmed for a toggle or set-reset operation.
   d. Enable bits are programmed as desired. If a simple set-reset function is desired, use the always enabled value (EVOUTx[ENABLE] = 1111). If the set-reset operation is enabled by another event on the chip, the enable bits can be set so the set-reset unit built from the two event multiplexers is enabled accordingly.
2. Event MUX i detects the reset and is programmed as follows:
   a. Set the EVOUTx[REN] bit (**page 15-34**) to a value of 1 to reset the multiplexer and prevent accidental triggering.
   b. Enable the desired inputs and auxiliary inputs for the set operation.
   c. Program the combine logic for set-reset (EVOUT[COMB] = 111).
   d. Program the INV bit to get a high-true or inverted waveform (EVOUTx[INV] = 1).

e. Program the enable bits as either always enabled (EVOUTx[ENABLE] = 1111) or enabled by MUX i+1 (EVOUTx[ENABLE] = 1110).

   There is a difference between these two only if the INV bit is set for this multiplexer. If INV is set and always enabled, a 1 is output before the first set occurs. If INV is set and enabled by MUX i + 1, a 0 is output before the first set.

3. The REN bit is cleared on event MUX i.
4. The REN bit is cleared on event MUX i + 1.

At this point, the set-reset logic is operational. Keep in mind that for a set-reset operation, the event multiplexer that generates the final signal must be programmed with EVOUTx[ENABLE] = 1111. The generated waveform shown in **Figure 15-8**, either high true or inverted, comes from event MUX i, although the output from Event MUX i+1 can also be used. The value of the event MUX i INV bit determines whether the waveform is inverted.

15.3.4 Toggle

The event port can be configured to toggle on the rising edges of two different triggering signals. Two adjacent event multiplexers are required, and they are set up as follows:

1. Program event MUX i+1:
   a. Set the EVOUTx[REN] bit (**page 15-34**) to a value of 1 to reset the multiplexer and prevent accidental triggering.
   b. Enable the desired inputs and auxiliary inputs for the set operation.
   c. Program the combine logic for OR, AND, or SET as desired. It cannot be programmed for a toggle or set-reset operation.
   d. The enable bits are typically programmed as always enabled, but you can program them with a different value for a more complex operation.
2. Event MUX i detects the reset and is programmed as follows:
   a. Set the EVOUTx[REN] bit (**page 15-34**) to a value of 1 to reset the multiplexer and prevent accidental triggering.
   b. Enable the desired inputs and auxiliary inputs for the set operation.
   c. Program the combine logic for a toggle operation (EVOUTx[COMB] = 011).
   d. The INV bit is typically cleared.
   e. The enable bits are typically programmed as always enabled (EVOUTx[ENABLE] = 1111), but you can program them with a different value for a more complex operation.
3. Clear the REN bit event MUX i.
4. Clear the REN bit on event MUX i+1.

At this point, the toggle logic is operational.

When an event multiplexer is configured for a toggle operation, the REN bit must be set:

- If the event multiplexer INV bit is cleared:
  - The output of the multiplexer while REN is still set is a 0.
  - When the REN bit is cleared to enable the multiplexer, the output remains a 0 until the first toggle is detected.
  - The first toggle detected can be one cycle after the REN bit is cleared.
- If the event multiplexer INV bit is set:
  - The output of the multiplexer while REN is still set is a 0.
  - When the REN bit is cleared to enable the multiplexer, the output remains a 0 for one clock cycle and then switches to a 1 to reflect the state of the toggle flop.
  - The first toggle detected can be one cycle after the REN bit is cleared.

When the toggle logic becomes operational, a detection of a toggle is immediately reflected on the output of the event multiplexer.

15.4 Event Port Actions

When a trigger is detected in an event multiplexer, the multiplexer can take one of the following actions:

- Request a DMA data transfer.
- Generate an interrupt.
- Send a trigger to an EVNT pin.
- Switch the crossbar to alternate priority registers.
- Wake up the SC1400 core from Stop mode. Event multiplexer 0 can be used to force an exit from Stop mode. This capability is available only for event multiplexer 0 and is covered in detail in **Section 11.4.5**, *Exit from Stop Mode*, on page 11-22.
- Provide status to an external host through the HDI16 port.

15.4.1 Event Port DMA Transfers

Each event multiplexer initiates DMA data transfers using the EVDMA0 request, the EVDMA1 request, or both. If more than one event multiplexer is configured to drive a particular EVDMAx signal, the triggers from each source (that is, the outputs of the optional invert blocks) are ORed together to form a single request. Software can determine which event multiplexer caused the DMA transfer by examining the EVCTL[EMUX] bits of the event multiplexers programmed to generate interrupts. When an event multiplexer triggers, its corresponding EMUX bit is set.

15.4.2 Event Port Interrupts

Each event multiplexer can generate an interrupt request on the EVINT0 signal, the EVINT1 signal, or both signals. When an event multiplexer is programmed to generate interrupts, it asserts the

EVINTx request signal as a high true, edge-triggered value. This value is captured and latched within each event multiplexer. The latched interrupt request remains asserted until it is explicitly cleared in the interrupt handler by setting the REN bit of the multiplexer that generated the interrupt request. Note that this action clears the edge detect latch only for this particular event multiplexer. An interrupt handler identifies the event multiplexer that caused the interrupt by examining the EVCTL[EMUX] bits. When an event multiplexer triggers, its corresponding EMUX bit is asserted. If more than one event multiplexer is configured to drive a particular EVINTx signal, the triggers from each source (that is, the outputs of the optional invert block) are ORed together to form a single request.

15.4.3 Crossbar Switch Priority Changes

When an event multiplexer is configured to drive the signal to switch the crossbar to alternate priority when the trigger occurs, the crossbar is switched to a new set of priorities via its alternate priority registers. The crossbar switch uses this alternate set of priorities as long as the signal is asserted. If the signal is a pulse, it may be useful to use the set capability within the combining logic so that the alternate priorities remain in use. Normal usage is restored when the REN bit of the appropriate event multiplexer is set.

If more than one event multiplexer is configured to switch the crossbar to alternate priority, the triggers from each source (that is, the outputs of the optional invert block) are ORED together to form a single request.

15.4.4 Forced Exit from Stop Mode

Only event multiplexer 0 can force an exit from Stop mode. See **Section 11.4.5**, *Exit from Stop Mode*, on page 11-22.

15.4.5 Status to an External Host

The event multiplexers can be configured so that a host processor can read their output in the HDI16 ISR register. The output of one multiplexer is read in ISR[EV1] and the output of a second is read in ISR[EV0]. Therefore, an external host can view any data visible through the event port. The outputs are latched by the IPBus clock before the signals are sent outside the event port.

15.4.6 Restrictions on Multiple Drivers

Each event multiplexer can independently drive any of the following signals:

- One of the TINx signals (TIN[0–3])
- One of the OCE10 on-chip emulator signals (EE[0–5], EED, EC0)
- One of the event pins (EVNT[0–4])
- One of the event general-purpose outputs

Problems would occur if multiple sources simultaneously attempt to drive a single signal. Therefore, the event port must be configured so that only one event multiplexer can drive one of these signals at a given time. For example, one multiplexer can drive EVNT2 while another drives EVNT3, but two different multiplexers cannot both drive EVNT3 at the same time. Any ORing operations should use the event multiplexer ORing capability.

15.5 Event Port and Debug Port Interaction

The event multiplexers detect system events such as interrupts, DMA activity, and timer events, whereas the SC1400 debug port contains event detection channels that detect instruction-based events, such as program addresses and data values written to a particular address.

Event multiplexer sequencing uses both on-chip events entering the event port and the debug port EDCA and EDCD breakpoint units, which can enable multiplexers via the EEx and EED signals.

The event port and the debug port communicate with each other through the OCE10 on-chip emulator EE[5–0], EED, and EC0 signals. Each EE signal and the EED signal can be individually configured as an input or output signal. A signal is selected by programming the EE_CTRL register in the emulator. When the emulator uses them as inputs, the EE[5–0] and EED signals are synchronized as they enter the emulator port. The EC0 signal is synchronized outside the emulator port with the core clock and can run at rates up to the core frequency. At reset, the EE signals are configured as inputs. An EE input signal can also be configured to perform more than one function. For example, the EE2 signal can enable both EDCA2 and the event counter and generate an emulator event.

Each EEx signal (or the EED signal) can be configured as an input to the breakpoint logic, which can enable an address (EDCAx) or data detection channel (EDCD):

- EED enables EDCD.
- EE0 enables EDCA0.
- EE1 enables EDCA1.
- EE2 enables EDCA2.
- EE3 enables EDCA3.
- EE4 enables EDCA4.
- EE5 enables EDCA5.

Each EEx signal (or the EED signal) can generate an emulator event (selected via the SC1400 core emulator Event Selector registers):

- Enter Debug mode.
- Generate a non-maskable debug exception.
- Enable/disable trace buffer capture.

Each EEx signal can also be configured as an output to indicate internal emulator events to devices outside the SC1400 core. They can signal a detection by an event detection channel (EDCAx or EDCD). An event is signalled by toggling the appropriate terminal:

- EED signals a detection by EDCD.
- EE0 signals a detection by EDCA0.
- EE1 signals a detection by EDCA1.
- EE2 signals a detection by EDCA2.
- EE3 signals a detection by EDCA3.
- EE4 signals a detection by EDCA4.
- EE5 signals a detection by EDCA5.

Via a logic analyzer, the EEx signals can be configured to measure the time between two events detected by event detection channels.

The special capabilities of each signal are as follows:

- EC0 is an input to the 31/62-bit emulator event counter.
- EE0 can bring the SC1400 core into Debug mode directly from reset or during normal execution.
- EE1 can indicate that the SC1400 is currently in Debug mode.
- EE2 can enable the emulator 31/62-bit event counter.
- EE3 can indicate that the ERCV register is full.
- EE4 can indicate that data is available in the ETRSMT register.

Note: When programmed as an output, EE5 can be used only to enable its address detection channel. It cannot be used to generate an emulator event.

The debug port can drive the EE[5–0] and EED signals as outputs to the event port to trigger an event multiplexer. When the EE[5–0] signals are configured as emulator outputs, only the EE4 signal, can connect to an EVNTx pin through an event multiplexer. You can view another emulator output signal indirectly by programming an event multiplexer to trigger from the desired EEx pin with the corresponding EMUX[i] bit set. As soon as the EEx signal asserts, the value of the EMUX[i] bit is set to perform an event port action. From the emulator port, the EE0 signal can be viewed on the EE0 pin. This EE0 connection is routed directly from the emulator to the pin, bypassing the event port. When the EE0 pin is configured as an input, its signal goes directly to the emulator port, ORed with the EE0 signal output from the event port.

You can also connect the EVNTx pins through an event multiplexer to the debug port EE[0–5], EED, and EC0 signals. You can connect an emulator output, EE4, through an event multiplexer to an EVNTx pin. When an EVNT pin is used as a timer input, its signal first passes through a synchronizer, introducing a delay of up to two core clocks. There is an additional synchronizer within the emulator port on the EEx, EED, and EC0 signals.

15.6 Software Management of Event Multiplexers

The event port multiplexers respond not only to incoming signals that trigger event port actions but also to software triggers. Each event multiplexer is accessible through the device programming model:

- Trigger an event multiplexer from an SC1400 instruction
- Read an event multiplexer to determine whether it has triggered. Use the EVCTL[EMUX] bits to check the status of the event multiplexers and generate triggers from a program through an event multiplexer when it is enabled.
- Reset an event multiplexer via the EVOUTx[REN] bit.

15.6.1 Trigger an Event Multiplexer

You can trigger an event multiplexer action by writing to its corresponding EVCTL[EMUX] bit via a bit set instruction in a program. That is, instructions in a program can be treated as events to be handled by the event port. Therefore, you can trigger an event multiplexer sequence either by a device-level event or a particular location in a program. In **Figure 15-3**, *Block Diagram of One Event Multiplexer,* on page 15-4, notice the connection to the OR gate immediately after the combining logic.

Writing a value of 1 to an EMUX bit sets the bit, and the action passes through the OR and optional invert functions and generates a trigger for the multiplexer. If the trigger is enabled, an event port sequence can advance to the next multiplexer in the sequence and/or cause an event port action to occur for the particular multiplexer. Writing a value of 0 clears the EMUX bit, removing the trigger condition for the multiplexer. However, writing a 0 does not affect the value of the EMUX[i] bit when the EVCTL register is read. The EMUX bit is also reset to zero when the multiplexer EVOUTx[REN] bit is set The EMUX bits are also useful for checking the trigger status of a multiplexer because they are set only when an enabled trigger occurs and remain set until it is cleared.

15.6.2 Reset An Event Multiplexer

The EVOUTx[REN] bit resets an event multiplexer. The REN bit is set when an event multiplexer is not in use or while it is programmed for operation. When an event multiplexer is disabled by setting its REN bit, the corresponding outputs for the event multiplexer are immediately disabled. No other bit values in the EVOUTx register should be modified. A BMSET.W instruction is used to ensure that no other fields in the EVOUTx register are modified, or you can write a value to the EVOUTx register that is equal to its current contents with the exception of setting the REN bit. Enabling an event multiplexer by clearing its REN bit takes effect one IPbus clock cycle later. There is a delay of one IPBus clock cycle before the output of the optional invert block is passed through the action select block. The direct connection modes in which the EVNTx pins directly connect to the timer module are unaffected by the REN bit. See **Section 15.2.2**, *Direct Connection Modes*, on page 15-7.

Table 15-6 summarizes the different capabilities of the REN bit in each event multiplexer.

Table 15-6. REN Bit Usage

Operation	Conditions	Description	REN Bit Definition	Cross Reference
Clear sticky bit in the combine logic.	EVOUTx[COMB] = 010	Set up an event multiplexer for the set mode of operation in the combine logic.	0 Bit not affected. 1 Clear sticky bit.	**Section 15.2.3**, *DMA Input Source Selection,* on page 15-8.
Clear toggle flop in the combine logic.	EVOUTx[COMB] = 011	Set up an event multiplexer for the toggle mode of operation in the combine logic.	0 Bit not affected. 1 Clear toggle bit.	**Section 15.2.3**, *DMA Input Source Selection,* on page 15-8.
Clear set-reset flop in the combine logic.	EVOUTx[COMB] = 111	Set up an event multiplexer for the set-reset mode of operation in the combine logic.	0 Bit not affected. 1 Reset bit.	**Section 15.2.3**, *DMA Input Source Selection,* on page 15-8.
Inhibit event multiplexer triggering	—	Set up an event multiplexer to prevent accidental triggering. Especially useful in sequenced triggering.	0 Event multiplexer can trigger. 1 Event multiplexer cannot trigger.	**Section 15.7**, *Event Sequencing,* on page 15-20.
Clear EMUX[i] status bits.	—	Clear an EMUX[i] bit, which indicates whether a particular event multiplexer has triggered. Also clears the EMUX[i] bit that is an input to the OR gate immediately following the combining logic.	0 EMUX[i] bit not affected. 1 Clear EMUX[i] bit.	See **page 15-27** for details on the Event Port Control (EVCTL) register.
Clear event port interrupt request, EVINTx.	EVINTx signal is asserted.	Used by an interrupt handler to clear an event port interrupt request.	0 Interrupt request not affected. 1 Interrupt request is cleared.	**Section 15.4.1**, *Event Port DMA Transfers,* on page 15-15.
Clear event port DMA request, EVDMAx	internal EVDMAx signal is asserted.	Software clears the DMA request.	0 DMA request not affected. 1 DMA request is cleared.	**Section 15.4.1**, *Event Port DMA Transfers,* on page 15-15.

15.7 Event Sequencing

Events can be sequenced through the event multiplexers with actions triggered at the appropriate location in the sequence. The EVOUTx[REN] bit ensures that an event multiplexer does not accidentally trigger before it is ready. For sequenced operations, the registers are programmed as follows:

1. Set the REN bit of the first multiplexer in the sequence.
2. Set the REN bit of the second multiplexer in the sequence.
3. Set the REN bit of the last multiplexer in the sequence.
4. Program the EVOUT register for the first multiplexer in the sequence and ensure that its REN bit is set.
5. Program the EVOUT register for the second multiplexer in the sequence and ensure that its REN bit is set.

6. Continue until the last multiplexer in the sequence is programmed and its REN bit is set.
7. Clear the REN bit of the last multiplexer in the sequence using a bit clear instruction.
8. Continue clearing the REN bits of each multiplexer in the sequence until the REN bit of the first multiplexer is cleared.

Steps 1–3 prevent the multiplexers from triggering early. Steps 4–6 clear the REN bit in order from the last to the first multiplexer to ensure that none of the units accidentally triggers during the set-up process. At this point, the sequence is enabled and ready.

15.7.1 Sequencing Through the Event Multiplexers

The following example presents three consecutive multiplexers (4, 3, and 2). The highest numbered one triggers first and the lowest numbered one triggers last. The last multiplexer trigger is programmed to toggle the EVNT2 pin.

1. Multiplexer 4: Trigger on the assertion of the EVNT3 or EVNT4 signal.
2. Multiplexer 3: Trigger on the start of DMA channel 2.
3. Multiplexer 2: Trigger on the assertion of the EVNT1 signal and drive the EVNT2 signal.

The desired sequencing is represented as: (EVNT3 || EVNT4) → DMA Channel 2 Start → EVNT1 → EVNT2. This sequence is programmed on three event multiplexers, MUX4, MUX3, and MUX2, as follows:

1. Set the EVOUT[REN] bit of MUX4 in the sequence.
2. Set the REN bit of MUX3 in the sequence.
3. Set the REN bit of MUX2 in the sequence.
4. Set up MUX4:
   a. Set the EVIN4[EVNT3] and EVIN4[EVNT4] bits.
   b. Combine these signals with an OR operation (EVOUTx[COMB] = 000).
   c. MUX4 is always enabled (EVOUT[ENABLE] = 1111).
   d. EVOUT[REN] = 1.
5. Set up MUX3:
   a. Select DMA Channel Start for DMA channel 2 (EVIN2[DMATYP] = 001).
   b. Use the OR for the combining operation (EVOUTx[COMB] = 000).
   c. MUX3 to be enabled by MUX i+1 (EVOUT[ENABLE] = 1110).
   d. EVOUT[REN] = 1.
6. Set up MUX2:
   a. EVNT1 bit set.
   b. Use OR for the combining operation (EVOUTx[COMB] = 000).

c. Set up MUX2 to be enabled by MUX i+1 (EVOUT[ENABLE] = 1110).
d. EVOUT[REN] = 1.
e. Program MUX2 to toggle EVNT2.

7. Clear the REN bit for MUX2, MUX3, and MUX4, respectively.

At this point, these multiplexers are ready. The sequence triggering is shown in **Figure 15-9**.

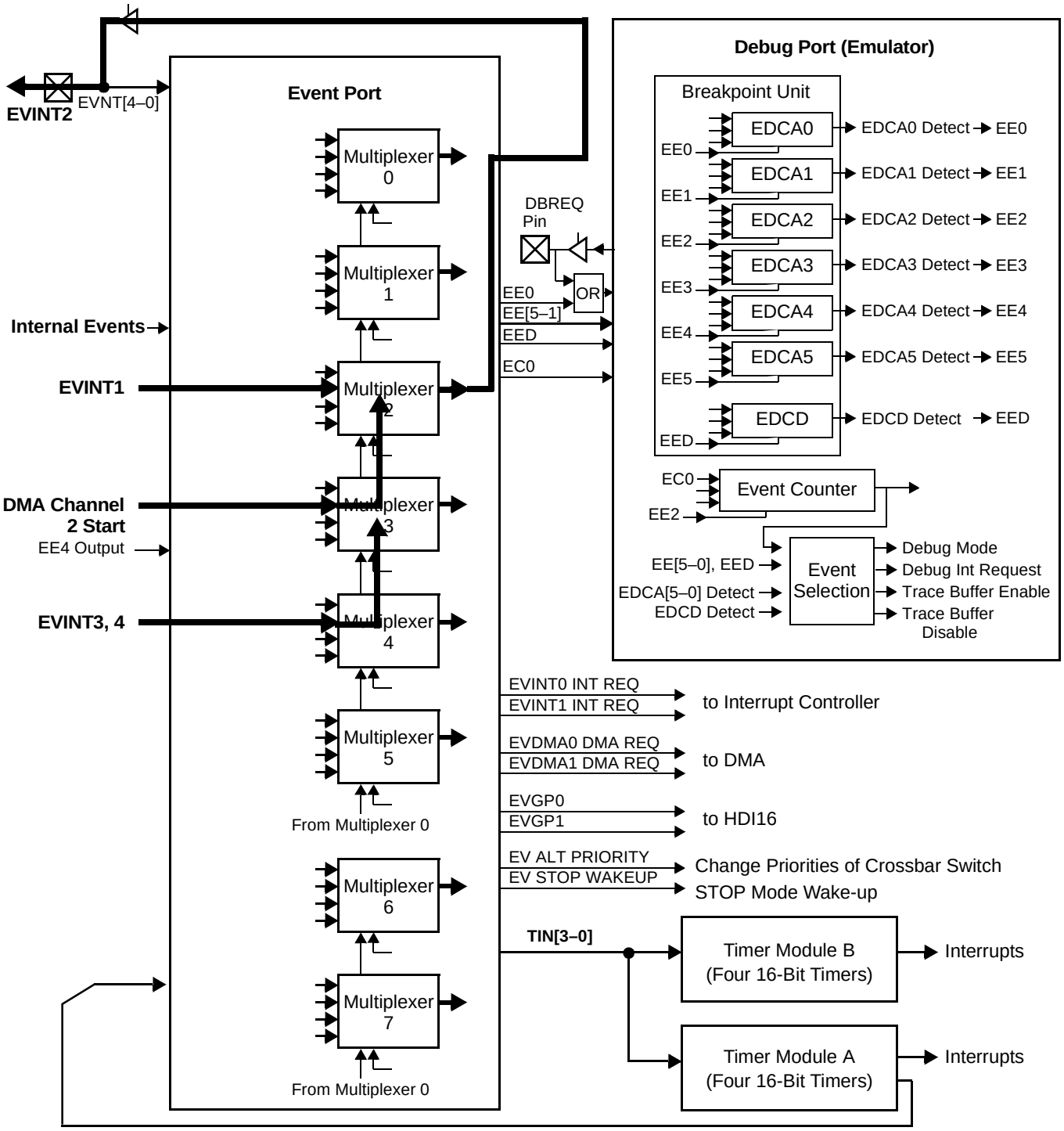


Figure 15-9. Event Multiplexer Sequencing

15.7.2 Sequencing from Event Multiplexer to Debug Port

You can use event port sequences to enable a sequence within the debug port breakpoint unit. Consecutive multiplexers must be used for sequencing events in the event port, although this is not required for sequences in the debug port. In the example shown here, a sequence of two events is created using multiplexers 3 and 2. When the second event is detected, a sequence of two breakpoints within the debug port is enabled, which, once detected, halts the SC1400 core. The desired triggering sequence (shown in **Figure 15-10**) is represented as: DMA Channel 15 Start → EVNT1 → (PAB== 0x00C00000) → (PAB==0x00000100) → Debug mode:

1. Trigger on the start of DMA channel 15.
2. Trigger on the assertion of the EVNT1 signal.
3. Upon detecting the last event in the sequence, enable the debug port breakpoint EDCA1 unit, which looks for a program address of 0x00C0 0000.
4. Upon detecting this first breakpoint, enable the debug port breakpoint EDCA0 unit which looks for a program address of 0x0000 0100.
5. Upon detecting this second breakpoint, place the SC1400 core into Debug mode.

15.7.3 Sequencing from Debug Port to Event Multiplexers

You can use a breakpoint sequence within the debug port breakpoint unit to enable an event port sequence. In the example shown here, a sequence of two breakpoints within the debug port is enabled. When the second breakpoint is detected, the event sequence is enabled and initiated, which generates an interrupt when the event sequence completes.

The desired triggering sequence is as follows:

1. The debug port breakpoint EDCA1 unit checks for a program address of 0x00C0 0000.
2. When this first breakpoint is detected, the debug port breakpoint EDCA0 unit is enabled, and it checks for a program address of 0x0000 0100.
3. When this second breakpoint is detected, the event port MUX3 is enabled, which triggers on a DMA channel request on channel 15.
4. MUX2 triggers on the assertion of the EVNT1. Upon detection, generate an interrupt.

The desired sequencing can be represented as: (PAB==0x00C00000) → (PAB==0x00000100) → DMA Channel 15 Request → EVNT1 → Generate Interrupt. This sequence triggering is shown in **Figure 15-11**.

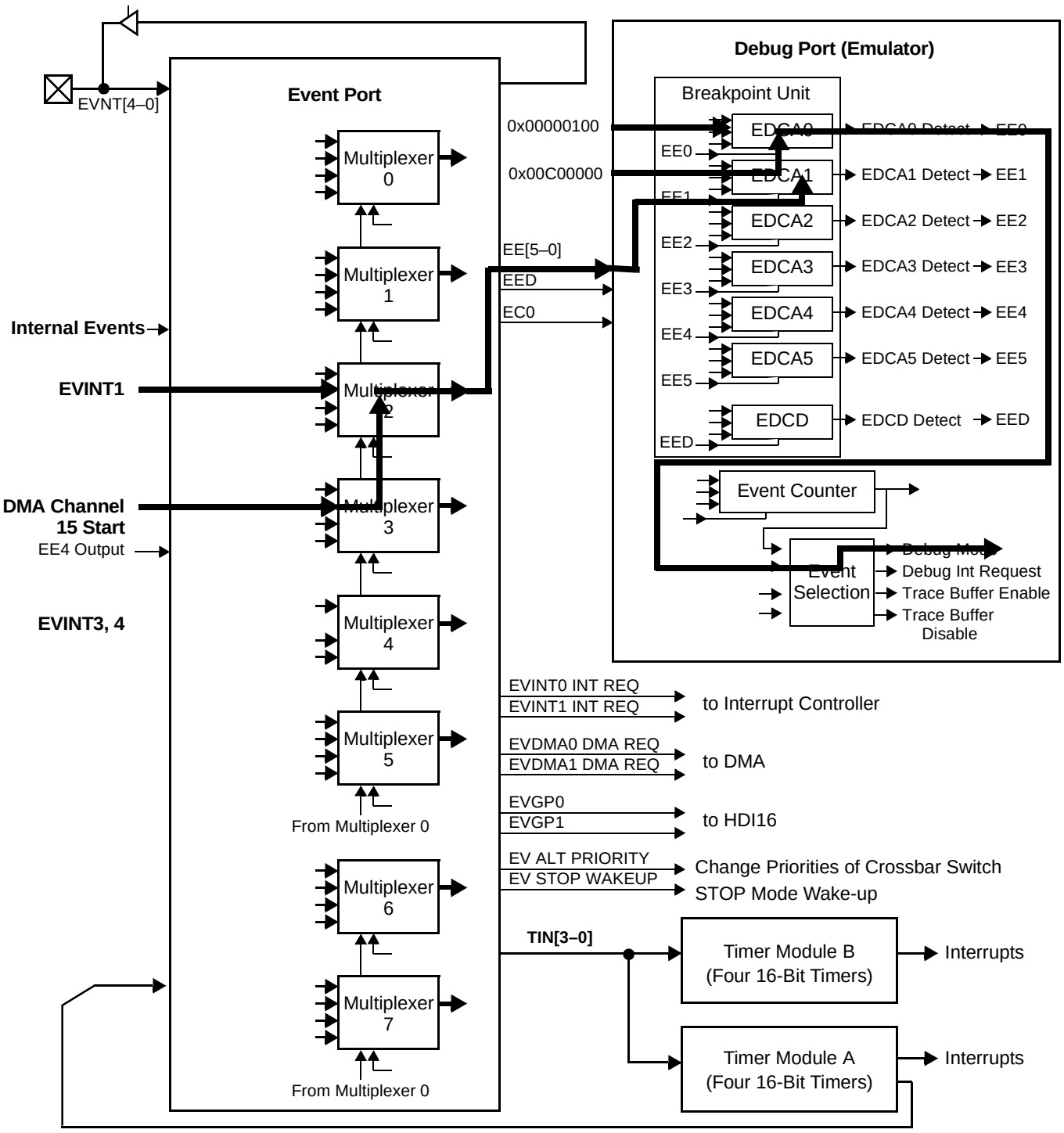


Figure 15-10. Event Multiplexer to Debug Port Sequencing

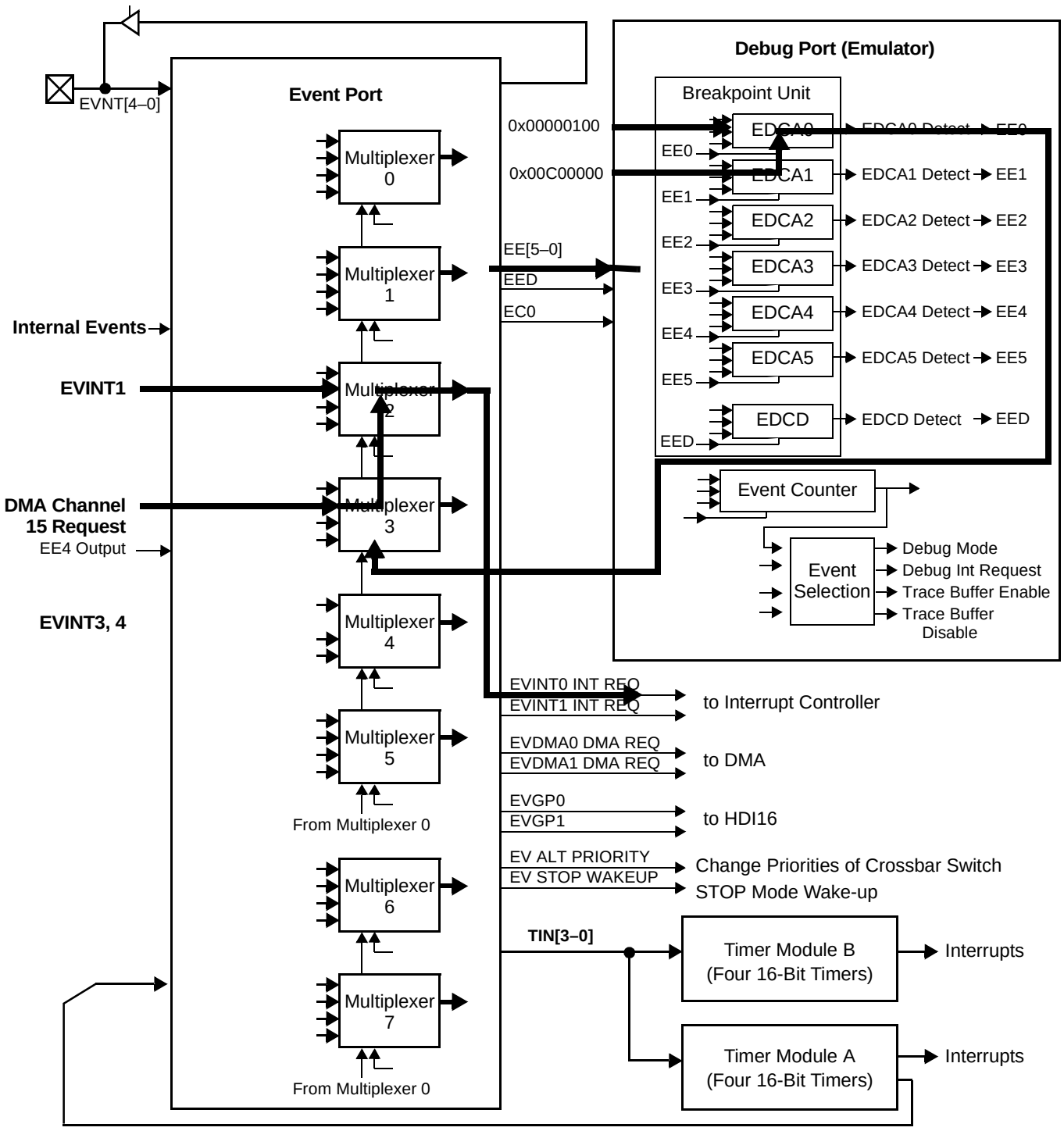


Figure 15-11. Debug Port to Event Multiplexer Sequencing

15.7.4 Instruction in a Triggering Sequence

A sequencing event can be an SC1400 instruction. The example presented here uses three consecutive multiplexers. MUX3 is triggered by the execution of an instruction.
The instruction, located at 0x00C90004, is a BMTSET.W instruction that sets the EMUX3 bit in

the EVCTL[EMUX] register, explicitly triggering an event multiplexer. The last multiplexer trigger is programmed to toggle the EVNT2 signal.

1. MUX4: Trigger on the assertion of the EVNT4 pin
2. MUX3: Trigger on any instruction that sets the EMUX3 bit
3. MUX2: Trigger on the assertion of the EVNT1 pin and drive the EVNT2 pin.

The desired sequencing is represented as: EVNT4 → (EMUX3 set by a SC1400 instruction) → EVNT1 → EVNT2.

15.7.5 Instruction ORed with an Event in a Triggering Sequence

In the example presented here, one of the sequencing events is a SC1400 instruction, there are three consecutive multiplexers, and MUX3 is triggered by the execution of an instruction or a trigger from Timer 1 in timer module A. Instructions at 0x00C90004, 0x00401004, and 0x0000A700 are BMTSET.W instructions that set the EVCTL[EMUX3] bit. They explicitly trigger an event multiplexer. The last multiplexer trigger is programmed to toggle EVNT2.

1. MUX4: Trigger on a DMA transfer completion on channel 5.
2. MUX3: Trigger on any instruction that sets the EMUX3 bit OR a time-out on Timer 1.
3. MUX2: Trigger on the assertion of the EVNT1 signal and drive the EVNT2 signal.

The desired sequencing is represented as: DMA Channel 5 Completed → (EMUX3 set by a SC1400 instruction -OR- TOUT1) → EVNT1 → EVNT2.

15.8 Event Port Programming Model

The event port registers, which configure the event selection multiplexers, are listed as follows, along with the number of the page where each register is discussed:

- Event Port Control Register (EVCTL), **page 15-27**.
- Event Port Selective Invert (EVSELINV), **page 15-30**.
- Event Input Selection Register (EVINx), **page 15-32**.
- Event Output Selection 0, 2, 4 (EVOUT[0, 2, 4, 6]), **page 15-34**.
- Event Output Selection 1, 3, 5 (EVOUT[1, 3, 5, 7]), **page 15-37**.

For the value of the base address for the EV_BASE register file, see **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. In addition to the input sources selected in the EVINx registers, the EVCTL register and the corresponding EVOUTx registers also provide input sources for the event multiplexers. There is only a minor difference between the EVOUT registers for multiplexers 0,2,4,6 versus those for multiplexers 1,3,5, 7. The first set drives the emulator EC0 signal, and the second set drives the emulator EED signal.

EVCTL Event Port Control Register EV_BASE + 0xC0

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—				AUX4				EMUX							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AUX3				AUX2				AUX1				AUX0			
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EVCTL provides the status and allows triggering of each event multiplexer via an SC1400 instruction. This register also provides additional bits for input selection. When any signal is selected via the AUX bits, the corresponding capability must be enabled in the EVINx[AUXEN] bits. The inputs come from different MSC711x peripherals. The CADEV and PADEV signals are from the address detection units, the HTRQ and HRRQ signals and signals from the HCVR register are from the HDI16. The PLL, OCE10 emulator, and Ethernet MAC (if present on the device) also provide inputs.

Table 15-7. EVCTL Bit Descriptions

Bit x	Reset	Description	Description
— 31–28	0	Reserved. Write to zero for future compatibility.	
AUX4 27–24	0	**AUX4 Signal Selection** Selects a signal as an extra input to the event multiplexers.	0000 Disabled. AUX signal not driven. 0001 EE4 signal. 0010 Ethernet Tx frame interrupt. 0011 Ethernet Rx frame interrupt. 0100 Ethernet summary interrupt signal. 0101 PLL loss of lock signal. 0110 PLL lock signal. 0111 Reserved. 1000 CADEV0 signal. 1001 CADEV1 signal. 1010 PADEV0 signal. 1011 PADEV1 signal. 1100 HTRQ signal. 1101 HRRQ signal. 1110 HCVR[EVIN0] bit. 1111 HCVR[EVIN1] bit.

Table 15-7. EVCTL Bit Descriptions (Continued)

Bit x	Reset	Description	Description
EMUX 23–16	0	**Event Multiplexer Bits 7–0** This field checks the status of event multiplexers 7–0 and generates triggers from a program through an event multiplexer, when enabled. EMUX is reset to zero when the multiplexer's REN bit is set. EMUX allows instructions in a program to be treated as events to be handled by the event port—for example, to trigger on the completion of a DMA channel or a program instruction. Program instructions can also be used to advance an event port sequence. Therefore, device-level events can not only be used for sequencing, but sequencing can also trigger on a particular location in a program. Setting EMUX sets an internal flop (shown in **Figure 15-3**, *Block Diagram of One Event Multiplexer,* on page 15-4) that passes through the OR and optional invert functions and can generate a trigger for the multiplexer. If the trigger is enabled, it can advance an event port sequence to the next multiplexer in the sequence and/or cause an event port action to occur for the specified multiplexer. Clearing this bit clears the internal flop and removes the trigger condition for the multiplexer. However, writing a 0 does not affect the value of the EMUX when the EVCTL register is read. Note that the internal flop is set by the *detection of a write* of a 1 to an EMUX bit, and not simply whether the value of the EMUX bit is 1.	0 Reads the status of the designated event multiplexer. 1 Reads the status of the designated event multiplexer. Writing a 1 to a bit in this field causes the event multiplexer to trigger.
AUX3 15–12	0	**AUX3 Signal Selection** Selects a signal as an extra input to the event multiplexers	0000 Disabled. AUX signal not driven. 0001 EE4 signal. 0010 Ethernet Tx frame interrupt. 0011 Ethernet Rx frame interrupt. 0100 Ethernet summary interrupt signal. 0101 PLL loss of lock signal. 0110 PLL lock signal. 0111 Reserved. 1000 Internal extended core event output 0 signal. 1001 Internal extended core event output 1 signal. 1010 Internal peripheral extended core output 0 signal. 1011 Internal peripheral extended core output 1 signal. signal. 1100 HTRQ signal. 1101 HRRQ signal. 1110 HCVR[EVIN0] bit. 1111 HCVR[EVIN1] bit.

Table 15-7. EVCTL Bit Descriptions (Continued)

Bit x	Reset	Description	Description
AUX2 11–8		**AUX2 Signal Selection** Selects a signal as an extra input to the event multiplexers	0000 Disabled. AUX signal not driven. 0001 EE4 signal. 0010 Ethernet Tx frame interrupt. 0011 Ethernet Rx frame interrupt. 0100 Ethernet summary interrupt signal. 0101 PLL loss of lock signal. 0110 PLL lock signal. 0111 Reserved. 1000 CADEV0 signal. 1001 CADEV1 signal. 1010 PADEV0 signal. 1011 PADEV1 signal. 1100 HTRQ signal. 1101 HRRQ signal. 1110 HCVR[EVIN0] bit. 1111 HCVR[EVIN1] bit.
AUX1 7–4	0	**AUX1 Signal Selection** Selects a signal as an extra input to the event multiplexers. Before an AUX1 selected signal can be used, this capability must also be enabled in the EVINx register AUXEN1 bit.	0000 Disabled. AUX signal not driven. 0001 EE4 signal. 0010 Ethernet Tx frame interrupt. 0011 Ethernet Rx frame interrupt. 0100 Ethernet summary interrupt signal. 0101 PLL loss of lock signal. 0110 PLL lock signal. 0111 Reserved. 1000 CADEV0 signal. 1001 CADEV1 signal. 1010 PADEV0 signal. 1011 PADEV1 signal. 1100 HTRQ signal. 1101 HRRQ signal. 1110 HCVR[EVIN0] bit. 1111 HCVR[EVIN1] bit.

Table 15-7. EVCTL Bit Descriptions (Continued)

Bit x	Reset	Description	Description
AUX0 3–0	0	**AUX0 Signal Selection** Selects a signal as an extra input to the event multiplexers. Before an AUX0 selected signal can be used, this capability must also be enabled in the EVINx register AUXEN0 bit.	0000 Disabled. AUX signal not driven. 0001 EE4 signal. 0010 Ethernet Tx frame interrupt. 0011 Ethernet Rx frame interrupt. 0100 Ethernet summary interrupt signal. 0101 PLL loss of lock signal. 0110 PLL lock signal. 0111 Reserved. 1000 CADEV0 signal. 1001 CADEV1 signal. 1010 PADEV0 signal. 1011 PADEV1 signal. 1100 HTRQ signal. 1101 HRRQ signal. 1110 HCVR[EVIN0] bit. 1111 HCVR[EVIN1] bit.

EVSELINV Event Selective Invert Register(Multiplexer 0) EV_BASE + 0xC8

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PRVM	DHIGH	IMISS	IMISSE	TDM0	TDM1	NMICH	M1CON	—	—	DMA	AUXEN4	AUXEN3	AUXEN2	AUXEN1	AUXEN0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TOUT				EVNT					—	INTSV	INTEV				
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EVSELINV contains bits that bits allow event port input signals to be selectively inverted before the event multiplexers use them. Each bit enables one source for this event multiplexer. If multiple sources are selected, they are combined as specified in the corresponding output action register.

Table 15-8. EVSELINV Bit Descriptions

Bit x	Reset	Description	Description
PRVM 31	0	**Previous Multiplexer**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.

Table 15-8. EVSELINV Bit Descriptions (Continued)

Bit x	Reset	Description	Description
DHIGH 30	0	**DMA Priority Elevation**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
IMISS 29	0	**ICache Miss All Cases**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
IMISSE 28	0	**ICache Miss to External Memory**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
TDM0 27	0	**Receive Interrupt Request**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
TDM1 26	0	**Receive Interrupt Request**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
NMICH 25	0	**NMI Interrupt Request**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
M1CON 24	0	**Contention at M1 Memory**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
— 23–22	0	Reserved. Write to zero for future compatibility.	
DMA 21	0	**DMA Event**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
AUXEN4 20	0	**AUX4 Input Enable**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
AUXEN3 19	0	**AUX3 Input Enable**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
AUXEN2 18	0	**AUX2 Input Enable**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
AUXEN1 17	0	**AUX1 Input Enable**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.

Table 15-8. EVSELINV Bit Descriptions (Continued)

Bit x	Reset	Description	Description
AUXEN0 16	0	**AUX0 Input Enable**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
TOUT[3:0] 15–12	0	**Timer Output Signals (timer 3 — 0 from Timer Module A)**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
EVNT[4:0] 11–7	0	**EVNT Pin Enables**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
— 6	0	Reserved. Write to zero for future compatibility.	
INTSV 5	0	**Interrupt Service**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.
INTEV 4–0	0	**Interrupt Event Signals**	0 Input is not inverted when used by all event multiplexers. 1 Input is inverted when used by all event multiplexers.

EVINx Event Multiplexer Input Selection Register

EVIN0 (Multiplexer 0) EV_BASE + 0x00
EVIN1 (Multiplexer 1) EV_BASE + 0x08
EVIN2 (Multiplexer 2) EV_BASE + 0x10
EVIN3 (Multiplexer 3) EV_BASE + 0x18
EVIN4 (Multiplexer 4) EV_BASE + 0x20
EVIN5 (Multiplexer 5) EV_BASE + 0x28
EVIN6 (Multiplexer 6) EV_BASE + 0x30
EVIN7 (Multiplexer 7) EV_BASE + 0x38

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PRVMX	DMACH					DMATYP			SWDRQ	DMAEN	AUXEN3	AUXEN3	AUXEN2	AUXEN1	AUXEN0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TOUT				EVNT					—	INTSV	INTEV	INTEVM			
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EVINx contains bits that, along with their corresponding bits in the EVOUTx register, select the active event(s) for the specified event multiplexer. Each bit enables one source for this event multiplexer. If more than one source is selected, they are combined as specified in the corresponding output action register. Additional input selection is available in the EVCTL and EVOUTx registers.

Table 15-9. EVINx Bit Descriptions

Bit	Reset	Description	Description
PRVMX 31	0	**Previous Event Multiplexer** Uses the output from the previous event multiplexer in the cascade. The output passes through a latch before it is used as an input.	0 Disable detection of the previous multiplexer signal. 1 Enable detection of the previous multiplexer signal.
DMACH 30–26	0	**DMA Channel** Selects one of the 32 DMA channels.	
DMATYP 25–23	0	**DMA Type** Specifies which signal from the DMA channel is selected as an input to this event multiplexer. See **Section 15.2.3**, *DMA Input Source Selection,* on page 15-8.	000 DMA channel request. 001 DMA channel start. 010 DMA channel completed. 011 DMA channel deprioritization. 100 Reserved. 101 Reserved. 110 Reserved. 111 Reserved.
SWDRQ 22	0	**Software DMA Request** Setting this bit notifies the event multiplexer of a software-initiated DMA channel request. Setting SWDRQ is required for event port triggering on any DMATYP setting for software-initiated channels. Setting this bit is not required for DMA channels initiated by hardware events.	0 Inactive. 1 Signals a request for a software-initiated DMA channel.
DMAEN 21	0	**DMA Event Enable** Enables the event multiplexer DMA input signal.	0 Disable detection of selected DMA signal. 1 Enable detection of selected DMA signal.
AUXEN4 20	0	**AUX4 Input Enable** An enable for the AUX4 signal.	0 Event multiplexer ignores the AUX4 signal. 1 Event multiplexer uses the AUX4 signal.
AUXEN3 19	0	**AUX3 Input Enable** An enable for the AUX4 signal.	0 Event multiplexer ignores the AUX3 signal. 1 Event multiplexer uses the AUX3 signal.
AUXEN2 18	0	**AUX2 Input Enable** An enable for the AUX4 signal.	0 Event multiplexer ignores the AUX2 signal. 1 Event multiplexer uses the AUX2 signal.
AUXEN1 17	0	**AUX1 Input Enable** An enable for the AUX1 signal.	0 Event multiplexer ignores the AUX1 signal. 1 Event multiplexer uses the AUX1 signal.

Table 15-9. EVINx Bit Descriptions (Continued)

Bit	Reset	Description	Description
AUXEN0 16	0	**AUX0 Input Enable** An enable for the AUX0 signal.	0 Event multiplexer ignores the AUX0 signal. 1 Event multiplexer uses the AUX0 signal.
TOUT 15–12	0	**Timer Output Signals** For timers 3–0 from timer module A.	0 Disable detection for the designated timer. 1 Enable detection for the designated timer.
EVNT[4–0] 11–7	0	**EVNT Signals** Enables/disables detection for signals from the EVNT[4–0] pins.	0 Disable detection for the specified event pin. 1 Enable detection for the specified event pin.
— 6	0	Reserved. Write to zero for future compatibility.	
INTSV 5	0	**Interrupt Service** Generates a pulse when the SC1400 core is servicing an interrupt.	0 Disable detection for interrupt service. 1 Enable detection for interrupt service.
INTEV 4	0	**Interrupt Event Signals** Enables/disables signals for non-maskable interrupt events.	0 Disable detection for interrupt event i. 1 Enable detection for interrupt event i.
INTEVM **3–0**	0	**Interrupt Event Signals Masked** Enables/disables signals for maskable interrupt events.	0 Disable detection for interrupt event i. 1 Enable detection for interrupt event i.

EVOUTx Event Output Register

EVOUT0	EV_BASE + 0x40
EVOUT2	EV_BASE + 0x50
EVOUT4	EV_BASE + 0x60
EVOUT6	EV_BASE + 0x70

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INV	ENABLE				REN	COMB			DHIGH	IMSS	IMSSE	TDM0	TDM1	NMID	M1C
TYPE	R/W															
RESET	0	0xF				1	0x0			0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	DTIN			DEVNT			GEVIN1	GEVIN0	DEC0	DEE			GDRACP		
TYPE	R/W															
RESET	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0

EVOUTx selects how sources are combined and programs the resulting action(s) for event multiplexers 0, 2, 4 and 6. Bits 22 through 16 are for input selection, not output action. More than one action is permitted at a time. For details, see **Section 15.4.6**, *Restrictions on Multiple Drivers*, on page 15-16.

Table 15-10. EVOUTx Bit Descriptions

Bit x	Reset	Description	Description
INV 31	0	**Invert Result** Specifies whether an invert operation is performed on the signal or set of signals.	0 Result not inverted. 1 Result inverted.
ENABLE 30–27	0xF	**Event Enable** Determines whether the event multiplexer operates independently (always enabled) or in a cascaded sequence. In the sequence, this multiplexer is enabled by a signal from either the emulator or the next event multiplexer. When an event multiplexer is enabled, it remains enabled until a reset occurs or its REN bit is set. The REN bit overrides the enable.	0000 Enabled by the EE0 signal. 0001 Enabled by the EE1 signal. 0010 Enabled by the EE2 signal. 0011 Enabled by the EE3 signal. 0100 Enabled by the EE4 signal. 0101 Enabled by the EE5 signal. 0110–0111 Reserved. 1000 By EED ANDed with EE0. 1001 By EED ANDed with EE1. 1010 By EED ANDed with EE2. 1011 By EED ANDed with EE3. 1100 By EED ANDed with EE4. 1101 By EED ANDed with EE5. 1110 Enabled by MUX i + 1. 1111 Always enabled.
REN 26	1	**Reset Enable** Resets the enable sequences. The full functionality of the REN bit is described in **Table 15-6**. When the REN bit is cleared, there is a delay of one IPBus clock cycle. The procedure for setting this bit correctly is described in **Section 15.6.2**, *Reset An Event Multiplexer,* on page 15-19.	0 Normal operation of enable. 1 Resets the enable; the event multiplexer cannot trigger or pass any event through.
COMB 25–23	0	**Combination Selection** Combines the input sources to an event multiplexer. See **Section 15.3**, *Event Multiplexer Combining Logic*, for a description of signal combining and the operation of these bits.	000 Sources are ORed together. 001 Sources are ANDed together. 010 Set operation. 011 Toggle operation. 100 Sources are XORed together. 101 Reserved. 110 Reserved. 111 Set-reset operation.
DHIGH 22	0	**DMA Priority Elevation** Detects a DMA priority elevation of transfer control descriptors (TCDs).	0 Disable detection. 1 Enable detection.
IMSS 21	0	**ICache Miss All Cases** Detects an ICache miss that is not a prefetch hit.	0 Disable detection. 1 Enable detection.
IMSSEX 20	0	**ICache Miss to External Memory** Detects an ICache miss to external memory that is not a prefetch hit.	0 Disable detection. 1 Enable detection.
TDM0 19	0	**Receive Interrupt Request** Detects a receive interrupt request for TDM0.	0 Disable detection. 1 Enable detection.
TDM1 18	0	**Receive Interrupt Request** Detects a receive interrupt request for TDM1.	0 Disable detection. 1 Enable detection.

Table 15-10. EVOUTx Bit Descriptions (Continued)

Bit x	Reset	Description	Description
NMID 17	0	**NMI Interrupt Request** ORs device-level NMI requests.	0 Disable detection. 1 Enable detection.
M1C 16	0	**Contention at M1 Memory** Detects an M1 memory contention.	0 Disable detection. 1 Enable detection.
— 15	0	Reserved. Write to zero for future compatibility.	
DTIN 14–12	0	**Drive TINx Pins** Drives the timer input pins by the detected result. Drives the inputs for all timer modules.	000 Disabled. TINx signals not driven. 001 Reserved. 010 Reserved. 011 Reserved. 100 Drive TIN0 signal with result. 101 Drive TIN1 signal with result. 110 Drive TIN2 signal with result. 111 Drive TIN3 signal with result.
DEVNT 11–9	0x7	**Drive EVNT Pins** Drives the EVNT pins either by the detected result or by a direct connection between an EVNT pin and a timer. The timer connections provide a direct path for high-speed counter operation.	000 Drive EVNT0 signal with result. 001 Drive EVNT1 signal with result. 010 Drive EVNT2 signal with result. 011 Drive EVNT3 signal with result. 100 Drive EVNT4 signal with result. 101 Directly connects TOUTx to EVNTx: – TOUT0, EVNT0 for MUX0. – TOUT2, EVNT2 for MUX2. – TOUT0, EVNT0 for MUX4. – TOUT2, EVNT2 for MUX6. **Note:** TOUTx from timer module A only. 110 Directly connects EVNTx to TINx: – EVNT0, TIN0 for MUX0. – EVNT2, TIN2 for MUX2. – EVNT0, TIN0 for MUX4. – EVNT2, TIN2 for MUX6. 111 Disabled. EVNT pins not driven.
GEVIN1 8	0	**Generate EVINT1 Request** Requests interrupt servicing.	0 Interrupt request not driven. 1 Interrupt request driven.
GEVIN0 7	0	**Generate EVINT0 Request** Requests interrupt servicing.	0 Interrupt request not driven. 1 Interrupt request driven.
DEC0 6	0	**Drive EC0 Signal** Specifies whether the emulator receives EC0 from the event multiplexer.	0 Signal not driven. 1 Drive emulator signal with result.

Table 15-10. EVOUTx Bit Descriptions (Continued)

Bit x	Reset	Description	Description
DEE 5–3	0x7	**Drive EE[0–5] Signals** Drives the emulator EEx signals when a trigger is detected.	000 Drive EE0 signal with result. 001 Drive EE1 signal with result. 010 Drive EE2 signal with result. 011 Drive EE3 signal with result. 100 Drive EE4 signal with result. 101 Drive EE5 signal with result. 110 Reserved. 111 Disabled. EEx signals not asserted.
GDRACP 2–0	0x0	**Generate DMA Requests, Alternate Crossbar Priority** Request a DMA transfer or switch to the crossbar switch alternate priority register set. The value of the EVGP0 signal is latched in the HDI16 ISR[EV0] bit. The value of the EVGP1 signal is latched in the HDI16 ISR[EV1] bit.	000 Disabled. No requests are driven. 001 Switch crossbar to alternate priority. 010 Drive the DMA EVDMA0 request. 011 Drive the DMA EVDMA1 request. 100 Reserved. 101 Reserved. 110 Drive EVGP0 request. 111 Drive EVGP1 request.

EVOUTx Event Output Register

EVOUT1 EV_BASE + 0x48
EVOUT3 EV_BASE + 0x58
EVOUT5 EV_BASE + 0x68
EVOUT7 EV_BASE + 0x78

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INV	ENABLE				REN	COMB			DHIGH	IMSS	IMSSE	TDM0	TDM1	NMID	M1C
TYPE	R/W															
RESET	0	0xF				1	0x0			0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	DTIN			DEVNT			GEVIN1	GEVIN0	DEED	DEE			GDRACP		
TYPE	R/W															
RESET	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0

EVOUTx select how sources are combined and programs the resulting action(s) for event multiplexers 1, 3, 5, and 7. This register provides additional bits for input selection. Bits 22 through 16 are used for input selection, not output action. More than one action is permitted at a time. For details, see **Section 15.4.6**, *Restrictions on Multiple Drivers*, on page 15-16.

Table 15-11. EVOUTx Bit Descriptions

Bit x	Reset	Description	Description
INV 31	0	**Invert Result** Specifies whether an invert operation is performed on the signal or set of signals.	0 Result not inverted. 1 Result inverted.
ENABLE 30–27	0xF	**Event Enable** Determines whether the event multiplexer operates independently (always enabled) or in a cascaded sequence. In the sequence, this multiplexer is enabled by a signal from either the emulator or the next event multiplexer. When an event multiplexer is enabled, it remains enabled until a reset occurs or its REN bit is set. The REN bit overrides the enable.	0000 Enabled by the EE0 signal. 0001 Enabled by the EE1 signal. 0010 Enabled by the EE2 signal. 0011 Enabled by the EE3 signal. 0100 Enabled by the EE4 signal. 0101 Enabled by the EE5 signal. 0110–0111 Reserved. 1000 By EED ANDed with EE0. 1001 By EED ANDed with EE1. 1010 By EED ANDed with EE2. 1011 By EED ANDed with EE3. 1100 By EED ANDed with EE4. 1101 By EED ANDed with EE5. 1110 Enabled by MUX i + 1. 1111 Always enabled.
REN 26	1	**Reset Enable** Resets the enable sequences. The functionality of the REN bit is described in **Table 15-6**, *REN Bit Usage,* on page 15-20. The procedure for setting this bit correctly is described in **Section 15.6.2**, *Reset An Event Multiplexer,* on page 15-19.	0 Normal operation of enable. 1 Resets the enable; the event multiplexer cannot trigger or pass any event through.
COMB 25–23	0	**Combination Selection** Combines the input sources to an event multiplexer. See **Section 15.3**, *Event Multiplexer Combining Logic*, for a description of signal combining and the operation of these bits.	000 Sources are ORed together. 001 Sources are ANDed together. 010 Set operation. 011 Toggle operation. 100 Sources are XORed together. 101 Reserved. 110 Reserved. 111 Set-reset operation.
DHIGH 22	0	**DMA Priority Elevation** Detects a DMA priority elevation of transfer control descriptors (TCDs).	0 Disable detection. 1 Enable detection.
IMSS 21	0	**ICache Miss - All Cases** Detects an ICache miss that is not a prefetch hit.	0 Disable detection. 1 Enable detection.
IMSSEX 20	0	**ICache Miss to External Memory** Detects an ICache miss to external memory that is not a prefetch hit.	0 Disable detection. 1 Enable detection.
TDM0 19	0	**Receive Interrupt Request** Detects a receive interrupt request for TDM0.	0 Disable detection. 1 Enable detection.
TDM1 18	0	**Receive Interrupt Request** Detects a receive interrupt request for TDM1.	0 Disable detection. 1 Enable detection.
NMID 17	0	**NMI Interrupt Request** ORs device-level $\overline{\text{NMI}}$ requests.	0 Disable detection. 1 Enable detection.

Table 15-11. EVOUTx Bit Descriptions (Continued)

Bit x	Reset	Description	Description
M1C 16	0	**Contention at M1 Memory** Detects an M1 memory contention.	0 Disable detection. 1 Enable detection.
— 15	0	Reserved. Write to zero for future compatibility.	
DTIN 14–12	0	**Drive TINx Pins** Drives the timer input pins by the detected result. Drives the inputs for all timer modules.	000 Disabled. TINx signals not driven. 001 Reserved. 010 Reserved. 011 Reserved. 100 Drive TIN0 signal with result. 101 Drive TIN1 signal with result. 110 Drive TIN2 signal with result. 111 Drive TIN3 signal with result.
DEVNT 11–9	0x7	**Drive EVNT Pins** Drives the EVNT pins either by the detected result or by a direct connection between an EVNT pin and a timer. The timer connections provide a direct path for high-speed counter operation.	000 Drive EVNT0 signal with result. 001 Drive EVNT1 signal with result. 010 Drive EVNT2 signal with result. 011 Drive EVNT3 signal with result. 100 Drive EVNT4 signal with result. 101 Directly connects TOUTx to EVNTx: – TOUT1, EVNT1 for MUX1. – TOUT3, EVNT3 for MUX3. – TOUT1, EVNT1 for MUX5. – TOUT3, EVNT3 for MUX7. **Note:** TOUTx from timer module A only. 110 Directly connects EVNTx to TINx: – EVNT1, TIN1 for MUX1. – EVNT3, TIN3 for MUX3. – EVNT1, TIN1 for MUX5. – EVNT3, TIN3 for MUX7. 111 Disabled. EVNT pins not driven.
GEVIN1 8	0	**Generate EVINT1 Request** Requests interrupt servicing.	0 Interrupt request not driven. 1 Interrupt request is driven.
GEVIN0 7	0	**Generate EVINT0 Request** Requests interrupt servicing.	0 Interrupt request not driven. 1 Interrupt request is driven.
DEED 6	0	**Drive EED Signal** Specifies whether the emulator receives EED from multiplexer.	0 EED signal not driven. 1 Drive emulator signal with result.
DEE 5–3	0x7	**Drive EE[0–5] Signals** Drives the OCE10 on-chip emulator EEx signals when a trigger is detected.	000 Drive EE0 signal with result. 001 Drive EE1 signal with result. 010 Drive EE2 signal with result. 011 Drive EE3 signal with result. 100 Drive EE4 signal with result. 101 Drive EE5 signal with result. 110 Reserved. 111 Disabled. EEx signals not asserted.

Table 15-11. EVOUTx Bit Descriptions (Continued)

Bit x	Reset	Description	Description	
GDRACP 2–0	0x0	**Generate DMA Requests, Alternate Crossbar Priority** Request a DMA transfer or switch to the crossbar switch alternate priority register set. The value of the EVGP0 signal is latched in the HDI16 ISR[EV0] bit. The value of the EVGP1 signal is latched in the HDI16 ISR[EV1] bit.	000 001 010 011 100 101 110 111	Disabled. No requests are driven. Switch crossbar to alternate priority. Drive the DMA EVDMA0 request. Drive the DMA EVDMA1 request. Reserved. Reserved. Drive EVGP0 request. Drive EVGP1 request.

Debugging 16

The MSC711x dedicated user-accessible test access port (TAP) is fully compatible with the **IEEE**® Std 1149.1™ standard test access port and boundary scan architecture. Problems associated with testing high-density circuit boards led to development of this standard under the sponsorship of the test technology committee of **IEEE** and the Joint Test Action Group (JTAG). The MSC711x supports circuit-board test strategies based on this standard. This chapter covers aspects of JTAG that are specific to the MSC711x. It includes the items that the standard requires to be defined, with additional information specific to the MSC711x. For details on the standard, refer to the **IEEE** Std 1149.1 documentation.

The JTAG port also provides access to the OCE10 on-chip emulator module, a dedicated block for debugging applications. This on-chip emulator module is simply referred to as the “emulator” in the remainder of this chapter, which presents information on registers and functionality of the emulator that are specific to the MSC711x. For details on emulator functionality, consult the *OCE10 On-Chip Emulator Reference Manual*, which is available at the web site listed on the back of this manual.

The SC1400 core emulator interfaces with the SC1400 core and its peripherals non-intrusively so that you can examine registers, memory, or on-device peripherals, thus facilitating hardware and software development on the SC1400 core-based devices. Special circuits and dedicated signals on the SC1400 core protect user-accessible internal resource. As the DSP applications grow in both size and complexity, the emulator breakpoints, conditional breakpoints, breakpoints on data-bus values, and event detection offer you non-intrusive access to peripherals, variety in profiling, a program tracing buffer, and real-time access to memory. 1KB of the boot ROM is reserved for use by the development tools.

16.1 Debugging Modes

It is important to understand the different debugging modes on an MSC711x device:

- *SC1400 Debug mode*. The SC1400 core is halted during a debug session by breakpoints, single stepping, and so on. For details, see **Section 16.2**, *Emulator*, on page 16-2 for details.
- *Device Debug mode*. Portions of the MSC711x device are halted during a debug session initiated by entry into SC1400 Debug mode. For example, you can optionally hold off operations in the following modules in Device Debug mode:
  - Interrupt controller (see **Section 12.1.3**, *Operation in Debug Mode*, on page 12-3)

  - DMA controller via the DMACR[EDBG] bit (see **page 8-27**)
  - Software watchdog timer (see **Section 7.3.1.2**, *Pause Mechanism*, on page 7-6)
- *Cache Debug mode*. For accessing the contents of the ICache array, tag array, and valid bit array. This mode is independent of the SC1400 core and device Debug modes. That is, this mode can be used when the SC1400 core is in the normal processing state or in the debug processing state. For details, see **Section 4.5.5**, *Debugging Support*, on page 4-23.

16.2 Emulator

The emulation and debug capability on the MSC711x devices eliminates the need for expensive and complicated stand-alone in-circuit emulators (ICEs). This section describes the emulation environment for use in debugging real-time embedded applications. You can use the emulator to perform the following tasks:

- Examine or modify the contents of any SC1400 core or memory-mapped peripheral register.
- Examine and modify program or data memory.
- Step at full speed on one or more instructions.
- Save a programmable change-of-flow instruction capture to the trace buffer.
- Display the contents of the real-time instruction trace buffer.
- Transfer data in real time between the SC1400 core and an external host using peripheral-mapped transmit and receive registers.
- Access emulator registers and programming model:
  - From the programming model of the MSC711x device (via SC1400 code).
  - Through the development system that accesses the resources through the JTAG port.
- Provide status of emulator events in a status register or on an output pin from the SC1400 core.
- Perform event counting.
- Enter debug mode via:
  - Execution of a SC1400 instruction.
  - The actions of the emulator.
  - The core JTAG port.
  - A special debug request input pin to the SC1400 core.
- Interrupt or break into debug mode on program memory addresses (fetch, read, write, or read and write access).
- Interrupt or break into debug mode on accesses to data memory or on-device peripheral registers (read, write, or read and write access) and for byte, word, or long data type accesses.
- Save or restore the current state of the device pipeline.

- Return to normal user mode from Debug mode.

Note: For details on the emulator features, consult the *OCE10 On-Chip Emulator Reference Manual*.

16.2.1 Emulator System-Level View

As **Figure 16-1** shows, the emulator can be viewed as a separate module that acts concurrently with the SC1400 core. Alternatively, SC1400 software can directly program, control, and communicate with the emulator.

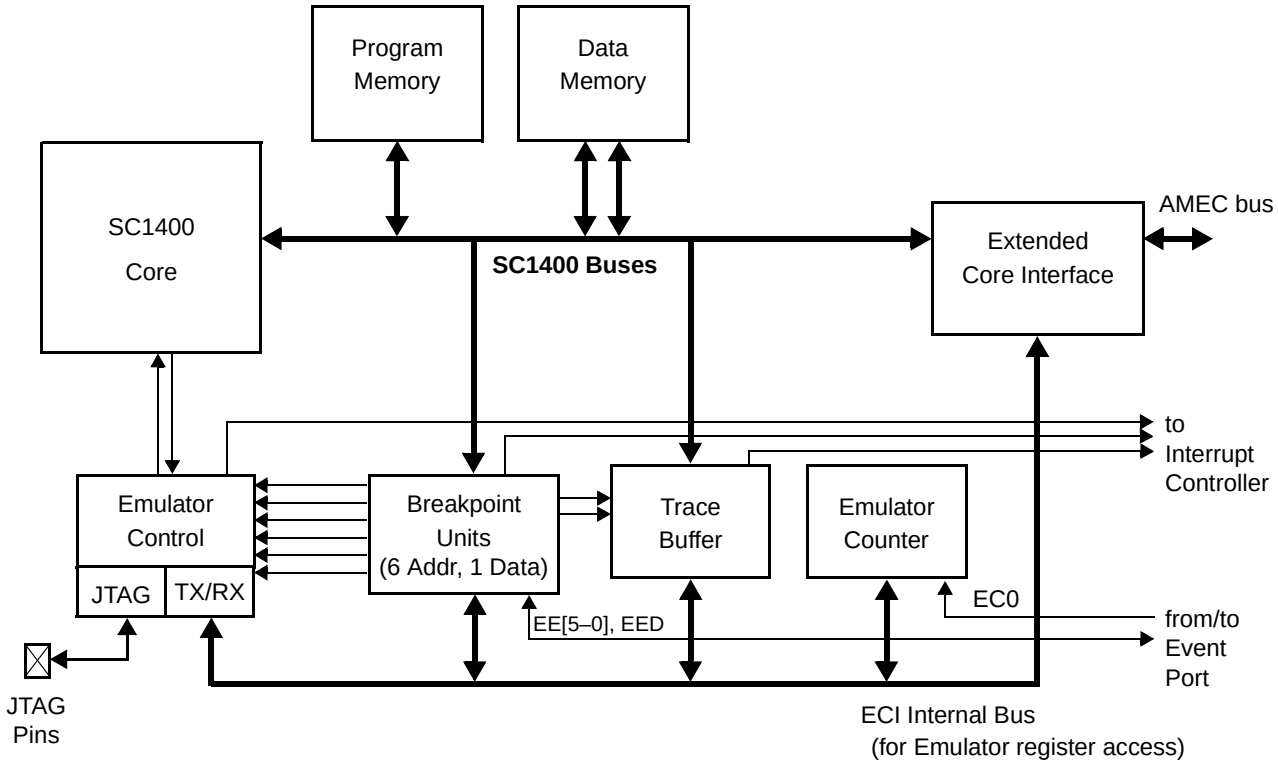


Figure 16-1. SC1400 Device with Debug Port

After it is properly initialized and programmed for breakpoint triggering and associated actions, the emulator operates in parallel with the SC1400 core. As the SC1400 core executes instructions, the emulator performs the following tasks:

- Receive new emulator commands.
- Read/Write emulator registers through the JTAG interface (also accessible through the SC1400 core system buses).
- Monitor SC1400 buses for breakpoint conditions.
- Capture SC1400 program addresses in the trace buffer.
- Generate an emulator interrupt request.
- Halt the SC1400 core upon a certain debug event so it enters the Debug processing state.

When the SC1400 core is halted during Debug processing, the emulator can still receive new commands and read or write any emulator register.

16.2.2 Accessing the Emulator

Resources in the emulator are accessed either through the JTAG port or under software program control through the MSC711x programming model. Therefore, debugging activity can be controlled either by a host development system or by a program executing on the MSC711x device.

16.2.2.1 Access through the JTAG Port

Development and debugging systems control emulator debugging actions by communicating with the emulator through the JTAG port. All emulator resources are available serially through the normal JTAG access protocol. While they are interacting, the MSC711x JTAG and emulators are tightly coupled, and the JTAG port handles the interface for both modules, communicating with the host software development and debug systems. The emulator uses the JTAG external serial interface to send and receive debugging commands and data. **Figure 16-2** shows a block diagram of the JTAG/emulators and the JTAG terminals used in the external interface. The JTAG port can also act as a completely independent module. When it is disabled, it has no impact on the function of the SC1400 core.

Note: For details on accessing the emulator from the JTAG port, see **Section 16.5**, *Accessing the Emulator Through the JTAG Port*, on page 16-21.

16.2.2.2 Access from the MSC711x Memory Map

The SC1400 core and other MSC711x masters can access the emulator through the MSC711x programming model as memory-mapped registers, independently of the JTAG port. All emulator resources are available through the memory-mapped registers, allowing access to the port via normal instruction execution. The SC1400 core can initialize the emulator, use its resources, and monitor its actions under program control. Data is uploaded or downloaded between the SC1400 core and each emulator sub-module. Both polled and interrupt driven communications can occur between the SC1400 core and the emulator.

16.3 System-Level Debugging

The emulator port and the event port work closely together to provide system-level debugging. The emulator port gives access to SC1400 core resources within the extended core. Using the emulator, you can perform the following tasks:

- Set hardware breakpoints.
- Capture information in the trace buffer.
- Halt the SC1400 core and enter Debug mode on debug-related events.

- Generate a debug exception on debug-related events.
- Use the full speed event counter (can clock at the core frequency).
- Enable or view emulator events via pins on the MSC711x device.

As a centralized location for processing debug information, the event port works closely with the emulator debug port, EVNTx pins, timer modules, interrupt controller, and clock controller.

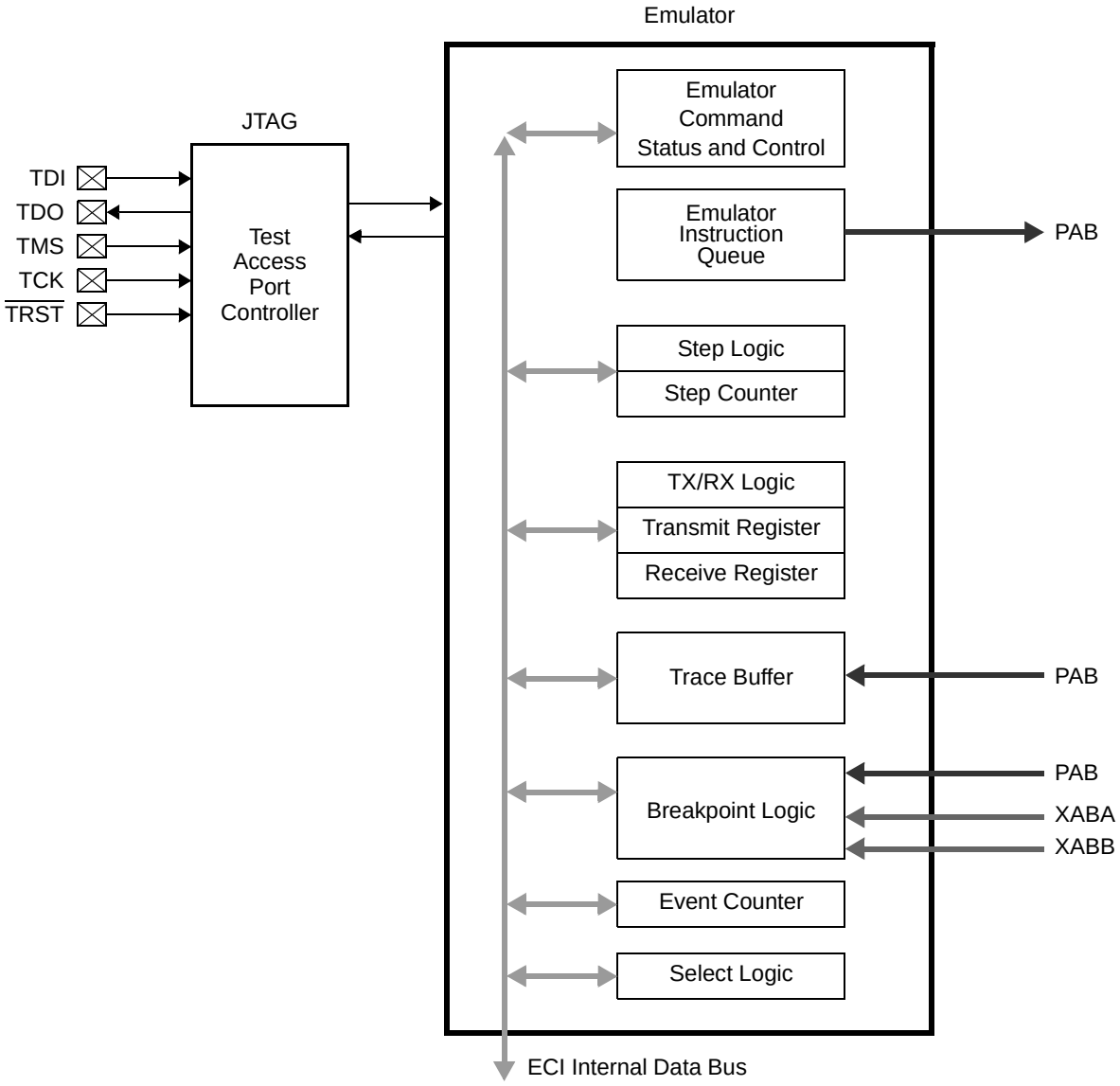


Figure 16-2. JTAG/OCE10 On-Chip Emulator Interface Block Diagram

The event port collects system events, such as DMA activity, system interrupts, or loss of PLL lock, combines this information as specified by the user, and sends the resulting signal to the appropriate module. Because the emulator is interlocked with the event port, it can:

- Enable or disable the trace buffer.
- Enable the emulator event counter.

- Force entry into Debug mode.

Breakpoint activity can be sequenced in the emulator, as can events in the event port, so you can set up triggering sequences using both emulator and event port resources. For examples, see **Section 15.7**, *Event Sequencing*, on page 15-20.

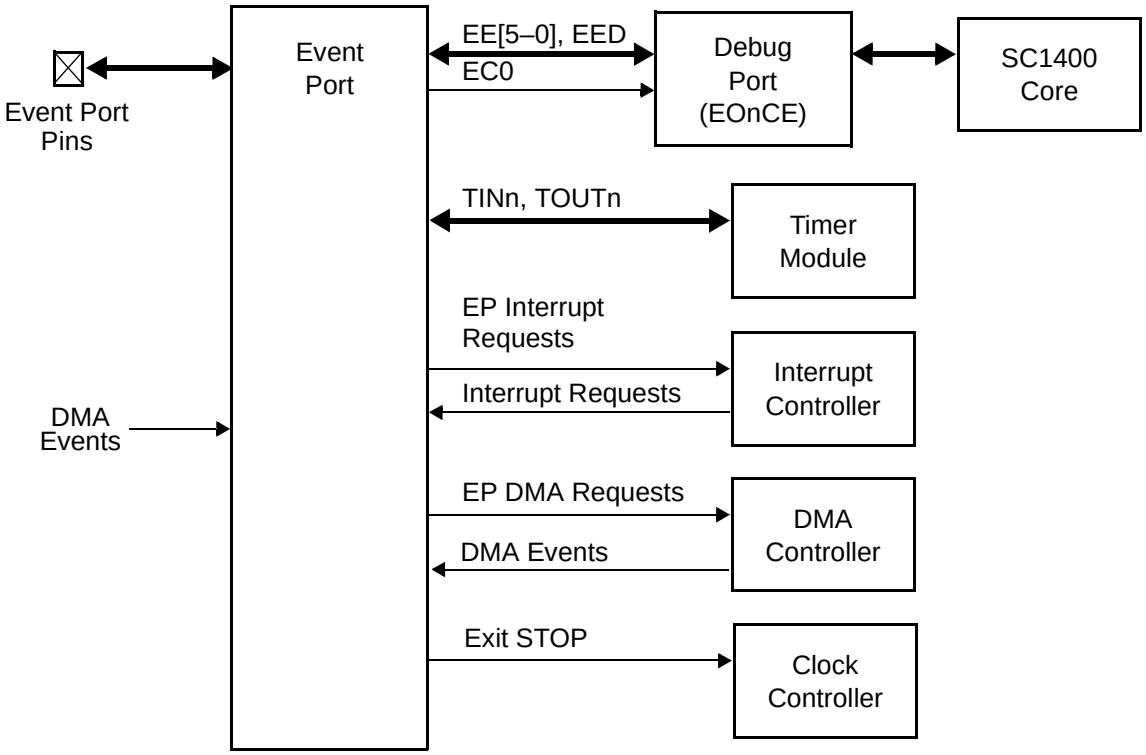


Figure 16-3. Centralized Debugging with the MSC711x Event and Debug Ports

Figure 16-4 presents a system-level view of the event port within an MSC711x device. This diagram shows how the event port interacts with the debug port (emulator), the EVNTx pins, timer modules, and interrupt controller. The close connectivity between the event port and the emulator gives the emulator debugger access to the following resources that are available to the event port:

- EVNTx pins (input or output)
- General-purpose timers
- Sequencing hardware withint the event port
- Trigger signals in the event port
- Event port interrupts

16.3.1 System-Level Emulator Signals

The emulator communicates with the rest of the MSC711x device via its EEn, EED, and EC0 signals, which pass through the event port to provide direct communication between the emulator and the MSC711x EVNT[0–5] pins. The event port has access to many different system-level signals to indicate DMA data transfers and other activity, as well as access to system-level resources such as the timers or EVNT[4–0] pins. The EE[0–5] and EED signals are used as follows:

- Configured as inputs:
  - Enable the emulator breakpoint unit.
  - Generate an emulator action.
  - Directly enter Debug mode (EE4 signal).
- Configured as outputs:
  - Enable an event port multiplexer.
  - Serve as an input to the event port multiplexers (EE4 signal only).

The EC0 signal is used only as an input to the emulator for counting event port (system-level) events or events occurring on the EVNT[4–0] pins.

16.3.2 SC1400 Emulator Instructions

Certain SC1400 instructions trigger actions within the emulator:

- *DEBUGEV*. Software breakpoint trigger.
- *DEBUGHLT*. Halt SC1400 core and enter Debug mode.
- *MARK*. Mark value in trace buffer.

DEBUGEV and DEBUGHLT trigger actions in the emulator. The MARK instruction explicitly updates the trace buffer when an SC1400 program is running.

16.3.3 Halting the SC1400 Core and Entering Debug Mode

The SC1400 core enters Debug mode in response to an emulator event, which halts all activity on the core. Debug mode is one of the processing states of the SC1400 core. When SC1400 instructions are single-stepped, the SC1400 core returns to Debug mode after executing each instruction. Debug has the following system-level effects:

- The watchdog clock to the software watchdog timer pauses, preventing an accidental time-out.
- Maskable and non-maskable interrupt requests can be optionally gated off, via the MIPR[DDBG] bit, so that interrupts do not accumulate during the many clock cycles the MSC711x device is in Debug mode (see **Section 12.1**, *Interrupt Controller Architecture*, on page 12-1).
- The DMA controller can optionally be configured, via the DMACR[EDBG] bit, to halt after completion of the current channel’s minor loop.
- All of these halt events are initiated when the emulator controller successfully halts the SC1400 core, and they remain in effect until the SC1400 core leaves Debug mode and returns to normal instruction processing. This is true when the SC1400 single-steps through lines of assembly code, except when stepping *over* a jsr statement, which results in the execution of an entire subroutine.

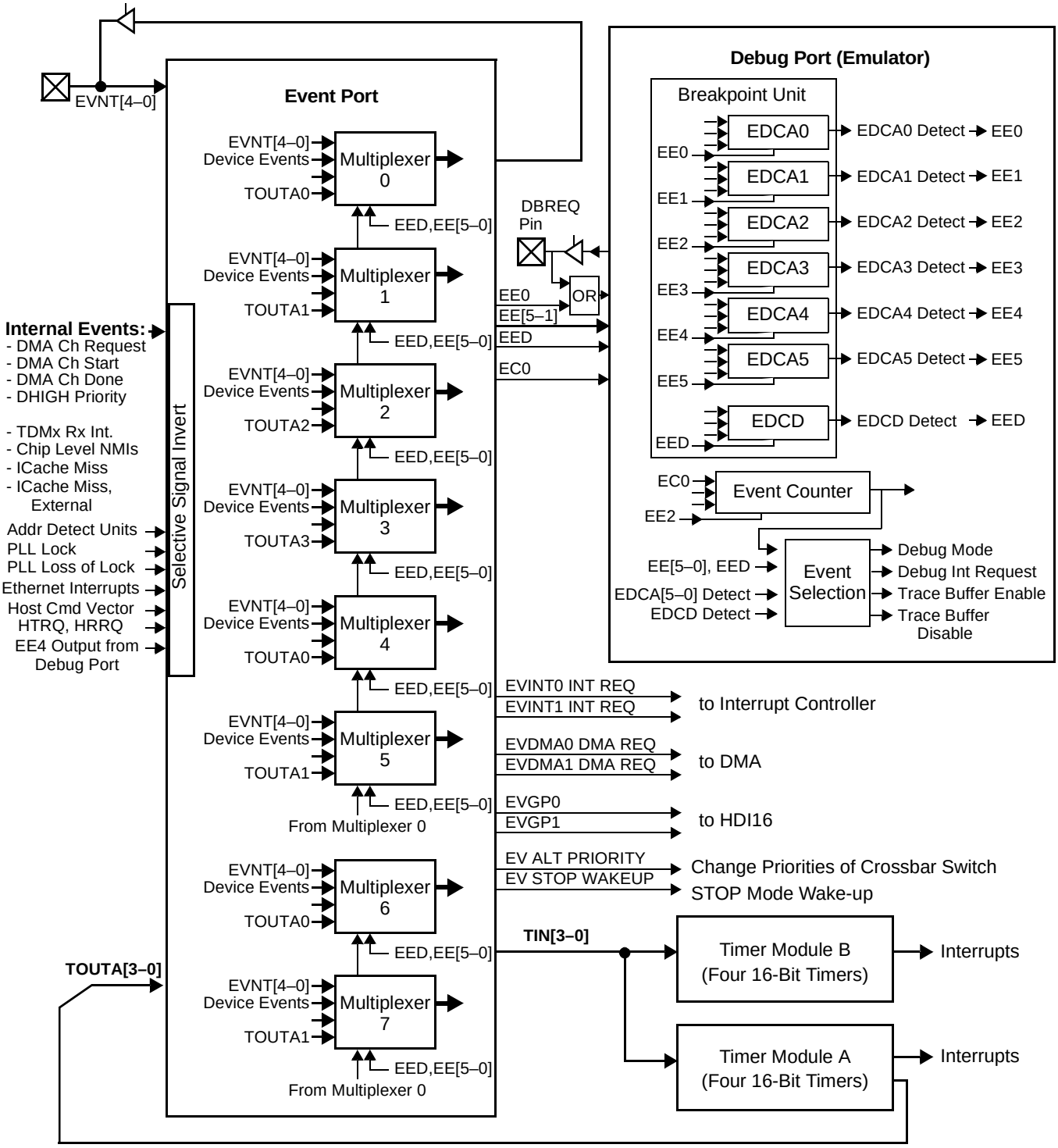


Figure 16-4. System-Level View of MSC711x Event Port

However, setting breakpoints on source-level code or single-stepping through lines of C code uses a different mechanism than is used for single-stepping assembly instructions. In these cases, chip-level events resume operation. For example, when single stepping through lines of a C program using the SC1400 development environment, these events resume activity while one line of C source code executes.

The crossbar switch remains active in Debug mode so that the debugger can access peripheral registers and memory locations throughout the system. The SC1400 core supports six processing states, as shown in **Table 16-1**.

Table 16-1. SC1400 Core Processing States

State	Description
Normal	The state of the SC1400 core when instructions execute normally.
Reset	The SC1400 core is forced into a known reset state. The first program instruction is fetched when the SC1400 core exits this state.
Exception	The state of interrupt processing. The SC1400 core transfers program control from its current location to an interrupt service routine using the interrupt vector table.
Wait	A low-power state in which the SC1400 core is shut down but the peripherals and the interrupt machine remain active.
Stop	A low-power state in which the SC1400 core, the interrupt machine, and most (if not all) of the peripherals are shut down. Care must be taken when the SC1400 core is in Stop state. All core clocks are disabled and the emulator is inaccessible. The device status is polled through the JTAG interface (sampled in the capture-IR state). The core JTAG TAP brings the SC1400 core out of Stop or Wait modes when DEBUG_REQUEST is decoded in the TAP IR. A small amount of additional power above the minimum possible is expended by the core TAP logic if the core TAP is used in Stop mode.
Debug	The SC1400 core halts and all emulator registers are accessible for program debug.

When a breakpoint is set on a line of source code, the applications runs until it reaches the breakpoint in the line, and the MSC711x halts upon reaching it. In the Debug state, breakpoints and other resources are initialized and set up for debugging, and MSC711x registers and memory locations are examined and modified. The device is often placed into the Debug state to initialize the emulator for a debug system. Also, the SC1400 core can enter the Debug state immediately upon exiting reset to set up a debug session before it begins executing instructions. Any of the following actions can put the SC1400 core into the Debug state:

- Hardware reset with JTAG DEBUG_REQUEST in the JTAG Instruction Register (IR).
- JTAG DEBUG_REQUEST placed into the JTAG IR during STOP mode or WAIT mode or wait states.
- Execution of the DEBUGHLT instruction while the emulator is powered up.
- The step counter expires while configured for a debug request.
- The trace buffer is full and configured for a debug request.
- A breakpoint trigger occurs when programmed for debug request.

The SC1400 core can be in any of processing states listed in **Table 16-1** when a request to enter Debug mode arrives. However, you do not have to place the SC1400 core into the Debug processing state to initialize the module. Alternatively, you can set up the desired emulator resources and enable them through the JTAG port or through SC1400 core access via set-up routines in an application, typically executed in the normal processing state.

The emulator can also generate interrupt requests in response to debug events. When the emulator exception trap detects a debug event, an interrupt can be generated and the program can initiate the appropriate handler routine. The SC1400 core performs many different actions in response to Debug events without halting while an event is serviced by a dedicated interrupt service routine.

Note: Take care when the SC1400 core operates in the Stop processing states. The core clock is disabled and the emulator is inaccessible. The JTAG interface is used to poll the device status (sampled in the capture-IR state). The core JTAG TAP brings the SC1400 out of Stop or Wait mode when DEBUG_REQUEST is decoded in the TAP IR. The SC1400 core TAP logic expends a small amount of power above the minimum if the SC1400 core TAP is used in Stop mode.

16.3.4 Exiting SC1400 Debug Mode

When the SC1400 core enters Debug mode, its emulator EE0 signal is masked, preventing further debug requests. The status of the SC1400 core is updated in the Debug Instruction Register CORES field (**Table 16-3**), which is accessible through the JTAG port.

When an SC1400 core exits Debug mode, the EE0 internal signal is unmasked, enabling further debug requests. A GO instruction restarts the SC1400 core. No retriggering occurs through EE0. For stepping, the same arrangement is used with the STEP instruction. The SC1400 core is enabled via the CHOOSE_EONCE command, and then a STEP instruction is scanned into the SC1400 core. When the scan is complete, the update launches the SC1400 core. No retriggering occurs through EE0.

16.4 MSC711x JTAG Port

The JTAG test access port (TAP) provides boundary scan for the MSC711x device as well as an interface for accessing the emulators through the device JTAG pins. **Table 16-2** lists the test access port (TAP) signals.

Table 16-2. TAP Signals

Signal	Description
TCK	A test clock input to synchronize the test logic.
TMS	A test mode select input (with an internal pull-up resistor) that is sampled on the rising edge of TCK to sequence the TAP controller state machine.
TDI	A test data input (with an internal pull-up resistor) that is sampled on the rising edge of TCK.
TDO	A data output that can be tri-stated and actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO changes on the falling edge of TCK.
$\overline{\text{TRST}}$	An asynchronous reset (with an internal pull-up resistor) that initializes the TAP controller and other logic required by the standard.

There are two TAP controllers on MSC711x devices: a boundary scan TAP controller and a debug TAP controller. The TAP controller accessed through the JTAG pins on the MSC711x device is selected via the TPSEL pin. The capabilities of each controller are covered in **Section 16.4.3**, *JTAG Instruction Decoding*, on page 16-14.

16.4.1 Boundary Scan TAP Controller

The boundary scan TAP consists of five dedicated signal lines, a 16-state TAP controller, and three test data registers. The test logic, which uses static logic design, is independent of the device system logic. The MSC711x JTAG module performs the following tasks:

- Bypass the MSC711x for a given circuit board test by effectively reducing the Boundary Scan Register (BSR) to a single cell.
- Provide access to the OCE10 emulator controller and its circuits to control a target system.
- Provide direct entry to the SC1400 core Debug mode.
- Query identification information (manufacturer, technology process, part, and version numbers) from an MSC711x-based device.
- Force test data onto the outputs of an MSC711x-based device while replacing its BSR in the serial data path with a single-bit register.

Note: Take precautions to ensure that the **IEEE** 1149.1-like test logic does not interfere with non-test operation.

The JTAG port consists of a serial communications interface, a command decoder and interpreter, and a device ID register (see **Figure 16-5**).

The serial interface provides the communication link between the core and the host development or debug system. All JTAG data is sent over this interface. Emulator commands and data from the host system can also be sent over this interface if they are accessed via JTAG. This interface is a serial interface that minimizes the number of external pins used on the device. For a full description of the interface signals, consult the user's manual for the specific device.

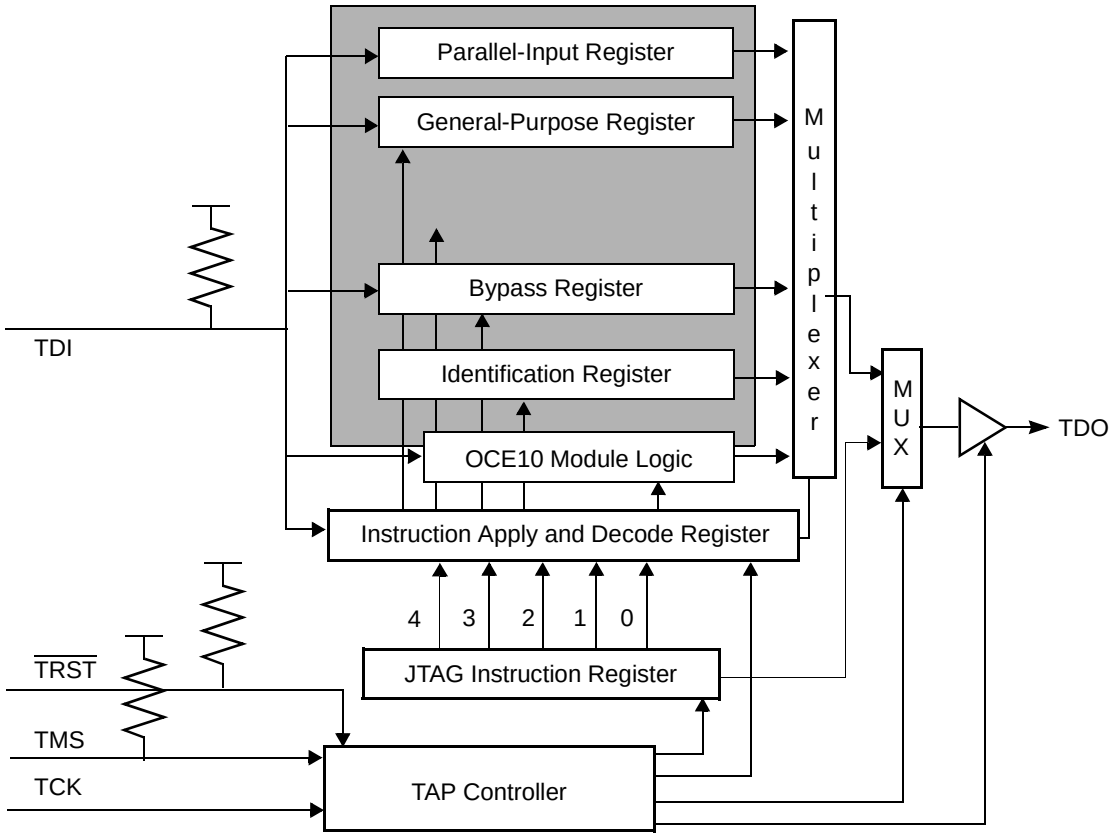


Figure 16-5. Test Logic Block Diagram, Boundary Scan Controller

The command decoder decodes and processes commands sent to the JTAG module. Commands for the JTAG port are completely independent of the SC1400 instruction set and execute in parallel. The JTAG device identification register provides a unique ID for each revision of each MSC711x device. This register enables a development system to determine the manufacturer, process technology, part, and revision numbers of a device through the JTAG port. To access the JTAG registers, shift the appropriate command into the JTAG instruction register and then shift the required value into the register. See **Section 16.4.3** for a discussion of the JTAG instructions. **Figure 16-5** shows the 30-bit boundary scan controller JTAG Instruction Register and the following test registers:

- 1-bit Bypass Register
- 32-bit Device Identification Register (DEVID)
- 32-bit JTAG General-Purpose Register (JGPR)
- 32-bit Parallel Input Register (PIREG)

16.4.2 TAP Controller Operation

The TAP controller is a sixteen state synchronous finite state machine that sequences the JTAG port through its valid operations:

- Serially shift in or out via a JTAG instruction.

- Update (and decode) the JTAG Instruction Register.
- Serially output the ID code.
- Serially shift in or out and update the emulator registers.

The JTAG port oversees the shifting of data to and from the emulator port through the TDI and TDO pins. The shifting is guided by the same tap controller that shifts JTAG Instruction Register (IR) information. The TAP controller (shown in **Figure 16-6**) is a synchronous state machine to control the operation of the JTAG logic. For a description of the TAP controller states, refer to the **IEEE** 1149.1 documentation. Transitions from one state to another occur on the rising edge of TCK. The TAP controller interprets the sequence of logical values on the TMS signal. The value shown adjacent to each state transition in **Figure 16-6** represents the value of the TMS signal at the rising edge of TCK. There are two paths through the 16-state machine. The Shift-IR_Scan path captures and loads JTAG instructions into the JTAG IR. The Shift-DR_Scan path captures and loads data into the other JTAG registers. The TAP controller provides direct access to the JTAG IR through the Select-IR_SCAN state. However, the other JTAG registers must be individually selected by the JTAG IR before the data registers are accessed through the Select-DR_SCAN state.

The TAP controller executes the last instruction decoded until a new instruction is entered in the Update-IR state or until it enters the Test-Logic-Reset state. When emulator registers are accessed through the JTAG port, accesses are first enabled by shifting the ENABLE_EOnCE instruction into the JTAG IR. The emulator registers and commands are read and written through the JTAG pins using the Shift-DR_Scan path. From any TAP controller state, you can return to the Test Logic Reset state by asserting the TMS signal. When TMS is asserted, the state machine transitions through its different states back to the reset state.

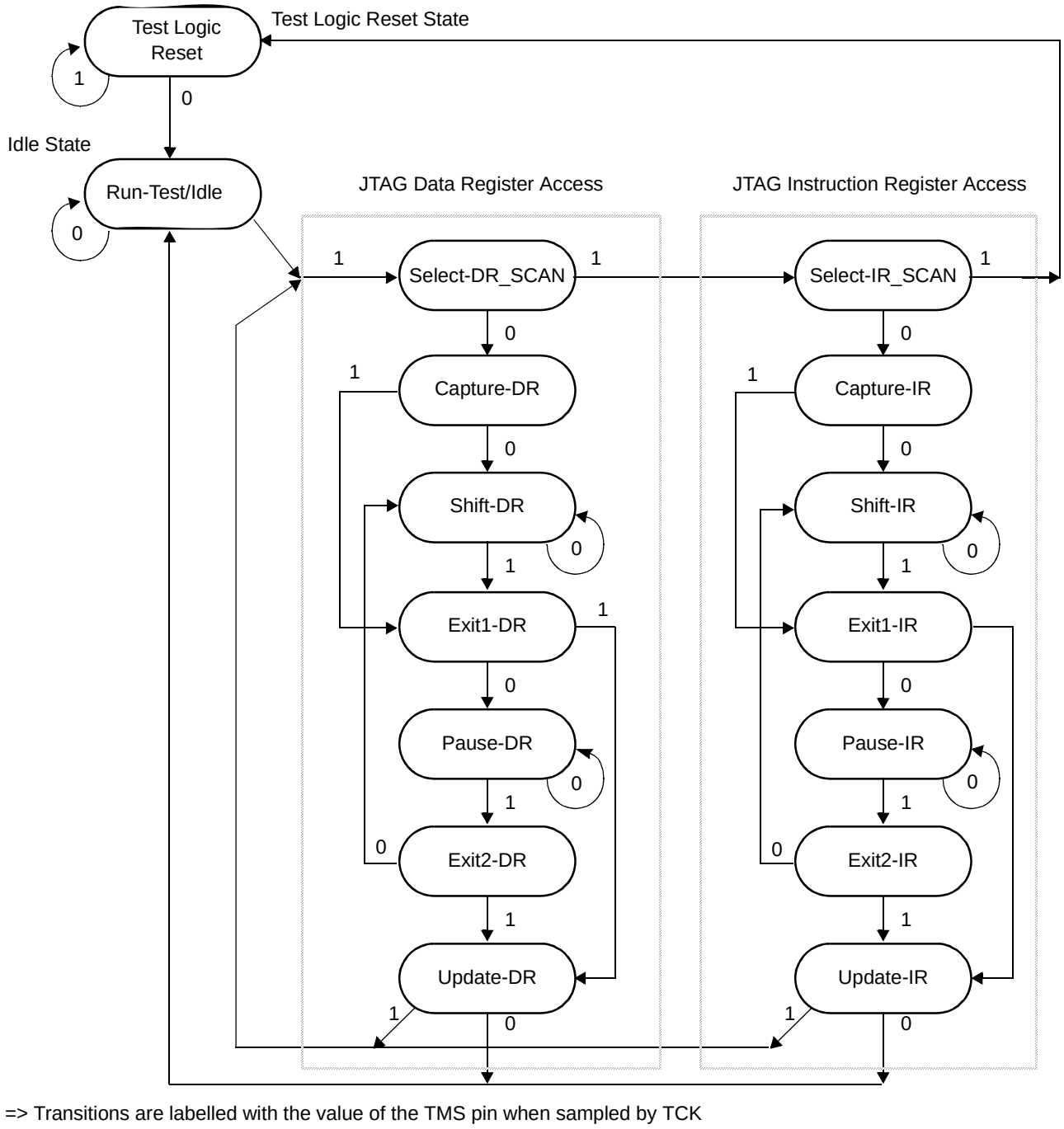


Figure 16-6. TAP Controller State Machine

16.4.3 JTAG Instruction Decoding

Each TAP controller on an MSC711x device supports a different set of JTAG instructions. Each TAP controller contains an instruction register (IR) without parity, consisting of a shift register with five parallel outputs. Data transfers from the shift register to the parallel outputs during the UPDATE-IR controller state. The instruction register bits are decoded to select one of the unique instructions for that TAP controller. All other encodings are reserved for future enhancements and are decoded as BYPASS. **Figure 16-7** shows the structure of a JTAG IR. In the

test-logic-reset controller state, the parallel output of the IR in each TAP controller is reset to 0b00010, which is equivalent to the IDCODE instruction.

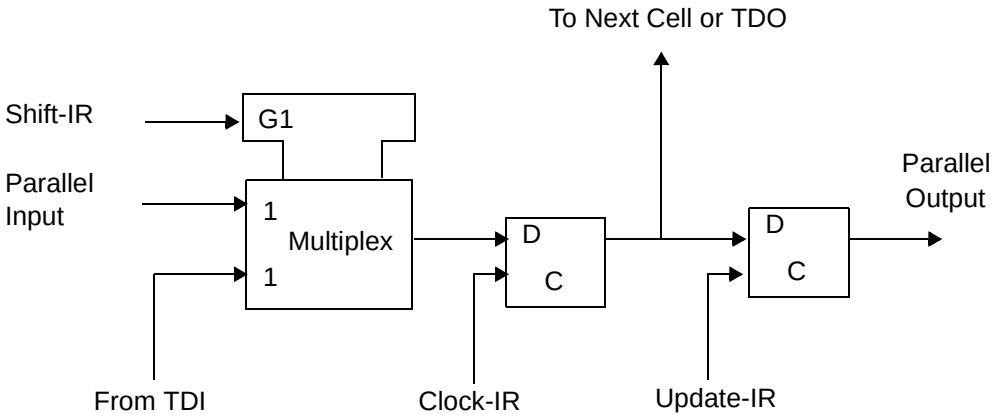


Figure 16-7. Instruction Register (IR) Configuration

During the CAPTURE-IR controller state, the parallel inputs to the 30-bit boundary scan instruction shift register are loaded with the value of 01 in the least significant bits, as required by the standard. The upper bits of this IR capture status information from the SC1400 core. The most significant bits are loaded with the values UPD_ACK, CORES, as shown in **Table 16-3**. Two bits of the GPR are configured to select an SC1400 core, whose status is output from the multiplexer. The SC1400 core can be viewed from the PIREG. For details, refer to the *SC1400 DSP Core Reference Manual*.

JDIR JTAG Debug Instruction Register JTAG port access only

Bit	4	3	2	1	0
	UPD_ACK	CORES		0	1
TYPE	R/W				
RESET	0	0	0	0	1

JDIR is unique among the JTAG registers because it is the only register accessible using the IR path through the state machine (**Figure 16-6**, *TAP Controller State Machine*, on page 16-14). All other JTAG registers are accessed through the DR path of the state machine. The instruction register for the boundary scan controller is not shown.

Table 16-3. JDIR Bit Descriptions

Name/bits	Description	Settings
UPD_ACK 4	**Update Acknowledge** Indicates whether the selected SC1400 emulator has executed the last instruction dispatched to it.	0 Emulator has executed the last instruction. 1 Emulator has not executed the last instruction.
CORES 3–2	**Core Status** Reflects the status of the SC1400 core.	00 Core is executing instructions. 01 Core is in WAIT or STOP mode. 10 Core is waiting for bus. 11 Core is in debug mode.
— 1–0	Contains a value required by the JTAG standard.	Read-only.

16.4.3.1 Boundary Scan TAP Controller Instruction Decoding

The boundary scan TAP controller has three mandatory public instructions: EXTEST, SAMPLE/PRELOAD, and BYPASS. The optional instructions defined by **IEEE** 1149.1 are CLAMP and HIGHZ. **Table 16-4** describes the 30-bit instructions supported by the Boundary Scan Instruction Register:

Table 16-4. JTAG Instruction Decoding for the Boundary Scan TAP Controller

Bits 30-0	Instruction	Description
00000	EXTEST	Selects the Boundary Scan Register (BSR). EXTEST also asserts internal reset for the MSC711x system logic to force a predictable internal state while external boundary scan operations are performed. Using the TAP, the register can: • Scan user-defined values into the output buffers • Capture values presented to inputs • Control the direction of bidirectional signals • Control the output drive of tri-statable outputs For details on the function and use of EXTEST, refer to the **IEEE** 1149.1 documentation.
00001	SAMPLE/PRELOAD	Initializes the BSR output cells prior to the selection of EXTEST. This initialization ensures that known data appears on the outputs when an EXTEST instruction is entered. SAMPLE/PRELOAD also provides a means to obtain a snapshot of system data and control signals. **Note:** There is no internal synchronization between the TCK and CLKOUT. Therefore, to achieve meaningful results, you must provide some form of external synchronization between the JTAG operation at TCK frequency and the system operation CLKOUT frequency.

Table 16-4. JTAG Instruction Decoding for the Boundary Scan TAP Controller (Continued)

Bits 30-0	Instruction	Description
00010	IDCODE	Selects the ID Register. This public instruction allows the manufacturer, part number, and version of a component to be determined through the TAP. The ID Register configuration is as follows: • Bits 31–28: Version Information • Bits 27–12: Customer Part Number • Bits 11–1: Manufacturer Identity One application of the ID Register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. As more components emerge that conform to the **IEEE** 1149.1 standard, it is desirable to allow for a system diagnostic controller unit to interrogate a board design blindly and determine the type of each component in each location. This information is also available for factory process monitoring and for failure mode analysis of assembled boards. The Freescale manufacturer identity number is 0b00000001110. The customer part number consists of two parts: design center number (bits 27–22) and a sequence number (bits 21–12). The design center number is 0b000110. When the IDCODE instruction is decoded, it selects the 32-bit ID Register. The Bypass Register loads a logic zero at the start of a scan cycle, whereas the ID Register loads a logic 1 into its least significant bit. Consequently, examination of the first bit of data shifted out of a component during a test data scan sequence, immediately following exit from test-logic-reset controller state, shows whether such a register is included in the design. As required by the **IEEE** 1149.1 standard, the operation of the test logic has no effect on the operation of the internal system logic when the IDCODE instruction is selected.
00011	CLAMP	Optional in the **IEEE** 1149.1 standard. This public instruction selects the one-bit Bypass Register as the serial path between TDI and TDO, while allowing signals driven from the component to be determined from the Boundary Scan Register. During testing of ICs on PCBs, it may be necessary to place static guarding values on signals that control operation of logic not involved in the test. The EXTEST instruction could be used for this purpose, but since it selects the BSR, the required guarding signals would be loaded as part of the complete serial data stream shifted in, both at the start of the test and each time a new test pattern is entered. Since the CLAMP instruction allows guarding values to be applied using the BSR of the appropriate ICs while selecting their Bypass Registers, it allows much faster testing than EXTEST. Data in the boundary scan cell remains unchanged until a new instruction is shifted in. The CLAMP instruction also asserts internal reset for the MSC711x system logic to force a predictable internal state while external boundary scan operations are performed.
00100	HIGHZ	Optional in the **IEEE** 1149.1 standard. It is a manufacturer's public instruction to prevent back-drive of the outputs during circuit-board testing. When HIGHZ is invoked, all output drivers, including the two-state drivers, are turned off (that is, high impedance). The HIGHZ instruction selects the Bypass Register. It also asserts internal reset for the MSC711x system logic to force a predictable internal state while external boundary scan operations are performed.
00101	—	Reserved
00110	—	Reserved
00111	—	Reserved

Table 16-4. JTAG Instruction Decoding for the Boundary Scan TAP Controller (Continued)

Bits 30-0	Instruction	Description
01000	—	Reserved
01001	—	Reserved
01010	—	Reserved
01011	—	Reserved
01100	—	Reserved
01101	—	Reserved
01110	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
01111	—	Reserved
...	...	Reserved
11100	---	Reserved
11101	—	Reserved
11110	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
11111	BYPASS	Selects the single-bit Bypass Register, which creates a shift-register path from TDI to the Bypass Register and finally to TDO, circumventing the 573-bit BSR register. This instruction enhances test efficiency when a component other than the MSC711x-based device is under test. When the current instruction selects the Bypass Register, the shift-register stage is set to a logic zero on the rising edge of TCK in the CAPTURE-DR controller state. Therefore, the first bit to be shifted out after the Bypass Register is selected is always a logic zero.

16.4.3.2 Debug TAP Controller Instruction Decoding

The debug TAP controller supports the debug port with the following public instructions used by the JTAG port:

- *ENABLE_EONCE*. Enables the EOnCE circuitry.
- *DEBUG_REQUEST*. Forces the SC1400 core into Debug mode.
- *CHOOSE_EONCE*. Selects an emulator port.

For a better understanding of how to access the emulator through the JTAG port using the debug TAP controller, see **Section 16.5**, *Accessing the Emulator Through the JTAG Port*, on page 16-21. **Table 16-4** describes the 5-bit instructions supported by the Debug IR.

Table 16-5. JTAG Instruction Decoding for the Debug TAP Controller

Bits 4-0	Instruction	Description
00000	—	Reserved
00001	—	Reserved
00010	IDCODE	Selects the ID Register. This instruction is a public instruction to allow the manufacturer, part number and version of a component to be determined through the TAP. The ID Register configuration is as follows: • Bits 31–28: Version Information • Bits 27–12: Customer Part Number • Bits 11–1: Manufacturer Identity One application of the ID Register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. As more components emerge that conform to the **IEEE** 1149.1 standard, it is desirable to allow for a system diagnostic controller unit to interrogate a board design blindly and determine the type of each component in each location. This information is also available for factory process monitoring and for failure mode analysis of assembled boards. Freescale's manufacturer identity number is 0b00000001110. The customer part number consists of two parts: design center number (bits 27–22) and a sequence number (bits 21–12). The design center number is 0b000110. When the IDCODE instruction is decoded, it selects the 32-bit ID Register. The Bypass Register loads a logic zero at the start of a scan cycle, whereas the ID Register loads a logic one into its least significant bit. Consequently, examination of the first bit of data shifted out of a component during a test data scan sequence, immediately following exit from test-logic-reset controller state, shows whether such a register is included in the design. As required by the **IEEE** 1149.1 standard, the operation of the test logic has no effect on the operation of the internal system logic when the IDCODE instruction is selected.
00011	—	Reserved
00100	—	Reserved
00101	—	Reserved
00110	ENABLE_EONCE	Not included in the **IEEE** 1149.1 standard. This public instruction allows you to perform system debug functions. When the ENABLE_EONCE instruction is decoded, TDI and TDO connect directly to the EOnCE registers. The EOnCE controller selects the specific EOnCE register connected between TDI and TDO, depending on the EOnCE instruction being executed. All communication with the EOnCE controller occurs through the SELECT-DR-SCAN path of the JTAG TAP Controller. Before the ENABLE_EONCE instruction is selected, the CHOOSE_EONCE instruction should be executed to define which EOnCE is to be activated.
00111	DEBUG_REQUEST	Not included in the **IEEE** 1149.1 standard. This public instruction allows you to generate a debug request signal to the MSC711x. When the DEBUG_REQUEST instruction is decoded, TDI and TDO connect to the EOnCE registers. In addition, ENABLE_EONCE is active and forced to request Debug mode from the MSC711x, in order to perform system debug functions. Before the DEBUG_REQUEST instruction is selected, the CHOOSE_EONCE instruction should be executed to define which EOnCE is to be selected for DEBUG_REQUEST.

Table 16-5. JTAG Instruction Decoding for the Debug TAP Controller (Continued)

Bits 4-0	Instruction	Description
01000	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
01001	CHOOSE_EONCE	Not included in the **IEEE** 1149.1 standard. This instruction enables selected SC1400 emulators. All instructions executed after this one target only the selected emulator set. Therefore, this instruction always executes, regardless of the selected emulator set.
01010	—	Reserved
01011	—	Reserved
01100	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
01101	LOAD_GPR	Not included in the **IEEE** 1149.1 standard: LOAD GPR **Note:** When programming the GPR, use only the bits permitted in **Table 16-8**.
01110	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
01111	—	Reserved
...	...	Reserved
11100	---	Reserved
11101	READ_PIREG	Not included in the **IEEE** 1149.1 standard: read Parallel Input Register (PIREG). **Note:** Use only the bits specified in **Table 16-9**. Other bits should be disregarded.
11110	PRIVATE	Manufacturer's private instruction. **Note:** Selecting this instruction many cause *unpredictable* operation of the device.
11111	BYPASS	Selects the single-bit Bypass Register, which creates a shift-register path from TDI to the Bypass Register and finally to TDO, circumventing the 573-bit BSR register. This instruction enhances test efficiency when a component other than the MSC711x-based device is under test. When the current instruction selects the Bypass Register, the shift-register stage is set to a logic zero on the rising edge of TCK in the CAPTURE-DR controller state. Therefore, the first bit to be shifted out after the Bypass Register is selected is always a logic zero.

16.4.4 JTAG Mode Restrictions

The control afforded by the output enable signals using the BSR and the EXTEST instruction requires a compatible circuit board test environment to avoid device-destructive configurations. You must avoid situations in which the MSC711x output drivers are enabled into actively driven networks. There are two constraints on the JTAG interface.

- The TCK input does not include an internal pull-up resistor and, to preclude mid-level inputs, should not be left unconnected.
- There are two methods to ensure that the JTAG test logic does not conflict with the system logic by forcing TAP into the test-logic-reset controller state. During power-up, $\overline{\text{TRST}}$ must be externally asserted to force the TAP controller into this state. After power-up, TMS must be sampled as a logic one for five consecutive TCK rising edges. If TMS either remains unconnected or is connected to V_{CC}, the TAP controller cannot leave the test-logic-reset state, regardless of the state of TCK.

The SC1400 core enters a low-power stop mode when it executes a STOP instruction. Since device clocks are selectively disabled in Stop mode, the JTAG interface allows polling of device status (sampled in the capture-IR state). The JTAG TAP brings the SC140 core out of Stop or Wait modes when DEBUG_REQUEST is decoded in the TAP IR. A small amount of additional power above the minimum possible is expended if the TAP is used in Stop mode. Save power in Low-Power Stop mode when JTAG is not in use, as follows:

- The TAP controller must be in the test-logic-reset state. Leaving the TAP controller test-logic-reset state negates the ability to achieve low power but does not otherwise affect device functionality.
- The TCK input is not blocked. To consume minimal power, the TCK input should externally connect to V_{CC} or ground.
- TMS and TDI include internal pull-up resistors. These two signals should remain either unconnected or connected to V_{CC} to achieve minimal power consumption.

16.5 Accessing the Emulator Through the JTAG Port

When the emulator is accessed through the JTAG port, the debug TAP controller instruction register must be loaded correctly so that the emulator can be controller through the JTAG pins. **Table 16-6** shows how JTAG must be configured to access the emulator registers.

Table 16-6. JTAG Instruction Decoding for the Boundary Scan TAP Controller

Task	JTAG Instruction Placed into Debug Instruction Register
Issue a debug request to an emulator to halt the SC1400 core.	DEBUG_REQUEST
Write an emulator command to the EOnCE Command Register (ECR).	DEBUG_REQUEST or ENABLE_EOnCE
Read or write any of the emulator registers through the JTAG port.	DEBUG_REQUEST or ENABLE_EOnCE

After the command is shifted in, the JTAG TAP state machine must enter the UPDATE-DR state. The data shifted via the TDI is sampled into the ECR. If, for example, the command written into the ECR is *Write EDCA0_CTRL*, the host must again enter the JTAG into SHIFT-DR and shift the required data, which is to be written into the EDCA0_CTRL, via TDI. If the command is *read some register*, the DR chain must be passed again and the contents of the register are shifted out through the TDO output signal. When JTAG shifts data to the emulator, the LSB of the data is shifted first. See **Figure 16-8**.

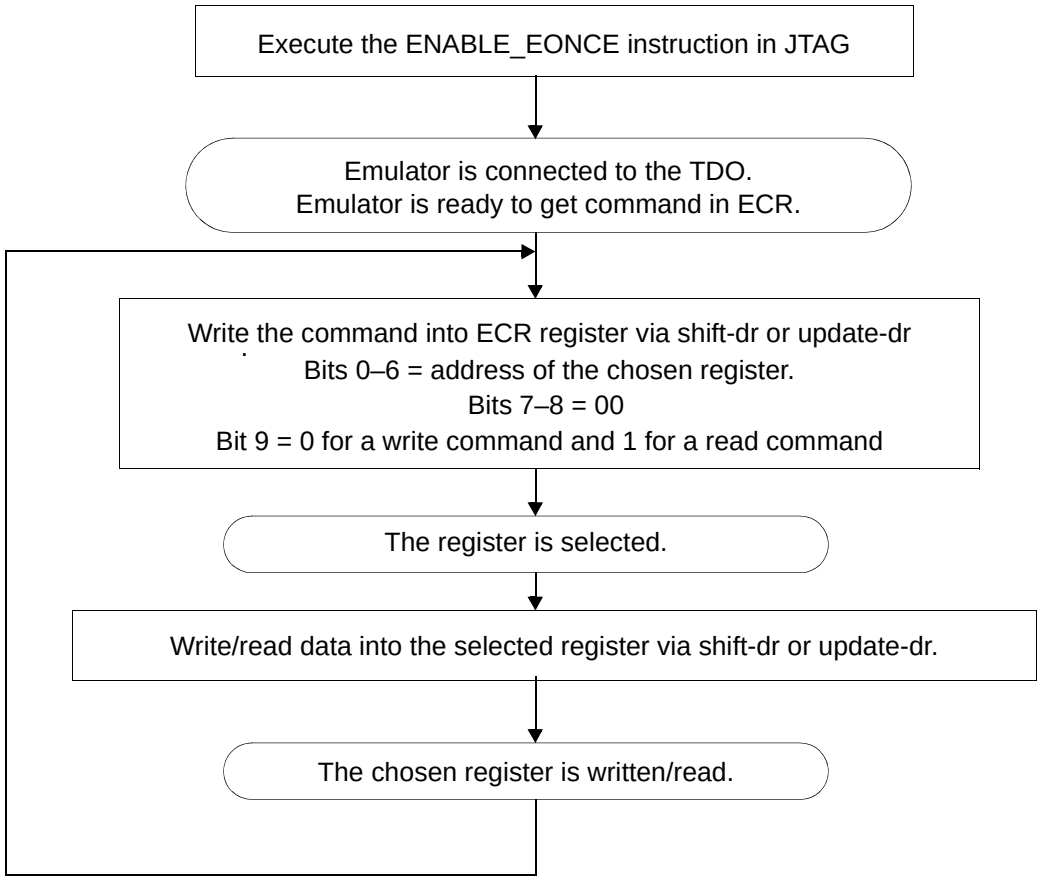


Figure 16-8. Reading and Writing Emulator Registers Via the JTAG TAP

16.6 OCE10 On-Chip Emulator and JTAG Programming Model

This section lists the emulator registers and describes the JTAG registers, which are available only through the JTAG signal pins.

16.6.1 Emulator Registers

For the emulator registers accessed as memory-mapped registers, the value of the base address for the register file, EONCE_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. **Table 16-7** lists the emulator registers, which are discussed in detail in the *OCE10 On-Chip Emulator Reference Manual*. The *Register Number* column of the table provides the value for accessing the register through the Emulator Command Register REGSEL field when the emulator registers are accessed through the JTAG port.

Table 16-7. Emulator Registers

Name	Address Offset	Register Number	Description
Control and Status			
Emulator Command Register (ECR)	—	—	Control register when the emulator is accessed through the JTAG port. **Note:** This register is accessible only through the JTAG port.
Emulator Status Register (ESR)	0x000	0x00	Status register providing status for the emulator and the associated core. **Note:** This register is read only.
Trace Buffer			
Trace Buffer Control (TB_CTRL)	0x140	0x50	Control register for the trace buffer.
Trace Buffer Read Pointer (TB_RD)	0x144	0x51	Accesses the read pointer, which points to the next value to read from the trace buffer.
Trace Buffer Write Pointer (TB_WR)	0x148	0x52	Accesses the write pointer, which points to the next value to write to the trace buffer.
Trace Buffer (TB_BUFF)	0x14C	0x53	Reading this register returns the next value to be read from the trace buffer. **Note:** This register is read only.
Emulator Event Counter			
Event Counter Control (ECNT_CTRL)	0x100	0x40	Selects events to be counted and enables the counter.
Event Counter Value (ECNT_VAL)	0x104	0x41	Value of the 31-bit event counter. When the counter is cascaded, provides the lower 31-bits of the counter.
Event Counter Extension Value (ECNT_EXT)	0x108	0x42	When the counter is cascaded, provides the upper 31-bits of the counter. Not used if counters are not cascaded.

Table 16-7. Emulator Registers (Continued)

Name	Address Offset	Register Number	Description
Hardware Breakpoints — Address Breakpointing			
Event Detection Channel n (EDCAn_CTRL)	0: 0x040 1: 0x044 2: 0x048 3: 0x04C 4: 0x050 5: 0x054	0: 0x10 1: 0x11 2: 0x12 3: 0x13 4: 0x14 5: 0x15	Control register for an address hardware breakpoint unit.
EDCAn Reference Value A (EDCAn_REFA)	0: 0x060 1: 0x064 2: 0x068 3: 0x06C 4: 0x070 5: 0x074	0: 0x18 1: 0x19 2: 0x1A 3: 0x1B 4: 0x1C 5: 0x1D	Holds a 32-bit reference value to be compared when breakpoints are detected.
EDCAn Reference Value B (EDCAn_REFB)	0: 0x080 1: 0x084 2: 0x088 3: 0x08C 4: 0x090 5: 0x094	0: 0x20 1: 0x21 2: 0x22 3: 0x23 4: 0x24 5: 0x25	Holds a 32-bit reference value to be compared when breakpoints are detected.
EDCAn Mask Value (EDCAn_MASK)	0: 0x0C0 1: 0x0C4 2: 0x0C8 3: 0x0CC 4: 0x0D0 5: 0x0D4	0: 0x30 1: 0x31 2: 0x32 3: 0x33 4: 0x34 5: 0x35	Mask applied to the address before a comparison.
Hardware Breakpoints — Data Breakpointing			
Event Detection Channel n (EDCD_CTRL)	0x0E0	0x38	Control register for an address hardware breakpoint unit.
EDCD Reference Value (EDCD_REF)	0x0E4	0x39	Holds a 32-bit reference value to be compared when breakpoints are detected.
EDCD Mask Value (EDCD_MASK)	0x0E8	0x3A	Mask applied to the address before a comparison.
Event Selector			
Event Selector Control (ESEL_CTRL)	0x120	0x48	Selects whether sources are ORed or ANDed together.
Event Selector Mask: Debug Mode (ESEL_DM)	0x124	0x49	Select sources to place the SC1400 core into Debug mode when asserted.
Event Selector Mask: Debug Exception (ESEL_DI)	0x128	0x4A	Select sources to generate a debug exception when asserted.
Event Selector Mask: Enable Trace Buffer (ESEL_ETB)	0x130	0x4C	Select sources to enable trace buffer operation when asserted.
Event Selector Mask Disable Trace Buffer (ESEL_DTB)	0x134	0x4D	Select sources to disable trace buffer operation when asserted.

Table 16-7. Emulator Registers (Continued)

Name	Address Offset	Register Number	Description
Emulator Data Transfer Unit (ETX / ERX)			
Emulator Monitor and Control Register (EMCR)	0x004		Status register providing status for the emulator and the associated core.
Emulator Receive Register (ERCV)	0x008	0x02	Lowest 32-bits of the Emulator Receive register. This register is read only.
Emulator Receive Register (ERCV)	0x00C		Highest 32-bits of the Emulator Receive register. This register is write only.
Emulator Transmit Register (ETRSMT)	0x010	0x04	Lowest 32-bits of the EOnCE Data Transmit register. This register is write only.
Emulator Transmit Register (ETRSMT)	0x014		Highest 32-bits of the EOnCE Data Transmit register. This register is read only.
Emulator EEn Signal Control			
EE Signals Control Register (EE_CTRL)	0x018		Defines EE[5–0] and EED pins as inputs or outputs. Also enables special capabilities for individual signals.
Program Counter Registers			
Exception VLES PC Register (PC_EXCP)	0x01C	0x07	Enables you to determine which VLES caused an internal exception. It is a read-on register that is accessed through the JTAG port or by core software.
Next VLES PC Register (PC_NEXT)	0x020	0x08	A 32-bit register that stores the address of the VLES to be executed next. It is a read-only register that is read through the JTAG port.
Last VLES PC Register (PC_LAST)	0x024	0x09	Contains the program counter (PC) of the last executed VLES. It is used in debug mode to define which PC triggered a PC breakpoint. It is a read-only register that is read through the JTAG port.
PC Breakpoint Detection Register (PC_DETECT)	0x028	0x0A	Captures the program counter (PC) of the first VLES that caused an entry into debug mode based on a data-memory event in EDCA or EDCD.

16.6.2 JTAG Registers

The JTAG boundary scan registers are accessible only through the JTAG pins of the device when the boundary scan TAP is selected. All of these registers are accessible via the DR path through the state machine (see **Figure 16-6**, *TAP Controller State Machine*, on page 16-14). These registers are listed as follows, along with the number of the page where each register is discussed:

- JTAG Identification Register (JTAGID), **page 16-26**.
- Boundary Scan Register (BSR), **page 16-27**.
- Bypass Register (JBYP), **page 16-29**.

- Identification Register (JID), **page 16-30**.

The JTAG debug registers are accessible only through the JTAG pins of the device when the debug TAP is selected. These registers are listed as follows, along with the number of the page where each register is discussed:

- JTAG Identification Register (JTAGID), **page 16-26**.
- Bypass Register (JBYP), **page 16-29**.
- Identification Register (JID), **page 16-30**.
- JTAG General-Purpose Register (JGPR), **page 16-30**.
- Parallel Input Register (PIREG), **page 16-31**.

Note: The JTAG instruction register, not listed here, is accessible only through the IR path through the state machine. This register is described on **page 16-15**.

JTAGID JTAG Identification (ID) Register JTAG Port Access Only

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Version				Design Center						Sequence Number					
TYPE	R															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Sequence Number				Manufacturer Identity											1
TYPE	R															1
RESET																

JTAGID is a read-only factory-programmed register that distinguishes the device on a board according to the **IEEE** 1149.1 standard.

- Version information corresponds to the revision number. The first version number for the MSC711x family of devices is 0b0000.
- The design center number is 0b100001.
- The sequence number is the device ID, which differs for each device in the MSC711x, as follows:
  - MSC7110: 0b0011
  - MSC7112: 0b0111
  - MSC7113: 0b0110
  - MSC7115: 0b0001
  - MSC7116: 0b0010
  - MSC7118: 0b1001

— MSC7119: 0b1010

- The Freescale manufacturer identity is 0b00000001110.
- The final 1 is required by the **IEEE** Std. 1149.1.
- The total JTAGID value for each member of the MSC711x family is as follows:
  - — MSC7110: 0x0840301D
  - — MSC7112: 0x0840701D
  - — MSC7113: 0x0840601D
  - — MSC7115: 0x0840101D
  - — MSC7116: 0x0840201D
  - — MSC7118: 0x
  - — MSC7119: 0x

BSR Boundary Scan Register

BSR contains bits for most device signals and control signals. All MSC711x bidirectional signals have two registers for boundary scan data and are controlled by an associated control bit in the BSR. The boundary scan bit definitions vary according to the specific device implementation of the MSC711x and are described in the BSDL file on the product web site. **Figure 16-12** through **Figure 16-12** show various BSR cell types.

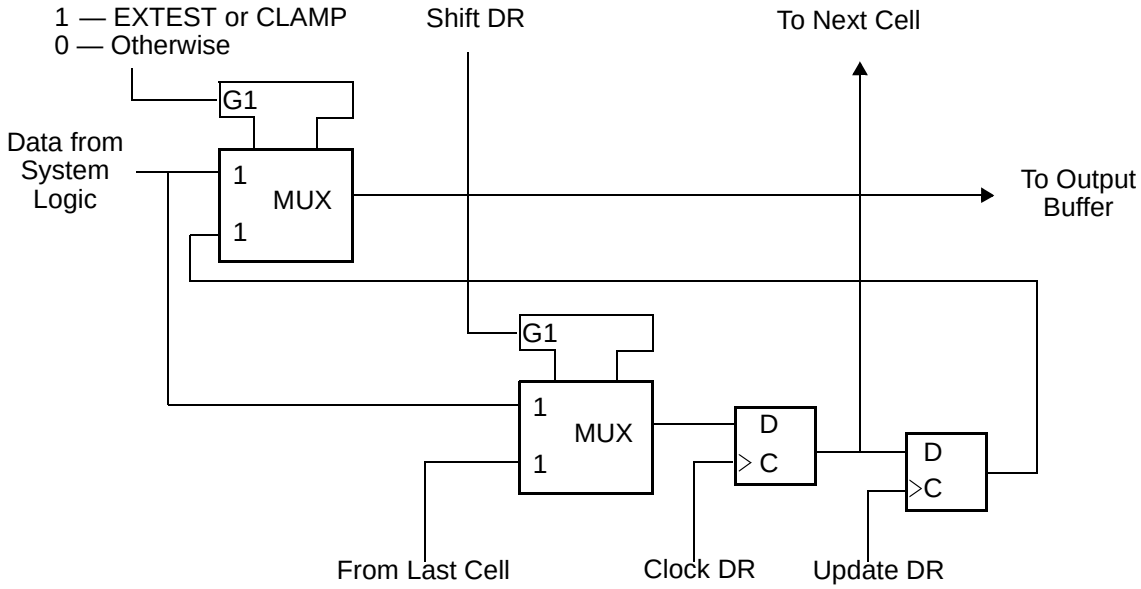


Figure 16-9. Output Signal Cell (O.PIN)

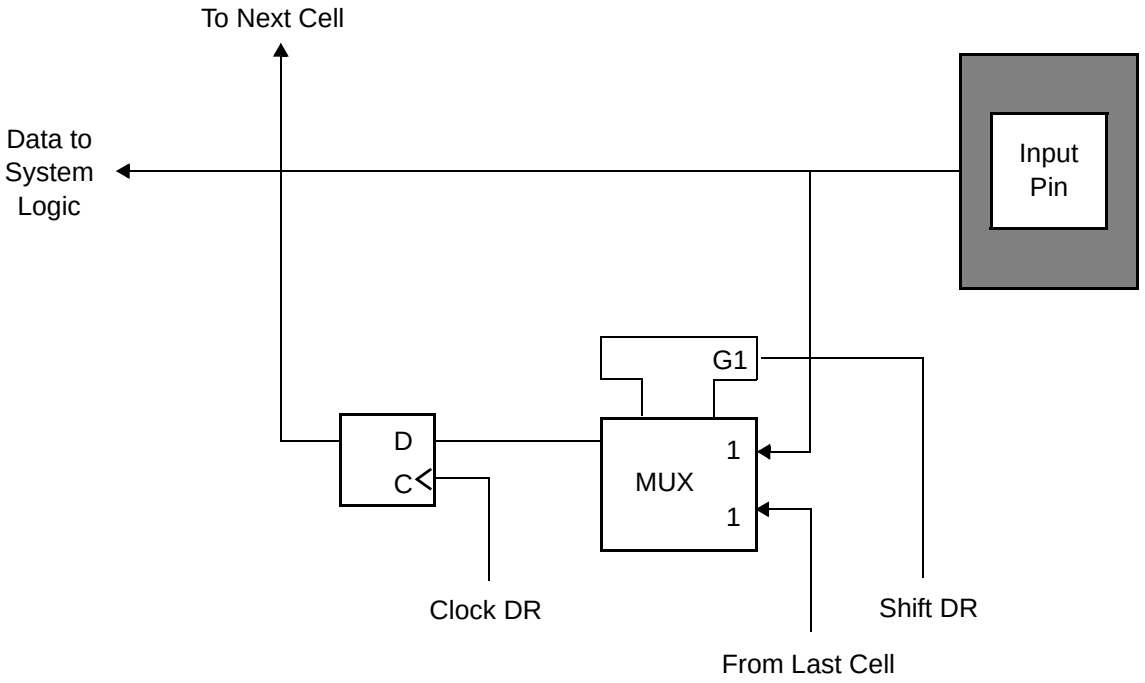


Figure 16-10. Observe-Only Input Signal Cell (I.OBS)

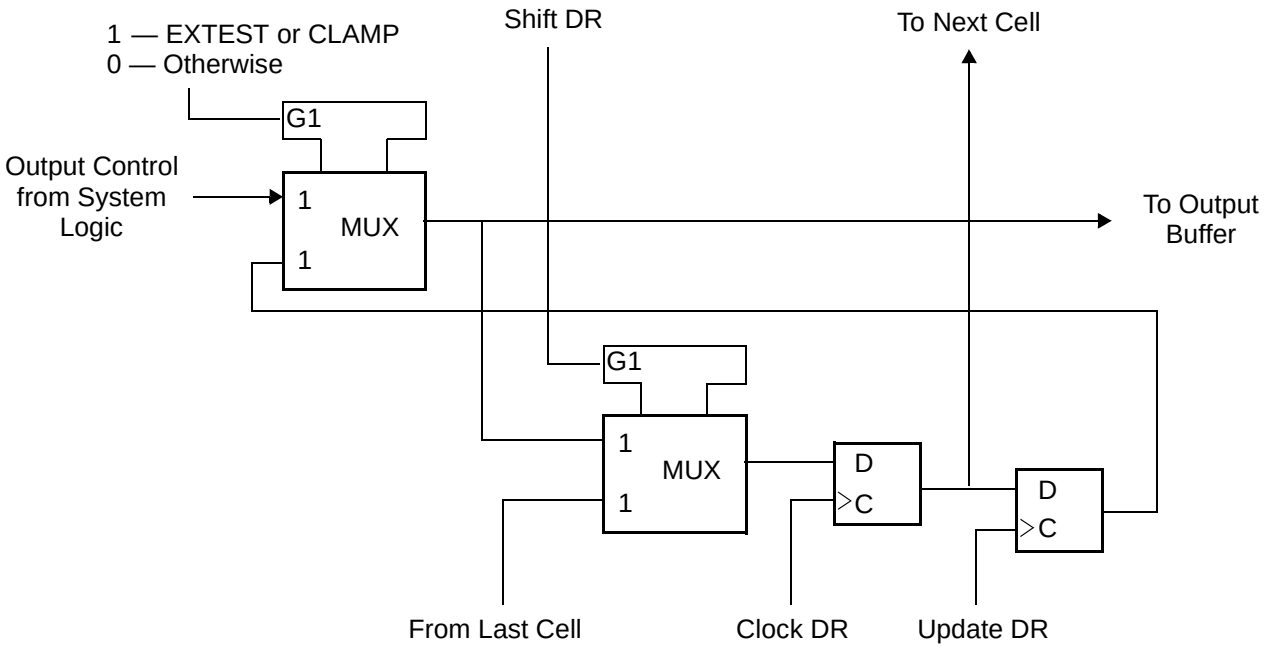


Figure 16-11. Output Control Cell (IO.CTL)

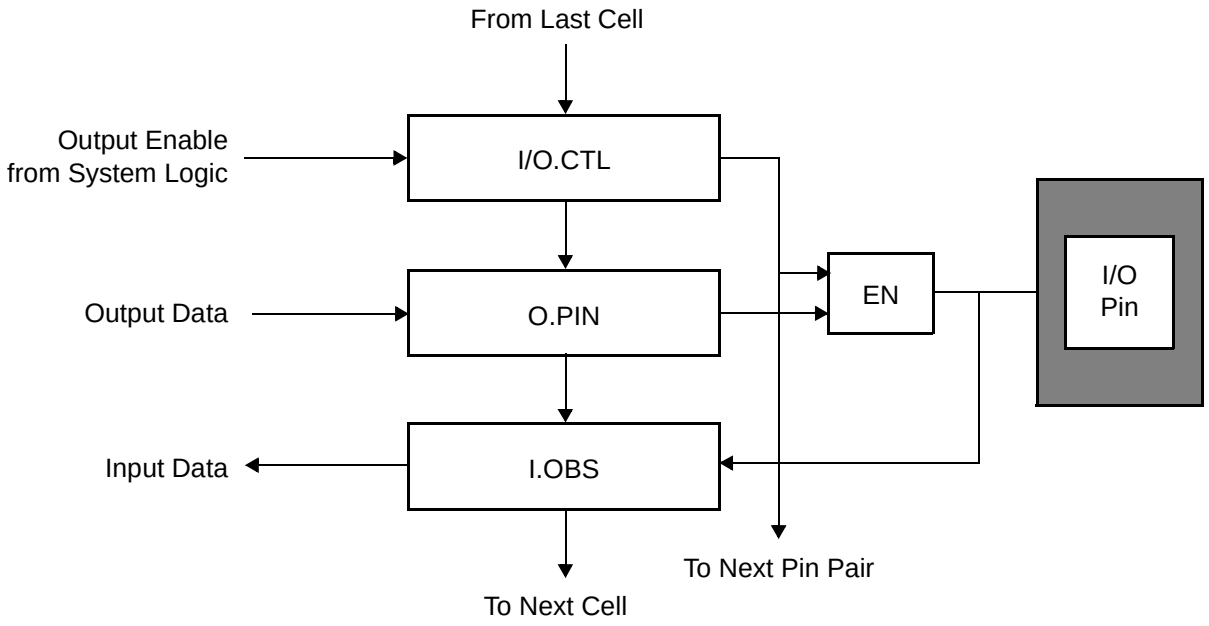


Figure 16-12. General Arrangement of Bidirectional Signal Cells

The control bit value controls the output function of the bidirectional signal. One or more bidirectional data cells can be serially connected to a control cell. Bidirectional signals include two scan cells for data (IO.Cell), as shown in **Figure 16-12**, and these bits are controlled by the cell shown in **Figure 16-11**. It is important to know the boundary scan bit order and signals associated with them. The BSDL file on the product web site describes the boundary scan serial string. The three MSC711x cell types described in this file are depicted in **Figure 16-9** through **Figure 16-11**.

JBYP Bypass Register

JBYP is a single-bit shift register (see **Figure 16-13**) that creates a shift-register path of one bit from TDI to TDO. When JBYP is selected, the shift-register stage is set to a logic zero on the rising edge of TCK in the CAPTURE-DR controller state.

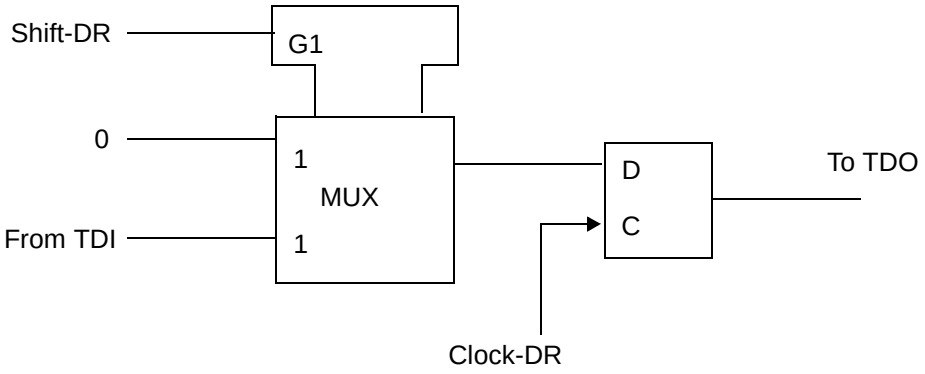


Figure 16-13. Bypass Register Configuration

JID Identification Register

JID sets the shift-register stage to a logic value equal to IDCODE on the rising edge of TCK in the CAPTURE-DR controller state. It is then shifted out in the SHIFT-DR controller state. See **Figure 16-14**.

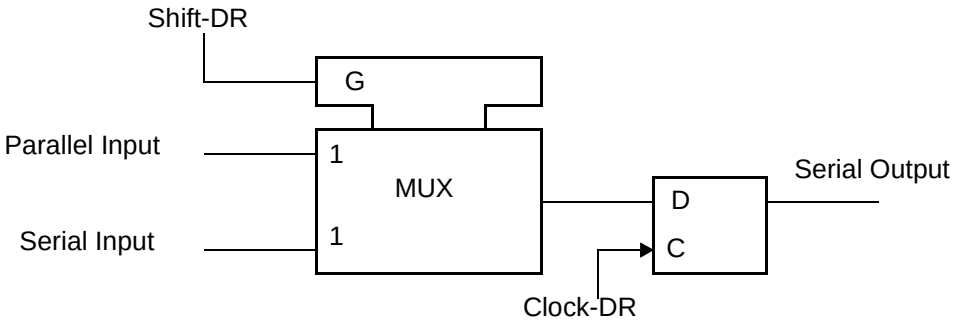


Figure 16-14. Identification Register Configuration (JID)

JGPR JTAG General-Purpose Register JTAG Port Access Only

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—														ISRSEL1	ISRSEL0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

During a shift in of any JTAG instruction, the bits shifted out reflect the status of the SC1400 core (see **Table 16-3**, *JDIR Bit Descriptions*, on page 16-16). Two bits in the JTAG GPR select the SC1400 core to which these bits belong. The GPR shifts in from TDI and out to TDO. The GPR[ISRSEL] bits select which emulator and SC1400 core status bits are reflected in the capture of any JTAG instruction. All other encodings are reserved and should be written with a value of 0. Writing a 1 to any of these bits may result in improper operation.

Table 16-8. GPR Bit Descriptions

Name	Reset	Description	Settings
— 31–10	0	Reserved. Write to zero for future compatibility.	
ISRSEL 1–0	0	**Instruction Status Core Select** Defines the SC1400 core to which the bits output during instruction register shifts belong.	00 Core 0. 01 Reserved. 10 Reserved. 11 Reserved.

PIREG JTAG Parallel Input Register JTAG Port Access Only

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—													COREST		CORACK
TYPE	R															
RESET	0															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE	R															
RESET	0															

PIREG enables you to observe the status of the SC1400 core by programming the JGPR and shifting in a JTAG instruction four times. To select PIREG, execute the READ PIREG command and shift out 32 bits from PIREG. The bits shifted out reflect the status of the SC1400 core.

Table 16-9. PIREG Bit Descriptions

Number	Reset	Description	Settings
— 31–19	0	Reserved.	
COREST 18–17	0	**Core 0 Core Status** Reflects the status of core 0.	00 Core is executing instructions. 01 Core is in Wait or Stop mode. 10 Core is waiting for bus. 11 Core is in debug mode.
CORACK 16	0	**Core 0 Update Acknowledge** Indicates whether the SC1400 core 0 emulator has executed the last instruction dispatched to it.	0 Emulator has executed the last instruction dispatched to it. 1 Emulator has not executed the last instruction dispatched to it.
— 15–0	0	Reserved. Write to zero for future compatibility.	

Programmable Address Detection 17

The programmable address detection units provide system protection, debug, and patch capability through user-programmable memory regions. This functionality goes beyond the “fixed” illegal access detection that causes interrupts based on attempted accesses to reserved memory regions. Programmable address detection enables you to allocate regions of memory as data and program memory. There are programmable units for the extended core and for peripherals:

The set for the extended core connects to the SC1400 core program address (PAB) and data address (XABA, XABB) buses and is programmed through the extended core QBus. The second set connects to the peripheral subsystem on the AMENT and AMDMA AHB master ports and is programmed through the IPBus. The high-level features for both sets of units are as follows:

- Range-based illegal access detection
- Range- or value-based event generation
- Value-based ROM patching
- Capture of status and detected addresses

17.1 Extended Core Programmable Address Detection

The fixed illegal-access detection in the extended core detects misaligned and fixed out-of-range addresses. The programmable units are far more powerful, allowing you to specify not only the modes of address detection but also the capture and resultant actions such as maskable and non-maskable interrupts and events. Since fixed out-of-range detection already protects against accesses to reserved areas of memory, the programmable units can define ranges based on how M1, M2 and DDR memories are partitioned into the application for program code, read-only data, and read-write data. You can define a data access to a program region as “illegal.” Detection occurs on the following address buses:

- SC1400 P bus (program accesses) with four independent program detection units
- SC1400 XA and XB buses (data accesses) with four independent data detection units, each containing an XA and XB detector
- The units are disabled out of reset

17.1.1 Detection Comparison Types

Based on the operating mode, the comparators in each detection unit performs either a range comparison or a value comparison:

- Range comparison:
  - A successful compare means that the address falls within the range specified by $ADUPR_x$ and $ADLWR_x$:
  - $ADLWR_x$ <= Address <= $ADUPR_x$.
- Value comparison:
  - A successful compare means that the address equates with either of the range values.
  - (Address == $ADLWR_x$) || (Address == $ADUPR_x$).

17.1.2 Detection Action Types

Depending on the operating mode, each detection unit can cause the following actions:

- *Interrupts*. Both maskable and non-maskable (NMI). Upon a successful detection, either CADSR[AOR] or CADSR[AVAL] is set. These status bits connect directly to non-maskable and maskable interrupts, respectively. Interrupts are cleared when you write a 1 to the appropriate status bit. Interrupts are available only in the non-event based modes
- *Events*. Events can be tied to a particular detection unit through masking. Event triggering is available only in the event modes. The event signal pins are EVNT[0–4].

17.1.3 Detection Modes

A mode in the detection units is a particular combination of comparison type, capture mechanism, and actions taken. The available modes are as follows:

- Disabled. No detection occurs.
- Address out of range.
- Value
- Event: address range.
- Event: value.

These modes are described in the context of comparison/capture/action in **Table 17-1**. The mode is selected through the CADCTL[0–1] registers.

Table 17-1. Detection Unit Modes/Configurations

Detection Type	Capture	Action	Comments
Disabled	—	—	—
Address out-of-range	PAB, XA, XB	Non-maskable interrupt	Detects invalid accesses in a range of addresses and generates a non-maskable interrupt. XA and XB are captured only if there are valid accesses on these buses at the moment of detection. CADCTL1[DATx] has no effect on capture rules.
Value	No	Maskable interrupt	Detects accesses to a particular program or data addresses and generates a maskable interrupt. Addresses are not captured in this mode
Event: address range	Bus with detection only	Event port trigger	Detects any accesses within an address range and triggers an event in the event port. Captures only the bus with a range match.
Event: value	No	Event port trigger	Detects accesses to a particular program or data addresses and triggers an event in the event port. Addresses are not captured in this mode

Notes: **1.** Capture registers CADCPTP, CADCPTXA, CADCPTXB and the status register CADSR are updated with the capture address/status on the core cycle following the detection. The detection occurs on the core address phase cycle.

2. When CADSR[AOR] is set, the capture registers CADCPTP, CADCPTXA, and CADCPTXB are not updated until CADSR[AOR] is cleared.

17.1.4 Extended Core Address Detection Architecture

Figure 17-1 shows the system architecture of extended core address detection. Notice the following:

- The control logic sends control signals to all the comparator units and multiplexes (P, XA, XB).
- Interrupts are not generated from the detection units. Instead, the status bits are set to generate either maskable or non-maskable interrupts. Interrupts are cleared when the appropriate status register bits are cleared.
- For event-based modes, eight event sources must be reduced to two outputs. This reduction is achieved via the masking register fields CADCTL0[EVME] and CADTCL0[EVMO].
- For maskable interrupts, there can be up to eight sources.

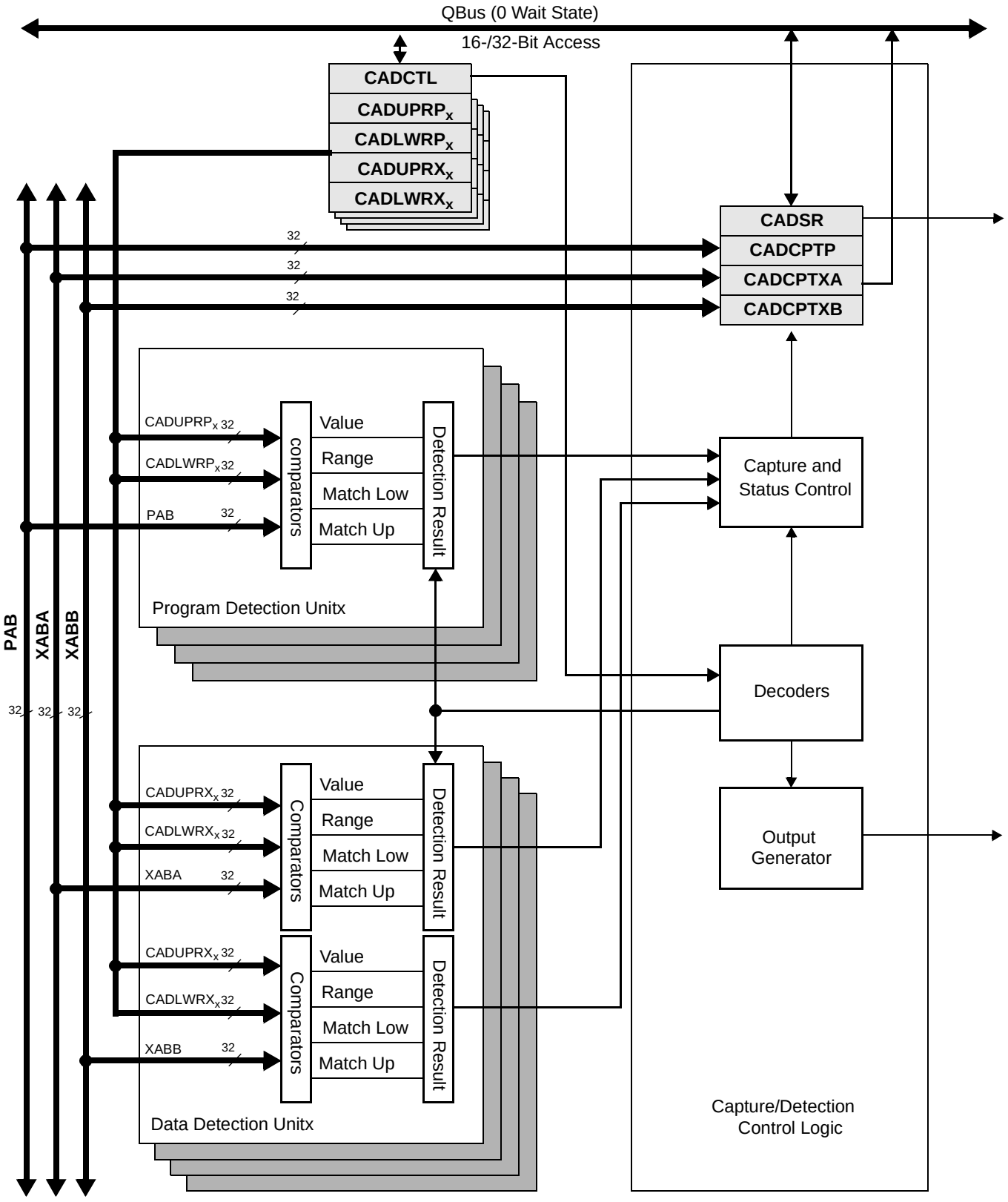


Figure 17-1. Extended Core Programmable Address Detection

17.2 Peripheral Programmable Address Detection

The peripheral address detection units perform much like those of the extended core, but they provide detection on the FEC (AMENT) and DMA (AMDMA) master AHB ports. Since fixed out-of-range detection already protects against access to reserved areas of memory, the programmable units can define ranges based on how M1, M2 and DDR memories are partitioned into an application for program code, read-only data, and read-write data as defined in the context of Ethernet and DMA accesses. You can define a DMA access to an Ethernet region as “illegal”. The detection occurs on the following address buses:

- DMA AMDMA AHB master bus with four independent program/data detection units
- Ethernet AMENT AHB master bus with 4 independent program/data detection units
- The units are disabled out of reset

17.2.1 Detection Comparison Types

Depending on the operating mode, the comparators in each detection unit performs a range comparison or a value comparison:

- Range comparison:
  - A successful compare means that the address falls within the range specified by $ADUPR_x$ and $ADLWR_x$:
  - $ADLWR_x$ <= Address <= $ADUPR_x$
- Value comparison:
  - A successful compare means that the address equates with either of the range values
  - (Address == $ADLWR_x$) || (Address == $ADUPR_x$)

17.2.2 Detection Action Types

Depending on the operating mode, each detection unit can cause the following actions:

- *Non-maskable interrupts (NMI)*. Upon a successful out-of-range detection, PADSR[AOR] is set. This status bit connects directly to the non-maskable interrupt request PADNMI. Interrupts are cleared when the appropriate status bit is cleared. Interrupts are available only in the non-event based modes.
- *Events*. Events can be tied to a particular detection unit through masking. Event triggering is available only in the event modes.

17.2.3 Detection Modes

A mode is a particular combination of comparison type, capture mechanism, and actions taken. The available modes are as follows:

- Disabled. No detection occurs.

- Address out of range
- Event: address range
- Event: value

Table 17-2 describes these modes in the context of comparison/capture/action. The mode is selected through the PADCTL0 and PADCTL1 registers.

Table 17-2. Peripheral Detection Unit Modes/Configurations

Detection Type	Capture	Action	Comments
Disabled	—	—	—
Address out-of-range	Bus with detection	Non-maskable interrupt	Detects invalid accesses in a range of addresses and generates a non-maskable interrupt.
Event: address range	Bus with detection	Event port trigger	Detects any accesses within an address range and triggers an event in the event port. Captures only the bus with a range match.
Event: value	No	Event port trigger	Detects accesses to particular program or data addresses and triggers an event in the event port. Addresses are not captured in this mode

Notes: **1.** Capture registers PADCPTD, PADCPTE, and the status register PADSR are updated with the capture address/status on the AHB cycle following the detection. The detection occurs on the AHB address phase cycle.

2. When PADSR[AOR] is set, the capture registers PADCPTD and PADCPTE do not update until PADSR[AOR]. is cleared.

17.2.4 Peripheral Address Detection Architecture

Figure 17-2 shows the system architecture for peripheral address detection. Notice the following:

- The control logic sends control signals to all the comparator units and multiplexes (AMENT, AMDMA).
- Interrupts are not generated from the detection units. Instead, the PADSR[AOR] bit is set to generate an $\overline{\text{NMI}}$. Interrupts are cleared when the appropriate status register bits are cleared.
- For event-based modes, eight event sources must be reduced to two outputs. This reduction is achieved via the masking register PADCTL0[EVME] and PADTCL0[EVMO] bits.

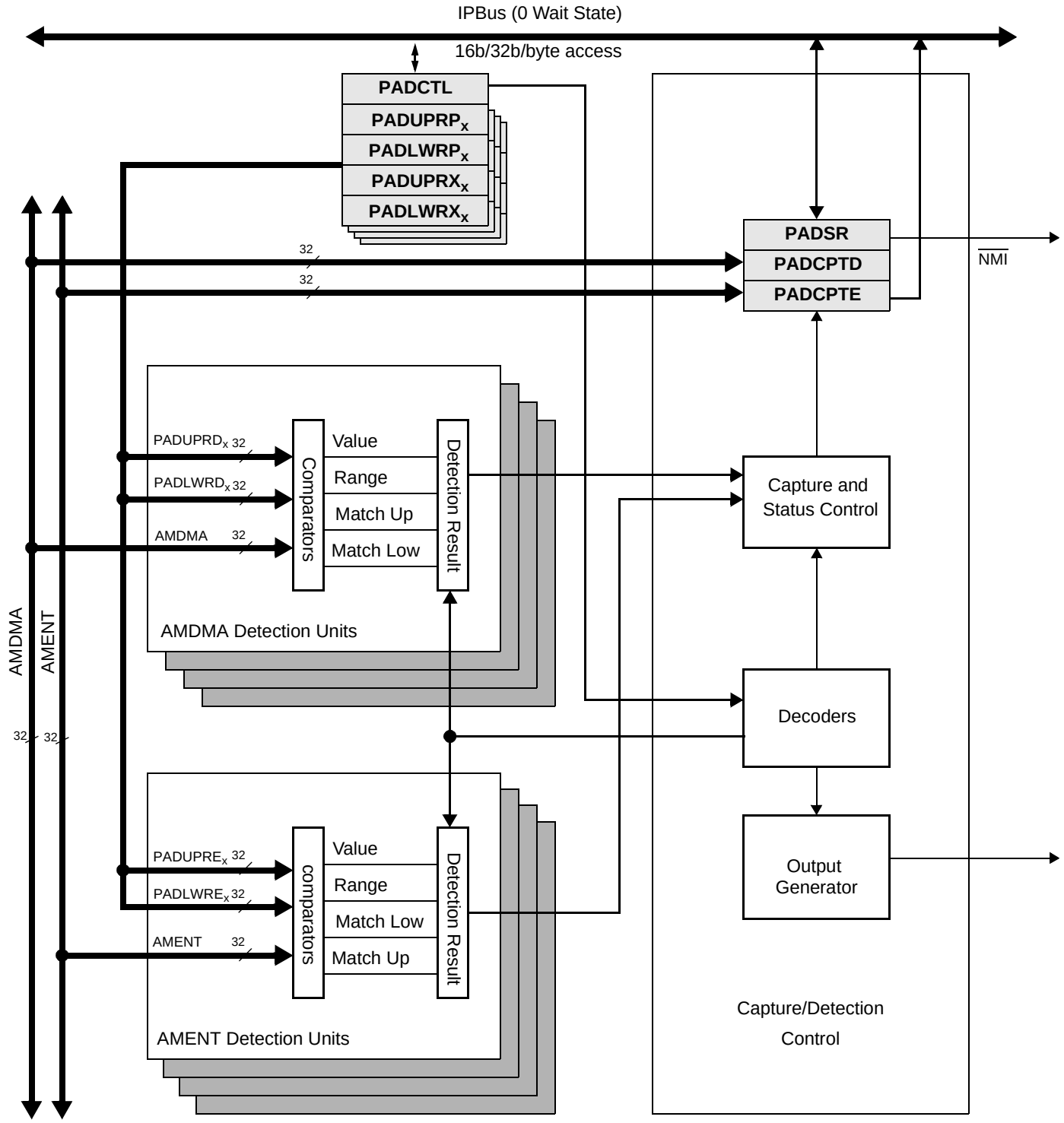


Figure 17-2. Peripheral Programmable Address Detection

17.3 Address Detection Unit Programming Model

There are two sets of registers for the address detection unit, one for the extended core and one for the peripherals. The values of the base addresses for the extended core address detection register file, CAD_BASE, and the peripheral address detection register file, PAD_BASE are listed in **Section 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4.

Note: You can access information in the address detection registers using 16-bit or 32-bit accesses from the SC1400 core.

17.3.1 Extended Core Address Detection Registers

The extended core address detection registers are as follows:

- Extended Core Address Detection Control Register 0 (CADCTL0), **page 17-9**.
- Extended Core Address Detection Control Register 1 (CADCTL1), **page 17-10**.
- Extended Core Address Detection Status Register (CADSR), **page 17-11**.
- Extended Core Address Detection PAB Lower Bound Register (CADLWRPx), **page 17-14**.
- Extended Core Address Detection PAB Upper Bound Register (CADUPRPx), **page 17-15**.
- Extended Core Address Detection XAB Lower Bound Register (CADLWRXx), **page 17-15**.
- Extended Core Address Detection XAB Upper Bound Register (CADUPRXx), **page 17-16**.
- Extended Core Address Detection Capture Program Address Register (CADCPTP), **page 17-17**.
- Extended Core Address Detection Capture XA Address Register (CADCPTXA), **page 17-17**.
- Extended Core Address Detection Capture XB Address Register (CADCPTXB), **page 17-18**.

Note: When you are using the program detection units from the upper bound registers (CADUx), be aware that the SC1400 core performs a read of three fetch sets ahead from M1 memory. Otherwise, you may experince unexpected NMIs.

CADCTL0 Extended Core Address Detection Control Register 0 CAD_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—								EVME				EVMO			
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—				MP3			MP2			MP1			MP0		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADCTL0 programs general settings for both units (event port multiplexing) and the mode settings for the program unit.

Table 17-3. CADCTL0 Bit Descriptions

Name	Reset	Description	Settings
31–24	0x00	Reserved. Write to zero for future compatibility.	
EVME 23–20	0x0	**Event Mask for Even Detection Units** Event mask for generation of CADEV0: • Bit 23 Data unit 2 enable/disable. • Bit 22 Data unit 0 enable/disable. • Bit 21 Program unit 2 enable/disable. • Bit 20 Program unit 0 enable/disable This field has no effect in non-event modes	0 Detection unit does not trigger 1 Detection unit triggers.
EVMO 19–16	0x0	**Event Mask for Odd Detection Units** Event mask for generation of CADEV1: • Bit 19 Data unit 3 enable/disable. • Bit 18 Data unit 1 enable/disable. • Bit 17 Program unit 3 enable/disable. • Bit 16 Program unit 1 enable/disable. This field has no effect in non-event modes.	0 Detection unit does not trigger. 1 Detection unit triggers.
15-12	0x00	Reserved. Write to zero for future compatibility.	
MP[3–0] 11–0	000	**Mode Select — Program Unit x** Specifies the mode/configuration for detection.	000 Detection unit disabled. 001 Address out of range. 010 Value. 011 Event address range. 100 Event value. 101– 111 Reserved

CADCTL1 Extended Core Address Detection Control Register 1 CAD_BASE + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—								DAT3		DAT2		DAT1		DAT0	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—				MD3			MD2			MD1			MD0		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADCTL1 programs mode settings for the data unit.

Table 17-4. CADCTL1 Bit Descriptions

Name	Reset	Description	Settings
31-24	0x00	Reserved. Write to zero for future compatibility.	
DAT[3–0] 23–16	00	**Data Detection Unit x Access Type** Controls the conditions for detection of data accesses. This setting does not affect XA or XB capture during a program address out-of-range detect.	00 Detect on access. 01 Reserved. 10 Detect only on reads. 11 Detect only on writes.
15-12	0x00	Reserved. Write to zero for future compatibility.	
MD[3–0] 11–0	000	**Mode Select, Data Unit x** Specifies the mode/configuration for detection per **Table 17-1**.	000 Detection unit disabled. 001 Address out of range. 010 Value. 011 Event: address range. 100 Event: value. 101–111 Reserved.

CADSR Extended Core Address Detection Status Register CAD_BASE + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	AOR	AVAL1	AVAL0	EVAR	EVVAL	—	CPTXA	CPTXB	—							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DDU3	DD3	DDU2	DD2	DDU1	DD1	DDU0	DD0	PDU3	PD3	PDU2	PD2	PDU1	PD1	PDU0	PD0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADSR reports status information for the extended core address detection units. Both the user and the internal control unit of the ADU can write to the CADSR. When both the user and the control unit write to ADSR, the user write has higher priority.

- Each bit is set when its associated detection occurs; these bits cannot be set by writing to the register.
- The control unit cannot clear the CADSR bits. Only the user program can clear them.
- Each bit can only be cleared by writing a 1 to the bit, which indicates an active mask for clearing. For example, if you want to clear the AOR and DDU3 bits, you would write a value of 0x8000_8000 to CADSR.
- The update behavior of the CPTXA and CPTXB bits follows the capture behavior of the CADCPTXA and CADCPTXB capture registers. For an address-out-of-range detection, the values of the CPTXA and CPTXB bits remains (sticky) regardless of further detects until the user program clears the AOR bit.
- The AVAL, EVAR, EVVAL bits and bits DDU3–PD0 continue to update even after the AOR bit is set. These fields accumulate each cycle (if detections occur), and only the user program clears them.

Table 17-5. CADSR Bit Descriptions

Name	Reset	Description	Settings
AOR 31	0	**Address Out of Range Detection** Captures the status of detection for the program and data detection units.	0 No detection. 1 Detection.
AVAL1 30	0	**Address Value Detection 1** Captures the status of detection for the even-numbered program and data detection units where the UPR_x matched the value.	0 No detection. 1 Detection.

Table 17-5. CADSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
AVAL0 29	0	**Address Value Detection 0** Captures the status of detection for the odd-numbered program and data detection units, where the UPR_x matches the value and all units where the value matches LWR_x.	0 No detection. 1 Detection.
EVAR 28	0	**Event Address Range Detection** Captures the status of detection for all the program and data detection units.	0 No detection. 1 Detection.
EVVAL 27	0	**Event Value Detection** Captures the status of detection for all the program and data detection units.	0 No detection. 1 Detection.
— 26	0	Reserved. Write to zero for future compatibility.	
CPTXA 25	0	**Capture on XA** A status bit that indicates whether a valid XA access occurred at the time of capture.	0 No detection. 1 Detection.
CPTXB 24	0	**Capture on XB** A status bit that indicates whether a valid XB access occurred at the time of capture.	0 No detection. 1 Detection.
— 23–16	0	Reserved. Write to zero for future compatibility.	
DDU3 15	0	**Data Detect Upper, Unit 3** Captures the status of the detection for the corresponding data address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD3 14	0	**Data Detect, Unit 3** Captures status of detection for the corresponding data address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.
DDU2 13	0	**Data Detect Upper, Unit 2** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD2 12	0	**Data Detect, Unit 2** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.
DDU1 **11**	0	**Data Detect Upper, Unit 1** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.

Table 17-5. CADSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
DD1 10	0	**Data Detect, Unit 1** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
DDU0 9	0	**Data Detect Upper, Unit 0** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD0 8	0	**Data Detect, Unit 0** Captures the status of detection for the corresponding data address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
PDU3 7	0	**Program Detect Upper, Unit 3** Captures status of detection for the corresponding program address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
PD3 6	0	**Program Detect, Unit 3** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.
PDU2 5	0	**Program Detect Upper, Unit 2** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
PD2 4	0	**Program Detect, Unit 2** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.
PDU1 3	0	**Program Detect Upper, Unit 1** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.

Table 17-5. CADSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
PD1 2	0	**Program Detect, Unit 1** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.
PDU0 1	0	**Program Detect Upper, Unit 0** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for value detection, it captures whether a detection occurred in the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
PD0 0	0	**Program Detect, Unit 0** Captures the status of detection for the corresponding program address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred in the LWRPx register.	0 No detection. 1 Detection.

CADLWRPx Extended Core Address Detection PAB Lower Bound Register

CADLWRP0	CAD_BASE + 0x14
CADLWRP1	CAD_BASE + 0x18
CADLWRP2	CAD_BASE + 0x1C
CADLWRP3	CAD_BASE + 0x20

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LWRP															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	LWRP															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADLWRPx specifies the lowest address program detection region. It can also specify one of two value addresses.

Table 17-6. CADLWRPx Bit Descriptions

Name	Reset	Description
LWRP 31–0	0	**Lower Bound, PAB** In normal operation, specifies the lowest address used in PAB address detection for one of the P-address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

CADUPRPx Extended Core Address Detection PAB Upper Bound Register

CADUPRP0 CAD_BASE + 0x2C
CADUPRP1 CAD_BASE + 0x30
CADUPRP2 CAD_BASE + 0x34
CADUPRP3 CAD_BASE + 0x38

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	UPRP															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	UPRP															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADUPRPx specifies the highest address in a program detection region. It can also specify one of two value addresses.

Table 17-7. CADUPRPx Bit Descriptions

Name	Reset	Description
UPRP 31–0	0	**Upper Bound, PAB** In normal operation, specifies the highest address used in PAB address detection for one of the P-address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

CADLWRX Extended Core Address Detection XAB Lower Bound Register

CADLWRX0 CAD_BASE + 0x44
CADLWRX1 CAD_BASE + 0x48
CADLWRX2 CAD_BASE + 0x4C
CADLWRX3 CAD_BASE + 0x50

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LWRX															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	LWRX															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADLWRXx specifies the lowest address in a data detection region. It can also specify one of two value addresses.

Table 17-8. CADLWRXx Bit Descriptions

Name	Reset	Description
LWRX 31–0	0	**Lower Bound, XAB** In normal operation, specifies the lowest address used in XAB address detection for one of the data address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

CADUPRX Extended Core Address Detection XAB Upper Bound Register

CADUPRX0 CAD_BASE + 0x5C
CADUPRX1 CAD_BASE + 0x60
CADUPRX2 CAD_BASE + 0x64
CADUPRX3 CAD_BASE + 0x68

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	UPRX															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	UPRX															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADUPRXx specifies the highest address in a data detection region. It can also specify one of two value addresses.

Table 17-9. CADUPRXx Bit Descriptions

Name	Reset	Description
UPRX 31–0	0	**Upper Bound, XAB** In normal operation, specifies the highest address used in XAB address detection for one of the data address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

CADCPTP Extended Core Address Detection PAB Capture CAD_BASE + 0x74

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CPTP															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CPTP															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADCPTP captures the PAB address when a range detection occurs.

Table 17-10. CADCPTP Bit Descriptions

Name	Reset	Description
CPTP 31–0	0	**Capture Address, PAB** The value of the PAB address is captured in this register when an address detection occurs. This register captures values only for a range detection between a lower and upper value. It does not capture the address when one of the units is programmed in value mode, so only a value mode detection occurs.

CADCPTXA Extended Core Address Detection XABA Capture CAD_BASE + 0x78

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CPTXA															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CPTXA															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADCPTXA captures the XABA address when a range detection occurs.

Table 17-11. CADCPTXA Bit Descriptions

Name	Reset	Description
CPTXA 31–0	0	**Capture Address, XABA** The value of the XABA address is captured in this register when an address detection occurs. This register captures values only for a range detection between a lower and upper value. It does not capture the address when one of the units is programmed in value mode, so only a value mode detection occurs.

CADCPTXB Extended Core Address Detection XABB Capture CAD_BASE + 0x7C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CPTXB															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CPTXB															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CADCPTXB captures the XABB address when a range detection occurs.

Table 17-12. CADCPTXB Bit Descriptions

Name	Reset	Description
CPTXB 31–0	0	**Capture Address, XABB** The value of the XABB address is captured in this register when an address detection occurs. This register captures values only when a range detection occurs between a lower and upper value. It does not capture the address when one of the units is programmed in value mode, so only a value mode detection occurs.

17.3.2 Peripheral Address Detection Registers

The extended core address detection registers are as follows:

- Peripheral Address Detection Control Register 0 (PADCTL0), **page 17-19**.
- Peripheral Address Detection Control Register 1 (PADCTL1), **page 17-20**.
- Peripheral Address Detection Status Register (PADSR), **page 17-21**.
- Peripheral Address Detection AMDMA Lower Bound Register (PADLWRDx), **page 17-24**.
- Peripheral Address Detection AMDMA Upper Bound Register (PADUPRDx), **page 17-25**.

- Peripheral Address Detection AMENT Lower Bound Register (PADLWREx), **page 17-25**.
- Peripheral Address Detection AMENT Upper Bound Register (PADUPREx), **page 17-26**.
- Peripheral Address Detection Capture AMDMA Address Register (PADCPTD), **page 17-27**.
- Peripheral Address Detection Capture AMENT Address Register (PADCPTE), **page 17-27**.

PADCTL0 Peripheral Address Detection Control Register 0 PAD_BASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DAT3		DAT2		DAT1		DAT0		EVME				EVMO			
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—				MD3		MD2		MD1		MD0					
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADCTL0 programs general settings for both the extended core and peripheral address detection units (event port multiplexing) and for the AMDMA unit.

Table 17-13. PADCTL0 Bit Descriptions

Name	Reset	Description	Settings
DAT3–DAT0 31–24	00	**AMDMA Detection Unit x Access Type** Controls the conditions in which detection takes place for AMDMA accesses.	00 Detect on access. 01 Reserved. 10 Detect only on reads. 11 Detect only on writes.
EVME 23–20	0000	**Event Mask for Even Detection Units** Event mask for generation of even-numbered address detection units. • Bit 23 AMENT unit 2 enable/disable. • Bit 22 AMENT unit 0 enable/disable. • Bit 21 AMDMA unit 2 enable/disable. • Bit 20 AMDMA unit 0 enable/disable. This field has no effect in non-event modes.	0 Detection unit does not trigger. 1 Detection unit triggers.

Table 17-13. PADCTL0 Bit Descriptions (Continued)

Name	Reset	Description	Settings
EVMO 19–16	0000	**Event Mask for Odd Detection Units** Event mask for generation of odd-numbered detection units. • Bit 19 AMENT unit 3 enable/disable. • Bit 18 AMENT unit 1 enable/disable. • Bit 17 AMDMA unit 3 enable/disable. • Bit 16 AMDMA unit 1 enable/disable. This field has no effect in non-event modes.	0 Detection unit does not trigger. 1 Detection unit triggers.
15–12	0x0	Reserved. Write to zero for future compatibility.	
MD3–MD0 11–0	000	**Mode Select, AMDMA Unit x** Specifies the mode/configuration for detection.	000 Detection unit disabled. 001 Address out of range. 010 Reserved. 011 Event: adress range. 100 Event: value. 101–111 Reserved.

PADCTL1 Peripheral Address Detection Control Register 1 PAD_BASE + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	EAT3		EAT2		EAT1		EAT0		—							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—				ME3		ME		ME1		ME0					
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADCTL1 programs the AMENT unit.

Table 17-14. PADCTL1 Bit Descriptions

Name	Reset	Description	Settings
EAT3–EAT0 31–24	00	**AMENT Detection Unit x Access Type** Controls the condition under which detection takes place for AMENT accesses.	00 Detect on access. 01 Reserved. 10 Detect only on reads. 11 Detect only on writes.

Table 17-14. PADCTL1 Bit Descriptions (Continued)

Name	Reset	Description	Settings
23-16	0x00	Reserved. Write to zero for future compatibility.	
ME3–ME0 11–0	000	**Mode Select, AMENT Unit x** Specifies the mode/configuration for detection.	000 Detection unit disabled. 001 Address out of range. 010 Reserved. 011 Event: address range. 100 Event: value. 101– 111 Reserved.

PADSR Peripheral Address Detection Status Register PAD_BASE + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	AOR	—	EVAR	EVVAL	—											
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EDU3	ED3	EDU2	ED2	EDU1	ED1	EDU0	ED0	DDU3	DD3	DDU2	DD2	DDU1	DD1	DDU0	DD0
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADSR reports status information for the peripheral address detection units.

- Each bit is set when its associated detection occurs; these bits cannot be set by writing to the register.
- The control unit cannot clear the PADSR bits. Only the user program can clear them.
- Each bit can only be cleared by writing a 1 to the bit, which indicates an active mask for clearing. For example, if you want to clear AOR and EDU3 bits, you would write a value of 0x8000_8000 to PADSR.
- Both the user and the internal control unit can write to the PADSR. When both the user and the control unit write to PADSR, the user write always has higher priority.
- The EVAR and EVVAL bits and bits DDU3–PD0 continue to update even after the AOR bit is set. These fields accumulate each cycle (if detections occur) and only the user program clears them.

Table 17-15. PADSR Bit Descriptions

Name	Reset	Description	Settings
AOR 31	0	**Address Out of Range Detection** Captures the status of detection for all AMDMA and AMENT detection units.	0 No detection. 1 Detection.
— 30	0	Reserved. Write to zero for future compatibility.	
EVAR 29	0	**Event Address Range Detection** Captures the status of detection for all AMDMA and AMENT detection units.	0 No detection. 1 Detection.
EVVAL 28	0	**Event Value Detection** Captures the status of detection for all AMDMA and AMENT detection units.	0 No detection. 1 Detection.
— 27-16	0	Reserved. Write to zero for future compatibility.	
EDU3 15	0	**Data Detect Upper, Unit 3** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
ED3 14	0	**Data Detect, Unit 3** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
EDU2 13	0	**Data Detect Upper, Unit 2** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
ED2 12	0	**Data Detect, Unit 2** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
EDU1 **11**	0	**Data Detect Upper, Unit 1** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
ED1 10	0	**Data Detect, Unit 1** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.

Table 17-15. PADSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
EDU0 9	0	**Data Detect Upper, Unit 0** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRXx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
ED0 8	0	**Data Detect, Unit 0** Captures the status of detection for the corresponding AMENT address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
DDU3 7	0	**Program Detect Upper, Unit 3** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD3 6	0	**Program Detect, Unit 3** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
DDU2 5	0	**Program Detect Upper, Unit 2** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD2 4	0	**Program Detect, Unit 2** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.
DDU1 3	0	**Program Detect Upper, Unit 1** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD1 2	0	**Program Detect, Unit 1** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.

Table 17-15. PADSR Bit Descriptions (Continued)

Name	Reset	Description	Settings
DDU0 1	0	**Program Detect Upper, Unit 0** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for value detection, it captures whether a detection occurred on the UPRPx register. No action is taken if the unit is programmed for range detection.	0 No detection. 1 Detection.
DD0 0	0	**Program Detect, Unit 0** Captures the status of detection for the corresponding AMDMA address detection unit. If the unit is configured for range detection, it captures whether a detection occurred in the specified range. If the unit is configured for value detection, it captures whether a detection occurred on the LWRPx register.	0 No detection. 1 Detection.

PADLWRD Peripheral Address Detection AMDMA Lower Bound Register

PADLWRD0 PAD_BASE + 0x14
PADLWRD1 PAD_BASE + 0x18
PADLWRD2 PAD_BASE + 0x1C
PADLWRD3 PAD_BASE + 0x20

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LWRD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LWRD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADLWRD specifies the lowest address program detection region. It can also specify one of two value addresses.

Table 17-16. PADLWRDx Bit Descriptions

Name	Reset	Description
LWRD 31–0	0	**Lower Bound, AMDMA** In normal operation, specifies the lowest address used in AMDMA address detection for one of the AMDMA address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

PADUPRD Peripheral Address Detection AMDMA Upper Bound Register

PADUPRD0 PAD_BASE + 0x2C
PADUPRD1 PAD_BASE + 0x30
PADUPRD2 PAD_BASE + 0x34
PADUPRD3 PAD_BASE + 0x38

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	UPRD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	UPRD															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADUPRDx specifies the highest address in a AMDMA detection region. It can also specify one of two value addresses.

Table 17-17. PADUPRDx Bit Descriptions

Name	Reset	Description
UPRD 31–0	0	**Upper Bound — AMDMA** In normal operation, specifies the highest address used in AMDMA address detection for one of the AMDMA address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

PADLWRE Peripheral Address Detection AMENT Lower-Bound Register

PADLWRE0 PAD_BASE + 0x44
PADLWRE1 PAD_BASE + 0x48
PADLWRE2 PAD_BASE + 0x4C
PADLWRE3 PAD_BASE + 0x50

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LWRE															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LWRE															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADLWREx specifies the lowest address in the AMENT detection region. It can also specify one of two value addresses.

Table 17-18. PADLWREx Bit Descriptions

Name	Reset	Description
LWRE 31–0	0	**Lower Bound — AMENT** In normal operation, specifies the lowest address used in AMENT address detection for one of the AMENT address detection regions. When the detection unit is used in value mode, this register provides one of the two value addresses available in each detection region.

PADUPRE Peripheral Address Detection AMENT Upper Bound Register

PADUPRE0 PAD_BASE + 0x5C
PADUPRE1 PAD_BASE + 0x60
PADUPRE2 PAD_BASE + 0x64
PADUPRE3 PAD_BASE + 0x68

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	UPRE															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	UPRE															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADUPREx specifies the highest address in the AMENT detection region. It can also specify one of two value addresses.

Table 17-19. PADUPREx Bit Descriptions

Name	Reset	Description
UPRE 31–0	0	**Upper Bound — AMENT** In normal operation, specifies highest address used in AMENT address detection for one of the AMENT address detection regions. When the detection unit is used in ‘value mode’, this register provides one of the two value addresses available in each detection region.

PADCPTD AMDMA Capture Register PAD_BASE + 0x74

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CPTD															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CPTD															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADCPTD captures the AMDMA address when a range detection occurs.

Table 17-20. PADCPTD Bit Descriptions

Name	Reset	Description
CPTD 31–0	0	**Capture Address, AMDMA** The value of the AMDMA address is captured in this register when an address detection occurs. This register captures values only for a range detection between a lower and upper value. It does not capture the address when one of the units is programmed in value mode, so only a value mode detection occurs.

PADCPTE AMENT Capture Register PAD_BASE + 0x78

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CPTE															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	CPTE															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PADCPTE captures the XABA address when a range detection occurs.

Table 17-21. PADCPTE Bit Descriptions

Name	Reset	Description
CPTE 31–0	0	**Capture Address, AMENT** The value of the AMENT address is captured in this register when an address detection occurs. This register captures values only for a range detection between a lower and upper value. It does not capture the address when one of the units is programmed in value mode, so only a value mode detection occurs.

Fast Ethernet Controller (FEC) 18

The FEC supports 10/100 Mbps Ethernet as defined by **IEEE** Std. 802.3. It has two MAC-PHY interfaces: the-media independent interface (MII) and the reduced MII (RMII) to provide MII functionality on a reduced pin count (10 instead of 18), and a 7-Wire Interface mode. The media access controller (MAC) handles the MII interface FIFOs and DMA functionality. An MII gasket (MIIGSK) module supports the RMII interface. An FEC RISC microcontroller manages DMA buffer descriptors (BDs), minimizing processor usage. Also, a management information base (MIB) module tracks network activity on the MAC-PHY interface.

Note: Not all MSC711x devices have an FEC. To determine whether your device includes this module, consult **Table 1-2**, *MSC711x Device-Specific Feature Comparison*, on page 1-8.

18.1 Features

FEC features are as follows:

- Designed to comply with **IEEE** Std. 802.3, 802.3u™, 802.3x™, and 802.3ac™.
- Internal receive and transmit FIFOs and a FIFO controller.
- Direct access to internal MSC711x memories via its own DMA controller.
- Support for the 10/100 Mbps media independent interface (MII)
- Support for the 10/100 Mbps reduced media independent interface (RMII).
- Support for the 10 Mbps 7-wire interface.
- Full duplex (200 Mbps throughput with a minimum system clock rate of 25 MHz) and half duplex operation (100 Mbps throughput).
- Programmable maximum frame length supports **IEEE** Std. 802.1™ VLAN tags and priority.
- Retransmission from transmit FIFO following a collision.
- CRC generation and verification for inbound and outbound packets.
- Automatic internal flushing of the receive FIFO for runt receive frames and receive frames rejected by address recognition.
- **IEEE** Std. 802.3 full duplex flow control.
- Address recognition including promiscuous, broadcast, individual address. hash/exact match, and multicast hash match:

— frames with broadcast address can be always accepted or always rejected
— exact match for single 48-bit individual (unicast) address
— 64-bit hash check of individual (unicast) addresses.
— 64-bit hash check of group (multicast) addresses.

It is important to note that the range of addresses used by the Ethernet MAC as a master on the crossbar switch to access M1 memory over the AMENT bus differs from the range of addresses used by the SC1400 core to access these same locations. See **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4 and **Table 5-2**, *MSC711x Detailed Memory Map*, on page 5-5, which shows the correct address ranges.

18.2 FEC Architecture

Figure 1 shows the FEC block diagram, with the network depicted at the bottom of the diagram. The IPBus and interrupt interfaces comply with V2 of the IPBus specification, and the AHB master interface connected to the AMENT bus complies with Rev. 2.0 of the AHB-Lite specification. The external Ethernet interfaces comply with industry and **IEEE** 802.3 standards.

The RISC-based descriptor controller, shown in **Figure 1**, performs the following functions:

- Initializes the internal registers not initialized by the user or hardware.
- Controls the DMA channels at a high level, initiating DMA data transfers.
- Interprets buffer descriptors (BDs).
- Provides address recognition for receive frames.
- Generates random numbers for the transmit collision back-off timer.

The RAM is the focal point of all FEC data flow. The RAM is divided into transmit and receive FIFOs with a boundary that is programmed in the FIFO Receive Start Register (FRST) register discussed on **page 18-49**. User data flows to/from the DMA unit from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block, and receive data flows from the receive block into the receive FIFO.

The bus controller selects the TBus master on each clock. All modules receive their control information over the TBus and provide status information over the TBus.

The user controls the FEC by writing through the slave interface (SIF) module into control registers in each block. The control and status registers provide global control, such as Ethernet reset and enable and interrupt handling.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the MDC (clock) and MDIO bidirectional data lines of the MII interface.

The multiple-channel DMA unit allows transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

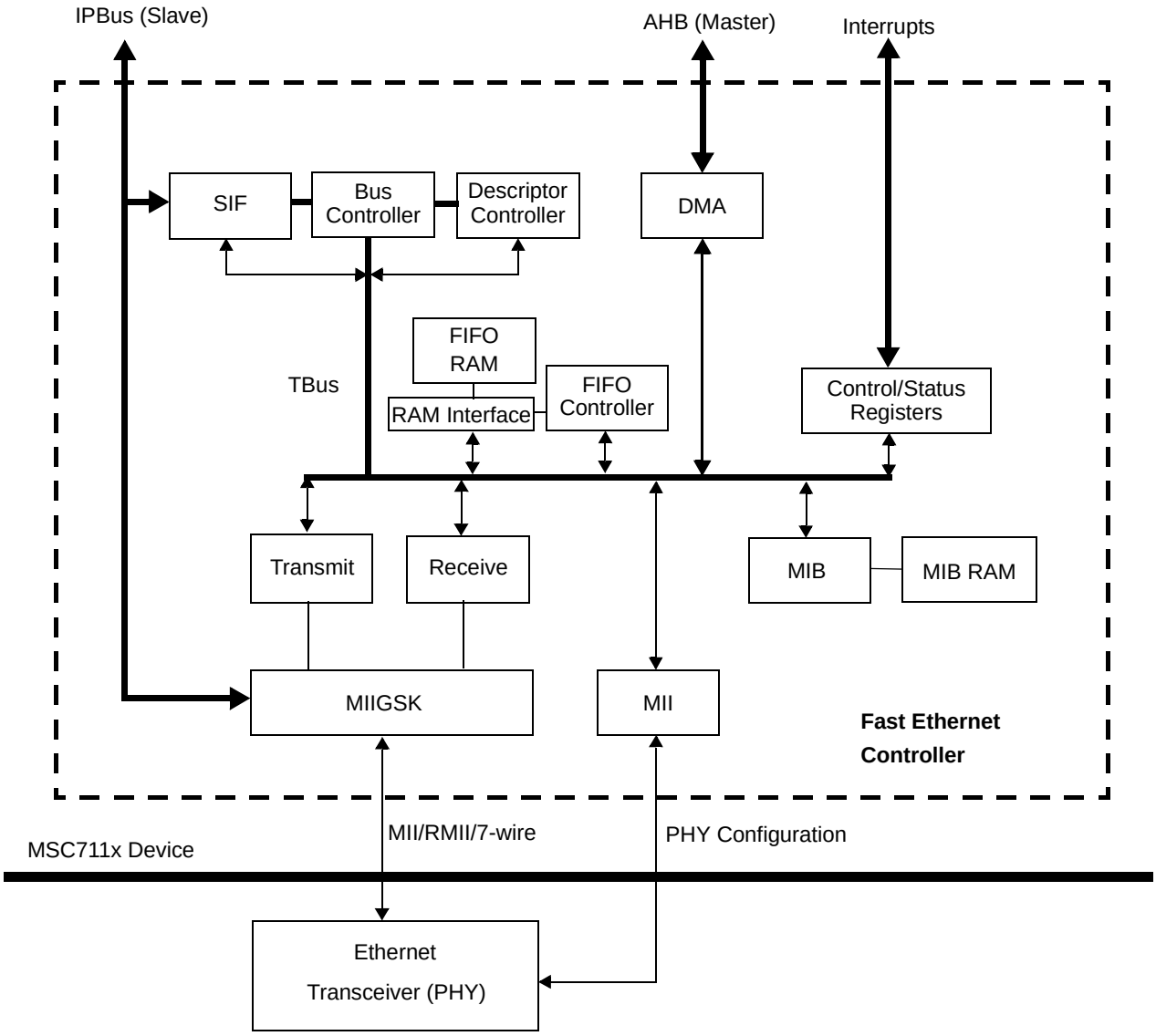


Figure 17-1. FEC Block Diagram

The transmit and receive blocks provide the Ethernet MAC functionality. Internal to these blocks are clock domain boundaries between the system clock and the network clocks.

The management information base (MIB) maintains counters for a variety of network events and statistics. It is not necessary for FEC operation but provides valuable counters for network management. The counters are the remote monitoring (Rmon) RFC 1757 Ethernet Statistics group and some **IEEE** 802.3 counters.

The MII gasket (MIIGSK) converts the data path portion of the MII interface to the RMII interface, reducing the MII data path pin count from 16 to 8 pins.

18.3 FEC MAC-PHY Interface Signal Pins

Table 18-1 summarizes the MAC-PHY interface signal pins for each physical interface mode. All the pins, as shown in **Table 18-1**, are used for MII operation, and subsets of these pins are used for RMII and 7-Wire operation. For descriptions of the individual FEC MAC-PHY interface pins, see **Section 2.5**, *Ethernet MAC Interface Port*, on page 2-16.

Table 18-1. FEC MAC-PHY Interface Pins

Pin	Direction	MII	RMII	7-Wire	Description
TXCLK	Input	TXCLK	REF_CLK	TXCLK	MII/7-Wire: transmit clock RMII: shared reference clock
TXD3	Output	TXD3	—	—	MII: transmit data bit 3
TXD2	Output	TXD2	—	—	MII: transmit data bit 2
TXD[1]	Output	TXD1	TXD1	—	MII/RMII: transmit data bit 1
TXD[0]	Output	TXD0	TXD0	TXD0	MII/RMII/7-Wire: transmit data bit 0
TX_EN	Output	TX_EN	TX_EN	TX_EN	MII/RMII/7-Wire: transmit enable
TX_ER	Output	TX_ER	—	—	MII: transmit error
RXCLK	Input	RXCLK	—	RXCLK	MII/7-Wire: receive clock
RXD3	Input	RXD3	—	—	MII: receive data bit 3
RXD2	Input	RXD2	—	—	MII: receive data bit 2
RXD[1]	Input	RXD1	RXD1	—	MII/RMII: receive data bit 1
RXD[0]	Input	RXD0	RXD0	RXD0	MII/RMII/7-Wire: receive data bit 0
RX_DV	Input	RX_DV	CRS_DV	RX_DV	MII/7-Wire: receive data valid RMII: carrier sense, receive data valid
RX_ER	Input	RX_ER	RX_ER	—	MII/RMII: receive error
COL	Input	COL	—	COL	MII/7-Wire: collision
CRS	Input	CRS	—	—	MII: carrier sense
MDC	Output	MDC	MDC	—	MII/RMII: PHY configuration clock
MDIO	Input/ Output	MDIO	MDIO	—	MII/RMII: PHY configuration data
18 Pins	—	18 Pins	10 Pins	7 Pins	

18.3.1 MII MAC-PHY Signal Pins

The MII MAC-PHY interface uses eighteen signal pins. The transmit and receive functions require seven pins each: four data signals, a delimiter, error, and clock. Additionally, two pins indicate the status of the media: one indicates the presence of a carrier, and the second indicates a collision. The remaining two pins provide a management interface for configuring the Ethernet PHY. Each MII signal is described in **Table 18-2**.

Table 18-2. MII MAC-PHY Pins

Signal	Description
TX_CLK	Input clock that provides a timing reference for TX_EN, TXD[3–0], and TX_ER.
TXD[3–0]	Represents a nibble of valid data when TX_EN is asserted. When TX_EN is de-asserted, TXD has no meaning.
TX_EN	Indicates that valid data is being presented on TXD. This pin asserts with the first nibble of the preamble and is negated prior to the first TX_CLK rising edge following the final nibble of the frame.
TX_ER	Assertion of this pin for one or more clock cycles while TX_EN is asserted causes the PHY to transmit one or more illegal symbols. Asserting TX_ER has no affect at 10 Mbps or when TX_EN is deasserted. TX_ER asserts due to a transmit FIFO underflow, deassertion of Ethernet enable (ECTL[EEN]) during frame transmission, or a forced append of a bad CRC to a transmit frame (set TxBD [ABC]).
RX_CLK	Input clock that provides a timing reference for RX_DV, RXD[3–0], and RX_ER.
RXD[3–0]	Represents a nibble of valid data to be transferred from the PHY to the MAC when RX_DV is asserted. When RX_DV is not asserted, RXD has no meaning.
RX_DV	This input pin indicates a valid nibble on RXD[3–0] when the PHY asserts it. This pin remains asserted from the first recovered nibble of the frame through the last nibble. Assertion of RX_DV must start no later than the SFD.
RX_ER	Indicates PHY detection of an error when asserted with RX_DV. When RX_DV is not asserted, RX_ER has no effect. RX_ER assertion during frame reception sets the CRC error or non-octet aligned error in the corresponding RxBD depending on whether the frame is octet aligned.
CRS	Indicates that the transmit or receive medium is not idle. CRS remains asserted through the duration of a collision. This pin is not required to transition synchronously with TX_CLK or RX_CLK.
COL	Indicates detection of a collision in half-duplex operation. The behavior of this signal is not specified in full-duplex mode. This signal is not required to transition synchronously with TX_CLK or RX_CLK.
MDC	This output pin provides a timing reference to the PHY for data transfers on the MDIO pin.
MDIO	Transfers control/status information between the PHY and FEC synchronously with MDC. The MDIO pin is bidirectional.

18.3.2 RMII MAC-PHY Pins

The RMII provides the functionality of the MII interface on a total of 10 pins instead of 18. The MDC and MDIO pins of the MII interface are used with RMII just as with MII. The 8 RMII data path pins are defined in **Table 18-3**.

Table 18-3. RMII MAC-PHY Pins

Signal	Description
REF_CLK	Continuous input clock that provides a timing reference for CRS_DV, RXD[1–0], TX_EN, TXD[1–0], and RX_ER. The PHY also has an input corresponding to this clock.
TXD[1–0]	Valid transmit data when TX_EN is asserted. When TX_EN is deasserted, TXD has no meaning. When TX_EN is asserted, the PHY accepts TXD[1–0] for transmission.
TX_EN	This output indicates that the data on TXD[1–0] is valid and should be accepted by the PHY. TX_EN asserts at the frame preamble and remains asserted during the frame transmission.
RXD[1–0]	Valid receive data when CRS_DV is asserted. For each clock period in which CRS_DV is asserted, RXD[1–0] transfers two bits of recovered data from the PHY.
CRS_DV	The PHY asynchronously asserts this input pin, carrier sense/receive data valid, when the receive medium is non-idle (carrier detected). When the carrier is lost, this pin deasserts synchronously to REF_CLK. This pin toggles at 25 MHz in100 Mbps mode if the PHY has additional data to transfer after the carrier is lost. This behavior allows for recovery of carrier sense and receive data valid.
RX_ER	This input pin indicates that the PHY has detected an error in the current frame.

18.3.3 7-Wire MAC-PHY Interface Pins

The 7-Wire MAC-PHY interface provides 10 Mbps operation on seven pins. The pin descriptions for this interface are the same as those in **Section 18.3.1** except that TXD[3–1], RXD[3–1], TX_ER, RX_ER, CRS, MDC, and MDIO are unused.

18.4 FEC Operation

This section describes the FEC software initialization sequence and the software (Ethernet driver) interface for transmitting and receiving frames, as well as other FEC functionality.

18.4.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC descriptor controller, and which locations you must initialize before you enable the FEC. Hardware resets some FEC registers, specifically those that generate interrupts and cause outputs to be asserted. Other registers are reset when the ECTL[EEN] enable bit is cleared, disabling the FEC. To halt operation, ECTL[EEN] is cleared by a hard reset or by software. When ECTL[EEN] is cleared, the configuration control registers such as TCTL and RCTL are not reset, but the entire data path is reset, as summarized in **Table 18-4**..The procedure for shutting down the FEC is described in **Section 11.4.4.3**, *Complete Halt of the Ethernet MAC*, on page 11-19.

Table 18-4. Effect on FEC of ECTL[EEN] = 0

Register/Machine	Reset Value
Transmit block	Transmission aborts and a bad CRC is appended.
Receive block	Receive activity is aborted.
Descriptor controller	Halts and resets to the initialization sequence.
Receive Control Descriptor Active (RDA), **page 18-34**.	Cleared.
Transmit Control Descriptor Active (TDA), **page 18-35**.	Cleared.
DMA unit	All DMA activity terminates, and the control logic and AHB interfaces are reset.

You must initialize portions the FEC before you set the ECTL[EEN] bit to enable the FEC. The exact values depend on the application. The sequential order is not important. FEC registers requiring initialization are listed in **Table 18-5**.

Table 18-5. Registers to Initialize Before ECTL[EEN] = 1

Register	Page
Initialize the Interrupt Enable Register (IMASK).	**page 18-31**
Clear the Interrupt Event Register (IEVENT) by writing a value of 0xFFFF_FFFF to it.	**page 18-29**
Initialize the Transmit FIFO Watermark Register (TWMRK) (Optional).	**page 18-47**
Initialize Descriptor Individual Address Registers 1 and 2 (IAADR1/IADDR2).	**page 18-45**
Initialize Descriptor Group Address Registers 1 and 2 (GADDR1/GADDR2).	**page 18-46**
Initialize the Descriptor Physical Address Low Register (PADDRL) and the Descriptor Physical Address High Register (PADDRH).	**page 18-44** **page 18-44**
Initialize the Opcode/Pause Duration Register (OPPAUSE) (needed only for FDX flow control)	**page 18-45**
Initialize the Receive Control Register (RCTL).	**page 18-40**
Initialize the MIIGSK Configuration Register (MIIGSKCFG) (needed only for RMII).	**page 18-51**
Initialize the Transmit Control Register (TCTL).	**page 18-42**
Initialize the MII Speed Register (MIISPEED) (optional)	**page 18-38**
Clear MIBRAM (at locations 200–2FC).	
Initialize the FIFO Receive Start Register (FRST) (optional)	**page 18-49**
Initialize the Receive Buffer Size Register (RBSZ).	**page 18-50**
Initialize the Receive Descriptor Ring Start Register (DMARDST).	**page 18-49**
Initialize the Transmit Descriptor Ring Start Register (DMATDST).	**page 18-50**
Initialize the DMA Control Register (DMACTL).	**page 18-51**
Initialize (Empty) the Transmit Descriptor Ring.	
Initialize the MIIGSK Enable Register (MIIGSKEN).	**page 18-52**

After setting ECTL[EEN], you can set up the buffer/frame descriptors and write to the Receive Descriptor Active (RDA) and Transmit Descriptor Active (TDA) registers. If continuous buffer

descriptor polling is desired, configure the RDCP and TDCP fields of the Descriptor Ring Poll Control Register before writing to the RDA and TDA registers. Keep in mind that the use of these bits can result in a significant increase in system loading.

18.4.2 Operating Modes

Table 18-6 lists the FEC operating modes.

Table 18-6. FEC Operating Modes

Mode	Description
Full Duplex and Half Duplex	Full duplex operation is selected when the TCTL[FDEN] bit is set. Full duplex mode is for use on point-to-point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Full duplex flow control can be enabled in full duplex mode. Refer to the TCTL[RFCP, TFCP] bits and the RCTL[FCE] bit, and **Section 18.4.7**, *Full Duplex Flow Control*, on page 18-17.
10 Mbps and 100 Mbps MII mode	The MAC-PHY interface operates in Media-Independent Interface (MII) mode when the RCTL[MIIM] bit is set. The speed of operation is determined by the TXCLK and RXCLK signals driven by the transceiver. The transceiver either auto-negotiates the speed or software controls the speed via the serial management interface to the transceiver (MDC/MDIO pins). Refer to the MIIDATA and MIISPEED register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.
10 Mbps and 100 Mbps RMII mode	The MAC-PHY interface operates in Reduced Media-Independent Interface (RMII) mode when the MIICFG[RMII] bit is set. The MIICFG[FCTL] bit determines the speed of operation. The reference clock for RMII is always 50 MHz, but this clock can be divided by 10 within the MIIGSK to support 10 Mbps operation. The PHY must be configured accordingly.
10 Mbps 7-Wire Interface mode	The MAC-PHY interface operates in 7-Wire Interface mode when the RCTL[MIIM] bit is cleared. The speed of operation is determined by the TXCLK and RXCLK signals driven by the transceiver. The FEC supports only 10 Mbps 7-Wire Interface mode.
Address recognition	The address recognition options are set in the Receive Control Register (RCTL), which is described on **page 18-40**. See also **Section 18.4.6**, *Ethernet Address Recognition*, on page 18-12.
Internal Loopback mode	Internal Loopback mode is selected via the RCTL[LOOPB] bit (see **page 18-40**).

18.4.3 Buffer Descriptors

The data for FEC frames must reside in memory external to the FEC. The data for a frame is placed into one or more buffers, each with an associated BD that contains a starting address (pointer) and the data length for the buffer. The most significant bit of the BD is an ownership bit that defines the current state of the buffer. Other bits in the BD communicate status/control information between the Ethernet MAC and the driver (see the definition of RxBD on

page 18-23). To permit maximum user flexibility, the BDs reside in external memory and are read by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing BDs. Setting the R/E (ownership) bit in the most significant word of the transmit (receive) BD produces the buffer. Writing to either the TDA and RDA (transmit/receive active) registers, software notifies the FEC that a buffer has been placed into external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the DMA data transfer completes and the DMA engine writes the BD status bits, hardware clears the R(E) bit to signal that the contents of the buffer have been emptied. Software either polls the BDs to detect when the buffers are emptied or relies on the buffer/frame interrupts. The driver can then process these buffers and return them to the free list.

The ECTL[EEN] bit acts as a reset to the BD/DMA logic. When ECTL[EEN] is cleared, the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. During reset, hardware does not initialize the BDs. Software must initialize at least one transmit and receive BD (write 0x00000000 to the most significant word of the BD) before the ECTL[EEN] bit is set.

The BDs operate as two separate rings. RDESST defines the starting address for receive BDs and TDESST defines the starting address for transmit BDs. The last BD in each ring is defined by the Wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location to which the RDESST and TDESST registers point for the receive and transmit rings, respectively. BD rings must start on a 32-bit boundary.

18.4.3.1 Driver/DMA Operation with Transmit BDs

Typically, a transmit frame is divided between multiple buffers. For example, an application could place an application payload in one buffer, the TCP header in a second buffer, the IP header in a third buffer, and the Ethernet/802.3 header in a fourth buffer. The Ethernet MAC does not prepend the Ethernet header, which contains the destination address, source address, length/type field(s), and so on. The driver must provide this header in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. The TC bit in the transmit BD, which is set by the driver, determines whether the CRC is appended by the MAC or by the driver.

In an end station application, the value of the TC bit is always 1. For a switch/router application, the TC bit can have a value of 1 or 0, depending on the type of port on which the frame arrives and whether the frame contents are modified. The Append Bad CRC (ABC) bit has a value of 0 unless an error, such as a data parity error during a DMA transfer, results in data corruption.

The driver should set up Tx BDs to present a complete transmit frame to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the ownership (R) bits should be set = 1 in reverse order (third, second, first BD) to ensure that the complete frame is ready in memory before the DMA transfer

begins. If the TxBDs are set up in order, the DMA controller can transfer the first BD before the second is made available, potentially causing a transmit FIFO underrun.

The driver notifies the FEC DMA controller that new transmit frame(s) are available by writing to the TDESST register. The FEC descriptor controller notifies the DMA controller to read the next transmit BD in the ring. The transmit BDs are read and interpreted in order, and the DMA controller transfers the associated buffers until a transmit BD is encountered with the R bit = 0. The FEC polls this BD one more time. If the R bit = 0 the second time, the descriptor controller stops the transmit descriptor read process until software sets up another transmit frame and writes to TDESST.

After each buffer is transmitted, the DMA controller writes back to the BD to clear the R (ownership) bit, indicating that hardware is finished with the buffer. A second driver task (Tx BD software consumer) processes the transmit descriptor ring and returns buffers consumed by the hardware to the free list. Setting up a transmit BD ring with a single BD is not advised. A BD ring as such can result in multiple transmissions of the same frame before BD closure because the DMA controller processes it in a pipeline. For frame transmission via a single BD, a ring of at least two BDs should be set up.

18.4.3.2 Driver/DMA Operation with Receive BDs

The length of the receive frame is unavailable to the driver ahead of time. Therefore, the driver must set a variable to define the length of all receive buffers. This variable is written to the RBSZ register.

The driver sets up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The DMA controller consumes these buffers by filling them with data as frames are received. It clears the E bit and writes to the L bit (1 indicates last buffer in frame), the frame status bits (if L = 1), and the length field.

If a receive frame spans multiple receive buffers, the L bit is set only for the last buffer in the frame. For the other buffers, the DMA controller writes the length field in the receive BD with the default receive buffer length value when the E bit is cleared. For end-of-frame buffers the receive BD is written with L = 1. Additionally, the status bits (M, BC, MC, LG, NO, SH, CR, OV, TR) are updated with frame status. Some of the status bits are error indicators that are set to indicate the receive frame should be discarded and not given to higher layers.

The length field for the end-of-frame buffer is written with the length of the entire frame, not just the length of the last buffer. For simplicity the driver may assign a default receive buffer length large enough to contain an entire frame, keeping in mind that a malfunction could result in giant frames. The FEC truncates frames of 2k (2048) bytes or larger at 2047 bytes, so software is guaranteed never to receive a frame larger than 2047 bytes.

The FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the Receive Descriptor Active Register (RDA). As frames are received, the FEC fills receive buffers

and updates the associated BDs, then reads the next BD in the ring. If the FEC reads a receive BD and finds the E bit = 0, it polls this BD once more. If the BD = 0 a second time, the FEC stops reading receive BDs until the driver writes to RDA.

18.4.4 FEC Frame Transmission

The Ethernet transmitter works with almost no software intervention. When ECTL[EEN] is set and data is read into the transmit FIFO, the Ethernet MAC can transmit onto the network.

The Ethernet controller transmits bytes least significant bit first. When the transmit FIFO fills to the watermark defined in the Transmit FIFO Watermark Register (TWMRK), the MAC transmit logic asserts TXEN and starts transmitting the preamble sequence, the start frame delimiter, and the frame information from the FIFO. However, the controller defers the transmission if the network is busy (that is, carrier sense is asserted). Before transmitting, the controller waits for carrier sense to become inactive for 60 bit times. Then the transmission begins after a further wait of 36 bit times, for a total of 96 bit times after carrier sense became inactive.

In Half Duplex mode, if a collision occurs during frame transmission, the Ethernet controller follows the specified back-off procedures and attempts to retransmit the frame until the retry limit threshold is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame so that they do not have to be retrieved from system memory in case of a collision. This practice improves bus usage and latency.

When all frame data is transmitted, the FCS (32-bit CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data, regardless of the TC bit value. After the CRC is transmitted, the Ethernet controller writes the frame status information to the MIB. Short frames are automatically padded if the TC bit in the transmit BD for the end of frame buffer = 1.

Both buffer and frame interrupts can be generated as determined by the settings in the IMASK register. Transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds RCTL[MAXFL] bytes, the BABT interrupt is asserted. However, the entire frame is transmitted (no truncation).

To pause transmission, set the Graceful Transmit Stop (GTS) bit in the TCTL register. When the GTS is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame either finishes or terminates with a collision. After the transmitter stops, the GRA interrupt is asserted. If GTS is cleared, the FEC resumes transmission with the next frame.

18.4.5 FEC Frame Reception

The FEC receiver works with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking and maximum frame length checking.

When the driver sets the ECTL[EEN] bit to enable the FEC receiver, it immediately starts processing receive frames. When RXDV asserts, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the receiver processes the frame. If a PA/SFD is not available, the frame is ignored.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored. After the first 6 bytes of the frame are received, the FEC performs address recognition on the frame. When a collision window (64 bytes) of data is received and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame is accepted and can be passed on to the DMA controller. If the frame is a fragment due to collision or is rejected by address recognition, the receive FIFO is notified to reject the frame. Thus, no collision fragments are presented to the user, except late collisions, which indicate serious LAN problems.

Receive buffer (RXB) and frame (RFINT) interrupts can be generated if enabled by the IMASK register. Receive error interrupts are BABR. Receive frames are not truncated if they exceed the RCTL[MAXFL] byte length; however, the BABR interrupt occurs and the LG bit in the RxBD is set.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (IEVENT[RFINT], maskable by IMASK[RFIEN]), indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

18.4.6 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type: individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on receive frames is presented in **Figure 18-1** and **Figure 18-2**.

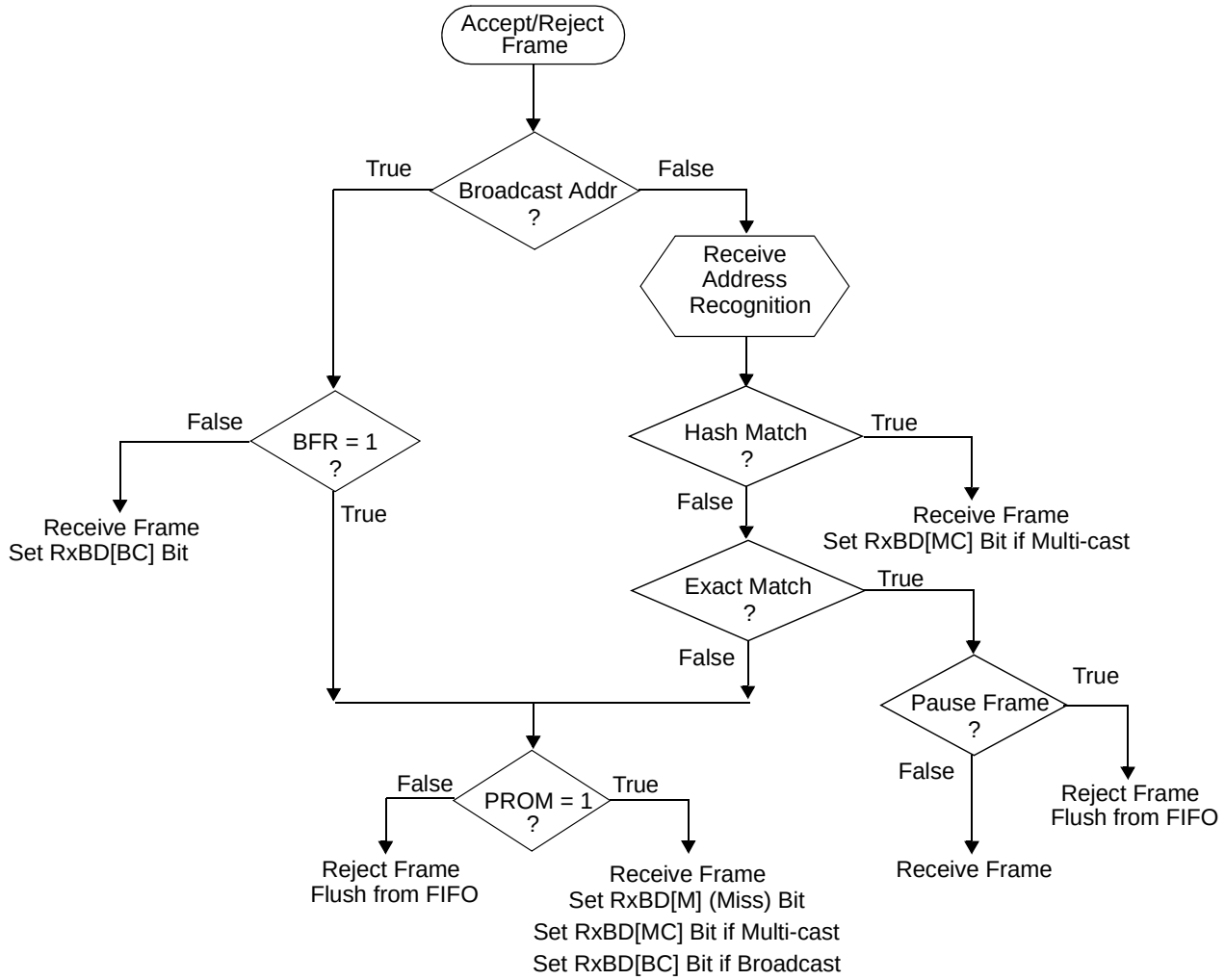
Address recognition is accomplished through the use of the receive block and the code running on the SC1400 core. The flowchart in **Figure 18-1** illustrates the address recognition decisions made by the receive block, and **Figure 18-2** illustrates the decisions made by the SC1400 core.

If the DA is a broadcast address and broadcast reject, RCTL[BFR], is cleared, then the frame is accepted unconditionally. If the DA is a group (multicast) address and flow control is disabled, a group hash table look-up uses the 64-entry hash table programmed in the Descriptor Group Address 1 and 2 Registers (GADDR1 and GADDR2). If a hash match occurs, the receiver accepts the frame.

If the DA is the individual (unicast) address, an individual exact match comparison is performed between the DA and 48-bit physical address that the user programs in the Physical Address Low (PADDRL) and Physical Address High (PADDRH) registers. If there is not an exact match, an

individual hash table look-up is performed using the 64-entry hash table programmed in registers, IADDR1 and IADDR2. If there is a hash table match, the receiver accepts or rejects the frame based on PAUSE frame detection. Detected PAUSE frames are rejected. When a frame is rejected, it is flushed from the FIFO. If neither a hash match nor an exact match occurs, then the Promiscuous mode setting determines frame acceptance or rejection. If Promiscuous mode is enabled (RCTL[PROM] = 1), the frame is accepted and the RxBD[MISS] bit is set. Otherwise, the frame is rejected and the MISS bit is cleared.

Similarly, if the DA is a broadcast address, broadcast reject (RCTL[BFR) is set, and Promiscuous mode is enabled, the frame is accepted and the RxBD[MISS] bit is set. Otherwise, the frame is rejected and the MISS bit is cleared.



NOTES:

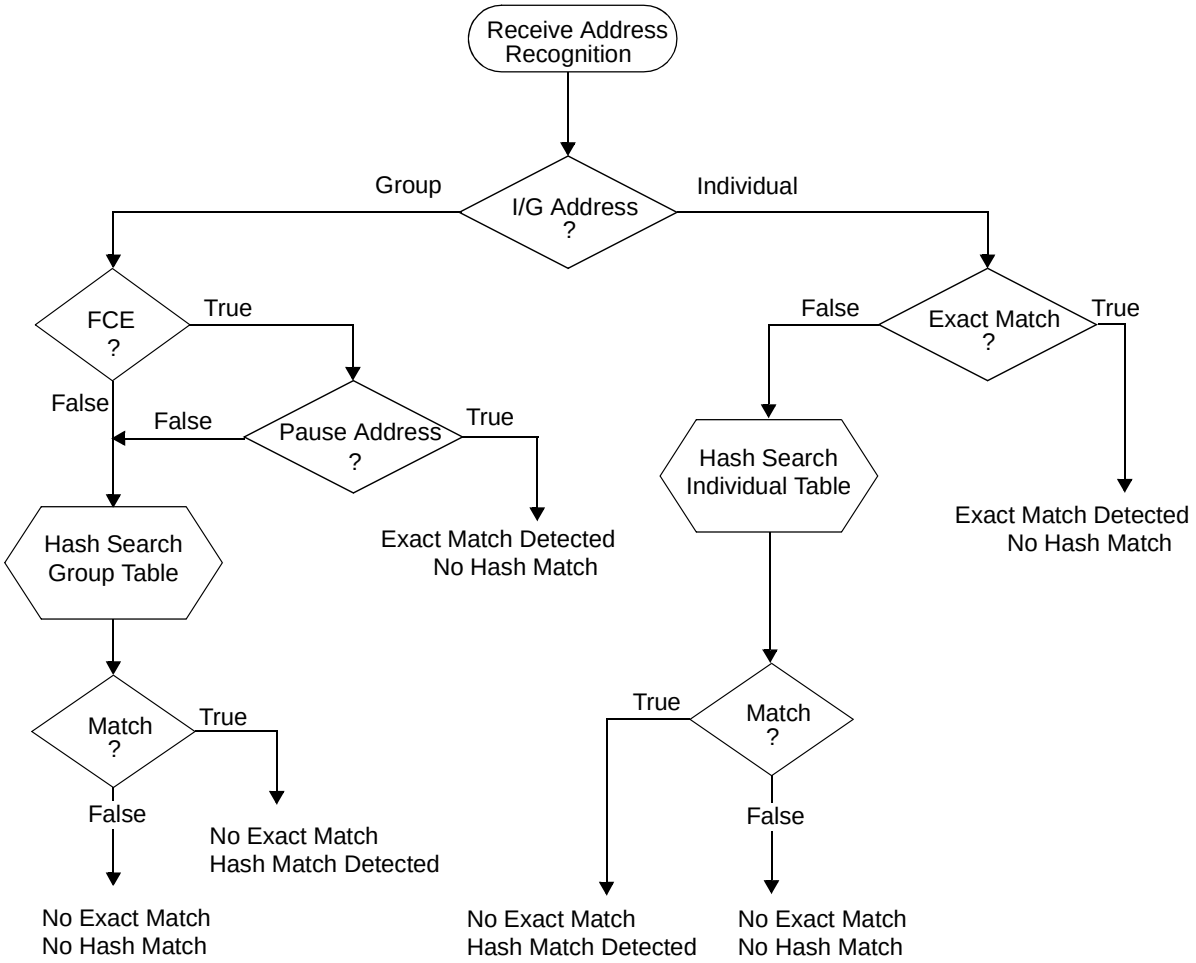
BFR is a field in the RCTL register (Broadcast Frame Reject)

PROM is a field in the RCTL register (Promiscuous Mode).

Pause frame queries whether a valid PAUSE frame is received.

Figure 18-1. Ethernet Address Recognition, Decisions on the Receive Side

The hash table algorithm used in group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GADDR1,2 (group address hash match) or IADDR1,2 (individual address hash match). This mapping is performed by passing the 48-bit address through the internal 32-bit CRC generator and selecting the six most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GADDR1 (MSB = 1) or GADDR2 (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, it is rejected.



Notes:

FCE is a field in the RCTL Register (flow control enable).
I/G is the Individual/Group bit in destination address (least significant bit in first byte received in the MAC frame).

Figure 18-2. Ethernet Address Recognition, Decisions on the Microcode Side

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5 percent) of the group address frames from reaching memory. The processor further filters those that do reach memory to determine

whether they truly contain one of the eight desired addresses. The effectiveness of the hash table declines as the number of addresses increases.

You must initialize the hash table registers. You can compute the hash for a particular address in software and use it to program the FEC hash table registers. The CRC32 polynomial to use in computing the hash is as follows:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Table 18-7 shows example destination addresses and corresponding hash values.

Table 18-7. Destination Address to 6-Bit Hash

48-Bit Destination Address	6-Bit Hash (Hexadecimal)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff:ff	0xA	10
8b:ff:ff:ff:ff:ff	0xB	11
0b:ff:ff:ff:ff:ff	0xC	12
3b:ff:ff:ff:ff:ff	0xD	13
7b:ff:ff:ff:ff:ff	0xE	14
5b:ff:ff:ff:ff:ff	0xF	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24

Table 18-7. Destination Address to 6-Bit Hash (Continued)

48-Bit Destination Address	6-Bit Hash (Hexadecimal)	Hash Decimal Value
b9:ff:ff:ff:ff:ff	0x19	25
f9:ff:ff:ff:ff:ff	0x1A	26
c9:ff:ff:ff:ff:ff	0x1B	27
59:ff:ff:ff:ff:ff	0x1C	28
79:ff:ff:ff:ff:ff	0x1D	29
29:ff:ff:ff:ff:ff	0x1E	30
19:ff:ff:ff:ff:ff	0x1F	31
d1:ff:ff:ff:ff:ff	0x20	32
f1:ff:ff:ff:ff:ff	0x21	33
b1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7f:ff:ff:ff:ff:ff	0x28	40
4f:ff:ff:ff:ff:ff	0x29	41
1f:ff:ff:ff:ff:ff	0x2A	42
3f:ff:ff:ff:ff:ff	0x2B	43
bf:ff:ff:ff:ff:ff	0x2C	44
9f:ff:ff:ff:ff:ff	0x2D	45
df:ff:ff:ff:ff:ff	0x2E	46
ef:ff:ff:ff:ff:ff	0x2F	47
93:ff:ff:ff:ff:ff	0x30	48
b3:ff:ff:ff:ff:ff	0x31	49
f3:ff:ff:ff:ff:ff	0x32	50
d3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3d:ff:ff:ff:ff:ff	0x38	56
0d:ff:ff:ff:ff:ff	0x39	57
5d:ff:ff:ff:ff:ff	0x3A	58
7d:ff:ff:ff:ff:ff	0x3B	59
fd:ff:ff:ff:ff:ff	0x3C	60

Table 18-7. Destination Address to 6-Bit Hash (Continued)

48-Bit Destination Address	6-Bit Hash (Hexadecimal)	Hash Decimal Value
dd:ff:ff:ff:ff:ff	0x3D	61
9d:ff:ff:ff:ff:ff	0x3E	62
bd:ff:ff:ff:ff:ff	0x3F	63

18.4.7 Full Duplex Flow Control

Full-duplex flow control allows you to transmit pause frames and to detect received pause frames. When a pause frame is detected, MAC data frame transmission stops for a specified pause duration. To enable pause frame detection, the FEC must operate in Full-Duplex mode (TCTL[FDEN] = 1) with flow control enabled (RCTL[FCE] = 1). The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in **Table 18-8**. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 18-8. Pause Frame Field Specification

48-Bit Destination Address	0x0180_c200_0001 or Physical Address
48-bit source address	Any
16-bit type	0x8808
16-bit opcode	0x0001
16-bit PAUSE duration	0x0000–0xFFFF

When a pause frame is detected, the FEC asserts a graceful transmit stop internally. When transmission pauses, the Graceful Stop Complete interrupt is asserted and the pause timer begins to increment. The receive flow control pause (TCTL[RFCP]) status bit is set while the transmitter pauses due to a pause frame. To transmit a pause frame, the FEC must operate in Full-Duplex mode and flow control pause must be asserted (TCTL[TFCP] = 1). The transmitter asserts graceful transmit stop internally. After GRA assertion, the pause frame is transmitted. When pause frame transmission completes, flow control pause (TCTL[TFCP]) and the graceful transmit stop are deasserted internally.

During pause frame transmission, the transmit hardware places data into the transmit data stream from the registers shown **Table 18-9**. You must specify the desired pause duration in the OPPAUSE register (see **page 18-45**).

Table 18-9. Transmit Pause Frame Registers

Pause Frame Fields	FEC Register	Register Contents
48-bit source address	PADDRL[0–31], PADDRH[0–15]	Physical address
16-bit type	PADDRH[16–31]	8808
16-bit opcode	OPPAUSE[0–15]	0001
16-bit pause duration	OPPAUSE[16–31]	0x0000–0xFFFF

When the transmitter is paused because of receiver pause frame detection, transmit flow control pause (TCTL[TFCP]) can still be asserted and cause the transmission of a single pause frame. In this case, the GRA interrupt is not asserted.

18.4.8 Inter-Packet Gap Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After a transmission or after the back-off algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission can begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs. The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the receiver may discard the next frame.

18.4.9 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern is sent after the preamble sequence ends. If a collision occurs within 64 byte times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 64 byte times, no retransmission is performed and the end of frame buffer is closed with an LC error indication.

18.4.10 Internal and External Loopback

In internal and external Loopback mode, both of the FIFOs are used and the FEC operates in full duplex. Both internal and external loopback are configured using combinations of the RCTL[LOOPB, DRT] bits and the TCTL[FDEN] bit. For both internal and external loopback, set FDEN = 1.

For internal loopback, set LOOPB = 1 and DRT = 0. TXEN and TXER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than normal because the transmit and receive blocks use the internal system clock instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data transferred via the DMA controller to/from external memory. It may be

necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback set LOOPB = 0, DRT = 0 and configure the external transceiver for loopback.

18.4.11 Ethernet Transmission Error-Handling

The Ethernet controller reports frame reception and transmission error conditions using the FEC receive BDs, the IEVENT register, and the MIB counters. **Table 18-10** lists the Ethernet transmission errors.

Table 18-10. Ethernet Transmission Errors

Error	Description
Transmitter Underrun	If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for the frame are flushed and closed. The FEC continues to the next transmit BD and begins transmitting the next frame. If the IMASK[TFUEN] bit is set, an interrupt is enabled.
Carrier Sense Lost During Frame Transmission	When this error occurs and no collision is detected in the frame, the frame is transmitted normally. No retries are performed as a result of this error. No interrupt is generated as a result of this error.
Retransmission Attempts Limit Expired	When this error occurs, the FEC terminates transmission. All remaining buffers for the frame are then flushed and closed. The FEC then continues to the next transmit BD and begins transmitting the next frame. If the IMASK[CLREN] bit is set, an interrupt is generated.
Late Collision	When a collision occurs after the slot time (512 bits starting at the preamble), the FEC terminates transmission. All remaining buffers for that frame are then flushed and closed. The FEC then continues to the next transmit BD and begins transmitting the next frame. If the IMASK[LCEN] bit is set, an interrupt is generated.
Heartbeat	Some transceivers have a self-test feature called heartbeat or signal quality error (SQE). To signify a good self-test, the transceiver must indicate a collision to the FEC within 10 cycles (4 microseconds for 10 Mbps operation) after the Ethernet controller transmits a frame. This collision indication does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition. If the TCTL[HBC] bit is set and the FEC does not detect the heartbeat condition after a frame transmission, a heartbeat error occurs. The FEC closes the buffer and generates the IEVENT[HBERR] interrupt if it is enabled in IMASK[HBEEN].

18.4.12 Ethernet Reception Error Handling

The Ethernet controller reports frame reception and transmission error conditions using the FEC receive BDs, the IEVENT register, and the MIB counters. **Table 18-11** lists the Ethernet reception errors.

Table 18-11. Ethernet Transmission Errors

Error	Description
Overrun Error	If the receive block has data for the receive FIFO but the receive FIFO is full, the FEC sets the OV bit in the receive status word. All subsequent data in the frame is discarded and subsequent frames may also be discarded until the DMA controller services the receive FIFO and space is made available. At this point, the receive frame/status word is written into the FIFO with the OV bit set. The driver must discard this frame.
Non-Octet Error (Dribbling Bits)	The Ethernet controller handles up to seven dribbling bits when the receive frame terminates nonoctet aligned, and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, the frame nonoctet aligned (NO) error is reported in the RxBD. If there is no CRC error, no error is reported.
CRC Error	When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.
Frame Length Violation	When the receive frame length exceeds RCTL[MAXFL] bytes, the BABR interrupt is generated and the LG bit in the end of frame RxBD is set. The frame is not truncated.
Truncation	When the receive frame length exceeds 2047 bytes the frame is truncated and the TR bit is set in the RxBD.

18.4.13 Reset

An FEC hard reset occurs when the peripheral modules reset asserts as described in **Chapter 13**, *Reset* or when the ECTL[RESET] bit is set. A soft reset occurs when the ECTL[EEN] bit is set. Hardware automatically clears ECTL[RESET] after the internal soft reset signal is asserted for eight clock cycles. The ECTL[EEN] bit allows software to reset internal data path control logic without resetting the internal mode control bits. This bit is cleared following a hard reset, and software should set it only after the FEC registers are initialized. During operation, this bit can be cleared if a soft error condition requires the data path to be reset prior to restarting transmission and reception.

18.4.14 Interrupts

When an interrupt event occurs, a bit is set in the IEVENT register. Bits in the IEVENT register are set by the initial occurrence of the event and remain set until software clears them. If a bit in the IEVENT register is set and the corresponding bit is set in the IMASK register, the corresponding interrupt asserts. Software clears individual interrupts by writing a 1 to the corresponding bit in the IEVENT register. For information on the different types of interrupts on MSC711x devices, see IEVENT on **page 18-29**. Even though two requests may share an interrupt vector, the requests can be programmed for different interrupt priorities and can be separately enabled.

18.5 Fast Ethernet Controller Programming Model

The FEC programming model includes the management information base (MIB) counters, the Ethernet receive and transmit BDs, and the FEC registers.

18.5.1 Management Information Base (MIB) Counters

Table 18-12 defines the MIB counters memory map, which defines the locations in the MIB RAM space where hardware-maintained counters reside. These counters fall into the 0x200–0x3FC address offset range. The counters are divided into two groups, RMON counters and **IEEE** counters.

The RMON counters are the Ethernet statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet Statistics group, a counter is included to count truncated frames because the FEC supports only frame lengths up to 2047 bytes. The RMON transmit and receive counters are independent to ensure accurate network statistics in full duplex mode.

The **IEEE** counters support the mandatory and recommended counter packages defined in section 5 of ANSI/**IEEE** Std. 802.3 (1998 edition). The FEC supports the **IEEE** Basic Package objects, which do not require counters in the MIB block. In addition, some of the recommended package objects supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 18-12. MIB Counters (Address 0x0200–0x03FF)

Address	Mnemonic	Description
0200	RMON_T_DROP	Count of frames not counted correctly
0204	RMON_T_PACKETS	RMON Tx packet count
0208	RMON_T_BC_PKT	RMON Tx broadcast packets
020C	RMON_T_MC_PKT	RMON Tx multicast packets
0210	RMON_T_CRC_ALIGN	RMON Tx packets with CRC/align error
0214	RMON_T_UNDERSIZE	RMON Tx packets < 64 bytes, good CRC
0218	RMON_T_OVERSIZE	RMON Tx packets > MAX_FL bytes, good CRC
021C	RMON_T_FRAG	RMON Tx packets < 64 bytes, bad CRC
0220	RMON_T_JAB	RMON Tx packets > MAX_FL bytes, bad CRC
0224	RMON_T_COL	RMON Tx collision count
0228	RMON_T_P64	RMON Tx 64 byte packets
022C	RMON_T_P65TO127	RMON Tx 65–127 byte packets
0230	RMON_T_P128TO255	RMON Tx 128–255 byte packets
0234	RMON_T_P256TO511	RMON Tx 256–511 byte packets
0238	RMON_T_P512TO1023	RMON Tx 512–1023 byte packets
023C	RMON_T_P1024TO2047	RMON Tx 1024–2047 byte packets

Table 18-12. MIB Counters (Address 0x0200–0x03FF) (Continued)

Address	Mnemonic	Description
0240	RMON_T_P_GTE2048	RMON Tx packets with > 2048 bytes
0244	RMON_T_OCTETS	RMON Tx octets
0248	IEEE_T_DROP	Count of frames not counted correctly
024C	IEEE_T_FRAME_OK	Frames transmitted OK
0250	IEEE_T_1COL	Frames transmitted with single collision
0254	IEEE_T_MCOL	Frames transmitted with multiple collisions
0258	IEEE_T_DEF	Frames transmitted after deferral delay
025c	IEEE_T_LCOL	Frames transmitted with late collision
0260	IEEE_T_EXCOL	Frames transmitted with excessive collisions
0264	IEEE_T_MACERR	Frames transmitted with Tx FIFO underrun
0268	IEEE_T_CSERR	Frames transmitted with carrier sense error
026C	IEEE_T_SQE	Frames transmitted with SQE error
0270	T_FDXFC	Flow control pause frames transmitted
0274	IEEE_T_OCTETS_OK	Octet count for frames transmitted without error
0278–027C	Reserved	Reserved
0284	RMON_R_PACKETS	RMON Rx packet count
0288	RMON_R_BC_PKT	RMON Rx broadcast packets
028C	RMON_R_MC_PKT	RMON Rx multi-cast packets
0290	RMON_R_CRC_ALIGN	RMON Rx packets with CRC/align error
0294	RMON_R_UNDERSIZE	RMON Rx packets < 64 bytes, good CRC
0298	RMON_R_OVERSIZE	RMON Rx packets > MAX_FL bytes, good CRC
029C	RMON_R_FRAG	RMON Rx packets < 64 bytes, bad CRC
02A0	RMON_R_JAB	RMON Rx packets > MAX_FL bytes, bad CRC
02A4	RMON_R_RESVD_0	Reserved
02A8	RMON_R_P64	RMON Rx 64 byte packets
02AC	RMON_R_P65TO127	RMON Rx 65–127 byte packets
02B0	RMON_R_P128TO255	RMON Rx 128–255 byte packets
02B4	RMON_R_P256TO511	RMON Rx 256–511 byte packets
02B8	RMON_R_P512TO1023	RMON Rx 512–1023 byte packets
02BC	RMON_R_P1024TO2047	RMON Rx 1024–2047 byte packets
02C0	RMON_R_P_GTE2048	RMON Rx packets with > 2048 bytes
02C4	RMON_R_OCTETS	RMON Rx octets
02C8	IEEE_R_DROP	Count of frames not counted correctly
02CC	IEEE_R_FRAME_OK	Frames received OK
02D0	IEEE_R_CRC	Frames received with CRC Error
02D4	IEEE_R_ALIGN	Frames received with alignment error
02D8	IEEE_R_MACERR	Receive FIFO overflow count

Table 18-12. MIB Counters (Address 0x0200–0x03FF) (Continued)

Address	Mnemonic	Description
02DC	R_FDXFC	Flow control pause frames received
02E0	IEEE_R_OCTETS_OK	Octet count for frames received without error
02E4–03FC	Reserved	Reserved

18.5.2 Ethernet Receive and Transmit BDs

The Ethernet receive and transmit BDs are as follows:

- Ethernet Receive Buffer Descriptor (RxBD), **page 18-23**.
- Ethernet Transmit Buffer Descriptor (TxBD), **page 18-26**.

RxBD Fast Ethernet Receive Buffer Descriptor

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0x0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 0x2	Data Length (DL)															
Offset + 0x4	Rx Data Buffer Pointer, A[31–16]															
Offset + 0x6	Rx Data Buffer Pointer, A[15–0]															

RxBD reports information on the received data for each buffer. In the RxBD, you initialize the E and W bits in the first word and the pointer in second word. When the DMA controller transfers the buffer contents, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer into the first word. The Ethernet controller modifies the M, BC, MC, LG, NO, CR, OV, and TR bits in the first word of the BD are only when the L bit is set. The first word of the RxBD contains control and status bits. Its format is detailed in **Table 18-13**.

Table 18-13. RxBD Bit Descriptions

Offset	Name	Description	Settings
0x0	**E** 15	**Empty** Indicates whether the buffer associated with this RxBD is empty. The SC1400 core can read any fields of this RxBD. When E = 1, the SC1400 core should not write any fields of this RxBD. When this bit is cleared, the status and length fields are updated as required. Any time the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDA.	0 The buffer associated with this RxBD is full or reception terminated due to an error. 1 The associated buffer is empty, or reception is in progress.
	RO1 14	**Receive Software Ownership** Reserved for use by software. Hardware does not use or modify this bit.	
	W 13	**Wrap** (final BD in RxBD ring) Indicates that the BD is the final one in the RxBD ring and the next BD is at the location defined by RDESST. The number of RxBDs in this table is programmable and determined only by the W bit. The RxBD ring must contain more than one BD.	0 The next BD is in the consecutive location. 1 The next BD is at the location defined in RDESST (see **page 18-49**).

Table 18-13. RxBD Bit Descriptions (Continued)

Offset	Name	Description	Settings
0x0 cont.	**RO2** 12	**Receive Software Ownership** Reserved for use by software. Hardware does not use or modify this bit.	
	L 11	**Last in Frame** Set by the Ethernet controller when this buffer is the last in a frame.	0 Not the last buffer in a frame. 1 Last buffer in a frame.
	— 10–9	Reserved. Write to zero for future compatibility.	
	M 8	**Miss** Set by the Ethernet controller for frames that are accepted in Promiscuous mode but are flagged as a miss by the internal address recognition. Thus, in Promiscuous mode, use the miss bit to determine quickly whether the frame is destined for this station. Valid only if RxBD[L] and RCTL[PROM] bits are set.	0 The frame is received because the address is recognized. 1 The frame is received because of Promiscuous mode (address is not recognized).
	BC 7	**Broadcast Address** Set is the destination address is broadcast (FF-FF-FF-FF-FF-FF). Valid only for the last buffer in a frame (RxBD[L] = 1).	
	MC 6	**Multicast Address** The received frame address is a multicast address other than a broadcast address. Valid only for the last buffer in a frame (RxBD[L] = 1).	
	LG 5	**Rx Frame Length Violation** A frame length greater than the RCTL[MAXFL] (maximum frame length) defined for this FCC is recognized. The FEC sets this bit only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.	
	NO 4	**Rx Nonoctet Aligned Frame** A frame containing a number of bits not divisible by eight is received and the CRC check at the preceding byte boundary generates an error. This bit is valid only if the L-bit is set. If this bit is set, the CR bit is not set.	
	— 3	Reserved. Write to zero for future compatibility.	
	CR 2	**Rx CRC Error** This frame contains a CRC error and is an integral number of octets long. This bit is valid only if the L-bit is set. This bit is written by the FEC.	
	OV 1	**Overrun** A receiver overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, SH, CR, and CL lose their normal meaning and have a value of zero. This bit is valid only if the L-bit is set.	
	TR 0	**Truncated Frame** Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set, the frame should be discarded and the other error bits should be ignored because they may be incorrect.	

TxBD Fast Ethernet Transmit Buffer Descriptor

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0x0	R	TO1	W	TO2	L	TC	ABC									
Offset + 0x2	Data Length (DL)															
Offset + 0x4	Rx Data Buffer Pointer, A[31–16]															
Offset + 0x6	Rx Data Buffer Pointer, A[15–0]															

Data is arranged in buffers referenced by the TxBDs and presented to the FEC for transmission. The Ethernet controller confirms transmission by clearing an ownership bit (R bit) when DMA transfer of the buffer is complete. In the TxBD, you initialize the R, W, L, and TC bits and the length (in bytes) in the first word and the buffer pointer in the second word. The FEC clears the R bit = 0 in the first word of the BD when the DMA controller has transferred the buffer contents. Status bits for the buffer/frame are not included in the transmit BDs. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB.

When the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in the set-up is to set the R bit in the first BD for the frame. The driver should follow this step with a write to TDA to trigger the FEC to poll the next BD in the ring.

Table 18-14. TxBD Bit Descriptions

Offset	Name	Description	Settings
0x0	**R** 15	**Ready** Both the FEC and the user write to this bit. When this bit is cleared, you are free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. You must not write to this BD after this bit is set.	0 The data buffer associated with this BD is not ready for transmission. 1 The data buffer, prepared by the user for transmission, has not been transmitted or is being transmitted.
	TO1 14	**Transmit Software Ownership Bit** Reserved for use by software. Hardware does not use or modify this bit.	
	W 13	**Wrap** This bit is written by the user and indicates the location for the next BD in the ring. The TxBD ring must contain more than one BD.	0 The next BD is in the consecutive location. 1 The next BD is at the location defined in TDESST (see **page 18-50**).
	TO2 12	**Transmit Software Ownership Bit** Reserved for use by software. Hardware does not use or modify this bit.	
	L 11	**Last in Frame** This bit is written by the user and indicates that the buffer is the last in the transmit frame.	0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.

Table 18-14. TxBD Bit Descriptions (Continued)

Offset	Name	Description	Settings
0x0	**TC** 10	**Transmit CRC** This bit is written by the user and allows the CRC sequence to be appended to the transmit frame. Valid only if L = 1.	0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
	ABC 9	**Append Bad CRC** This bit is written by the user and enables a bad CRC sequence to be appended to the transmit frame.	0 No effect. 1 Transmit the CRC sequence inverted after the last data byte, regardless of the TC value.
	— 8–0	Reserved. Write to zero for future compatibility.	

18.5.3 FEC Registers

The FEC is programmed by a combination of control/status registers and BDs. The control/status registers are used for mode control, interrupts, and extraction of status information. The descriptors pass data buffers and related buffer or frame information between the hardware and software. All accesses (via IPBus) to and from the registers must be via 32-bit accesses, with the exception of the IEVENT and IMASK registers, which also support 16-bit accesses. The FEC registers are listed as follows, along with the number of the page where the discussion of each register begins:

- FEC Identification Register (FECID), **page 18-28**.
- Interrupt Event Register (IEVENT), **page 18-29**.
- Interrupt Enable Register (IMASK), **page 18-31**.
- Descriptor Ring Poll Control Register (DRPC), **page 18-33**.
- Receive Descriptor Active Register (RDA), **page 18-34**.
- Transmit Descriptor Active Register (TDA), **page 18-35**.
- Ethernet Control Register (ECTL), **page 18-35**.
- MII Management Frame Register (MIIDATA), **page 18-37**.
- MII Speed Control Register (MIISPEED), **page 18-38**.
- MIB Control/Status Register (MIBCTL), **page 18-39**.
- Receive Control Register (RCTL), **page 18-40**.
- Receive Hash Register (RHASH), **page 18-41**.
- Transmit Control Register (TCTL), **page 18-42**.
- Physical Address Low Register (PADDRL), **page 18-44**.
- Physical Address High Register (PADDRH), **page 18-44**.
- Opcode/Pause Duration Register (OPPAUSE), **page 18-45**
- Descriptor Individual Address 1 (IADDR1), **page 18-45**.
- Descriptor Individual Address 2 (IADDR2), **page 18-46**.

- Descriptor Group Address 1 (GADDR1), **page 18-46**.
- Descriptor Group Address 2 (GADDR2), **page 18-47**.
- FIFO ID Register (FIFOID), **page 18-47**.
- FIFO Transmit Watermark Register (TWMRK), **page 18-47**.
- FIFO Receive Bound Register (FRBND), **page 18-48**.
- FIFO Receive Start Register (FRST), **page 18-49**.
- Receive Descriptor Ring Start Register (RDESST), **page 18-49**.
- Transmit Descriptor Ring Start Register (TDESST), **page 18-50**.
- Receive Buffer Size Register (RBSZ), **page 18-50**.
- DMA Control Register (DMACTL), **page 18-51**.
- MIIGSK Configuration Register (MIIGSKCFG), **page 18-51**.
- MIIGSK Enable Register (MIIGSKEN), **page 18-52**.

FECID FEC Identification Register ENET_BASE + 0x000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	FECID															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—					DMA	FIFO	—	FECREV							
TYPE	R															
RESET	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

FECID is a read-only register that identifies the FEC and its revision.

Table 18-15. FECID Bit Descriptions

Name	Reset	Description	Settings
FECID 31–16	0	**FEC Identification** Unique value identifying the FEC (10/100 Ethernet MAC).	
— 15–11	0	Reserved. Write to zero for future compatibility.	
DMA 10	1	**DMA Present** Indicates whether DMA functionality is available in the FEC.	0 No DMA function. 1 DMA function present.

Table 18-15. FECID Bit Descriptions (Continued)

Name	Reset	Description	Settings
FIFO 9	1	**FIFO Present** Indicates whether FIFO functionality is included in the FEC.	0 No FIFO. 1 FIFO functionality present.
— 8	0	Reserved. Write to zero for future compatibility.	
FECREV 7–0	0	**FEC Revision** Value identifies the revision of the FEC.	

IEVENT Interrupt Event Register ENET_BASE + 0x004

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	HBERR	BABR	BABT	GRA	TFINT	TXB	RFINT	RXB	MII	—	LC	CRL	TFU	ROV	—	
TYPE	R/W									R	R/W				R	R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IEVENT contains bits that cause interrupts to be generated when they are set. When an event occurs that sets a bit in IEVENT, an interrupt is generated if the corresponding bit in the interrupt enable register (IMASK) is also set. The bit in IEVENT is cleared if a one is written to that bit position. A write of zero has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts that can occur in normal operation are GRA, TFINT, TXB, RFINT, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and CRL. The TFU interrupt results from a transmit FIFO underrun, and the ROV interrupt results from a receiver overrun. Some error interrupts are independently counted in the MIB block counters. Software can mask these interrupts since these errors are visible to network management via the MIB counters.

Table 18-16. IEVENT Bit Descriptions

Name	Reset	Description	Settings
HBERR 31	0	**Heartbeat Error** If the CTL[HBC] bit is set and the FEC does not detect a collision within the heartbeat window, a heartbeat error occurs. The FEC closes the buffer and generates the HBERR interrupt if it is enabled. The heartbeat condition is checked only in Half Duplex mode.	0 Heartbeat error disabled. 1 Heartbeat error enabled.
BABR 30	0	**Babbling Receive Error** Indicates a frame received with a length in excess of RCTL[MAXFL], which specifies the maximum frame length, in bytes.	0 No interrupt. 1 Babbling receive error interrupt.
BABT 29	0	**Babbling Transmit Error** Indicates a frame transmitted with a length in excess of RCTL[MAXFL]. This condition is usually caused when a frame that is too long is placed into the transmit data buffer(s). Truncation does not occur.	0 No interrupt. 1 Babbling transmit error interrupt.
GRA 28	0	**Graceful Stop Complete** Graceful stop means that the transmitter is put into a pause state after completion of the frame being transmitted.This interrupt is asserted for one of three reasons. 1. A graceful stop initiated by setting the TCTL[GTS] bit is now complete. 2. A graceful stop initiated by setting the TCTL[FCP] bit is now complete. 3. A graceful stop initiated by the reception of a valid full duplex flow control pause frame is now complete. Refer to **Section 18.4.7**, *Full Duplex Flow Control*, on page 18-17.	0 No interrupt. 1 Graceful stop complete interrupt.
TFINT 27	0	**Transmit Frame Interrupt** Indicates that a frame has been transmitted and that the last corresponding buffer descriptor (BD) has been updated. This interrupt is generated when the transmit block generates status for the just completed frame.	0 No interrupt. 1 Transmit frame completed interrupt.
TXB 26	0	**Transmit Buffer Interrupt** Indicates that a transmit BD has been updated. This interrupt is generated when a DMA transfer of a transmit buffer is complete.	0 No interrupt. 1 Transmit buffer interrupt.
RFINT 25	0	**Receive Frame Interrupt** Indicates that a frame has been received and that the last corresponding BD has been updated. This bit is set after the last receive buffer in a frame has been transferred via DMA.	0 No interrupt. 1 Receive frame interrupt.
RXB 24	0	**Receive Buffer Interrupt** Indicates that a receive BD has been updated that was not the last in the frame. This bit is set upon completion of a DMA transfer of a receive buffer that is not the last in the frame.	0 No interrupt. 1 Receive buffer interrupt.

Table 18-16. IEVENT Bit Descriptions (Continued)

Name	Reset	Description	Settings
MII 23	0	**MII Interrupt** Indicates that the MII has completed the requested data transfer. This bit is set if the transceiver register read/write operation controlled by the MIIDATA and MIISPEED registers is complete.	0 No interrupt. 1 MII interrupt.
— 22	0	Reserved. Write to zero for future compatibility.	
LC 21	0	**Late Collision** Indicates a collision beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC. and the remainder of the frame is discarded. In Full Duplex mode, the collision input is ignored.	0 No interrupt. 1 Late collision interrupt.
CRL 20	0	**Collision Retry Limit** Indicates a collision on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted, and transmission of the next frame commences. This interrupt can occur only in Half Duplex mode.	0 No interrupt. 1 Collision retry limit interrupt.
TFU 19	0	**Transmit FIFO Underrun** Indicates that the transmit FIFO emptied before the complete frame was transmitted. A bad CRC is appended to the frame fragment, and the remainder of the frame is discarded.	0 No interrupt. 1 Transmit FIFO underrun interrupt.
ROV 18	0	**Receiver Overrun** Indicates that the receiver is full and has dropped frame data during reception. The OV bit in the corresponding RxBD is set, indicating that the frame should be discarded.	0 No interrupt. 1 Receiver overrun interrupt.
— 17–0	0	Reserved. Write to zero for future compatibility.	

IMASK Interrupt Enable Register ENET_BASE + 0x008

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	HBEEN	BREN	BTEN	GRAEN	TFIEN	TBIEN	RFIEN	RBIEN	MIIEN	—	LCEN	CRLEN	TFUEN	ROVEN	—	
TYPE	R/W									R	R/W				R	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—				TFAC	—	RFAC	—								
TYPE	R				R/W	R	R/W	R								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IMASK controls which interrupt events can generate an actual interrupt. This register is cleared upon a hardware reset. If the corresponding bits in both the IEVENT and IMASK registers are

set, the interrupt is signalled to the SC1400 core. The interrupt signal remains asserted until a value of 1 is written to the IEVENT bit (write 1 to clear) or a 0 is written to the IMASK bit.

Additionally, IEVENT contains TFAC and RFAC so that TFINT and RFINT can be automatically cleared. With these bits, an interrupt can be generated for several frames using the event port and a timer configured to detect TFINT and/or RFINT rising edges.

Table 18-17. IMASK Bit Descriptions

Name	Reset	Description	Settings
HBEEN 31	0	**Heartbeat Error Enable** .	0 Not enabled. 1 Heartbeat error interrupt enabled.
BREN 30	0	**Babbling Receive Interrupt Enable** .	0 Not enabled. 1 Babbling receive interrupt enabled.
BTEN 29	0	**Babbling Transmitter Interrupt Enable**	0 Not enabled. 1 Babbling transmitter interrupt enabled.
GRAEN 28	0	**Graceful Stop Interrupt Enable**	0 Not enabled. 1 Graceful stop interrupt enabled.
TFIEN 27	0	**Transmit Frame Interrupt Enable**	0 Not enabled. 1 Transmit frame interrupt enabled.
TBIEN 26	0	**Transmit Buffer Interrupt Enable**	0 Not enabled. 1 Transmit buffer interrupt enabled.
RFIEN 25	0	**Receive Frame Interrupt Enable**	0 Not enabled. 1 Receive frame interrupt enabled.
RBIEN 24	0	**Receive Buffer Interrupt Enable**	0 Not enabled. 1 Receive buffer interrupt enabled.
MIIEN 23	0	**MII Interrupt Enable**	0 Not enabled. 1 MII interrupt enabled.
— 22	0	Reserved. Write to zero for future compatibility.	
LCEN 21	0	**Late Collision Enable**	0 Not enabled. 1 Late collision interrupt enabled.
CRLEN 20	0	**Collision Retry Limit Enable**	0 Not enabled. 1 Collision retry limit interrupt enabled.
TFUEN 19	0	**Transmit FIFO Underrun Enable**	0 Not enabled. 1 Transmit FIFO underrun interrupt enabled.
ROVEN 18		**Receiver Overrun Enable**	0 Not enabled. 1 Receiver overrun interrupt enabled.
— 17–12	0	Reserved. Write to zero for future compatibility.	

Table 18-17. IMASK Bit Descriptions (Continued)

Name	Reset	Description	Settings
TFAC 11	0	**Transmit Frame Interrupt Automatic Clear** Enables automatic clearing of the transmit frame interrupt after one cycle of assertion. TFAC allows an interrupt to be generated for several transmit frames using the event port and a timer configured to detect TFINT rising edges.	0 No automatic clear of TFINT. 1 Enable automatic clear of TFINT.
— 10	0	Reserved. Write to zero for future compatibility.	
RFAC 9	0	**Receive Frame Interrupt Automatic Clear** Enables automatic clearing of the receive frame interrupt after one cycle of assertion. RFAC allows an interrupt to be generated for several receive frames using the event port and a timer configured to detect RFINT rising edges.	0 No automatic clear of RFINT. 1 Enable automatic clear of RFINT.
— 8–0	0	Reserved. Write to zero for future compatibility.	

DRPC Descriptor Ring Poll Control Register ENET_BASE + 0x00C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—														RDCP	TDCP
TYPE	R														R/W	R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DRPC is a programmable control register that enables the FEC to poll the RxBD and TxBD rings continuously. Therefore, you do not have to update RDA and TDA registers when servicing the RxBD and TxBD rings. However, both RDA and TDA must be set one time initially. DRPC prevents the FEC from automatically clearing receive descriptor active (RDA[RDA]) and transmit descriptor active (TDA[TDA]).This register is cleared not only at reset but also when the ECTL[EEN] bit is cleared.

Table 18-18. DRPC Bit Descriptions

Name	Reset	Description	Settings
— 31–2	0	Reserved. Write to zero for future compatibility.	
RDCP 1	0	**Receive Descriptor Ring Continuous Poll** Enables the FEC to poll the RxBD ring continuously after RDA is set one time. RDCP prevents the FEC from automatically clearing the receive descriptor active (RDA[RDA]) bit. Set this bit with caution because it results in increased system loading.	0 Disables RxBD ring continuous polling. 1 Enables RxBD ring continuous polling.
TDCP 0	0	**Transmit Descriptor Ring Continuous Poll** Enables the FEC to poll the TxBD ring continuously after TDA is set one time. TDCP prevents the FEC from automatically clearing the transmit descriptor active (TDA[TDA]) bit. Set this bit with caution because it results in increased system loading.	0 Disables TxBD ring continuous polling. 1 Enables TxBD ring continuous polling.

RDA Receive Descriptor Active Register ENET_BASE + 0x010

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—							RDA	—							
TYPE	R							R/W	R							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 18-19. RDA Bit Descriptions

Name	Reset	Description	Settings
— 31–25	0	Reserved. Write to zero for future compatibility.	
RDA 24	0	**Receive Descriptor Active**	
— 23–0	0	Reserved. Write to zero for future compatibility.	

TDA Transmit Descriptor Active Register ENET_BASE + 0x014

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—							TDA	—							
TYPE	R							R/W	R							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 18-20. TDA Bit Descriptions

Name	Reset	Description	Settings
— 31–25	0	Reserved. Write to zero for future compatibility.	
TDA 24	0	**Transmit Descriptor Active**	
— 23–0	0	Reserved. Write to zero for future compatibility.	

ECTL Ethernet Control Register ENET_BASE + 0x024

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TAG3	TAG2	TAG1	TAG0	—	TMD	—									
TYPE	R/W				R	R/W	R									
RESET	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—														EEN	RESET
TYPE	R														R/W	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ECTL is a user-programmable register, though some fields can be altered by hardware. The ECTL register enables/disables the FEC.

Table 18-21. ECTL Bit Descriptions

Name	Reset	Description	Settings
TAG[3–0] 31–28	0b1111	**Tags 3–0** Programs and reads the TBus tag bits. This field, which is used for debug/test only, is composed of two separate 4-bit registers, tags-in and tags-out. A write from the IPBus to this field is written to tags-in. During a write cycle to any FEC register other than ECTL, the tags-in value is driven onto the TBus data bus tag field. During a read cycle, the TBus tag field bits are latched and saved in the tags-out register. When the ECTL register is read from the IPBus interface, the value from tags-out appears in the TAG[3–0] field.	
— 27	0	Reserved. Write to zero for future compatibility.	
TMD 26	0	**Test Mode**	
— 25–2	0	Reserved. Write to zero for future compatibility.	
EEN 1	0	**Ethernet Enable** Enables/disables the FEC. When this bit is set, the Ethernet can receive and transmit data. When this bit is cleared, reception immediately stops, and transmission stops after a bad CRC is appended to any frame being transmitted. The BD(s) for an aborted transmit frame are not updated following deassertion of EEN. When EEN is deasserted, the DMA controller, BD, and FIFO control logic is reset, including BD and FIFO pointers. When software writes a value of 1 to ECTL[RESET] or an AHB bus error is detected, the hardware clears EEN. The procedure for halting the FEC is described in **Section 11.4.4.3**, *Complete Halt of the Ethernet MAC,* on page 11-19.	0 Ethernet disabled. 1 Ethernet enabled.
RESET 0	0	**Ethernet Controller Reset** When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. EEN is cleared, and all other FEC registers take their reset values. Also, any transmission/reception in progress is abruptly aborted. Hardware automatically clears this bit during the reset sequence, which requires approximately eight clock cycles after RESET is written with a 1. **Note:** Before using the RESET bit to reset the FEC, shut down the FEC as described in **Section 11.4.4.3**, *Complete Halt of the Ethernet MAC,* on page 11-19	0 Normal operation. 1 Reset the FEC.

MIIDATA MII Management Frame Register ENET_BASE + 0x040

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ST		OP		PA					RA					TA	
TYPE	R/W															
RESET	Undefined															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DATA															
TYPE	R/W															
RESET	Undefined															

MIIDATA is a user-programmable register that communicates with the attached MII-compatible PHY device(s), providing read/write access to their MII registers. A write to MIIDATA causes a management frame to be sourced unless the MIISPEED register is programmed to 0. If MIISPEED = 0 and is then written to a non-zero value, an MII frame is generated with the data previously written to the MIIDATA register. MIIDATA and MIISPEED can therefore be programmed in either order if MIISPEED is currently zero.

To generate an 802.3-compliant MII management interface write frame (write to a PHY register) you must write {`01 01 PA RA 10 DATA`} to MIIDATA. In response, the control logic shifts out the data in the MIIDATA register after a preamble generated by the control state machine. The contents of MIIDATA alter as they are serially shifted, and they are unpredictable if you read them. When the write management frame operation completes, the MII configuration interrupt is generated, and the contents of MIIDATA now match the original value written.

To generate an MII management interface read frame (read a PHY register) you must write {`01 10 PHYAD REGAD 10 XXXX`} to MIIDATA (the contents of the DATA field do not matter). In response, the control logic shifts out the data in MIIDATA after a preamble generated by the control state machine. The contents of MIIDATA alter as they are serially shifted, and they are unpredictable if you read them. When the read management frame operation completes, the MII configuration interrupt is generated. The contents of MIIDATA now match the original value written, except for the DATA field whose contents are replaced by the value read from the PHY register. The contents of the TA field can also differ from the original if the PHY delays driving the MDIO pin by one or more MII management clock (MDC pin) cycles. If MIIDATA is written during frame generation, the frame contents are altered. Software should use the MIISTATUS register and/or the MII configuration interrupt to avoid writing to MIIDATA during frame generation.

Table 18-22. MIIDATA Bit Descriptions

Name	Description	Settings
ST 31–30	**Start of Frame Delimiter** These bits must be programmed to 01 for a valid MII management frame.	
OP 29–28	**Operation Code** This field must be programmed to 10 (read) or 01(write) to generate a valid MII management frame. A value of 11 produces a read frame operation, and a value of 00 produce a write frame operation, but these frames are not MII-compliant.	10 Read frame. 01 Write frame.
PA 27–23	**PHY Address** Specifies one of up to 32 attached PHY devices.	
RA 22–18	**Register Address** Selects one of up to 32 registers within the specified PHY device.	
TA 17–16	**Turnaround** Must be programmed to a value of 10 to generate a valid MII management frame.	
DATA 15–0	**Management Frame Data** Holds the data to be written to or read from a PHY register.	

MIISPEED MII Speed Control Register ENET_BASE + 0x044

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—								NPRE	MIISPEED						—
TYPE	R									R						
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MIISPEED provides control of the MII clock frequency (MDC pin), allows the preamble on the MII management frame to be dropped, and allows you to observe an internal counter used in generating the MDC clock signal (intended for manufacturing test).

Table 18-23. MIISPEED Bit Descriptions

Name	Reset	Description
— 31–8	0	Reserved. Write to zero for future compatibility.
NPRE 7	0	**No Preamble** The MII standard allows the preamble to be dropped if the attached PHY device(s) do not require it.

Table 18-23. MIISPEED Bit Descriptions (Continued)

Name	Reset	Description
MIISPEED 6–1	0	**MII Speed** Controls the frequency of the MII management interface clock (MDC) relative to the system clock. A value of 0 in this field turns off the MDC clock and leaves it in a low voltage state. Any non-zero value results in the MDC frequency of 1/(MIISPEED × 2) of the system clock frequency. MIISPEED must be programmed with a value to provide an MDC frequency of less than or equal to 2.5 MHz to comply with the IEEE MII specification. MIISPEED must be set to a non-zero value to source a read or write management frame. After the management frame completes, the MIISPEED register can optionally be set to zero to turn off the MDC. The MDC generated has a 50 percent duty cycle, except when MIISPEED is changed during operation. The change takes effect after either a rising or falling edge of MDC. If the system clock is 50 MHz, programming this register to 0x0000000A results in an MDC frequency of 50 MHz × 1/(10 × 2) = 2.5 MHz, as shown in the following table. **System Clock Frequency** / **MIISPEED Value** / **MDC Frequency** 25 MHz / 0x5 / 2.5 MHz 33 MHz / 0x7 / 2.36 MHz 40 MHz / 0x8 / 2.5 MHz 50 MHz / 0xA / 2.5 MHz
— 6–0	0	Reserved. Write to zero for future compatibility.

MIBCTL MIB Control Register ENET_BASE + 0x064

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	MIBD	MIBI	—													
TYPE	R/W		R													
RESET	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MIBCTL provides the means to control and observe the state of the management information base (MIB). Software accesses this register to disable the MIB. For example, to clear all MIB counters in RAM, disable the MIB, clear all the MIB RAM locations by writing a value of 0x00000000 to 0x200–0x2FC, and then enable the MIB.

Table 18-24. MIBCTL Bit Descriptions

Name	Reset	Description	Settings
MIBD 31	1	**MIB Disable** If this bit is set, the MIB logic halts and does not update any MIB counters.	0 Normal operation. 1 Disable the MIB.
MIBI 30	0	**MIB Idle**	
— 29–0	0	Reserved. Write to zero for future compatibility.	

RCTL Receive Control Register ENET_BASE + 0x084

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—					MAXFL										
TYPE	R					R/W										
RESET	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										FCE	BFR	PROM	MIIM	DRT	LOOP
TYPE	R										R/W					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

RCTL controls the operational mode of the receive block and should be written only when EEN = 0, that is, at initialization.

Table 18-25. RCTL Bit Descriptions

Name	Reset	Description	Settings
— 31–27	0	Reserved. Write to zero for future compatibility.	
MAXFL 26–16	0b101_1110_1110 (0x5EE)	**Maximum Frame Length** User-defined maximum frame length, where frame length is measured starting at DA and including the CRC at the end of the frame. Transmit frames longer than MAXFL cause the BABT interrupt. Receive frames longer than MAXFL cause the BABR interrupt to occur and set the LG bit in the end-of-frame BD. The recommended default to be programmed is the reset value decimal 1518 or decimal 1522 if VLAN tags are supported.	
— 15–6	0	Reserved. Write to zero for future compatibility.	

Table 18-25. RCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
FCE 5	0	**Flow Control Enable** Enables the receiver to detect pause frames. When a pause frame is detected, the transmitter stops transmitting data frames for a specified duration.	0 Normal operation. 1 Receiver detects pause frames.
BFR 4	0	**Broadcast Frame Reject** Causes frames with a destination address (DA) = FFFF_FFFF_FFFF to be rejected unless the PROM bit is set. If both BFR and PROM = 1, frames with broadcast DA are accepted and the miss (M) bit is set in the receive BD.	
PROM 3	0	**Promiscuous Mode** All frames are accepted, regardless of address matching.	
MIIM 2	0	**External Interface Mode** Selects between 7Wire Interface mode and MII mode, which applies to both the transmit and receive blocks.	0 7-Wire Interface operating mode. 1 MII operating mode.
DRT 1	0	**Disable Receive on Transmit** Disables reception of frames while transmitting, which is normally used for half duplex mode.	0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting.
LOOP 0	1	**Internal Loopback** Causes transmitted frames to be looped back internal to the device, and the transmit output signals are not asserted. The system clock is substituted for the TXCLK when LOOP is asserted. DRT must be set to zero when LOOP is asserted.	0 Normal operation. 1 Internal loopback.

RHASH Receive Hash Register ENET_BASE + 0x088

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	FCEDC	MCAST	HASH						—							
TYPE	R															
RESET	0	0	Undefined						0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RHASH provides address recognition information from the receive block about the frame being received.

Table 18-26. RHASH Bit Descriptions

Name	Reset	Description	Settings
FCEDC 31	0	**Read FCE** Gives a read-only view of the RCTL[FCE] bit.	
MCAST 30	0	**Multi-cast**	
HASH 29–24	—	**Hash** Corresponds to the hash value of the current receive frame destination address. The hash value is a six-bit field extracted from the least significant portion of the CRC register.	
— 23–0	0	Reserved. Write to zero for future compatibility.	

TCTL Transmit Control Register ENET_BASE + 0x0C4

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—											RFCP	TFCP	FDEN	HBC	GTS
TYPE	R												R/W			
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TCTL configures the transmit block.

Table 18-27. TCTL Bit Descriptions

Name	Reset	Description	Settings
— 31–5	0	Reserved. Write to zero for future compatibility.	
RFCP 4	0	**RFC Pause** Set when a full duplex flow control pause frame is received. The transmitter pauses for the duration defined in the pause frame This bit automatically clears when the pause duration is complete.	0 Normal operation. 1 Full duplex flow control pause frame received.

Table 18-27. TCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
TFCP 3	0	**TFC Pause** When this bit is set, the MAC stops transmitting data frames after the current transmission completes. The GRA interrupt in the IEVENT register is asserted. The MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFCP bit and resumes transmitting data frames. If the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC can still transmit a MAC Control PAUSE frame.	0 Normal operation. 1 Transmit a pause frame.
FDEN 2	0	**Full Duplex Enable** If set, frames are transmitted independently of carrier sense and collision inputs. This bit should be modified only when ECTL[EEN] is cleared.	0 Normal operation. 1 Enable full duplex.
HBC 1	0	**Heartbeat Control** If set, the heartbeat check is performed at the end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should be modified only when ECTL[EEN] is cleared.	0 Normal operation. 1 Perform heartbeat check.
GTS 0	0	**Graceful Transmit Stop** When this bit is set, the MAC stops transmission after the frame that is being transmitted is complete, and the GRA interrupt in the IEVENT register is asserted. If no frames are being transmitted, the GRA interrupt is asserted immediately. When transmission completes, the GTS bit is cleared to initiate a restart. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS = 1, transmission stops after the collision. The frame is transmitted again when GTS is cleared. If old frames are in the transmit FIFO, they are transmitted when GTS is reasserted. To avoid this, clear ECTL[EEN] after the GRA interrupt.	0 Normal operation. 1 Graceful transmit stop.

PADDRL Physical Address Low Register ENET_BASE + 0x0E4

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PADDRL															
TYPE	R/W															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PADDRL															
TYPE	R/W															
RESET	—															

PADDRL contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the destination address (DA) field of receive frames with an individual DA. In addition, this register is used in bytes 0–3 of the 6-byte source address field when PAUSE frames are transmitted. For a physical address of 0x112233445566, PADDRL and PADDRH should be configured as follows: PADDRL[31–0] = 0x11223344 and PADDRH[31–16] = 0x5566. Byte0 corresponds with the first byte received on the network at the start of the frame, when used by the internal address recognition logic. PADDRL is not reset and is a user-initialized register.

PADDRH Physical Address High Register ENET_BASE + 0x0E8

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PADDRH															
TYPE	R/W															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PADDRH															
TYPE	R/W															
VALUE	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

PADDRH contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the destination address (DA) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte source address field when PAUSE frames are transmitted. Bits 15–0 of PADDRH contain a constant type field (0x8808) for transmitting PAUSE frames. This register is not reset and bits 31–16 must be initialized by the user.

OPPAUSE Opcode/Pause Duration Register ENET_BASE + 0x0EC

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	OPCODE															
TYPE	R															
VALUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	PDUR															
TYPE	R/W															
RESET	—															

OPPAUSE contains the 16-bit Opcode, and 16-bit pause duration fields used in transmitting a PAUSE frame. The OPCODE field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration (PDUR) field. OPPAUSE is not reset and is a user-initialized register.

IADDR1 Descriptor Individual Address 1 ENET_BASE + 0x118

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	IADDR1															
TYPE	R/W															
RESET	—															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	IADDR1															
TYPE	R/W															
RESET	—															

IADDR1 contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for a possible match with the destination address (DA) field of receive frames with an individual DA. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32. IADDR1 is not reset and is a user-initialized register.

IADDR2 Descriptor Individual Address 2 ENET_BASE + 0x11C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	IADDR2															
TYPE	R/W															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	IADDR2															
TYPE	R/W															
RESET	—															

IADDR2 contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the destination address (DA) field of receive frames with an individual DA. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0. IADDR2 is not reset and is a user-initialized register.

GADDR1 Descriptor Group Address 1 ENET_BASE + 0x120

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GADDR1															
TYPE	R/W															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GADDR1															
TYPE	R/W															
RESET	—															

GADDR1 contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32. GADDR1 is a user-initialized register.

GADDR2 Descriptor Group Address 2 ENET_BASE + 0x124

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GADDR2															
TYPE	R/W															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GADDR2															
TYPE	R/W															
RESET	—															

GADDR2 contains the lower 32 bits of the 64 bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0. GADDR2 is a user-initialized register.

FIFOID FIFO ID Register ENET_BASE + 0x140

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	FREV								0							
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	FSIZ									0	0	0	0	0	0	0
TYPE	R															
RESET	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TWMRK FIFO Transmit Watermark Register ENET_BASE + 0x144

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0														TWMRK	
TYPE	R														R/W	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TWMRK controls the amount of data required in the transmit FIFO before a frame can be transmitted. This register helps you to minimize transmit latency (TWMRK = 0X) or allow for larger bus access latency due to contention for the system bus (TWMRK = 11). Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. You may need to modify the byte counts associated with the TWMRK field to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Table 18-28. TWMRK Bit Descriptions

Name	Reset	Description	Settings	
— 31–2	0	Reserved. Write to zero for future compatibility.		
TWMRK 1–0	0	**Transmit Watermark** Specifies the number of bytes that are written to the transmit FIFO before frame transmission can begin.	0X 10 11	64 bytes are written to the transmit FIFO. 128 bytes are written to the transmit FIFO. 192 bytes are written to the transmit FIFO.

FRBND FIFO Receive Bound Register ENET_BASE + 0x14C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0						RBND								0	
TYPE	R															
RESET	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

FRBND is a register that you can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the contents of the FRST register, to divide the available FIFO RAM appropriately between the transmit and receive data paths. The FIFO RAM address is at 0x00 and is not memory-mapped.

FRST FIFO Receive Start Register ENET_BASE + 0x150

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0						RFST								0	0
TYPE	R															
RESET	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

FRST is a user-programmable register to indicate the starting address (pointer to a 36-bit data word) of the receive FIFO. The RFST field marks the boundary between the transmit and receive FIFOs, and it provides the address of the first receive FIFO location. The transmit FIFO uses addresses from FTST to FRST. The receive FIFO uses addresses from FRST to FRBND. FRST is initialized by hardware at reset and need only be written to change the default value. The default value for the RFST field, 0x40, divides the FIFO RAM into receive and transmit FIFOs of equal size. If the value for RFST changes, it must be 0x12 or higher.

RDESST Receive Descriptor Ring Start Register ENET_BASE + 0x180

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RDESST															
TYPE	R															
RESET	—															

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RDESST														0	0
TYPE	R															
RESET	—															

RDESST provides a pointer to the start of the circular receive BD ring in external memory. This pointer must be 32-bit word aligned, but it can be made quad word aligned (evenly divisible by 16). Write a value of zero to bits 1 and 0 because hardware ignores non-zero values in these two bit positions. RDESST is not reset and is a user-initialized register.

TDESST Transmit Descriptor Ring Start Register ENET_BASE + 0x184

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TDESST															
TYPE	R															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TDESST														0	0
TYPE	R															
RESET	—															

TDESST provides a pointer to the start of the circular transmit BD ring in external memory. This pointer must be 32-bit word aligned, but it can be made quad word aligned (evenly divisible by 16). Write a value of zero to bits 1 and 0 because hardware ignores non-zero values in these two bit positions. TDESST is not reset and is a user-initialized register.

RBSZ Receive Buffer Size Register ENET_BASE + 0x188

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	—															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						RBSZ							0	0	0	0
TYPE	R					R/W							R			
RESET	—															

RBSZ specifies the maximum size of all receive buffers. Because receive frames are truncated at 2k – 1 bytes, only bits 10–4 are used. When specifying a value for this field, keep in mind that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, RBSZ must be set to RCTL[MAXFL] or larger. The value in the RBSZ field must be evenly divisible by 16, so bits 3–0 are low. To minimize descriptor fetches on the bus, the value of RBSZ should be >= 256 bytes. RBSZ does not reset and is a user-initialized register.

DMACTL DMA Control Register ENET_BASE + 0x1F4

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0								DMAREV							
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DMACTL contains a read-only field that indicates the DMA revision.

MIIGSKCFG MIIGSK Configuration Register ENET_BASE + 0x400

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0	0	0	0	0	0	0	0	0	FCTL	0	LOOPB	ECHO	0	0	RMII
TYPE	R									R/W	R	R/W		R		R/W
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MIIGSKCFG contains configuration bits for various features and modes of the MIIGSK module.

Table 18-29. MIIGSKCFG Bit Descriptions

Name	Reset	Description	Settings
— 31–7	0	Reserved. Write to zero for future compatibility.	
FCTL 6	0	**Frequency Control** Determines the clock frequency of the clock source to the MIIGSK RMII logic to support 10/100Mbps operations. This field has no effect in Pass-Through mode.	0 In RMII mode, the clock source (REF_CLOCK) for the MIIGSK RMII logic is 50 MHz to support 100 Mbps operation. 1 In RMII mode, the clock source (REF_CLOCK) for the MIIGSK RMII logic is divided by 10 (5 MHz) to support 10 Mbps operation.
— 5	0	Reserved. Write to zero for future compatibility.	

Table 18-29. MIIGSKCFG Bit Descriptions (Continued)

Name	Reset	Description	Settings
LOOPB 4	0	**Internal Loopback Mode** Causes the MIIGSK MII transmit inputs from the Ethernet controller to loop back to the MIIGSK MII receive outputs to the Ethernet controller through the MIIGSK RMII transmit/receive logic. Proper operation is guaranteed only when RMII = 1 and ECHO = 0.	0 Normal operation (default). 1 Internal loopback mode.
ECHO 3	0	**Echo Mode** Causes the MIIGSK MII receive inputs (from the MII PHY) to be looped back to the MIIGSK MII transmit outputs (to the MII PHY) without transferring through the RMII transmit/receive logic. Proper operation is guaranteed only when RMII = 0 and LOOPB = 0.	0 Normal operation (default). 1 Echo mode.
— 2–1	0	Reserved. Write to zero for future compatibility.	
RMII 0	0	**RMII Mode** Determines the type of interface to which the MIIGSK is connected.	0 Pass-through mode (used for MII 7-Wire Interface modes). 1 RMII mode.

MIIGSKEN MIIGSK Enable Register ENET_BASE + 0x408

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	0														MIIR	MIEN
TYPE	R														R/W	R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MIIGSKEN contains two bits to enable/disable MIIGSK operation and indicate when the MIIGSK is ready for use. The Ready bit ensures proper configuration of the MIIGSK.

Table 18-30. MIIGSKEN Bit Descriptions

Name	Reset	Description	Settings
— 31–2	0	Reserved. Write to zero for future compatibility.	

Table 18-30. MIIGSKEN Bit Descriptions (Continued)

Name	Reset	Description	Settings
MIIR 1	0	**MIIGSK Ready** Set when the MIIGSK is ready for use. This is a read-only bit.	0 Normal operation. 1 MIIGSK ready for use.
MIIEN 0	0	**MIIGSK Enable** Enables the MIIGSK to transmit/receive frames to/from the Ethernet controller. Clearing this bit prevents the transmission/reception of frames. This bit is cleared by default.	0 Transmission/reception of frames disabled. 1 Transmission/reception of frames enabled.

Time-Division Multiplexing (TDM) Interface 19

The TDM interface is a full-duplex serial port by which DSP devices communicate with a variety of serial devices, including industry-standard framers, codecs, other DSPs, and microprocessors. Typically, TDMs are used to transfer samples periodically. The TDM consists of independent transmitter and receiver sections with independent clock generation and frame synchronization.

Note: The number of TDM modules differs across MSC711x devices. For example, the MSC7115 device has three identical and independent TDM modules, whereas the MSC7110 device has only one. Check the device data sheet to verify how many TDMs reside on your MSC711x device, or check **Table 1-2**, *MSC711x Device-Specific Feature Comparison*, on page 1-8 of this manual.

The TDM module supports 128 channels running at up to 50 Mbps with 8- and 16-bit word size. The TDM bus connects gluelessly to most T1/E1 framers as well as to common buses such as the H.110, SCAS, and MVIP. The TDM also runs in I^2S mode. Each TDM module operates in independent or shared mode when receiving or transmitting data. In independent mode, there are different sync, clock, and data for receive and transmit. In shared sync and clock mode, the clock and the sync are shared between the transmit and receive with different receive and transmit data.

MSC711x devices can have anywhere from one to three identical and independent TDM modules, TDM 0, TDM 1, and TDM 2. The TDMs are clocked either from the TDM clocking pins, TCK and RCK, or from internal clocks generated from timer module B as determined by the settings of the CTS and RTS bits in the TDMx General Interface Register (TDMxGIR) described on **page 19-29**.

19.1 Features

TDM features are as follows:

- Independent (asynchronous) or shared (synchronous) transmit and receive sections with separate or shared internal/external clocks and frame syncs.
- TDM (network) mode operation allowing multiple devices to share the port with as many as 128 time slots.
- Single-channel operation using frame sync.
- Time-slot enable registers (receive and transmit).

- End-of-frame interrupt.
- Programmable internal clock divider.
- Programmable word length (8 or 16 bits).
- Program options for frame sync and clock generation.
- Programmable MSB or LSB first.
- TDM power-down.
- A-law/μ-law hardware conversion for 8-bit channels.
- Loopback mode.

19.2 Halting and Restarting a TDM

To halt the TDM completely, use the following procedure:

1. Follow the shut-down procedure in **Section 19.6**, *Software Programming Sequence*, on page 19-23. To verify that the TDM has shut down correctly, poll its status bits, TDMxRSR[RENS] (**page 19-49**) and TDMxTSR[TENS] (**page 19-50**).
2. Set the HLTREQ[TDMxCD] bit (**page 11-28**) to shut down the system clock to the module.

To restart the TDM, use the following procedure:

1. Clear the HLTREQ[TDMxCD] bit to re-enable the system clock to the module.
2. Follow the TDM start-up procedure in **Section 19.6**, *Software Programming Sequence*.

19.3 TDM Basics

Figure 19-1 illustrates the TDM block diagram, showing the control registers to set up the port, status registers, separate transmit and receive circuits with FIFO registers, and separate serial clock and frame sync generation for the transmit and receive sections. Multiple TDM channels are transferred sequentially in a frame. A frame sync signal is briefly asserted to identify the start of a frame. The TDM module can receive or transmit up to 128 channels at a granularity of two channels. There is also a single-channel mode. The number of receive channels is determined by the RNCF field in the TDMx Receive Frame Parameters Register (TDMxRFP). The number of transmit channels is determined by the TNCF field in the TDMx Transmit Frame Parameters Register (TDMxTFP). The size of all the channels is unified and can be 8 or 16 bits. The receive channel size is determined by the RCS field in the TDMxRFP; the transmit channel size is determined by the TCS field in the TDMxTFP.

When the TDM connects to a T1 framer, the RT1 field in the TDMx Receive Frame Parameters Register (TDMxRFP) and the TT1 field in the TDMx Transmit Frame Parameters Register (TDMxTFP) should be set. Also, the number of channels in the RCS and TCS fields should be set to 24. The T1 frame contains 193 bits (24 channels of 8 bits each), and the first bit of the frame is

an unused Frame Alignment bit. At the T1 received frame, the Frame Alignment bit is removed and does not transfer to the data registers. At the transmit T1 frame, the Frame Alignment bit is not driven out.

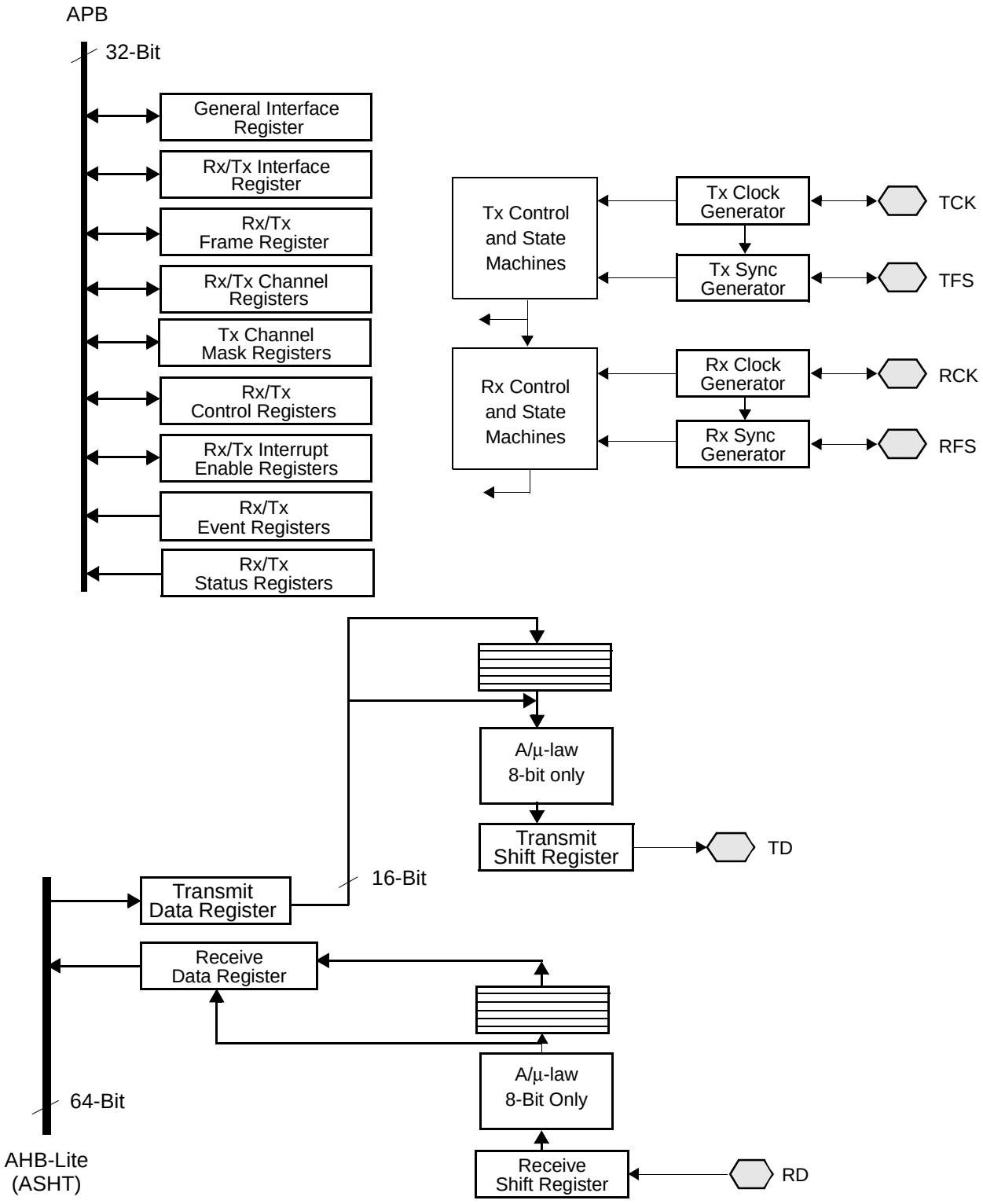


Figure 19-1. TDM Block Diagram

Figure 19-2 shows an example TDM interface. The receive frame contains two 8-bit channels. The transmit frame contains four 8-bit channels. **Figure 19-3** shows an example T1 frame. Note the first bit of the frame is not used by the receive and transmit TDM.

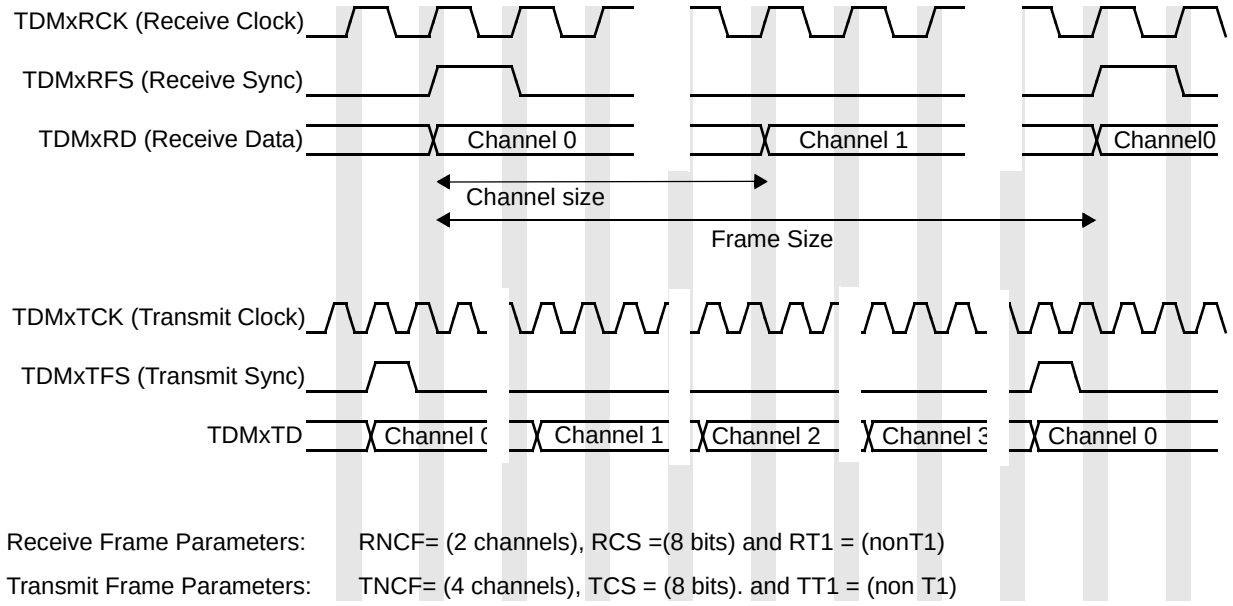


Figure 19-2. TDM Frames

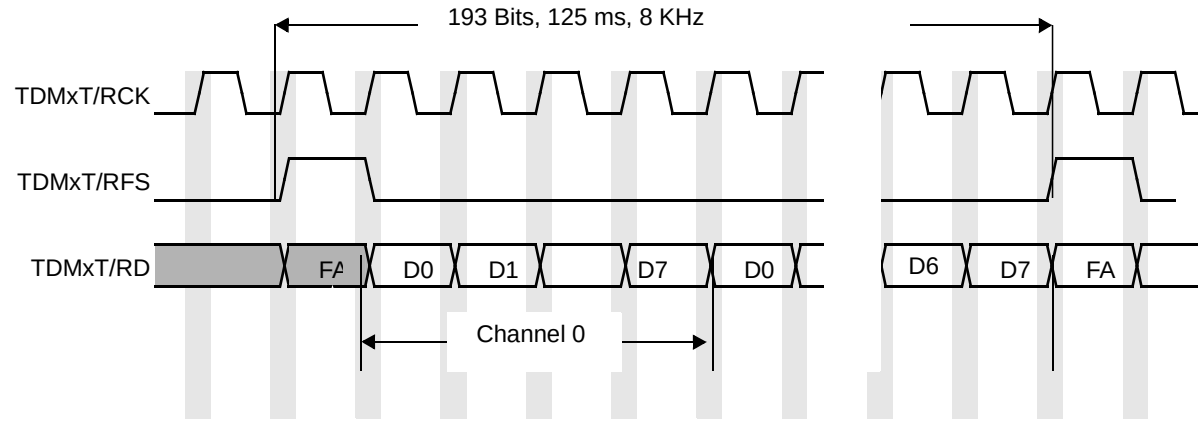


Figure 19-3. T1 Frame

19.3.1 Common Signals for the TDM Modules

The sync and clock signals can be independent or shared among the TDM modules. When the CTS bit of the TDMx General Interface Register is equal to 1, the TDM modules share sync and clock signals. In this mode, the common signals connect to the following signal lines:

- The transmit sync/frame sync connects to TFS (receive and transmit share the same sync signal for all TDMs).
- The transmit clock/frame clock connects to TCK (receive and transmit share the same clock signal for all TDMs).

The configuration registers should be identical for the TDM modules that share signals. **Figure 19-4** illustrates a common receive sync, receive clock, transmit sync, and transmit clock for TDM 0 and TDM 1.

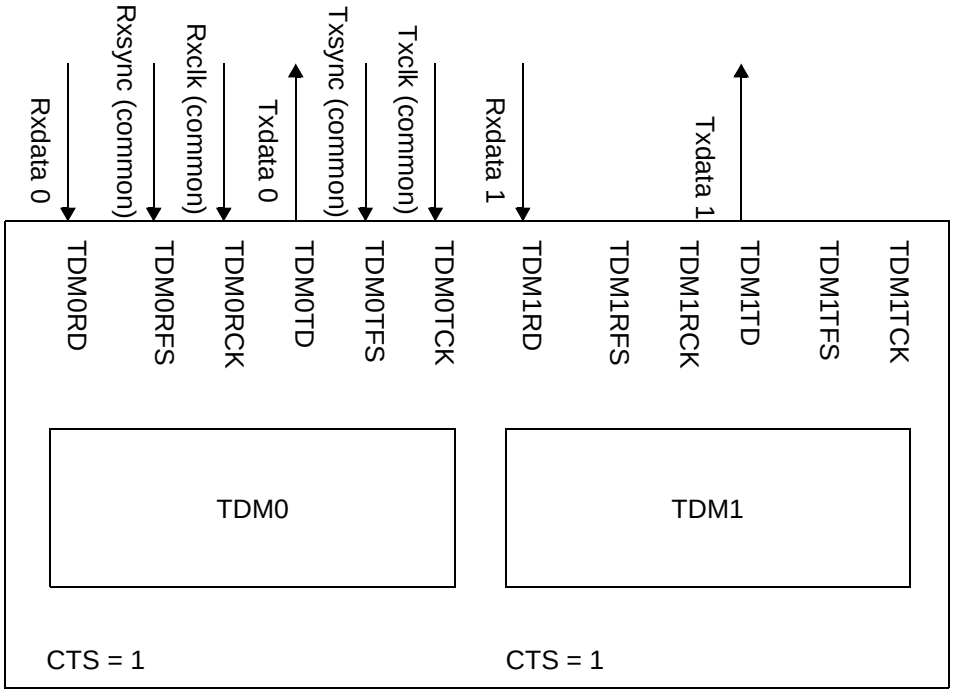


Figure 19-4. TDM Modules Shared Mode

19.3.2 Clocks

The TDM uses the following three clocks, illustrated in **Figure 19-5** and **Figure 19-6:**

- *Bit clock*. Serially clocks the data bits in and out of the TDM port.
- *Word clock*. Counts the number of data bits per word (8 or 16 bits).
- *Frame clock*. Counts the number of words in a frame.

The bit clock is visible on the TCK, for independent transmit or shared modes and RCK, for independent receive clock operation pins. The word clock is an internal clock used to determine

when transmission of an 8 or 16 bit word has completed. The word clock then clocks the frame clock, which marks the beginning of each frame. The frame clock can be viewed on the TFS and RFS pins; the maximum allowable rate for an external clock source is 50 MHz. The bit clock can be received from a TDM clock pin, or it can be generated from the peripheral clock passed through a divider, as illustrated in **Figure 19-5** and **Figure 19-6**.

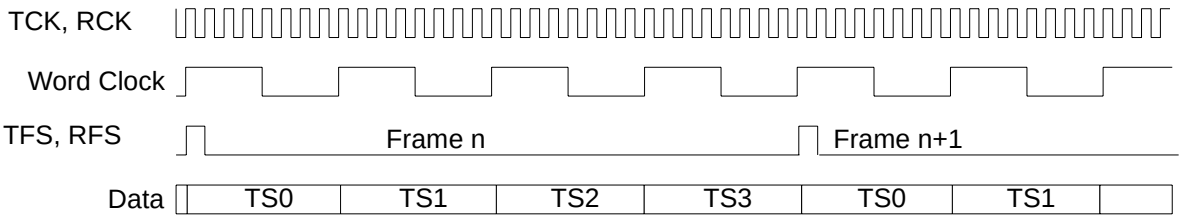


Figure 19-5. TDM Clocking (8-Bit Words, 4 Time Slots/Frame)

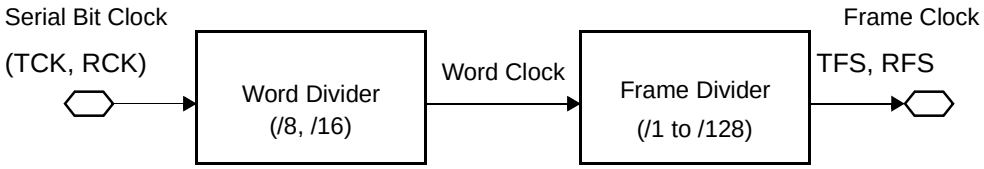


Figure 19-6. TDM Clock Generation

Table 19-1. Clock Summary

Clock	Source	Description
TCK	Internal/ External	Transmit data changes on the falling or rising edge of this clock. The control register can invert the clock, if required.
RCK	Internal/ External	Receive data is captured on the rising or falling edge of this clock. The control register can invert the clock, if required.
TFS	Internal/ External	Transmit frames begin with this signal. See the discussion of TDMxRIR on **page 19-31**. The control register can invert the clock, if required.
RFS	Internal/ External	Receive frames begin with this signal. See the discussion of TDMxRIR on **page 19-31**. The control register can invert the clock, if required.

19.3.3 TDM Clock and Frame Sync Generation

Data clock and frame sync signals are generated internally for the TDM from an internal clock source or are obtained from an external clock from a TDM pin. The maximum allowable rate for an external clock source is one half of the AMBA host bus clock, or 100 MHz/2 = 50 MHz. If a timer internally generates the clock, the TDM derives bit clock and frame sync signals from this internal timer. The sources change according to the receive and transmit sharing as well as common TDM definitions. A programmable frame rate divider and a word length divider are used for frame rate sync signal generation.

Figure 19-7 shows the clock generator block for the transmit section. The receive section contains an equivalent circuit for its frame sync generator.

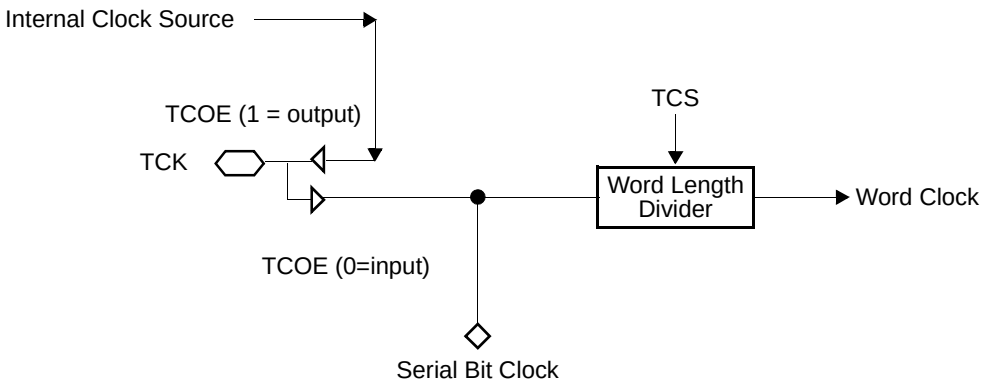


Figure 19-7. TDM Transmit Clock Generator Block Diagram

Figure 19-8 demonstrates the frame sync generator for the transmit section. When internally generated, both receive and transmit frame sync are generated from the word clock and are defined by the TDMx Transmit Interface Register (TDMxTIR) and the TDMx Transmit Frame Parameters Register (TDMxTFP). The receive section contains an equivalent circuit for its frame sync generator.

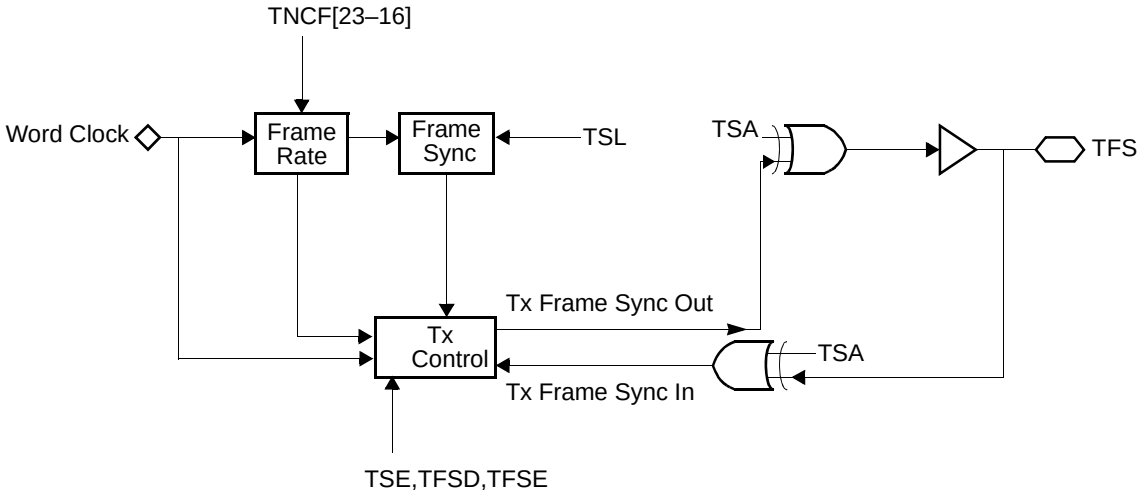


Figure 19-8. TDM Transmit Frame Sync Generator Block Diagram

19.3.4 TDM Configurations

Figure 19-9 and **Figure 19-10** illustrate the main TDM configurations. These pins support all transmit and receive functions as shown. Some operating modes do not require the use of all six pins, so you can use them as GPIO pins, if desired. The GPIO interface is a separate module that alternatively controls the function and state of the I/O pins. See **Chapter 24**, *General-Purpose Input/Output (GPIO)* for alternate functions of the I/O pins defined here

Figure 19-9 shows two MSC711x devices that connect point-to-point. Data transmits from the device on the left to the device on the right or *vice versa*.

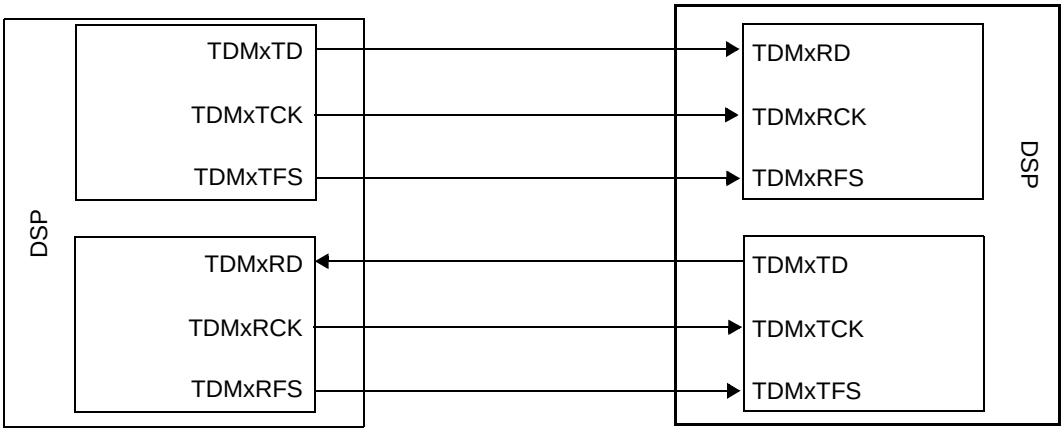


Figure 19-9. TDM Point-to-Point Configuration

Figure 19-10 depicts a TDM point to multi-point configuration. Multiple MSC711x devices connect on the same TDM bus, which connects to the network through a framer.

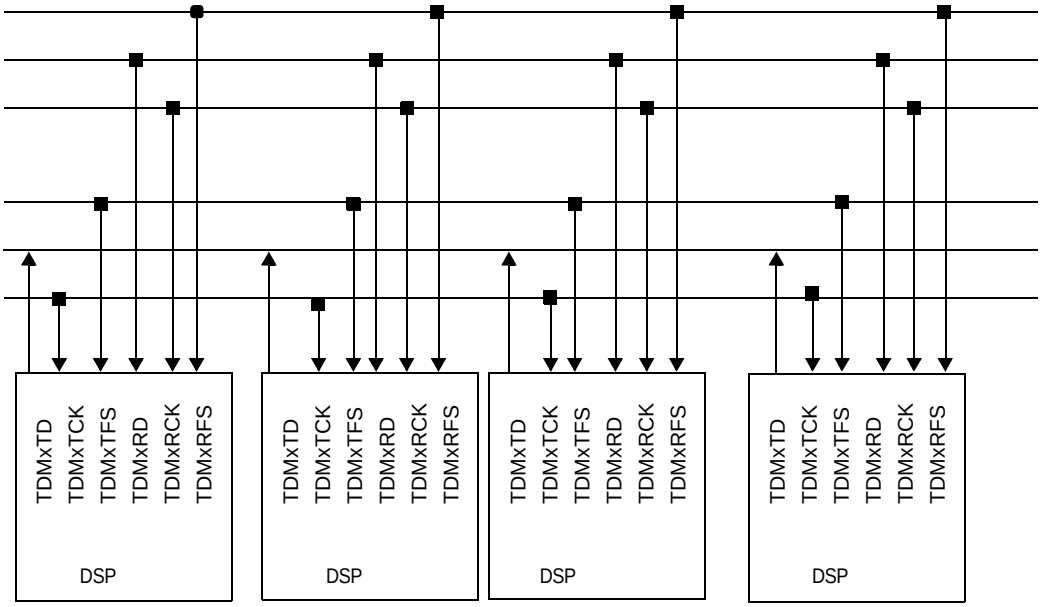


Figure 19-10. TDM Network Configuration

Figure 19-11 depicts an application in which all the TDM modules share the sync and the clock. Therefore, each TDM module supports one active link. Six receive links and six transmit links connect to three MSC711x devices—for devices with two TDM modules or two MSC711x devices with three TDM modules.

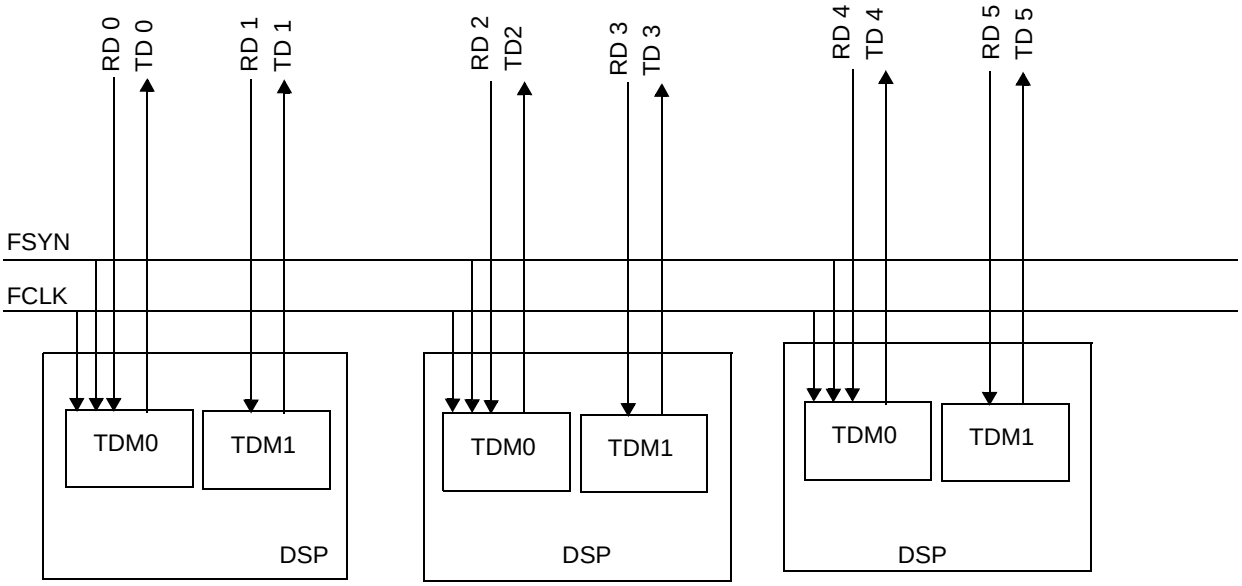


Figure 19-11. TDM Shared Network Configuration

19.4 TDM Serial Interface

This section covers issues related to the serial interface, such as how to configure the frame sync and how to control the data order of the bits in the channel word.

19.4.1 Sync Out Configuration

The TFS and RFS signals are programmed as either an input or output by writing a value 1 to the TSO bit in the Transmit Interface Register (TDMxTIR) or writing 1 to the RSO bit in the Receive Interface Register (TDMxRIR). When the TSO, RSO bit value is equal to 1, the TFS and RFS is internally generated. When the TDM modules share a sync and clock signals (the CTS bit is set), the TDMxTIR[TSO] bits should be equal for all the TDM modules. They determine whether the sync arrives externally or is generated by the TDM0 transmitter. Additionally, the edge on which the TFS and RFS signals are driven out is controlled by writing to the TSOE bit in the Transmit Interface Register (TDMxTIR) or writing to the RSOE bit in the Receive Interface Register (TDMxRIR). When these bits are set to 0, the corresponding TFS or RFS are driven out on the rising edge. When these bits are set to 1, the corresponding TFS or RFS are driven out on the falling edge. **Figure 19-12** illustrates how the TDMxRIR[RSL] and TDMxTIR[TSL] bits affect sync out generation.

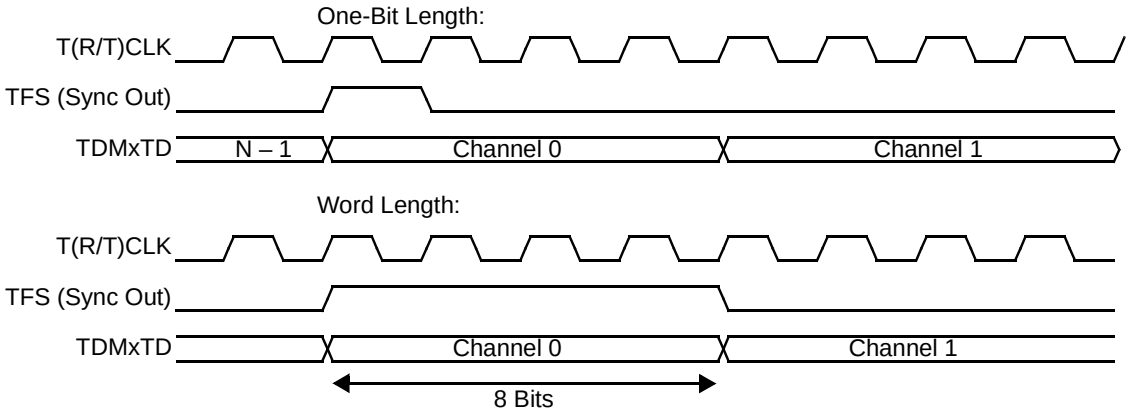


Figure 19-12. Sync Length Selection

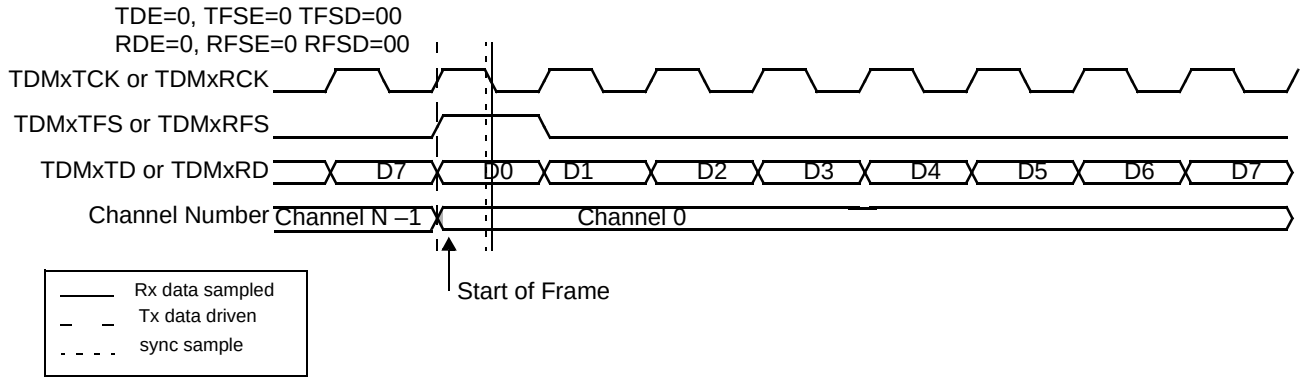
19.4.2 Sync In Configuration

The TFS and RFS signals are inputs that identify the beginning of a frame. **Figure 19-14** and **Figure 19-15** illustrate the relationship between the data, sync, and clock for various configurations. The receive data and frame sync are sampled with the rising or falling edge of the receive clock. The transmit frame sync, TFS, is sampled with the rising or falling edge of the transmit clock. The transmit data drives out at the rising or falling edge of the transmit clock. Determine the value of the (R/T)DE, (R/T)FSE, and the (R/T)FSD bits, as follows:

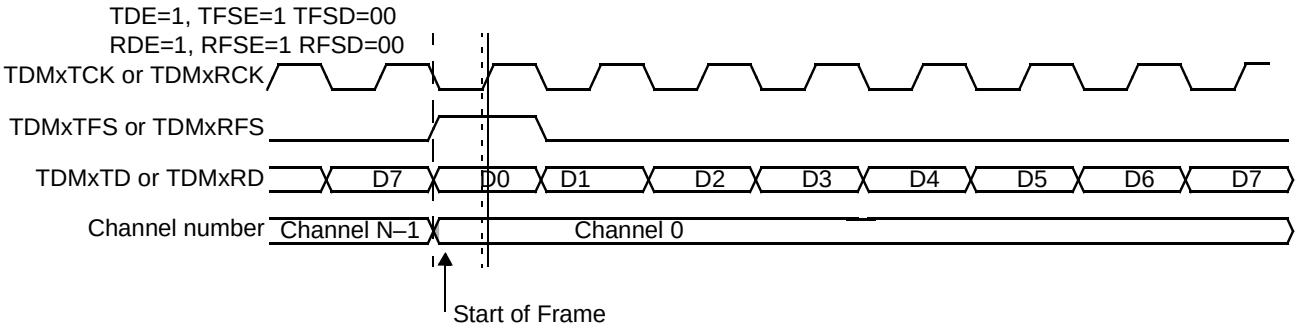
1. Determine the edge on which the frame sync is driven (or changes) and whether it is internally generated.
2. Determine the edge on which the data is driven (or changes) and whether it is internally generated.
3. If the (R/T)DE = (R/T)FSE, set the (R/T)FSD to the number of clock cycles before the first channel's first bit of data arrives after the first part of the Frame Sync edge. Otherwise, set the (R/T)FSD to the number of clock cycles before the first channel's first bit of data arrives after the first clock edge that the Frame Sync is active and stable.

When configured as inputs, the TDM frame syncs require only that the frame sync be inactive for two cycles before becoming active for a minimum of one cycle. Thus, the TDM can support frame syncs of any length that meet these requirements.

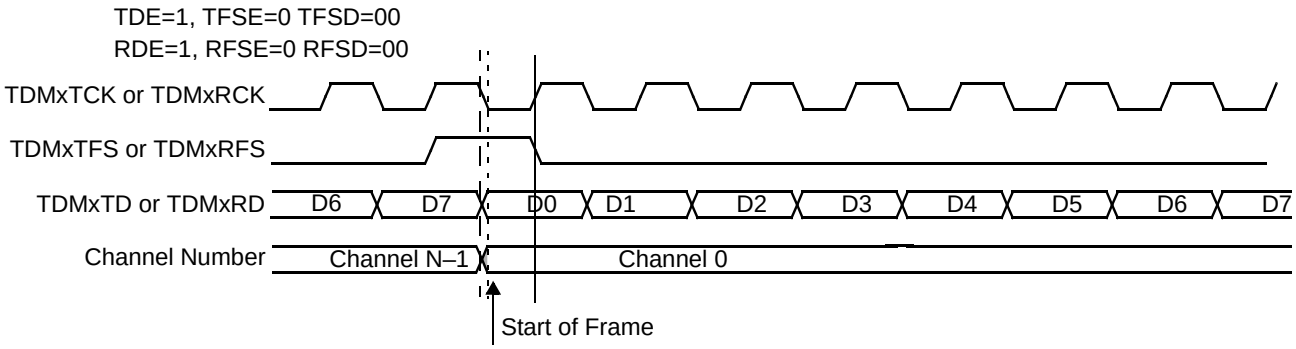
Data and Sync Change on the Rising Edge (No Sync Delay)



Data and Sync Change on the Falling Edge (No Sync Delay)



Data Changes on the Falling Edge, Sync on the Rising Edge (No Sync Delay)



Data Changes on the Rising Edge, Sync on the Falling Edge (No Sync Delay)

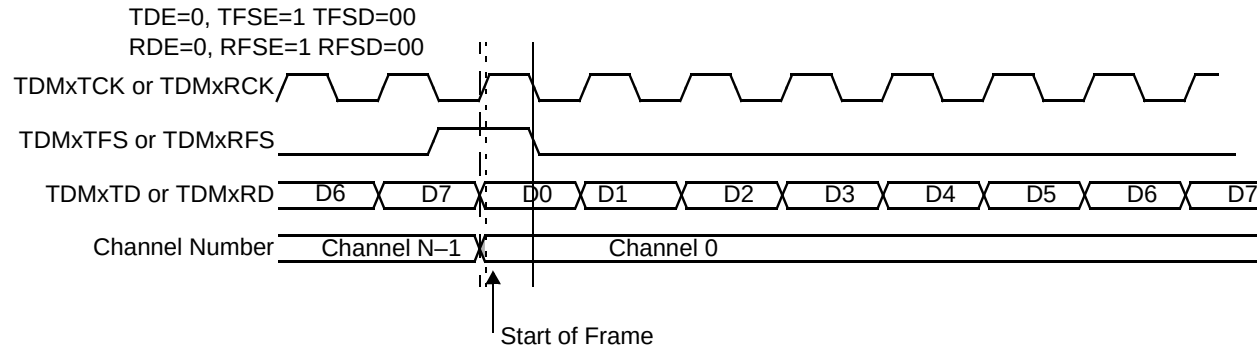


Figure 19-13. Frame Sync Configurations Without Sync Delays

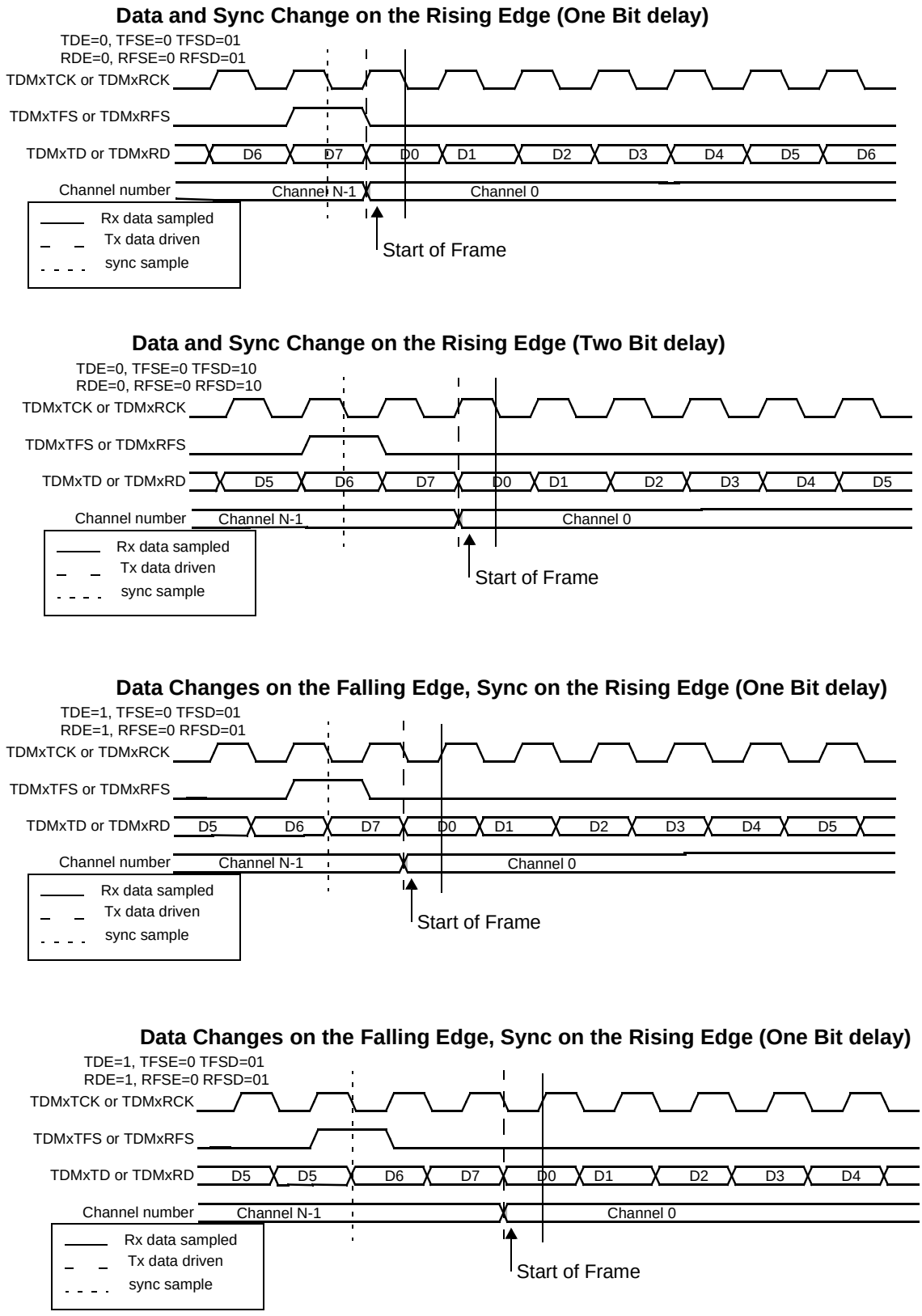


Figure 19-14. Frame Sync Configurations With Sync Delays

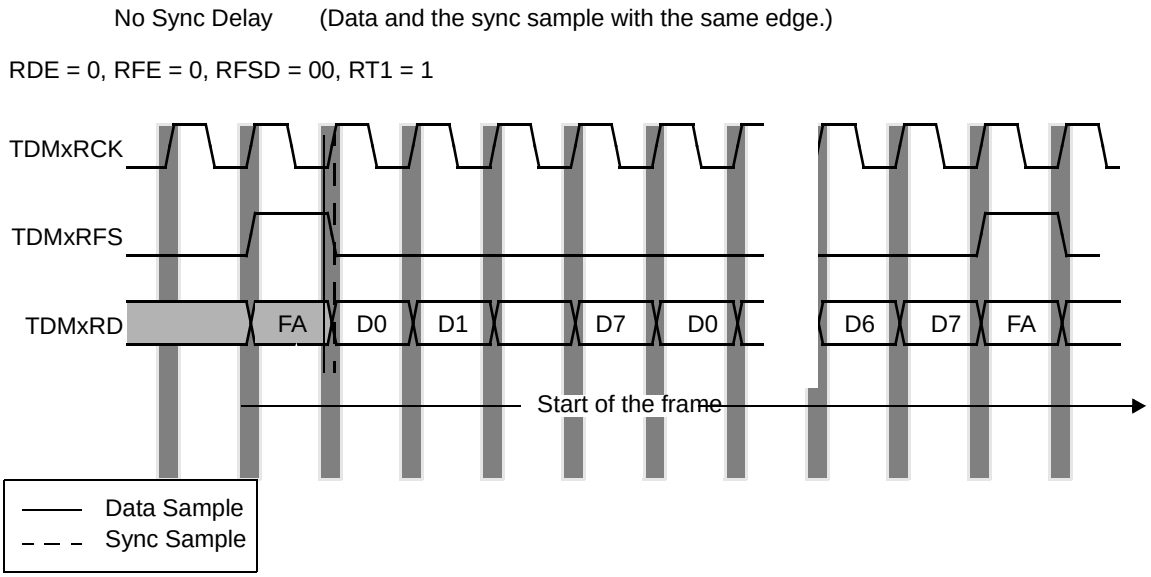


Figure 19-15. Frame Sync Configuration At T1 mode

Figure 19-16 illustrates how the TDMxRIR[RSA] and the TDMxTIR[TSA] bits control the polarity of the receive sync and the transmit sync signals.

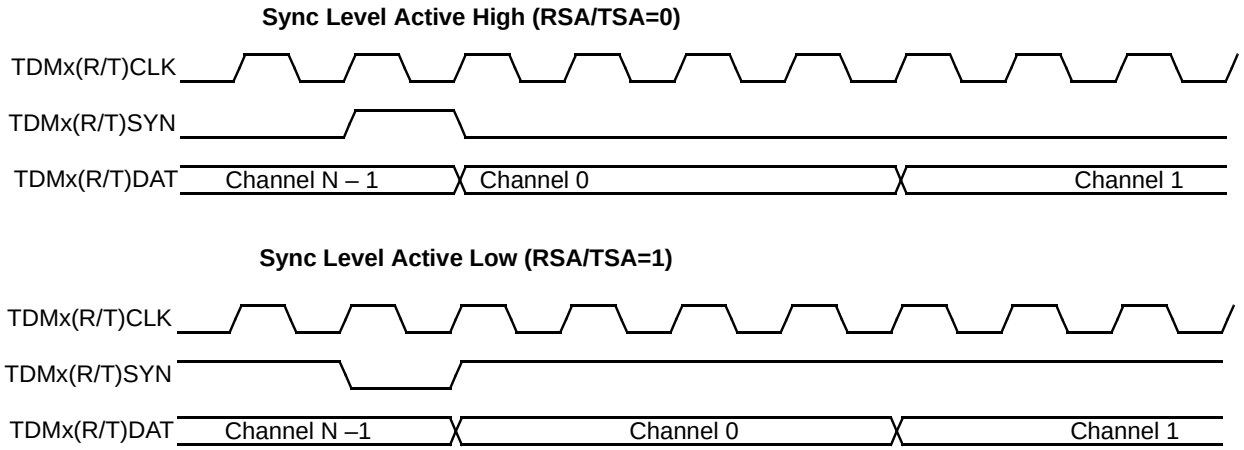


Figure 19-16. Frame Sync Polarity

19.4.3 Serial Interface Synchronization

The receive and transmit of each TDM frame is identified by a frame sync signal that is asserted at the beginning of every frame. The frame sync synchronization is necessary when more than one device drives the bus. **Figure 19-17** shows the state diagram of the frame sync synchronization.

The details of the state diagram are as follows:

- *HUNT* (0b00). A sync event is constantly sought. As soon the sync event is detected, the state machine changes to a WAIT state. During the HUNT state, data is neither received nor transmitted.
- *WAIT* (0b01). At least one sync is detected. The next sync event is accepted after one TDM frame. If the sync appears in the correct position, the state changes to the PRESYNC state (0b11). If the sync does not appear, the state returns to the HUNT state. During the WAIT state, data is neither received nor transmitted.
- *PRESYNC* (0b11). Two sync events are detected and the distance between the syncs is one TDM frame. If the sync event is recognized early, the state returns to the WAIT state. Otherwise, the machine transfers to the SYNC state at the last bit of the TDM frame. During PRESYNC state, data is neither received nor transmitted.
- *SYNC* (0b10). At least one sync event appears exactly where it is expected. This state is maintained as long as the sync event continues to appear where expected. If a sync is missed or a sync event is recognized early, the state changes to the HUNT state (0b00). During the SYNC state, data is both received and transmitted.

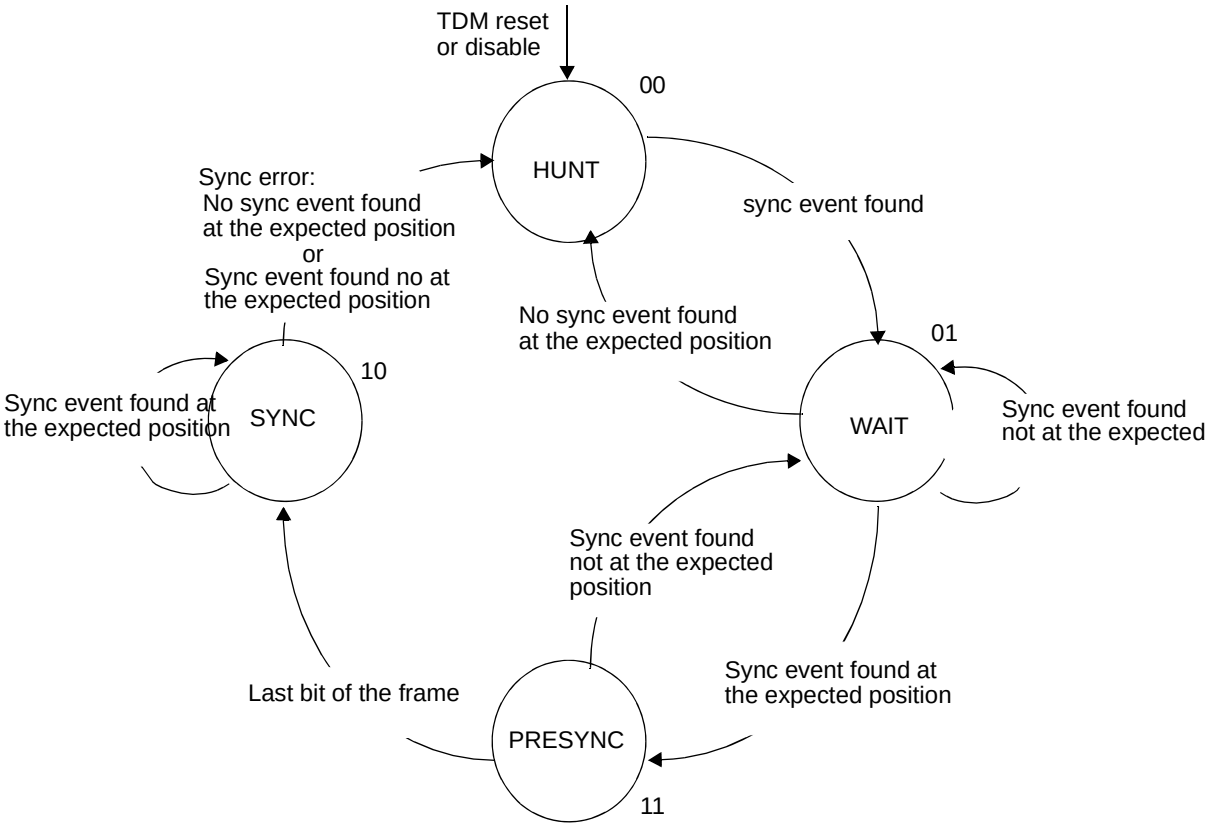


Figure 19-17. Frame Sync Synchronization State Diagram

The TDM receiver synchronizes on the receive frame sync (RFS). The state of the receive frame sync synchronization is indicated by the TDMxRSR[RSSS] field. During the HUNT, WAIT, and PRESYNC states, the received data is not transferred to the Receive Data Register. When the receive sync synchronization is lost, the state transfers from SYNC to HUNT (the TDMxRER[RSE] bit is asserted). If the TDMxREIR[RSEEE] bit is also set, a receive error interrupt is generated.

The transmit frame sync synchronization state is indicated by the TDMxTSR[TSSS] field. During the HUNT, WAIT, and PRESYNC states, new data is not driven out. If the Transmit Always Out (TDMxTIR[TAO]) field is set, then all 1s are driven out until the frame sync synchronization state returns to the SYNC state. If the TDMxTIR[TAO] bit is clear, data is not driven out and TD is tri-stated. When the transmit sync synchronization is lost, the TDMxTER[TSE] bit is asserted. If the TDMxTIER[TSEEE] bit is also set, a transmit error interrupt is generated.

The frame sync synchronization state can identify different problems. In the initial design stages, the frame sync summarization state indicates whether the TDM programming matches the actual TDM stream. During operation, the synchronization state and the error interrupts may indicate errors in the TDM module signal processing.

19.4.4 Reverse Data Order

Figure 19-18 illustrates how the bit order of the stored data relates to the bit order of the receive or the transmit data. The TDMxRIR[RRDO] bit defines how the receive channel data is stored in memory. If TDMxRIR[RRDO] is clear, the first bit of the received channel data is stored as the least significant bit. The TDMxTIR[TRDO] bit selects the transmit data bits order. If TDMxTIR[TRDO] is clear, the least significant bit of the memory is transmitted as the first transmit data.

19.5 Transmit and Receive Operation

The TDM operates with the transmit and receive operations running either independently or shared, as illustrated in **Figure 19-19**. When the RTS bit is cleared in the TDMx General Interface Register, the receive and the transmit are independent as illustrated on the left side of **Figure 19-19**. When the RTS bit is set in the TDMx General Interface Register, the transmit and receive share the Frame Sync (FSYN) and the Frame Clock (FCLK) signals with the transmitter being the input or the output, as illustrated on the right side of **Figure 19-19**. I

When the CTS bit is set in the TDMx General Interface Register, TDM0 is the source for the other TDM module’s clock and frame sync. Therefore the clock and frame sync source definitions must be the same for all TDMs that are sharing. When the CTS bit is clear, each TDM controls its own clock and frame sync configurations.

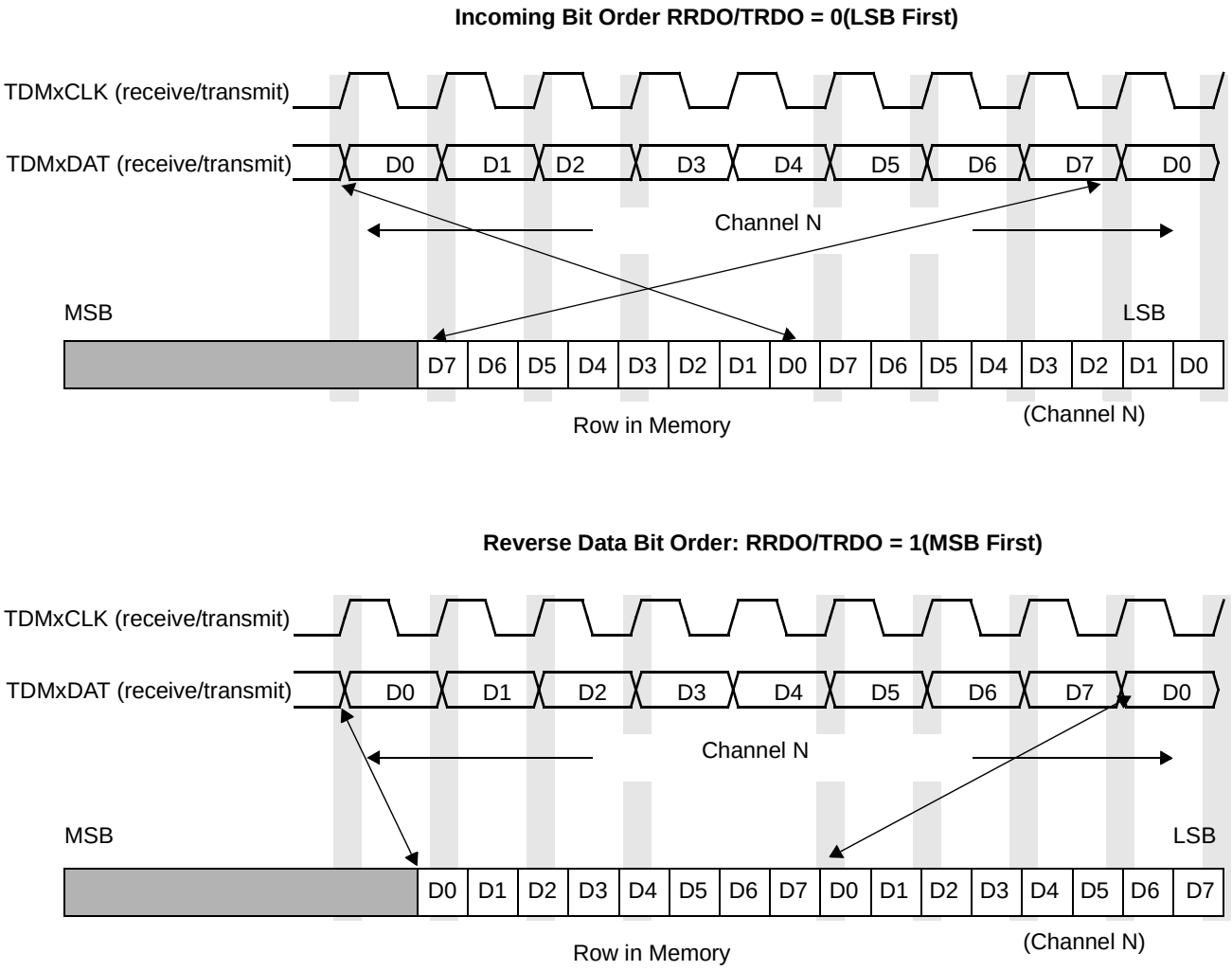
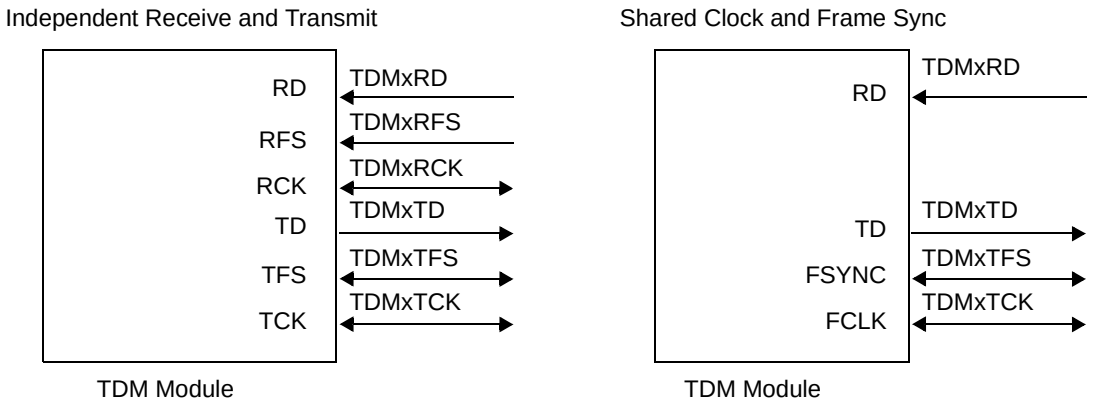


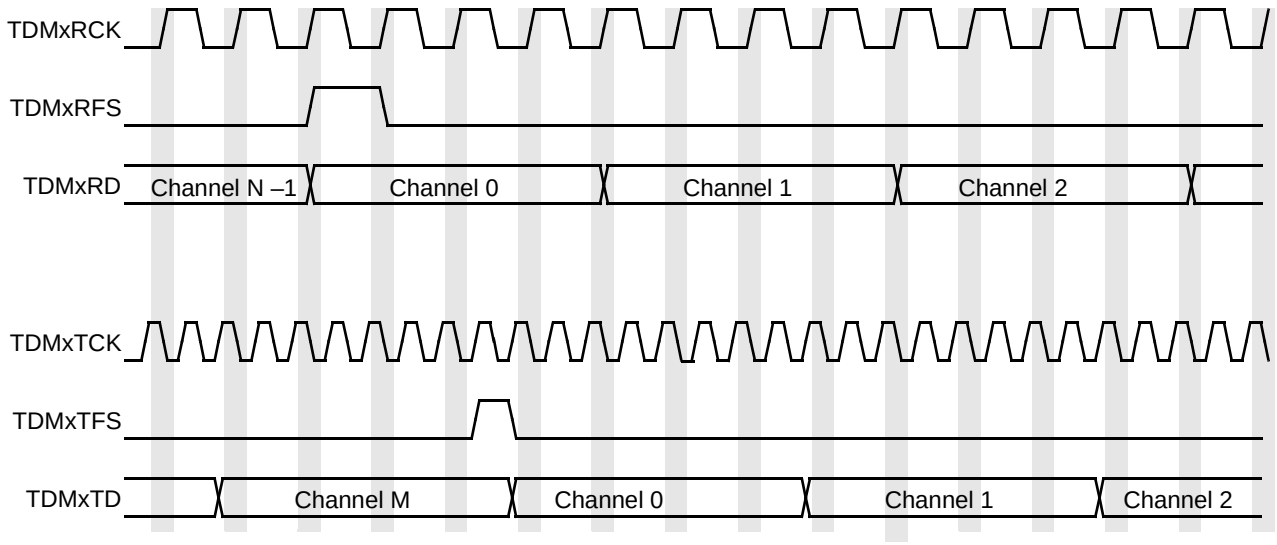
Figure 19-18. Reverse Bit Order



x Defines the TDM number.
FSYNC (frame sync) specifies that the receiver and transmitter share the same sync.
FCLK (frame clock) specifies that the receiver and transmitter share the same clock.

Figure 19-19. TDM Module Modes

Figure 19-20 shows the TDM interface when the receive and transmit are totally independent (TDMxGIR[RTS] = 0). The TDMxRCK is not synchronized with the TDMxTCK. They differ by sync location relative to the beginning of the frame and the number of bits in a word.



X The TDM number.
N The number of channels in the receive TDM frame.
M The number of channels in the transmit TDM frame.

Figure 19-20. Receive and Transmit Totally Independent

19.5.1 TDM Multi-Channel (Network) Mode

Figure 19-21 and **Figure 19-22** illustrate sample timing of TDM multi-channel (network) mode transfers. The numbered circles and arrows in the figures identify discussion notes in **Table 19-2** and **Table 19-3.**

19.5.1.1 Tx Channel Mask Register

If all bits of the TCMA registers are cleared, the TDM transmitter continues to operate solely by the TCEN registers. The TCMA registers place data on specific channels. Therefore, TCMA should be set only on disabled channels or channels with the corresponding TCEN bit already cleared. When TCEN is cleared, the TCMA is set to 1 in the desired channel bit location for data to be discarded by the TDM. Therefore, memory structures can keep channels open that are physically closed on the TDM or are transferred to the TDM via a DMA transfer and the TDM discards the data. If the TCMA is set on a channel that is enabled, the TDM discards the data for that channel and instead transmits all 1s.

When a channel is disabled in this way, the TDM ignores the time slot for that channel, and no data is transferred to the Transmit Shift Register. **Figure 19-21** illustrates the transmitter timing, using TCM registers for an 8-bit word and a continuous clock with disabled FIFO six words per frame sync in TDM Network mode**.** Explanatory notes for the transmit portion of the figure are

provided in **Table 19-2.** In this example there are only three transmit interrupts per frame instead of six as in the previous example where the TCM register is not used.

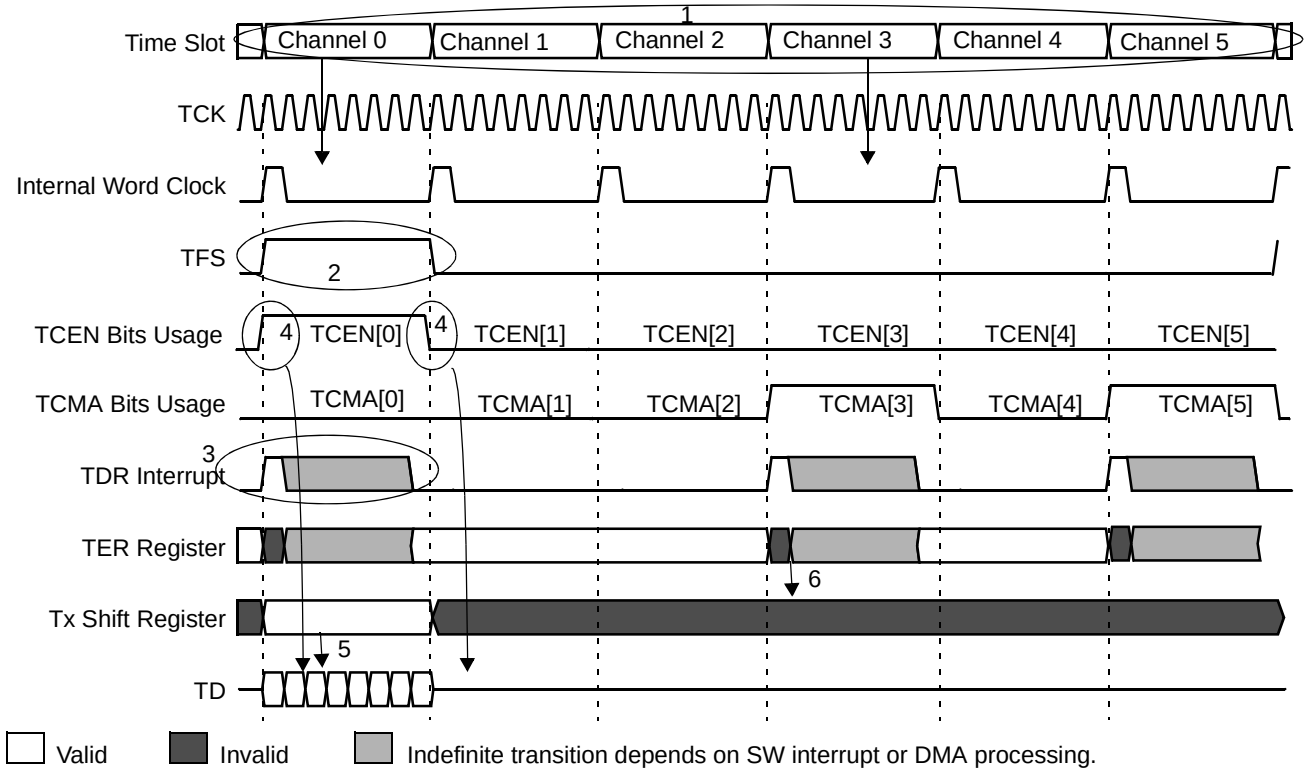


Figure 19-21. TDM Network Mode Transmit Timing with Mask Register

Table 19-2. Transmit Timing with the Mask Register

Note	Source Signal	Destination Signal	Description
1	—	—	Example of a 6 time-slot frames transmitting in Channels 0,3, and 5.
2	TFS	—	Example of a word length frame sync and standard timing. Frame timing begins with the rising edge of TFS.
3	—	TDR Interrupt	For enabled time slots, this flag is set at the beginning of each word to indicate that TDR data has been used and software should supply another data word. If the transmit interrupt is enabled, the processor is interrupted to request the data.
4	Tx Data Register	Tx Shift Register	On each word clock boundary, there is an indication of what to transmit on the next time slot. If the TCEN register bit is cleared to *zero* for the next time slot, the TD pin is either tri-stated or drives previous time-slot data and the time slot is ignored. When the TCEN bit is set to *one* for the next time-slot, the contents of the TDMxTDR are transferred to the TxSR register and this data is shifted out. If the TDMxTDR register was not written in the previous time slot, the previous data is reused. When neither of these registers were written in the previous time slot (where TCEN = 1), the TUE status bit is set and the hardware operates as if the TDMxTDR has been written. The TD pin is enabled and the contents of TDMxTDR[TDREG] are transmitted again.

Table 19-2. Transmit Timing with the Mask Register (Continued)

Note	Source Signal	Destination Signal	Description
5	Tx shift Register	TD Pin	On active time slots, the TxSR register contents are shifted out on the TD pin, one bit per rising edge of TCK. On inactive time slots, the TD pin is tri-stated so it can be driven by another device.
6	Tx Data Register	Tx Shift Register	On disabled time slots (TCEN = 0) and the channel mask is enabled (TCMA = 1), The contents of the TDMxTDR are discarded and the TDR status flag is set as if the data were shifted out.

19.5.1.2 Rx Channel Enable Register

Use of TDMxRCEN[0–3] (**page 19-39**) can reduce interrupt overhead. If all bits of a TDMxRCEN register are set, the TDM receiver receives all channels in the frame. TDMxRCEN automatically discards data from selected channels by writing the RCEN with 0 in the selected channel bit location. No data is transferred from the Receive Data Shift Register on these channels, no status flags change, and no interrupts are generated.

Figure 19-22 shows receiver timing using TDMxRCEN for an 8-bit word with a continuous clock, a disabled FIFO, and six words per frame sync. There are only three receive interrupts per frame instead of the six as in the previous example in which the RCEN register is not used. This process is explained in **Table 19-3.**

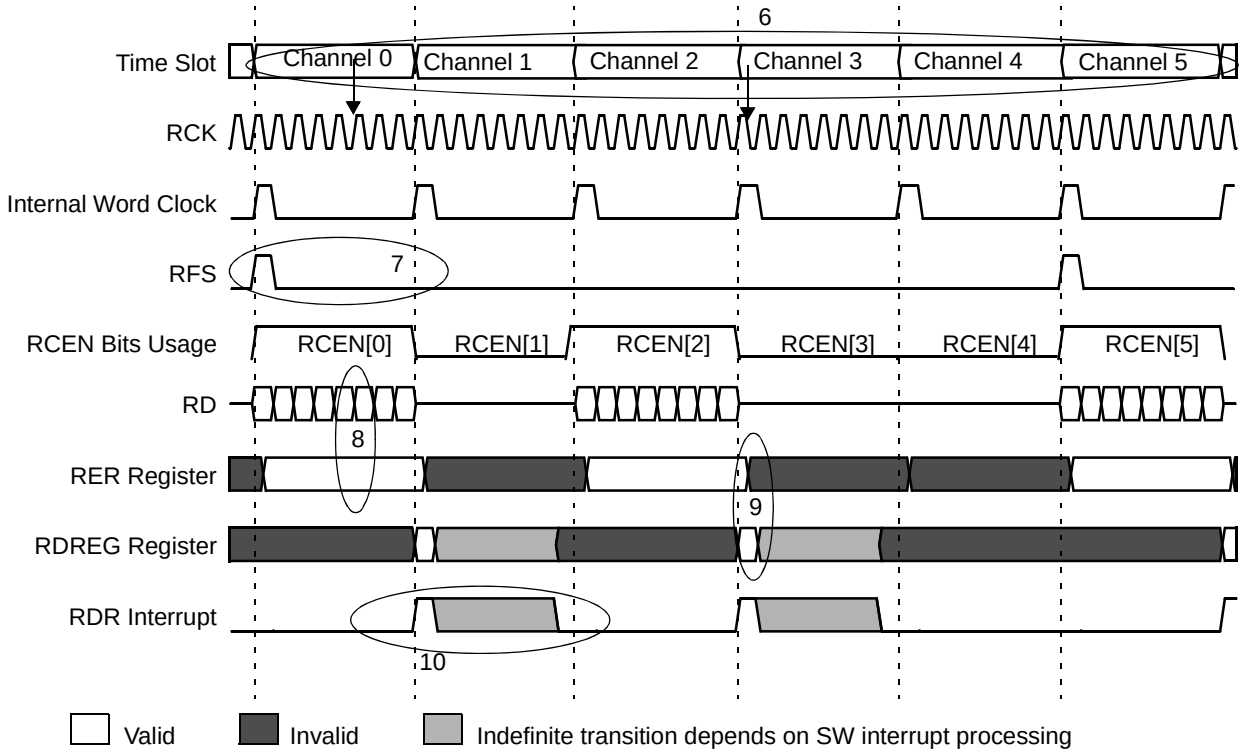


Figure 19-22. TDM Network Mode Receive Timing with Enable Register

Table 19-3. Receive Timing with Mask Register

Note	Source Signal	Destination Signal	Description
6	—	—	Example of a frame with six time slots and receive data from channels 0, 2, and 5. The receive hardware obtains data only on the RD pin when the RCEN bit is set to one.
7	RFS	—	Example with bit length frame sync and standard timing. Frame timing begins with the rising edge of RFS.
8	RD	Rx Shift Register	Data on the RD pin is sampled on the falling edge of RCK and shifted into the Rx shift register.
9	Rx shift Register	Rx Data Register	At the word clock, the data in the Rx shift register is transferred to the Rx register for enabled time slots.
10	RDR Interrupt	—	This flag is set for each word clock, or time slot, where the RCEN bit is set, indicating data is available to be processed. The software must keep track of the time slots so it knows which data it is processing. When the receive interrupts are enabled, an interrupt is generated when the status flag is set. Software reads TDMxRDR to clear the interrupt.

19.5.2 Data Structures

The TDM receives and transmits four types of data. There is 16- bit data unencoded, 8-bit data unencoded, 8 bit data A-law encoded and 8 bit data µ-law encoded. The A/µ-law conversion is performed according to the ITU-T recommendation G.711. The channel size and encoding is identical for all channels in the frame.

When the TDMxRFP[RCS] field (**page 19-36**) is set to receive an 8-bit A-law encoded channel, the 8-bit pulse code modulation (PCM) compressed sample is converted into a 13-bit PCM linear sample padded with three zeros on the right to create a 16 bit data structure. When the TDMxRFP[RCS] field is set to receive an 8-bit µ-law encoded channel, the 8-bit PCM compressed sample is converted into a 14-bit PCM linear sample padded with two zeros on the right to create a 16-bit data structure.

When the TDMxTFP[TCS] field (**page 19-38**) is set to transmit an 8-bit A-law encoded channel, the 16 bits to be transmitted (13-bit linear sample padded with three zeros on the right) are converted into an 8-bit PCM A-law compressed sample, which is transmitted out of the TDM. When the TDMxTFP[TCS] field is set to transmit an 8-bit µ-law encoded channel, the 16 bits to be transmitted (14-bit linear sample padded with three zeros on the right) are converted into a 8-bit PCM µ-law compressed sample, which is transmitted out of the TDM.

19.5.3 FIFO Configuration

The TDM has a FIFO in both the Tx and Rx paths. After reset, the FIFO is bypassed and disabled. To enable the FIFO, set the Rx FIFO Enable bit in the TDMxRIR register (**page 19-31**) and the Tx FIFO Enable bit in the TDMxTIR register (**page 19-34**). The FIFO watermark registers configure when the FIFO sets event bits and/or interrupts. The TDMxRIR[RFWM] and TDMxTIR[TFWM] bits are set according to how much data is held in the FIFO. The FIFO has a depth of four lines. The Rx FIFO is cleared when the TDM Rx is disabled, and the Tx FIFO is cleared when the TDM Tx is disabled.

The width of each line in the FIFO is set to either 1 sample of 16/8-bit data or 64 bits packed with eight 8-bit samples or four 16-bit samples, as shown in **Figure 19-23**. Data width must be accessed in one read/write transaction, as configured via the TDMxRIR[RWEN] and TDMxTIR[TWEN] bits. In Wide FIFO mode, the channels are packed from the lowest address to the highest address in the RDREG. When a new frame comes in, the Rx shift register stops filling the current line in the FIFO and moves to the next line in the FIFO. On the transmit side, Wide FIFO mode reads data sequentially from the lowest address to the highest address. If a frame ends in the middle of a FIFO line, the remainder of the line is bogus data that is not transmitted. The receiver does not put data in the remainder of the line and should be ignored on reads.

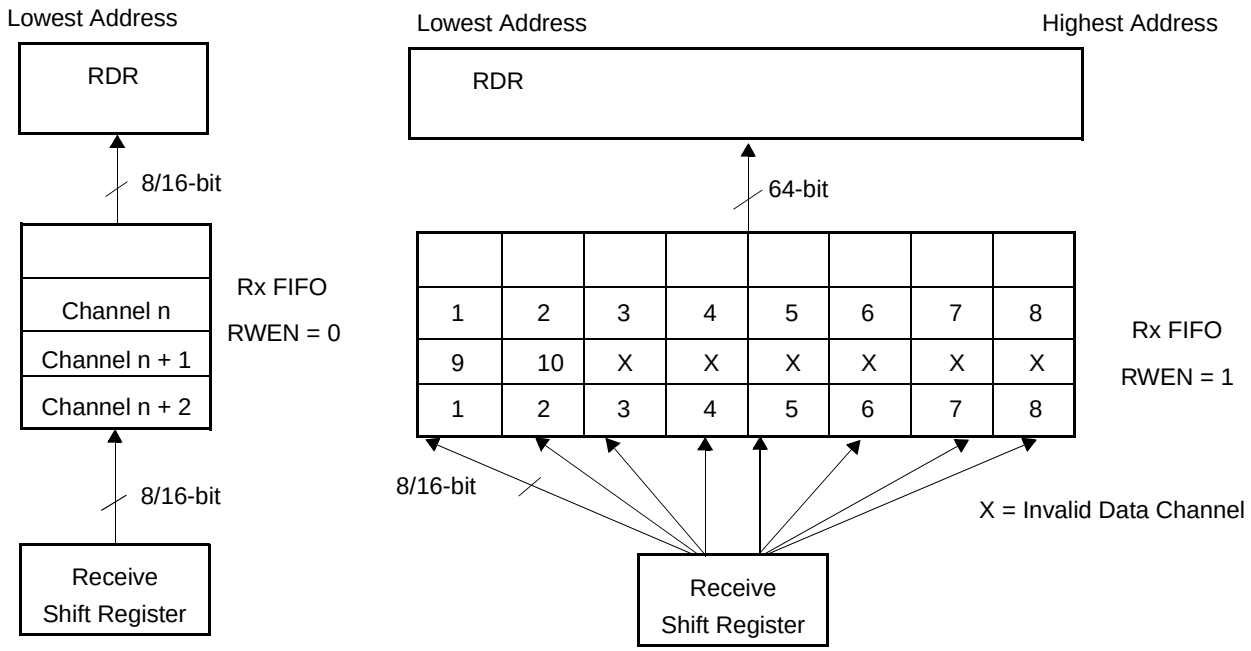


Figure 19-23. TDM Rx FIFO Using RWEN Bit with Ten 8-Bit Channels

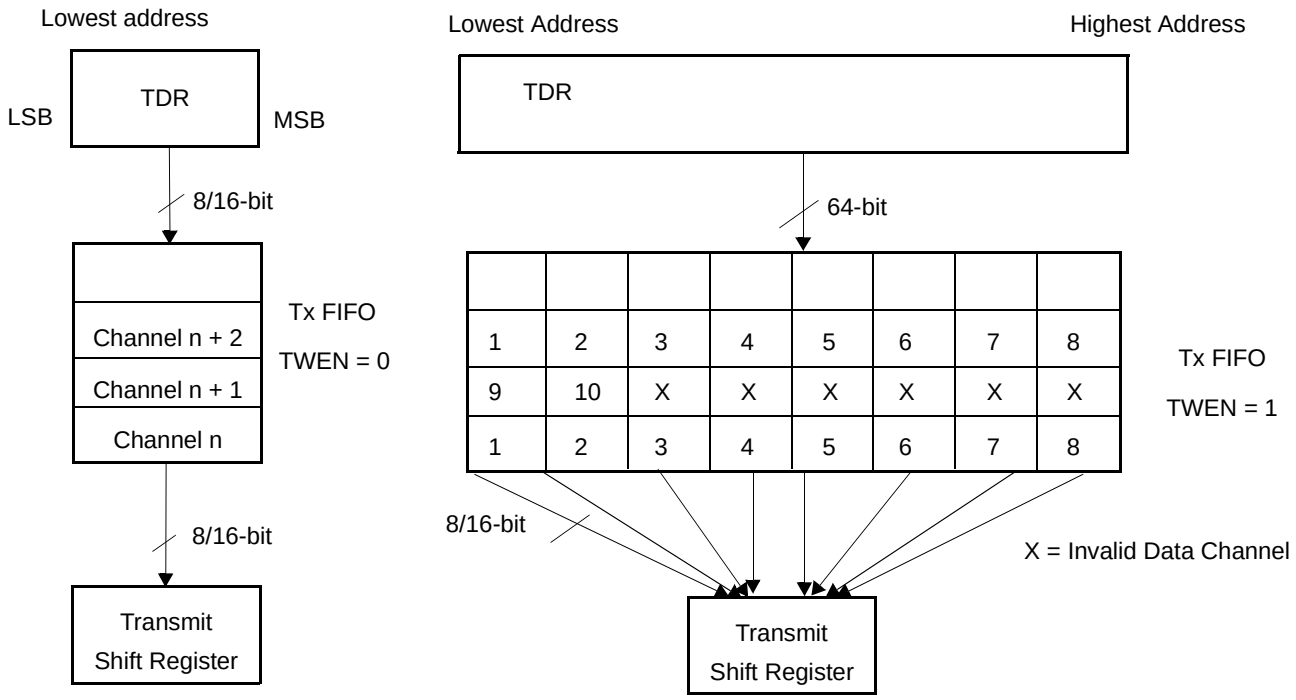


Figure 19-24. TDM Tx FIFO Using TWEN Bit with 10 8-Bit Channels

19.5.4 DMA Configuration

The DMA controller must be configured to transfer the appropriate amount of data for the TDMxTDR or TDMxRDR on the basis of channel size, encoding, and whether Wide FIFO mode is used. The DMA interface is enabled by setting the TDMxRIR[RDMA] bit (**page 19-31**) or the TDMxTIR[TDMA] bit (**page 19-34**). **Table 19-4** describes the conditions when the DMA controller is requested to service the TDM data registers. When the DMA controller asserts the DMA acknowledge signal, the requests deassert if the event flags clear (no more data to service). See **Chapter 8**, *DMA Controller* for details on DMA operation and channel priorities.

Table 19-4. DMA Request Channels for the TDM

Source of DMA Request	Event in TDM Causing Request	Description
TDM0 Rx	Rx Data Ready or Rx FIFO Full	If the Rx FIFO is not enabled, the Rx Data Ready flag causes the Rx DMA request to go active. If the Rx FIFO is enabled, the Rx FIFO Full flag causes the Rx DMA request to go active.
TDM0 Tx	Tx Data Empty or Tx FIFO Empty	If the Tx FIFO is not enabled, the Tx Data Register Empty flag causes the Tx DMA request to go active. If the Tx FIFO is enabled, the Tx FIFO Empty flag causes the DMA request to go active.

Table 19-4. DMA Request Channels for the TDM (Continued)

Source of DMA Request	Event in TDM Causing Request	Description
TDM1 Rx	Rx Data Ready or Rx FIFO Full	If the Rx FIFO is not enabled, the Rx data Ready flag causes the Rx DMA request to go active. If the Rx FIFO is enabled, the Rx FIFO Full flag causes the Rx DMA request to go active.
TDM1 Tx	Tx Data Empty or Tx FIFO Empty	If the Tx FIFO is not enabled, the Tx Data Register Empty flag causes the Tx DMA request to go active. If the Tx FIFO is enabled, the Tx FIFO Empty flag causes the DMA request to go active.

The DMA controller can access the TDM only when the DMA controller TCDx-2[NBYTES] field equals the number of bytes in one line of the FIFO (1, 2, or 8 bytes). That is, the Rx TCDx-1[SSIZE] and the Tx TCDx-1[DSIZE] must be equal to 000, 001, or 011, corresponding to 1, 2, or 8 bytes, respectively. Additionally, the TCDx-1[DSIZE] for Rx and TCDx-1[SSIZE] for Tx must be equal to or less than TCDx-2[NBYTES] to meet the DMA requirements on SSIZE and DSIZE.

19.5.5 Interrupts

Normal and error receiver interrupts can occur when receive interrupts are enabled via the TDMx Receive Interrupt Enable Register (TDMxRIER)(**page 19-43**). The bits in this register enable interrupts based on the event occurring. If more than one interrupt is enabled, an interrupt occurs for either event. To clear the interrupt, a one is written to the corresponding bit of the interrupt in the TDMx Receive Event Register (TDMxRER) (**page 19-45**) or the Receive Data Register (TDMxRDREG) (**page 19-51**) is read, depending on the type of interrupt.

Normal and error transmitter interrupts can occur when receive interrupts are enabled via the TDMx Transmit Interrupt Enable Register (TDMxTIER)(**page 19-44**). The bits in this register enable interrupts based on the event occurring. If more than one interrupt is enabled, an interrupt occurs for either event. To clear the interrupt, a one is written to the corresponding bit of the interrupt in the TDMx Transmit Event Register (TDMxTER)(**page 19-47**) or the Transmit Data Register (TDMxTDR)(**page 19-52**) is written, depending on the type of interrupt.

19.6 Software Programming Sequence

This section describes the programming procedures for the following tasks:

- Initializing a TDM for a shared operation starting on the same frame.
- Initializing a TDM for a non-shared operation.
- Dynamically configuring a channel for both shared and non-shared operations.
- Configuring the TDM for an I^2S operation.
- Powering down a TDM.

- Handling synchronization errors.
- Processing transmit normal and transmit error interrupts.

19.6.1 Initialization for a Shared Operation

The sequence for initializing the TDM for a shared operation (TDMxGIR[RTS] = 1) in which both the Tx and Rx machines must start on the same frame is as follows:

1. Bring the MSC711x device out of reset.
2. Program the following TDM configuration registers:
   - TDMxGIR (**page 19-29**)
   - TDMxRIR (**page 19-31**)
   - TDMxTIR (**page 19-34**)
   - TDMxRFP (**page 19-36**)
   - TDMxTFP (**page 19-37**)
3. Program all Channel Enable Registers and Transmit Channel Mask Registers as disabled:
   - TDMxRCEN[0–3] (**page 19-39**)
   - TDMxTCEN[0–3] (**page 19-40**)
   - TDMxTCMA[0–3] (**page 19-41**)

   These are the only registers whose values that can be changed during TDM operation. These registers are sampled at the beginning of the frame to determine how the current frame is to be processed.
4. Enable the TDM Rx and Tx by setting the TDMxRCR[REN] (**page 19-42**) and TDMxTCR[TEN] bits (**page 19-42**).
5. Wait until TSR[TENS] and RSR[RENS] are set.
6. If the TDM is serviced by DMA transfers, immediately use a software-started DMA channel to write data to the Tx data registers. This step must be completed before two Tx frame syncs arrive after the TDM is enabled.

   If the TDM is serviced by interrupts, immediately write data to the Tx data register(s). This step must be completed before two Tx frame syncs arrive after the TDM is enabled.
7. To enable channels, follow the steps outlined in **19.6.3**, *Dynamic Channel Configuration for a Shared Operation*.
8. All other writes and all reads are initiated either by an interrupt or a DMA request, depending on the configuration.

This procedure ensures proper operation of the TDM by changing the control bits before the TDM is enabled. Except for the channel enable and mask registers, these control bits should not be changed during TDM operation.

19.6.2 Initialization for a Non-Shared Operation

The sequence for initializing the TDM for a non-shared operation is as follows:

1. Bring the device out of reset.
2. Program the TDM configuration registers.
3. Enable the TDM Rx and Tx by setting the TDMxRCR[REN] (**page 19-42**) and TDMxTCR[TEN] bits (**page 19-42**).
4. Wait until TSR[TENS] and RSR[RENS] are set.
5. If the TDM is serviced by DMA transfers, immediately use a software-started DMA channel to write data to the Tx data registers. This step must be completed before two Tx frame syncs arrive after the TDM is enabled.

   If the TDM is serviced by interrupts, immediately write data to the Tx data register(s). This step must be completed before two Tx frame syncs arrive after the TDM is enabled.
6. All other writes and all reads are initiated either by an interrupt or a DMA request, depending on the configuration.

This procedure also applies for a shared operation when the Tx and Rx machines are not required to start on the same frame. The procedure ensures proper operation of the TDM by changing the configuration bit values before the TDM is enabled. The configuration bits values should not be changed during TDM operation.

19.6.3 Dynamic Channel Configuration for a Shared Operation

During a shared operation (TDMxGIR[RTS] = 1), the sequence for changing channels during TDM operation is as follows:

1. Enable the channel enable update interrupt (in TDMxRIER[RCEUE], **page 19-43**).
2. Read the values of TDMxRCEN0 into one of the SC1400 core data registers.
3. Immediately write this same value back out to the same register, TDMxRCEN0.

   This creates an interrupt for a receive on a frame boundary without modifying the values in this register.
4. Wait for the receive channel enable update interrupt, TDMxRER[RCEU]. Upon receiving this interrupt, turn ON/OFF the appropriate channels by writing the new values to the Transmit and Receive Channel Enable Registers and the Transmit Channel Mask Registers:
   - TDMxRCEN[0–3] (**page 19-39**)
   - TDMxTCEN[0–3] (**page 19-40**)
   - TDMxTCMA[0–3] (**page 19-41**)

   This step should be completed within half a frame after the interrupt is received.

5. Wait until the next receive channel enable update interrupt occurs. Read all enable and mask registers to verify that the writes in Step 4 completed in time.
6. Update any data structures for the TDM in memory.
7. Disable the channel enable update interrupt.
8. Process data as normal.

When there are only one or two physical channels per frame (that is, TDMxRFP[RNCF] = 0x0 or 0x1), the procedure for non-shared dynamic channel configuration must be used instead.

19.6.4 Dynamic Channel Configuration for a Non-Shared Operation

The sequence for changing channels during TDM operation is as follows:

1. Enable the channel enable update interrupt (in TDMxRIER, **page 19-43**, for a receive channel or in TDMxTIER, **page 19-44**, for a transmit channel).
2. Read the value of TDMxRCEN0 and TDMxTCEN0 into two of the SC1400 core data registers.
3. Immediately write these same values back out to the same registers, TDMxRCEN0 and TDMxTCEN0.

   This creates interrupts for receive and transmit on the frame boundaries without modifying the values in these registers.
4. Wait for the transmit channel enable update interrupt, TDMxRER[RCEU]. Upon receiving this interrupt, turn ON/OFF the appropriate channels by writing the new values to the Transmit Channel Enable Registers and the Transmit Channel Mask Registers:
   - TDMxTCEN[0–3] (**page 19-40**)
   - TDMxTCMA[0–3] (**page 19-41**)

   This step should be completed within half a frame after the interrupt is received.
5. Wait for the receive channel enable update interrupt, TDMxRER[RCEU]. Upon receiving this interrupt, turn ON/OFF the appropriate channels by writing the new values to the Receive Channel Enable Registers (TDMxRCEN[0–3]).

   This step should be completed within half a frame after the interrupt is received.

Note: Steps 4 and 5 can be completed in either order.

6. Wait until the next transmit channel enable update interrupt occurs. Read all enable and mask registers to verify that the writes in Step 4 completed in time.
7. Wait until the next receive channel enable update occurs. Read all enable registers to verify that the writes in Step 5 completed in time.

Note: Steps 6 and 7 can be completed in either order.

8. Update any data structures for the TDM in memory.
9. Disable the channel enable update interrupt.
10. Process data as normal.

19.6.5 Configuring a TDM for I²S Operation

To prepare a TDM for I2S operation, configure the TDMxTIR and TDMxRIR as follows:

1. Set the frame sync out length equal to the channel width:
   a. TDMxTIR[TSL] = 1.
   b. TDMxRIR[RSL] = 1.
2. Set the frame sync to active high on the right channel and active low on the left channel:
   a. TDMxTIR[TSA] = 0.
   b. TDMxRIR[RSA] = 0.
3. Set the data and synch change on the falling edge only (one bit delay):
   a. TDMxTIR[TDE] = 1, TFSE = 1, TFSD = 1.
   b. TDMxRIR[RDE] = 1, RFSE = 1, RFSD = 01.
4. Set as reversed data order, with the MSB first:
   a. TDMxTIR[TRDO] = 1.
   b. TDMxRIR[RRDO] = 1.

19.6.6 Powering Down a TDM

Shut off the TDM as follows:

1. Disable all channels by clearing all the Tx and Rx channel enable bits.
2. Read all data out of the receive data registers.
3. Disable the TDM Rx and Tx by clearing the TDMxRCR[REN] and TDMxTCR[TEN] bits.
4. Clear all TDMxRER and TDMxTER interrupt/status bits by writing 1s to the registers.
5. Verify that the TDM is disabled by reading 0 for both TDMxRSR[RENS] and TDMxTSR[TENS].

19.6.7 Handling Synchronization Errors

Handle a synchronization error as follows:

1. Read all data out of the receive data registers.
2. If you want to clear out the data in the Tx FIFO/ Tx data registers, disable the TDM Tx by clearing TDMxTCR[TEN] bit (**page 19-42**). Then write data to the Tx Data register(s) (**page 19-52**).

   Assuming the TDM synchronizes again, all other writes and all reads are initiated either by an interrupt or a DMA request, depending on the configuration.

19.7 TDM Programming Model

The handshake between the TDM module and the SC1400 core occurs via a set of registers, interrupts, and DMA requests. The TDM registers are mapped into the APB and AHB address space. For information on TDMx_BASE, refer to the memory map in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. The TDM is reset by either a AHB reset or a APB reset. When the TDM resets, all control registers are reset and the TDM is disabled.

19.7.1 TDM APB Interface Registers

This section discusses the TDM registers in the APB memory map. Refer to **Section 19.7.2**, *TDM AHB Interface Registers*, on page 19-51 for details on the registers in the AHB memory map. The APB registers are divided into three categories:

- *Configuration registers* to set the operation modes and provide indications for all channels. They are set before the TDM is enabled and should not be changed while the TDM is active.
- *Control registers* to set the channel specific parameters individually for each channel and to set the threshold pointers. These registers can be changed during operation. The channel enable and mask registers are sampled at the beginning of the frame.
- *Status registers*, which are read-only registers that can be accessed at any time.

The APB memory-mapped TDM registers are listed as follows, along with the number of the page where each is discussed:

- TDM configuration registers:
  - TDMx General Interface Register (TDMxGIR), **page 19-29**.
  - TDMx Receive Interface Register (TDMxRIR), **page 19-31**.
  - TDMx Transmit Interface Register (TDMxTIR), **page 19-34**.
  - TDMx Receive Frame Parameters Register (TDMxRFP), **page 19-36**.
  - TDMx Transmit Frame Parameters Register (TDMxTFP), **page 19-37**.
  - TDMx Receive Channel Enable Registers 0–3, TDMxRCEN[0–3], **page 19-39**.
- TDM control registers:
  - TDMx Transmit Channel Enable Registers 0–3, TDMxTCEN[0–3], **page 19-40**.

- TDMx Transmit Channel Mask Registers 0–3, TDMxTCMA[0–3], **page 19-41**.
- TDMx Receive Control Register (TDMxRCR), **page 19-42**.
- TDMx Transmit Control Register (TDMxTCR), **page 19-42**.
- TDMx Receive Interrupt Enable Register (TDMxRIER), **page 19-43**.
- TDMx Transmit Interrupt Enable Register (TDMxTIER), **page 19-44**.

■ TDM status registers:

- TDMx Receive Event Register (TDMxRER), **page 19-45**.
- TDMx Transmit Event Register (TDMxTER), **page 19-47**.
- TDMx Receive Status Register (TDMxRSR), **page 19-49**.
- TDMx Transmit Status Register (TDMxTSR), **page 19-50**.

19.7.1.1 Configuration Registers

TDMxGIR TDMx General Interface Register TDMxBASE + 0x00

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—													LPBK	CTS	RTS
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxGIR defines the operating mode of the TDMx interface.

Table 19-5. TDMxGIR Bit Descriptions

Name	Reset	Description	Settings
— 31–3	0	Reserved. Write to zero for future compatibility.	
LPBK 2	0	**TDM Loopback Mode** Enables/disables TDM Loopback mode. When this bit is set, the TDM internally loops back the TD output to the RD input. Disabled Tx channels get an idle pattern. Use this feature when the channels enabled in the Tx/Rx are the same, the Tx/Rx frame parameters are the same, and the RTS bit is set. The RD input pin data is ignored. **Note:** Shared operation is recommended in loopback mode. For details, see **19.6.1**, *Initialization for a Shared Operation*.	0 No loopback. 1 Loopback mode.
CTS 1	0	**Common TDM Signals** Defines whether the TDM shares frame sync and clock signals with other TDM modules. The TDMs can share signals as follows: • TDM0 and TDM1 do not share signals. (TDM0CTS=0, TDM1CTS=0). • TDM0 and TDM1 share signals. (TDM0CTS=1, TDM1CTS=1) with TDM0 pins being the source of the signals for TDM1. See **Table 19-7**. If the TDM modules share sync and clock signals, the TDMxRFP, TDMxTFP, TDMxRIR, and TDMxTIR registers should be configured the same way for all TDM modules, if the MSC711x device has multiple TDM modules.	0 TDM does not share signals with other TDM modules 1 TDM shares sync and clock signals with other TDM modules.
RTS 0	0	**Receive and Transmit Sharing** Defines the TDM serial interface operating mode. It determines whether the TDM transmit and receive paths are independent or share the same clock and sync. If frame sync and clock are shared, the transmitter pins are used and the receiver pins are not for frame sync and clock. Refer to **Table 19-7**.	0 Receive and transmit are independent. 1 Receive and transmit share the frame clock and frame sync.

See **Table 19-6** for the timer output source of the TDM clock in independent and shared modes.

Table 19-6. Timer Module B Inputs for the TDM Modules

CTS	RTS	TDM0		TDM1		TDM2		Notes
		TCK	RCK	TCK	RCK	TCK	RCK	
0	0	TOUT0	TOUT1	TOUT2	TOUT3	TOUT2	TOUT3	TDM is independent from other TDMs and Rx and Tx are Independent.
0	1	TOUT0	TOUT0	TOUT2	TOUT2	TOUT2	TOUT2	TDM is independent from other TDMs and Rx and Tx are shared.
1	0	TOUT0	TOUT1	TOUT0	TOUT1	TOUT0	TOUT1	TDM uses TDM0 as common source and Rx and Tx are Independent.
1	1	TOUT0	TOUT0	TOUT0	TOUT0	TOUT0	TOUT0	TDM uses TDM0 as common source and Rx and Tx are shared.

Table 19-7. Pins for TDMx Modules Based on CTS and RTS

CTS	RTS	TDMx				Comments
		TCK	RCK	TFS	RFS	
0	0	TDMxTCK	TDMxRCK	TDMxTFS	TDMxRFS	TDM is independent from other TDMs, and Rx and Tx are independent.
0	1	TDMxTCK	TDMxTCK	TDMxTFS	TDMxTFS	TDM is independent from other TDMs, and Rx and Tx are shared.
1	0	TDM0TCK	TDM0RCK	TDM0TFS	TDM0RFS	TDM uses TDM0 as a common source, and Rx and Tx are Independent.
1	1	TDM0TCK	TDM0TCK	TDM0TFS	TDM0TFS	TDM uses TDM0 as a common source, and Rx and Tx are shared.

TDMxRIR TDMx Receive Interface Register TDMxBASE + 0x04

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—														RFWM	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RFEN	RWEN	RSO	—	RSL	—	RCOE	—		RDMA	RFSD		RSA	RDE	RFSE	RRDO
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRIR defines the TDMx receiver interface operation.

Table 19-8. TDMxRIR Bit Descriptions

Name	Reset	Description	Settings
— 31–18	0	Reserved. Write to zero for future compatibility.	
RFWM 17–16	00	**Receive FIFO Watermark** Determines when the receive full FIFO event occurs.	00 The FIFO is full with 1 or more elements. 01 The FIFO is full with 2 or more elements. 10 The FIFO is full with 3 or more elements. 11 The FIFO is full with 4 or more elements.
RFEN 15	0	**Receive FIFO Enable** Determines whether the receive FIFO is used.	0 Rx FIFO is not used. 1 Rx FIFO is used.

Table 19-8. TDMxRIR Bit Descriptions (Continued)

Name	Reset	Description	Settings
RWEN 14	0	**Receive Wide FIFO Enable** Determines whether the receive FIFO is used as one sample (8/16 bit data) or as 64 bits packed with eight 8-bit data or four 16-bit data per entry in the FIFO. RWEN is set only when RFEN is set.	0 Rx FIFO is used as 16 bits or 8 bits wide. 1 Rx FIFO is used as 64 bits wide, packed with eight 8-bit data or four 16-bit data.
RSO 13	0	**Receive Sync Output** Determines whether the receive sync is driven out by the TDM receiver or is input to the TDM receiver. For details, see **Section 19.4.1**, *Sync Out Configuration,* on page 19-9.	0 Receive sync is input. 1 Receive sync is output.
— 12	0	Reserved. Write to zero for future compatibility.	
RSL 11	**0**	**Receive Sync Out Length** Indicates whether the TDMxRFS is asserted for one cycle of TDMxRCK or is asserted for the duration of the first channel in the frame. RSL must be cleared to 0 when there is only 1 receive channel in a frame. For details, see **Section 19.4.1**, *Sync Out Configuration,* on page 19-9.	0 The receive sync out width is one bit. 1 The receive sync out width is equal to the channel width.
— 10	0	Reserved. Write to zero for future compatibility.	
RCOE 9	**0**	**Receive Clock Output Enable** Determines whether the receive clock out signal, TDMxRCK, is driven out from the appropriate timer. For details, see **Section 11.2**, *Clock Synthesis Module Operation,* on page 11-4.	0 The receive clock is an input. 1 The receive clock is an output.
— 8-7	0	Reserved. Write to zero for future compatibility.	
RDMA 6	**0**	**Receive DMA Enable** Specifies whether the TDM requests DMA service for the receiver. When set, the TDM requests a DMA module transfer of the received data when: • The Rx Data Ready TDMxRER[RDR] bit is set (Rx FIFO disabled). • The Receive FIFO Full (TDMxRER[RFF] bit is set (Rx FIFO enabled).	0 The TDM does not request DMA service for the receiver. 1 The TDM requests DMA service for the receiver.
RFSD 5–4	**0**	**Receive Frame Sync Delay** With the RDE and the RFSE bits, determines the number of clocks between the receive sync signal and the first data bit of the receive frame. For examples, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	Refer to **Table 19-9**.
RSA 3	**0**	**Receive Sync Active** Determines the polarity of the receive sync signal. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 The receive sync is active on logic 1. 1 The receive sync is active on logic 0.
RDE 2	**0**	**Receive Data Edge.** Determines whether the receive data is driven out on the rising or falling edge of the receive clock. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 The receive data is driven out on the rising edge of the receive clock. 1 The receive data is driven out on the falling edge of the receive clock.

Table 19-8. TDMxRIR Bit Descriptions (Continued)

Name	Reset	Description	Settings
RFSE 1	**0**	**Receive Frame Sync Edge.** Determines whether the receive frame sync signal is sampled on the rising or falling edge of the receive clock. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 Driven on the rising edge of the receive clock. 1 Driven on the falling edge of the receive clock.
RRDO 0	**0**	**Receive Reversed Data Order.** For examples, see **Section 19.4.4**, *Reverse Data Order,* on page 19-15.	0 The first bit of a received channel is stored as the least significant bit in internal memory. 1 The first bit of a received channel is stored as the most significant bit in internal memory.

Table 19-9. Received Data Delay for Receive Frame Sync

Receive Frame Sync Delay (RFSD)	Receive Frame Sync Edge (RFSE)	Receive Data Edge (RDE)	Receive Clocks[1]
00	0	0	0
00	0	1	0.5
00	1	0	0.5
00	1	1	0
01	0	0	1
01	0	1	1.5
01	1	0	1.5
01	1	1	1
10	0	0	2
10	0	1	2.5
10	1	0	2.5
10	1	1	2
11	0	0	3
11	0	1	3.5
11	1	0	3.5
11	1	1	3

Notes: **1.** Receive clocks is the number of receive clocks between the first edge of the receive frame sync and the drive of the first data bit of the received frame.

TDMxTIR TDMx Transmit Interface Register TDMxBASE + 0x08

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—														TFWM	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TFEN	TWEN	TSO	TAO	TSL	—	TCOE	—		TDMA	TFSD		TSA	TDE	TFSE	TRDO
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTIR defines the TDMx transmitter interface operation.

Table 19-10. TDMxTIR Bit Descriptions

Name	Reset	Description	Settings
— 31–18	0	Reserved. Write to zero for future compatibility.	
TFWM 17–16	01	**Transmit FIFO Watermark** Determines when the transmit full FIFO event occurs.	00 FIFO empty with 1 or more empty slots. 01 FIFO empty with 2 or more empty slots. 10 FIFO empty with 3 or more empty slots. 11 FIFO empty with 4 or more empty slots.
TFEN 15	0	**Transmit FIFO Enable** Determines whether the transmit FIFO is used.	0 Tx FIFO not used. 1 Tx FIFO used.
TWEN 14	0	**Transmit Wide FIFO Enable** Determines whether the transmit FIFO is used as one sample (8/16-bit data) or as 64 bits packed with eight 8-bit data or four 16-bit data per entry in the FIFO. Set TWEN only when TFEN is set.	0 Tx FIFO used as 16-bit or 8-bit wide. 1 Tx FIFO is used as 64 bit wide, packed with eight 8-bit data or four 16-bit data.
TSO 13	0	**Transmit Sync Output** Determines whether the transmit sync is driven out by the TDM transmitter or it input to the TDM transmitter. For details, see **Section 19.4.1**, *Sync Out Configuration,* on page 19-9	0 Transmit sync is input. 1 Transmit sync is output.
TAO 12	0	**Transmit Always Output** Determines whether the TDM transmitter drives the TxTD pin for the inactive channels.	0 TDM transmitter does not drive the TDMxDAT for inactive channels. 1 TDM transmitter drives the TDMxDAT, regardless of whether the channel is active.

Table 19-10. TDMxTIR Bit Descriptions (Continued)

Name	Reset	Description	Settings
TSL 11	0	**Sync Out Length** Indicates whether the TDMxTFS is asserted for one cycle of TDMxTCK or is asserted for the duration of the first channel in the frame. For details, see **Section 19.4.1**, *Sync Out Configuration,* on page 19-9.	0 The sync out width is one bit. 1 The sync out width is equal to the channel width.
— 10	0	Reserved. Write to zero for future compatibility.	
TCOE 9	0	**Transmit Clock Output Enable** Determines whether the transmit clockout signal, TDMxTCK, is driven out from the appropriate timer.	0 Transmit clock is an input. 1 Transmit clock is an output.
— 8–7	0	Reserved. Write to zero for future compatibility.	
TDMA 6	**0**	**Transmit DMA Enable** Determines whether the TDM requests DMA service for the transmitter. When TDMA is set, the TDM requests a DMA module transfer of the transmit data when: • The Tx Data Register Empty TDMxTER[TDR] bit is set (Tx FIFO disabled). • The Transmit FIFO Empty TDMxTER[TFE] bit is set (Tx FIFO enabled).	0 The TDM does not request DMA service for the transmitter. 1 The TDM does request DMA service for the transmitter.
TFSD 5–4	**0**	**Transmit Frame Sync Delay** With the TDE and the TFSE bits, determines the number of clocks between the transmit sync signal and the first data bit of the transmit frame. For examples, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	Refer to **Table 19-11**.
TSA 3	**0**	**Transmit Sync Active** Determines the polarity of the transmit sync signal. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 The transmit sync is active on logic 1. 1 The transmit sync is active on logic 0.
TDE 2	**0**	**Transmit Data Edge.** Determines whether the transmit data is driven out on the rising or falling edge of the transmit clock. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 Transmit data is driven out on the rising edge of the transmit clock. 1 Transmit data is driven out on the falling edge of the transmit clock.
TFSE 1	**0**	**Transmit Frame Sync Edge.** Determines whether the transmit frame sync signal is sampled with the rising or falling edge of the receive clock. For details, see **Section 19.4.2**, *Sync In Configuration,* on page 19-10.	0 Driven out on the rising edge of the transmit clock. 1 Driven out on the falling edge of the transmit clock.
TRDO 0	**0**	**Transmit Reversed Data Order.** For examples, see **Section 19.4.4**, *Reverse Data Order,* on page 19-15.	0 The least significant bit of the memory is sent out as the first transmit data bit. 1 The most significant bit of the memory is sent out as the first transmit data bit.

Table 19-11. Transmit Data Delay for Transmit Frame Sync

Transmit Frame Sync Delay (TFSD)	Transmit Frame Sync Edge (TFSE)	Transmit Data Edge (TDE)	Transmit Clocks[1]
00	0	0	0
00	0	1	0.5
00	1	0	0.5
00	1	1	0
01	0	0	1
01	0	1	1.5
01	1	0	1.5
01	1	1	1
10	0	0	2
10	0	1	2.5
10	1	0	2.5
10	1	1	2
11	0	0	3
11	0	1	3.5
11	1	0	3.5
11	1	1	3

Notes: **1.** Transmit clocks is the number of transmit clocks between the first edge of the transmit frame sync and the first data bit of the frame that is driven out.

TDMxRFP TDMx Receive Frame Parameters TDMxBASE + 0x0C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—								RNCF							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										RCS		—		RT1	—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRFP defines the TDMx receive frame parameters.

Table 19-12. TDMxRFP Bit Descriptions

Name	Reset	Description	Settings
— 31–24	0	Reserved. Write to zero for future compatibility.	
RNCF 23–16	0	**Receive Number of Channels in a TDM Frame** Specifies the total number of channels that are received in the TDM modules. One TDM frame can contain 1–128 channels at a granularity of two. **Note:** If RNCF is *clear,* the minimum number of channels is one.	0x00 1 received channels. 0X01 2 received channels. 0x02 Reserved. 0x03 4 received channels. 0x04 Reserved . . . 0x7D 126 received channels. 0x7F 128 received channels. 0x80 – 0xFF Reserved **Note:** The even values are reserved, except x00.
— 15–6	0	Reserved. Write to zero for future compatibility.	
RCS 5-4	0	**Receive Channel Size.** Determines the receiver channel size for all channels in the frame, including decoding µ-Law and A-Law.	00 Receive channel size is 8 bits. 01 Receive channel size is 8 bits using µ-Law decoding. 10 Receive channel size is 8 bits using A-Law decoding. 11 Receive channel size is 16 bits.
— 3–2	0	Reserved. Write to zero for future compatibility.	
RT1 1	**0**	**Receive T1 frame** Determines whether the receive frame is T1 frame or non T1. In T1 mode the channel size must be 8 (**RCS** = 0) and the number of channels must be 24 (RNCF = 0x18). For details, see **Section 19.3**, *TDM Basics,* on page 19-2.	0 The receive frame is a non T1 frame. 1 The receive frame is a T1 frame.
— 0	0	Reserved. Write to zero for future compatibility.	

TDMxTFP TDMx Transmit Frame Parameters TDMxBASE + 0x10

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—								TNCF							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										TCS		—		TT1	—
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTFP defines the TDMx transmit frame parameters.

Table 19-13. TDMxTFP Bit Descriptions

Name	Reset	Description	Settings
— 31–24	0	Reserved. Write to zero for future compatibility.	
TNCF 23–16	0	**Transmit Number of Channels in a TDM Frame** Specifies the total number of channels that are transmitted by the TDM module. One TDM frame can contain 1–128 channels at a granularity of two. **Note:** If **TNCF** is *clear*, the minimum number of channels is one.	0x00 1 transmit channels. 0x01 2 transmit channels. 0x02 Reserved. 0x03 4 transmit channels. 0x04 Reserved. . . . 0x7D 126 transmit channels. 0x7F 128 transmit channels. 0x80 – 0xFF Reserved. **Note:** The even values are reserved, except 0x00.
— 15–6	0	Reserved. Write to zero for future compatibility.	
TCS 5-4	0	**Transmit Channel Size** Determines the transmitter channel size for all channels in the frame.	00 Transmit channel size is 8 bits. 01 Transmit channel size is 8 bits using μ-Law encoding. 10 Transmit channel size is 8 bits using A-Law encoding. 11 Transmit channel size is 16 bits.
— 3–2	0	Reserved. Write to zero for future compatibility.	
TT1 1	**0**	**Transmit T1 frame** Determines whether the transmit frame is T1 frame. In T1 mode, the channel size must be 8 (TCS = 0) and the number of channels must be 24 (TNCF= 0x18). For details, see **Section 19.3**, *TDM Basics,* on page 19-2.	0 Transmit frame is not a T1 frame. 1 Transmit frame is a T1 frame.
— 0	0	Reserved. Write to zero for future compatibility.	

19.7.1.2 Control Registers

TDMxRCENx TDMx Receive Channel Enable x

TDMxRCEN0	TDMxBASE + 0x20
TDMxRCEN1	TDMxBASE + 0x24
TDMxRCEN2	TDMxBASE + 0x28
TDMxRCEN3	TDMxBASE + 0x2C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RCENx															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RCENx															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 19-14. TDMxRCEN[0–3] Bit Descriptions

Name	Reset	Description	Settings
RCEN 31–0	0	**Receive Channel Active Enable Group 0–3 [31–0]** Set when the receive channel *n* is active. Each bit corresponds to each channel. For example, Group 0, bit 0 is the active bit for channel 0 and bit 1 is the active bit for channel 1.	0 The channel is not active. 1 The channel is active.
Notes: **1.** Group 0 = Channels 0–31. **2.** Group 1 = Channels 32–63. **3.** Group 2 = Channels 64–95. **4.** Group 3 = Channels 96–127.			

TDMxTCENx TDMx Transmit Channel Enable x

TDMxTCEN0	TDMxBASE + 0x40
TDMxTCEN1	TDMxBASE + 0x44
TDMxTCEN2	TDMxBASE + 0x48
TDMxTCEN3	TDMxBASE + 0x4C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TCEN															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TCEN															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 19-15. TDMxTCENx Bit Descriptions

Name	Reset	Description	Settings
TCEN 31–0	0	**Transmit Channel Active Enable Group 0–3** Set when the transmit channel *n* is active. Each bit corresponds to each channel. For example, Group 0, bit 0 is the active bit for channel 0 and bit 1 is the active bit for channel 1.	0 The channel is not active. 1 The channel is active.

Notes:
1. Group 0 = Channels 0–31.
2. Group 1 = Channels 32–63.
3. Group 2 = Channels 64–95.
4. Group 3 = Channels 96–127.

TDMxTCMAx TDMx Transmit Channel Mask x

TDMxTCMA0	TDMxBASE + 0x60
TDMxTCMA1	TDMxBASE + 0x64
TDMxTCMA2	TDMxBASE + 0x68
TDMxTCMA3	TDMxBASE + 0x6C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TCMA															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TCMA															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTCMA[0–3] permit data to be sent to the TDM but not transmitted. TDMxTCMA[0–3] account for invalid channel data coming into the TDMxTDR. This data is thrown away and the TDM does not drive the channel. The corresponding TDMxTCENx bit is cleared to disable the channel, and TDMxTCMA[0–3] allow locations in memory for inactive transmit channels. Therefore, the DMA controller transfers the invalid data into the TDM where the Transmit Channel Data Mask can discard it, and the transmit channel is not driven. If this type of functionality is not desired, do not set the bit to discard the data for the specific channel.

Table 19-16. TDMxTCMA[0–3] Bit Descriptions

Name	Reset	Description	Settings
TCMA 31–0	0	**Transmit Channel Mask Group x** Set when the transmit channel *n* data is to be ignored when received in the transmit data register. Each bit corresponds to each channel.For example, Group 0, bit 0 is the active bit for channel 0, and bit 1 is the active bit for channel 1.	0 The channel is transmitted according the corresponding TDMxTCEN[0–3] register. 1 The channel data is discarded.

Notes: **1.** Group 0 = Channels 0–31.
2. Group 1 = Channels 32–63.
3. Group 2 = Channels 64–95.
4. Group 3 = Channels 96–127.

TTDMxRCR TDMx Receive Control Register TDMxBASE + 0x80

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															REN
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRCR controls the activation/deactivation of the TDMx receiver. Activation of the receiver is valid only when the REN bit is clear.

Table 19-17. TDMxRCR Bit Descriptions

Name	Reset	Description	Settings
— 31–1	0	Reserved. Write to zero for future compatibility.	
REN 0	0	**Receive Enable.** Determines whether the receive TDM is enabled or disabled. Setting this bit is the last step in initializing the receiver.	0 Receiver is disabled. 1 Receiver is enabled.

TDMxTCR TDMx Transmit Control Register TDMxBASE + 0x84

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—															TEN
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTCR controls the activation/deactivation of the TDMx Transmitter. The activation of the transmitter is valid only when the TENS bit is clear.

Table 19-18. TDMxTCR Bit Descriptions

Name	Reset	Description	Settings
— 31–1	0	Reserved. Write to zero for future compatibility.	
TEN 0	0	**Transmit Enable** Determines whether the transmit TDM is enabled or disabled. Setting this bit is the last step in initializing the transmitter.	0 Transmitter is disabled. 1 Transmitter is enabled.

TDMxRIER TDMx Receive Interrupt Enable Register TDMxBASE + 0x88

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—							RCEUE	RLCEE	RFSEE	RFFEE	RDREE	RSEEE	ROEE	—	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRIER has the same bit format as the TDMxRER registers. If one of its bits is clear, the corresponding event in the TDMxRER registers is masked (see **page 19-45**).

Table 19-19. TDMxRIER Bit Descriptions

Name	Reset	Description	Settings
— 31–8	0	Reserved. Write to zero for future compatibility.	
RCEUE 8	0	**Receive Channel Enable Update Enable** Enables assertion of an interrupt when the Receive Channel Enable Update (RCEU) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.
RLCEE 7	0	**Receive Last Channel Event Enable** Enables assertion of an interrupt when the Receive Last Channel (RLC) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.

Table 19-19. TDMxRIER Bit Descriptions

Name	Reset	Description	Settings
RFSEE 6	0	**Receive Frame Sync Event Enable** Enables assertion of an interrupt when the Receive Frame Sync (RFS) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.
RFFEE 5	0	**Receive FIFO Full Event Enable** Enables assertion of an interrupt when the Receive FIFO Full (RFF) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.
RDREE 4	0	**Receive Data Ready Event Enable** Enables assertion of an interrupt when the Receive Data Ready (RDR) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.
RSEEE 3	0	**Receive Sync Error Event Enable** Enables assertion of the receive error interrupt when the Receive Sync Error (RSE) bit is set (see **page 19-45**).	0 Receive sync error is masked. 1 Receive sync error is enabled.
ROEE 2	0	**Receive Overrun Event Enable** Enables assertion of an interrupt when the Receive Overrun Event (ROE) bit is set (see **page 19-45**).	0 Event is masked. 1 Event is enabled.
— 1-0	0	Reserved. Write to zero for future compatibility.	

TDMxTIER TDMx Transmit Interrupt Enable Register TDMxBASE + 0x8C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—							TCEUE	TLCEE	TFSEE	TFEEE	TDREE	TSEEE	TUEE	—	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTIER has the same bit format as the TDMxTER registers. If a TDMxTIER bit is clear, the corresponding event in the TDMxTER is masked (see **page 19-49**).

Table 19-20. TDMxTIER Bit Descriptions

Name	Reset	Description	Settings
— 31-8	0	Reserved. Write to zero for future compatibility.	
TCEUE 8	0	**Transmit Channel Enable Update Enable** Enables assertion of an interrupt when the Transmit Channel Enable Update (TCEU) bit is set (see **page 19-47**).	0 Event is masked. 1 Event is enabled.
TLCEE 7	0	**Transmit Last Channel Event Enable** Enables assertion of an interrupt when the Transmit Last Channel (TLC) bit is set (see **page 19-47**).	0 Event is masked. 1 Event is enabled.
TFSEE 6	0	**Transmit Frame Sync Event Enable** Enables assertion of an interrupt when the Transmit Frame Sync (TFS) bit is set (see **page 19-47**).	0 Event is masked. 1 Event is enabled.
TFEEE 5	0	**Transmit FIFO Empty Event Enable** Enables assertion of an interrupt when the Transmit FIFO Empty (TFE) bit is set (see **page 19-47**).	0 Event is masked. 1 Event is enabled.
TDREE 4	0	**Transmit Data Register Empty Event Enable** Enables assertion of an interrupt when the Transmit Data Ready (TDR) bit is set (see **page 19-47**).	0 Event is masked. 1 Event is enabled.
TSEEE 3	0	**Transmit Sync Error Event Enabled** Enables assertion of the transmit error interrupt when the Transmit Sync Error (TSE) bit is set. See **page 19-47**	0 Transmit sync error interrupt is disabled. 1 Transmit sync error interrupt is enabled.
TUEE 2	0	**Transmitter Underrun Error Enabled** Enables assertion of an interrupt when the Transmitter Under-run Error (TUE) bit is set. See **page 19-47**.	0 Underrun error is masked. 1 Underrun error is enabled.
— 1-0	0	Reserved. Write to zero for future compatibility.	

19.7.1.3 Status Registers

TDMxRER TDMx Receive Event Register TDMxBASE + 0xA0

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—							RCEU	RLC	RFS	RFF	RDR	RSE	ROE	—	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRER contains the status of the receive data buffers and general receive events. The register can be read at any time. Depending on the event, some bits are cleared by writing a 1 to the register and some bits are cleared by reading the TDMxRDR register. Bits are cleared as defined for each bit in **Table 19-21**. If they are cleared by writing a 1, then writing a 0 has no effect. Only the Receive Sync Error and the Receive Overrun Error cause a receive error interrupt. All others cause a normal receive interrupt. Interrupts occur only when the Receive Synchronization Status is in the SYNC state (see TDMxRSR[RSSS] on **page 19-49**) except as defined in **Table 19-21**.

Table 19-21. TDMxRER Bit Descriptions

Name	Reset	Description	Settings
— 31–9	0	Reserved. Write to zero for future compatibility.	
RCEU 8	0	**Receive Channel Enable Update** Set when RxCHEN[0–3] is updated for the current frame, even if it is the same value. RCEU is cleared by writing a 1 to this bit location. Valid during all states except HUNT (see TDMxRSR[RSSS] on **page 19-49**).	0 Not updated for the current frame. 1 Updated for the current frame.
RLC 7	0	**Receive Last Channel** Set when the start of the last channel of the frame is being received, regardless of whether the last channel is enabled. It can happen between the first and fourth bit of the last received channel. The next word in the RDREG is the last channel, if the channel is enabled. Bit is cleared by writing a 1 to this bit location in the register.Valid while the receiver is enabled.	0 Last channel is not received. 1 The last channel is received.
RFS 6	0	**Receive Frame Sync** Set when the Rx frame sync is received. Bit is cleared by writing a 1 to this bit location in the register.Valid during all states except HUNT (see TDMxRSR[RSSS] on **page 19-49**).	0 First channel is not received. 1 First channel is received.
RFF 5	0	**Receive FIFO Full** When the receiver is programmed to use the Rx FIFO, this bit is set when the Rx FIFO has reached the Rx FIFO Watermark. RFF is cleared by reading the RDREG.	0 Watermark not reached. 1 Watermark reached.
RDR 4	0	**Receive Data Ready** Set when Receive Data Register (RDREG) or receive FIFO is loaded with a new value. RDR is cleared by reading the RDREG register. If the Rx FIFO is enabled, RDR is cleared when receive FIFO is empty.	0 No data loaded. 1 New data loaded.
RSE 3	0	**Receive Sync Error** Indicates a sync error. RSE is set when the receive frame synchronization is lost (the synchronization state change from the SYNC to HUNT state) because a frame sync arrived early or was not received at the expected position. This bit indicates glitches on the receive pins of the TDM module. For details, see **Section 19.4.3**, *Serial Interface Synchronization,* on page 19-13. RSE is cleared by writing a 1 to this bit location.	0 Normal operation. 1 Receive sync error.

Table 19-21. TDMxRER Bit Descriptions (Continued)

Name	Reset	Description	Settings
ROE 2	0	**Receive Overrun Error** Indicates an overrun event in the TDM when the Rx shift register loaded the current data by overwriting the oldest data in the TDMxRDR or the Rx FIFO. In wide mode, the entire line is thrown out when the current channel is loaded. This may cause a decrease in the Rx FIFO count until the current line is filled. This error should not occur during normal operation. ROE is cleared by reading from the TDMxRDR, and then writing a 1 to this bit location.	0 No overrun event. 1 Overrun event.
— 1–0	0	Reserved. Write to zero for future compatibility.	

TDMxTER TDMx Transmit Event Register TDMxBASE + 0xA4

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—							TCEU	TLC	TFS	TFE	TDR	TSE	TUE	—	
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTER contains the status of the transmit data buffers and general transmit events. It can be read at any time. Depending on the event, some bits are cleared by writing a 1 to the register and others s are cleared by writing to the TDMxTDR. Bits are cleared as defined for each bit in **Table 19-22**. If they are cleared by writing a 1, then writing a 0 has no effect. Only the Transmit Sync Error and the Transmit Under-run Error cause a transmit error interrupt. All others cause a normal transmit interrupt. Interrupts occur only when the Transmit Sync Synchronization Status is in the SYNC state (see TDMxTSR[TSSS] on **page 19-50**), except as defined in **Table 19-22**.

Table 19-22. TDMxTER Bit Descriptions

Name	Reset	Description	Settings
— 31–9	0	Reserved. Write to zero for future compatibility.	
TCEU 8	0	**Transmit Channel Enable Update** Set when the TCEN0-3 or TCMA0-3 are updated for the current frame, even if it is the same value. TCEU is cleared by writing a 1 to this bit location in the register. Valid during all states except HUNT (see TDMxTSR[TSSS] on **page 19-50**).	0 No update for the current frame. 1 Update for the current frame.
TLC 7	0	**Transmit Last Channel** Set when the start of the last channel of the frame is transmitted, regardless of whether the last channel is enabled. TLC is set between the first and fourth bits of the last transmitted channel. The next word in the TDMxTDR is the first channel, if that channel is enabled. TLC is cleared by writing a 1 to this bit location in the register. Valid during all states except HUNT (see TDMxTSR[TSSS] on **page 19-50**).	0 Last channel is not transmitted. 1 Last channel is transmitted.
TFS 6	0	**Transmit Frame Sync** Set when the Transmit Frame sync is received. TFS is cleared by writing a 1 to this bit location in the register.Valid while the transmitter is enabled.	0 First channel not transmitted. 1 First channel is transmitted.
TFE 5	0	**Transmit FIFO Empty** When the transmitter is programmed to use the Tx FIFO, TFE is set when the Tx FIFO falls below the Tx FIFO watermark. The TFF bit is cleared when data is written to the TXDxTDR or a TDM reset occurs.	0 At or above the Tx FIFO watermark. 1 Below the Tx FIFO watermark.
TDR 4	0	**Transmit Data Register Empty** Set when the Transmit Data Register (TDMxTDR) or transmit FIFO is empty and has no values to load into the Tx shift register. TDR is cleared on a write to TDMxTDR.	0 Data to be transmitted. 1 No data to be transmitted.
TSE 3	0	**Transmit Sync Error** Indicates a sync error. TSE is set when the transmit frame synchronization is lost (the synchronization state change from SYNC to HUNT state) because a transmit frame sync arrived early or not at the expected position. During operation, this bit indicates glitches on the transmit pins of the TDM module. For details, see **Section 19.4.3**, *Serial Interface Synchronization,* on page 19-13. TSE is cleared by writing a 1 to this bit location.	0 Normal operation. 1 A transmit sync error.
TUE 2	0	**Transmitter Underrun Error** Indicates an underrun event in the TDM. This error should not occur during normal operation. It indicates that the TDM has not received enough data to transmit. TUE is cleared by first writing data to the TDMxTDR and then writing a 1 to this bit location.	0 No underrun error. 1 Underrun error.
— 1–0	0	Reserved. Write to zero for future compatibility.	

TDMxRSR TDMx Receive Status Register TDMxBASE + 0xA8

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										RFCNT			RSSS		RENS
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRSR contains the receiver status. It indicates whether the receiver is synchronized on the receive sync, the receiver is enabled or disabled, and the Rx FIFO status.

Table 19-23. TDMxRSR Bit Descriptions

Name	Reset	Description	Settings
— 31–6	0	Reserved. Write to zero for future compatibility.	
RFCNT 5–3	0	**Receive FIFO Counter** Number of data words in the Rx FIFO.	000 0 words. 001 1 word. 010 2 words. 011 3 words. 100 4 words.
RSSS 2–1	0	**Receive Sync Synchronization Status** Indicates the status of the receive sync synchronization. When the synchronization state is SYNC, the serial part synchronized on the received sync and the received data transfer to the buffer in main memory for processing. For details, see **Section 19.4.3**, *Serial Interface Synchronization,* on page 19-13.	00 HUNT. 01 WAIT. 11 PRESYNC. 10 SYNC.
RENS 0	0	**Receive Enable Status** Indicates whether all the receiver parts are enabled/disabled. The propagation of the enable/disable may be delayed because of the different clock domains.	0 Receiver machine disabled. 1 Receiver machine enabled.

TDMxTSR TDMx Transmit Status Register TDMxBASE + 0xAC

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										TFCNT			TSSS		TENS
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTSR contains the status of the transmitter. It indicates whether the transmitter is synchronized on the transmit sync, whether it is enabled or disabled, and the Tx FIFO status.

Table 19-24. TDMxTSR Bit Descriptions

Name	Reset	Description	Settings
— 31–6	0	Reserved. Write to zero for future compatibility.	
TFCNT 5-3	0	**Transmit FIFO Counter** Number of data words in the Tx FIFO.	000 0 words. 001 1 words. 010 2 words. 011 3 words. 100 4 words.
TSSS 2-1	0	**Transmit Sync Synchronization Status** Indicates the transmit sync synchronization status. When the synchronization state is SYNC, the serial part is synchronized on the transmit sync and new transit data is driven out. For details, see **Section 19.4.3**, *Serial Interface Synchronization,* on page 19-13.	00 HUNT. 01 WAIT. 11 PRESYNC. 10 SYNC.
TENS 0	0	**Transmit Enable Status** Indicates whether all the transmitter parts are enabled/disabled. The propagation of the enable/disable may be delayed because of the different clock domains.	0 Transmit machine disabled. 1 Transmit machine enabled.

19.7.2 TDM AHB Interface Registers

This section discusses the TDM registers in the AHB memory map. For the value of TDMx_BASE, see **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. The TDM registers discussed in this section are listed as follows:

- TDMx Receive Data Registers (TDMxRDR), **page 19-51**.
- TDMx Transmit Data Registers (TDMxTDR), **page 19-52**

TDMxRDR TDMx Receive Data Register TDMxAHBBASE + 0x00

Bit	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	RDREG[63–48]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
	RDREG[47–32]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RDREG[31–16]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RDREG[15–0]															
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxRDR contains the data for the receiver, based on the channel size and any encoding features. If the Rx FIFO mode is not used and the data is 16 bits or 8 bits decoded to 16 bits, one word in the TDMxRDR must be read as 16 bits. If the Rx FIFO mode is not used and the data is 8 bits, one byte in the TDMxRDR must be read as 8 bits. If the Rx FIFO mode is used, there are either eight 8-bit data samples or four 16-bit samples in TDMxRDR, which must be read simultaneously in one 64-bit read. The data is loaded from the receive FIFO or receive shift

register when any read occurs. See **Section 19.5.3**, *FIFO Configuration*, on page 19-20 for information on the Rx FIFO wide mode.

Table 19-25. TDMxRDR Bit Descriptions

Name	Reset	Description	Settings
RDREG 63–0	0	**Receive Channel Data Register 0[63:0]** Contains the data received by TDMx. This data is loaded with top of stack FIFO data.	

TDMxTDR TDMx Transmit Data Register TDMxAHBBASE + 0x08

Bit	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	TDREG[63–48]															
TYPE	W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
	TDREG[47–32]															
TYPE	W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TDREG[31–16]															
TYPE	W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TDREG[15–0]															
TYPE	W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TDMxTDR contains the data to be transmitted. If the Tx Wide FIFO mode is not used, and if 16-bit data is to be encoded to 8 bits, one word must be written to the TDMxTDR. If the Tx Wide FIFO mode is not used and the data is 8 bits, one byte must be written to the TDMxTDR. If the Tx Wide FIFO mode is used, there are either eight 8-bit data samples or four 16-bit samples to be loaded into the TDMxTDR, which must be written simultaneously in one 64-bit write. The data

is loaded from the transmit FIFO or transmit shift register as needed after any write. See **Section 19.5.3**, *FIFO Configuration*, on page 19-20 for details on transmit Wide FIFO mode.

Table 19-26. TDMxTDR Bit Descriptions

Name	Reset	Description	Settings
TDREG 63–0	0	**Transmit Channel Data Register[63:0]** The data to be transmitted out of the TDMx. This data goes into the FIFO or shift register.	

Host Interface (HDI16) 20

The host interface (HDI16) is a 16-bit wide, full-duplex, double-buffered parallel port that can directly connect to the data bus of a host processor. The HDI16 supports a variety of buses and gluelessly connects with a number of industry-standard microcomputers, microprocessors, and DSPs. An external host processor can asynchronously access the HDI16 host bus, independently of the clocks on the MSC711x device because the HDI16 registers are divided into two banks:

- External host register bank accessible to an external host.
- MSC711x register bank accessible to the MSC711x device.

The HDI16 supports two classes of interfaces to external devices: a host processor/microcontroller (MCU) connection interface and a DMA controller interface. The HDI16 can also operate with an 8-bit host data bus, making it fully compatible with the DSP56300 HI08 (from the external host side, not from the MSC711x side).

Key aspects of the HDI16 module are its flexible interfacing to the pins of an external host, its ability to support external hosts with 8- or 16-bit data buses, its different modes of operation such as polled, interrupt driven, or serviced by a DMA controller, and its additional capabilities such as the host flags, command vector, and host non-maskable interrupt. **Table 20-1** summarizes the restrictions on the MSC711x HDI16.

Table 20-1. Restrictions on the MSC711x HDI16

Restriction	Description
HPE signal pin	Not implemented on the MSC711x device. Instead, the HPE signal is internally tied within the device to asserted.
HA3 signal pin	Not supported on the MSC711x devices, and this signal is internally tied within the device to deasserted.
HDSP and H8BIT signal pins	On MSC711x devices, these pins are sampled only out of reset.
HD[15–8] signal pins	Out of reset, these pins are configured as GPIO pins and not yet assigned to the HDI16. These pins must be configured for HDI16 operation if the HDI16 is used with a 16-bit data bus. **Note:** When the device is booted through the HDI port, the boot program configures all host port pins for host port operation in GPIO ports B and C. See **Figure 14-5**, *Host Port Loader,* on page 14-13.
Big-endian operation	HPCR[HLEND] must always be cleared to zero, which configures the HDI16 module for big-endian operation. This is the default value out of reset.

Table 20-1. Restrictions on the MSC711x HDI16 (Continued)

Restriction	Description
HDI16 Reset Configuration Registers	These HDI16 registers are not accessible on MSC711x devices.
DMA controllers	Some MSC711x devices have two different DMA controllers: the standard DMA controller and the Ethernet MAC DMA controller. Only the standard DMA controller can initiate MSC711x-side DMA transfers. The Ethernet MAC DMA controller cannot access the HDI16.

Table 20-2 summarizes the differences between the MSC711x HDI16 port and that of the MSC8101.

Table 20-2. MSC711x HDI16 Versus MSC8101 HDI16

Feature	MSC711x HDI16	MSC8101 HDI16
Bit Ordering		
Data bus bit ordering	Bits on HD bus are numbered with bit 0 as the LSB.	Bits on HD bus are numbered with bit 0 as the MSB.
Address bus bit ordering	Bits on HA bus are numbered with bit 0 as the LSB.	Bits on HA bus are numbered with bit 0 as the MSB.
Register bit ordering	Bits in HDI16 registers are numbered with bit 0 as the LSB.	Bits in HDI16 registers are numbered with bit 0 as the MSB.
Pinout		
Host Port Enable (HPE) pin	Not implemented. On MSC711x devices, the host port is disabled via its HPCR[HEN] bit.	Implemented. See also the HPCR[HEN] bit.
Host Data Strobe Polarity (HDSP) pin	Sampled only at reset. Otherwise, the pin is assigned to a different function. See also the HPCR[HDSP] bit.	Exists as a pin on the MSC8101 device. See also the HPCR[HDSP] bit.
H8BIT Pin (8-Bit Mode)	Sampled only at reset. Otherwise, the pin is assigned to a different function. See also the HPCR[H8BIT] bit.	Exists as a pin on the MSC8101 device. See also the HPCR[H8BIT] bit.
Host Address Bus	Uses 3 pins, HA[2–0].	Uses 4 pins, HA[3–0].
Host Data Bus	Upper 8 data pins are configured as GPIO out of reset. Must be configured for HDI16 usage if the port is used in 16-bit mode.	The HDI data pins are not configured as GPIO out of reset.
Programming Model		
Reset Configuration Registers (external host side)	These registers *are not* accessible to an external host and *are not* supported.	These registers *are* accessible to an external host and *are* supported.
Host Port Control Register [Host Enable]	During HDI16 initialization, HPCR[HEN] should be cleared.	During HDI16 initialization, HPCR[HEN] does not need to be cleared.
External Host DMA Mode		
Bidirectional transfers (external host side)	External Host DMA transfers *are not* supported when both ICR[TREQ] and ICR[RREQ] are set to 1 when HCR[HICR] is also set to 1.	External Host DMA transfers are supported when both ICR[TREQ] and ICR[RREQ] are set to 1 when HCR[HICR] is also set to 1.
HREQ, HACK pin polarities	When the HACK pin is used for acknowledge, does not support different polarities on the HREQ and HACK pins.	When the HACK pin is used for acknowledge, supports different polarities on the HREQ and HACK pins.

20.1 Features

Table 20-3 lists the features of the external host-to-MSC711x and MSC711x-to-external host interfaces. It also discusses the host interface signals and the overall structure of the HDI16 interface.

Table 20-3. HDI16 Features

Feature	HDI16-to-MSC711x	HDI16-to-External Host Processor
Data word	64 bits	16 or 8 bits
Transfer Modes	• MSC711x to host • External host to SC1400 core • External host command • External host NMI	• Mixed 16-bit, 32-bit, 48-bit and 64-bit data transfers, or 8-bit, 16-bit, 24-bit and 32-bit data transfers in 8-bit mode. These transfers are MSC711x-to-external host or external host-to-MSC711x. • Host command • Host NMI
Handshaking Protocols	• Software polled • Interrupt driven • MSC711x DMA accesses	• Software polled • Interrupt driven • Cycle-stealing DMA with initialization
Instructions	Memory-mapped registers allow the standard MOVE instruction to be used.	
Signals		• HD[15–0] host data bus (in 16-bit mode) • HD[7–0] host data bus (in 8-bit mode) • HA[3–0] host address line • HRW/$\overline{\text{HRD}}$ read/write select (HRW) or read strobe (HRD) • HDS/HWR data strobe (HDS) or write strobe (HWR) • HCS1 host chip-select 1 • HCS2 host chip-select 2 • HREQ/HTRQ host request (HREQ) or host transmit request (HTRQ) • HACK/HRRQ host acknowledge (HACK) or host receive request (HRRQ) • HDDS dual data strobe control input (multiplexed) • HDSP data strobe polarity control input • H8BIT 8-bit mode control input
Mapping		• The HDI16 registers are mapped as consecutive locations in address space of the external host processor. • The HDI16 acts as a memory or I/O-mapped peripheral for microprocessors, microcontrollers, and so on.
Dedicated Interrupts		• Separate interrupt lines for each interrupt source • Special host commands generate SC1400 interrupts under host processor control. These commands are useful for real-time production diagnostics, creating a debugging window for program development, and host control protocols.

Table 20-3. HDI16 Features (Continued)

Feature	HDI16-to-MSC711x	HDI16-to-External Host Processor
Interface Capabilities		Glueless interface (no external logic required) to the following: • MSC8102 • MPC860 • MPC8260 Minimal glue-logic required to interface to the following: • Freescale 603-750 core products • ISA bus • Freescale 68K family • Intel X86 family

20.2 HDI16 Host Port Pins

Table 20-4 describes the host port signals and their function.

Table 20-4. HDI16 Host Port Signals

Port Signal	Description	Settings
Host Bus		
HA[3–0]	**Host Address Lines 3–0** Lines 3–0 of the host address input bus. **Note:** Bit 0 corresponds to the LSB of the bus.	**Note:** The HDI16 does not use the HA3 pin.
HD[15–0]	**Host Data Bus Lines 15–0** Lines 15–0 of the bidirectional three-state host data bus. In 8-bit mode, only lines 7–0 (HD[7–0]) are used. **Note:** Bit 0 corresponds to the LSB of the bus.	
Data Strobes and Chip Selects		
HRW/HRD	**Host Write Select/Host Read Strobe** When the HDI16 is programmed to interface with a single data strobe host bus, this pin is the read/write input (HRW). When the HDI16 is programmed to interface with a double data strobe host bus, this pin is the read data strobe Schmitt trigger input (HRD).	The polarity of HRD is programmable.
HDS/HWR	**Host Data Strobe/Host Write Data Strobe** When the HDI16 is programmed to interface with a single data strobe host bus, this pin is the data strobe Schmitt trigger input (HDS). When the HDI16 is programmed to interface with a double data strobe host bus, this pin is the write data strobe Schmitt trigger input (HWR).	The polarity of HDS or HWR is programmable. See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
HCS1	**HDI16 Chip Select 1** One of the two chip-select (CS) inputs. See **Section 20.5.1**, *Host Port Chip Select Capability,* on page 20-8.	The polarity of HCS1 and HCS2 pins is programmable through the HPCR[HCSP] bit. This bit configures the polarity of both chip select pins.

Table 20-4. HDI16 Host Port Signals (Continued)

Port Signal	Description	Settings
HCS2	**HDI16 Chip Select 2** One of the two chip-select (CS) inputs. See **Section 20.5.1**, *Host Port Chip Select Capability,* on page 20-8. **Note:** The functionality of this pin can be modified via DEVCFG[HCOV]. See **Section 7.4.3**, *Device Identification and Configuration,* on page 7-16.	The polarity of the HCS1 and HCS2 pins is programmable through the HPCR[HCSP] bit. This bit configures the polarity of both chip-select pins.
Requests and Acknowledges		
HREQ/HTRQ	**Host Request/Host Transmit Request** When the HDI16 is programmed to interface to a single host request, this pin is the host request output (HREQ). This pin can be used for host DMA requests in host DMA mode. When the HDI16 is programmed to interface to a double host request, this pin is the transmit host request output (HTRQ).	The polarity of the host request is programmable. The HREQ/HTRQ request can be programmed as a driven or open-drain output.
HACK / HRRQ	**Host DMA Acknowledge/Host Receive Request** When the HDI16 is programmed to interface to a single host request, this pin is the host acknowledge Schmitt trigger input in host DMA mode (HACK). The polarity of the host DMA acknowledge is programmable. When the HDI16 is programmed to interface to a double host request, this pin is the receive host request output (HRRQ).	The polarity of the host request is programmable. The HRRQ request can be programmed as a driven or open-drain output.
HDI Configuration		
HDSP	**Data Strobe Polarity** This input defines the polarity of the data strobe.	See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
HDDS	**Dual Data Strobe** This input defines the data strobe mode as dual or single.	See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
H8BIT	**8-Bit Mode** This input defines whether 8-bit or 16-bit mode is enabled.	0 16-bit mode enabled, if the HPCR[H8BIT] bit is also zero 1 8-bit mode enabled

Note: Several pins in the HDI16 interface are configurable as outlined in **Section 20.5**, *Configuring the Host Interface Pins (External Host Side),* on page 20-6.

20.3 HDI16 Architecture

Figure 20-1 shows a block diagram of the HDI16 module configured for big-endian operation. In little-endian operation, the programming model for the external host side is modified correspondingly. Notice that there is a correspondence between the registers on the MSC711x side and the external host side. The MSC711x registers are accessible to MSC711x masters such as the SC1400 core and the DMA controllers. The external host-side registers are accessible only to the external host processor or external DMA controllers. Data is transferred through separate transmit and receive data FIFOs. The MSC711x resources synchronously access an MSC711x-side programming model, whereas the external host asynchronously accesses a host-side programming model. Synchronization logic handles the transfers between the MSC711x-side registers and the external host-side registers.

20.4 HDI16 Clocking

The HDI16 port receives two different clocks: ECore clock and AHB clock. The ECore clock runs at the core frequency and clocks the synchronizers and state machines in the HDI16. The interfaces of the HDI16 to the ASTH bus, APB bus, and DMA controller are clocked at the APB bus clock frequency, which runs at half the frequency of the ECore clock.

20.5 Configuring the Host Interface Pins (External Host Side)

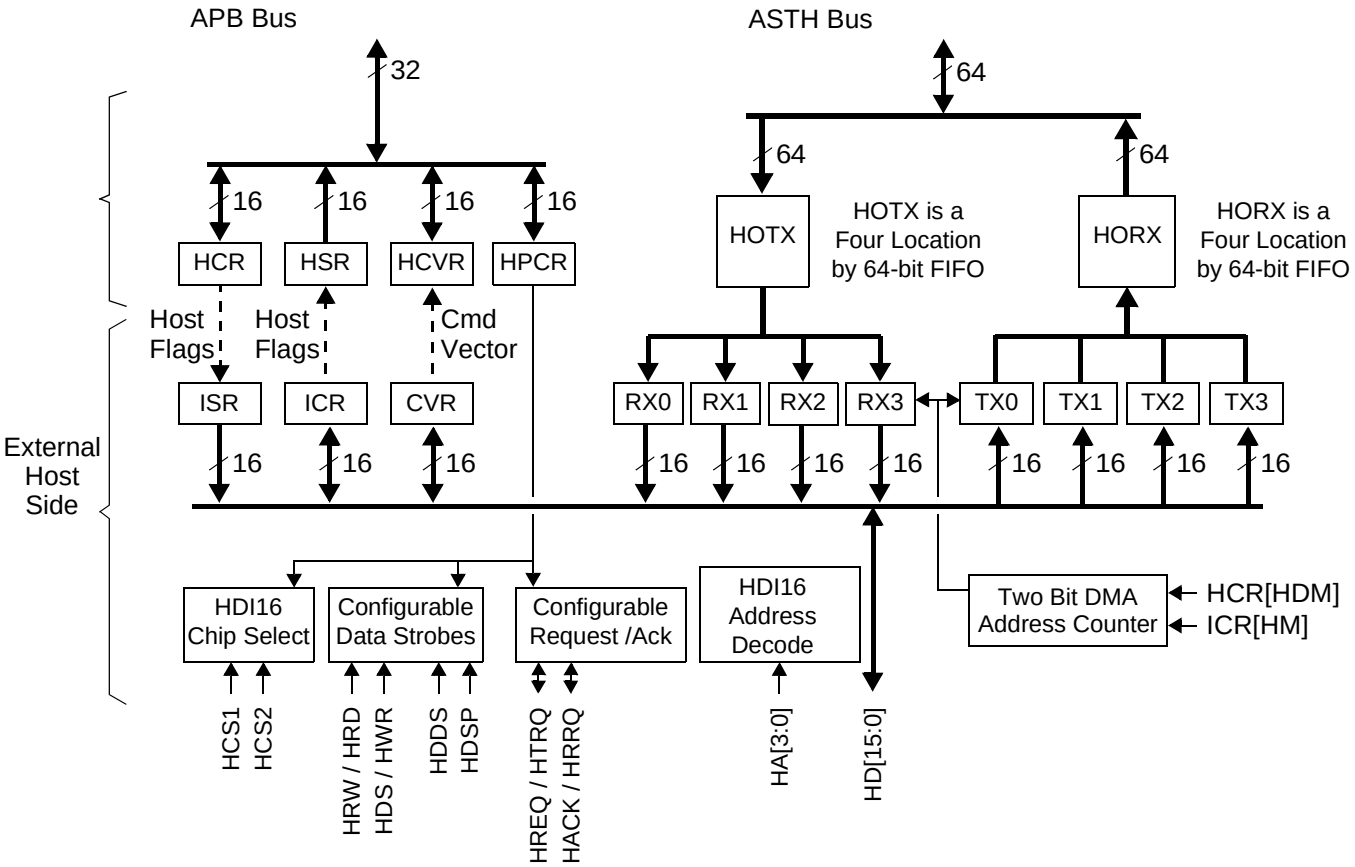
To support glueless interfacing between the HDI16 port pins and an external host, the functionality and polarity of the pins is programmable to meet the protocol on the host pins. Pin polarity is configured for either high or low true operation, as summarized in **Table 20-5**. In addition, interfacing to both an 8-bit or 16-bit external data bus is supported.

Table 20-5. Configuring HDI16 Pin Polarity

HDI16 Pin	Method to Configure Polarity	Alternate Method to Configure Polarity	Comments
HDS	HPCR[HDSP]	HDSP pin	See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
HRD	HPCR[HDSP]	HDSP pin	See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
HWR	HPCR[HDSP]	HDSP pin	See **Section 20.5.2**, *Data Strobe Pin Configuration,* on page 20-8.
HCS1, HCS2	HPCR[HCSP]	—	Both pins are configured for high or low true operation using this register bit. See **Section 20.5.1**, *Host Port Chip Select Capability,* on page 20-8.
HTRQ, HRRQ	HPCR[HRP]	—	Both pins are configured for high or low true operation using this register bit. See **Section 20.5.2.2**, *Host Request Pin Configuration*, on page 20-10.
HREQ	HPCR[HRP]	—	Configurable only from register. See **Section 20.5.2.2**, *Host Request Pin Configuration,* on page 20-10.
HACK	HPCR[HAP]	—	Configurable only from register. See **Section 20.5.2.1**, *Transfer Acknowledge Configuration,* on page 20-9.

Programming Model Accessible to MSC711x Device

Register	Register Name	Offset (APB)	Offset (ASTH Bus)
HCR	Host Control Register	0x0000	—
HSR	Host Status Register	0x0040	—
HCVR	Host Command Vector Register	0x0060	—
HPCR	Host Port Control Register	0x0020	—
HOTX	Host Transmit Data Register	—	0x0080
HORX	Host Receive Data Register	—	0x00A0
Note: Both HOTX and HORX are FIFOs with a capacity of four 64-bit words.			



Programming Model Accessible to External Host

Register	Register Name	Host Address (HA[3–0])			
ISR	Interface Status Register	0x2			
ICR	Interface Control Register	0x0			
CVR	Command Vector Register	0x1			
RX[0–3]	Receive Data Registers	**RX0:** 0x4	**RX1:** 0x5	**RX2:** 0x6	**RX3:** 0x7
TX[0–3]	Transmit Data Registers	**TX0:** 0x4	**TX1:** 0x5	**TX2:** 0x6	**TX3:** 0x7

Figure 20-1. HDI16 Block Diagram (Configured for Big-Endian Operation)

20.5.1 Host Port Chip Select Capability

The HDI16 provides a chip-select capability via the HCS[1–2] pins, either of which can enable the host port on a device. This chip select is useful, for example, when one select pin is used to select the HDI16 port for a single device on a board, and the other selects the HDI16 ports for all MSC711x devices on the board (broadcast mode). The polarity of these two pins is configured via the HPCR[HCSP] bit as follows:

- HPCR[HCSP] = 1: Module selected by HCS1 || HCS2
- HPCR[HCSP] = 0: Module selected by $\overline{\text{HCS1}}$ || $\overline{\text{HCS2}}$

The functionality of the HCS2 signal in can be disabled so that the pin is available for GPIO. For details, see **Table 7-10**, *DEVCFG Bit Descriptions*, on page 7-17.

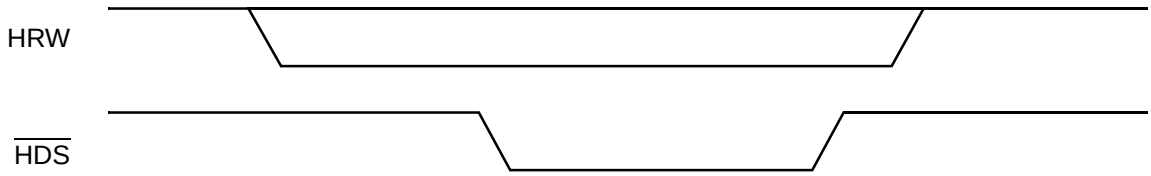
20.5.2 Data Strobe Pin Configuration

The HDI16 port pins can be configured to accept any of the following types of strobes from an external processor (**Table 20-6**):

Table 20-6. Programming Strobe Pin Functionality

Desired Usage of Pins	Pins Used	Configured via	Comments
HRW and $\overline{\text{HDS}}$	HRW/HRD and HDS/HWR	• HDDS pin = 0 and HPCR[HDDS] = 0. • HDSP pin = 0 and HPCR[HDSP] = 0.	**Single Strobe.** Uses single data strobe, $\overline{\text{HDS}}$, where strobe programmed as *low true*.
HRW and HDS	HRW/HRD and HDS/HWR	• HDDS pin = 0 and HPCR[HDDS] = 0. • HDSP pin = 1 or HPCR[HDSP] = 1.	**Single Strobe.** Uses single data strobe, HDS, where strobe programmed as *high true*.
$\overline{\text{HRD}}$ and $\overline{\text{HWR}}$	HRW/HRD and HDS/HWR	• HDDS pin = 1 or HPCR[HDDS] = 1. • HDSP pin = 0 and HPCR[HDSP] = 0.	**Dual Strobe.** Uses two data strobes, $\overline{\text{HDR}}$ and $\overline{\text{HWR}}$, where strobes programmed as *low true*.
HRD and HWR	HRW/HRD and HDS/HWR	• HDDS pin = 1 or HPCR[HDDS] = 1. • HDSP pin = 1 or HPCR[HDSP] = 1.	**Dual Strobe.** Uses two data strobes, HDR and HWR, where strobes programmed as *high true*.

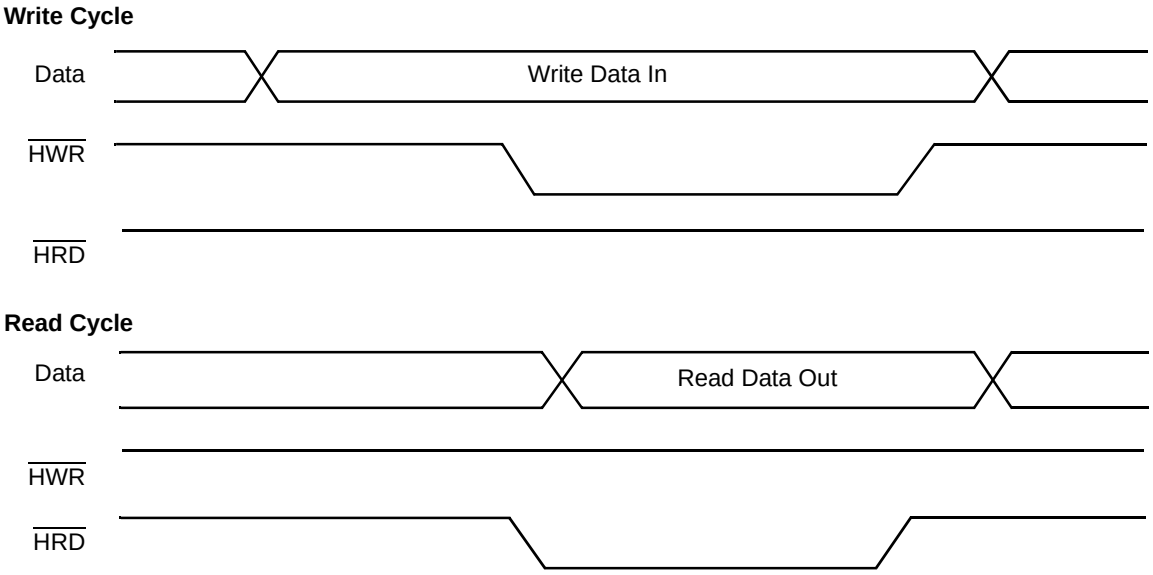
Figure 20-2 shows single-strobe operation.



In a single-strobe bus, $\overline{\text{HDS}}$ qualifies the access, while HRW specifies the type of access (if HRW is high, the access is a read and if HRW is low, the access is a write).

Figure 20-2. Single-Strobe Bus

Figure 20-3 shows dual-strobe bus operation.



In a dual-strobe bus, separate $\overline{\text{HRD}}$ and $\overline{\text{HWR}}$ signals specify the access as a read or write access, respectively.

Figure 20-3. Dual-Strobe Bus

20.5.2.1 Transfer Acknowledge Configuration

HDI16 transfer acknowledge is used only in DMA mode. The transfer acknowledge input indicates to the MSC711x device that DMA data was driven onto the host processor data bus. There are two different ways to indicate a transfer acknowledge, which are selected using the HPCR[OAD] bit.

Transfer acknowledge through the HACK input pin (HPCR[OAD] = 0) is as follows:

- For MSC711x-to-external host transfers, the HDI16 writes data onto the bus when HACK is asserted. The polarity of HACK must be set to use the same polarity as that of HREQ, as shown in **Table 20-7**.
- For external host-to-MSC711x transfers, there is valid data on the bus when HACK is asserted. In Host Acknowledge mode, the HRD/HWR pins are *not* used in read and write DMA accesses.

Transfer acknowledge through the external host accesses to host address 0x4 (HPCR[OAD] = 1):

- For MSC711x-to-external host transfers, the HDI16 writes data onto the bus when a *read operation* occurs at host address 0x4.
- For external host-to-MSC711x transfers, there is valid data on the bus when a *write operation* occurs at host address 0x4. In OAD mode, the HRD/HWR pins are used in read and write DMA accesses.

20.5.2.2 Host Request Pin Configuration

Table 20-7 shows how to configure the following parameters for non-DMA or DMA mode:

- Single request (HREQ) or separate requests for receive (HRRQ) and transmit (HTRQ)
- Polarity of the signals.

Table 20-7. Programming Host Request Pin Functionality

Pin Usage	Pins	HPCR[DMA]	ICR[HDRQ]	HPCR[HRP]	Comments
Non-DMA Mode					
HREQ	HREQ/HTRQ and HACK/HRRQ	0	0	0	**Single Host Request: $\overline{\text{HREQ}}$** $\overline{\text{Single}}$ request signal to the external host, $\overline{\text{HREQ}}$.
HREQ	HREQ/HTRQ and HACK/HRRQ	0	0	1	**Single Host Request: HREQ** Single request signal to the external host, HREQ.
$\overline{\text{HTRQ}}$ and $\overline{\text{HRRQ}}$	HREQ/HTRQ and HACK/HRRQ	0	1	0	**Dual Host Request: $\overline{\text{HTRQ}}$, $\overline{\text{HRRQ}}$** $\overline{\text{Dual}}$ request signals to the external host, $\overline{\text{HTRQ}}$ and $\overline{\text{HRRQ}}$.
HTRQ and HRRQ	HREQ/HTRQ and HACK/HRRQ	0	1	1	**Dual Host Request: HTRQ, HRRQ** Dual request signals to the external host, HTRQ and HRRQ.
DMA Mode					
HREQ and HACK	HREQ/HTRQ and HACK/HRRQ	1	—	1	**DMA Usage: HREQ, HACK** HREQ used as a DMA request signal and HACK as an acknowledge.
$\overline{\text{HREQ}}$ and $\overline{\text{HACK}}$	HREQ/HTRQ and HACK/HRRQ	1	—	0	**$\overline{\text{DMA}}$ Usage: $\overline{\text{HREQ}}$, $\overline{\text{HACK}}$** $\overline{\text{HREQ}}$ used as a DMA request signal and $\overline{\text{HACK}}$ as an acknowledge.

Notes: 1. $\overline{\text{In DMA}}$ mode, when configured for using the HACK pin, the HPCR[HAP] bit must be set so that $\overline{\text{HREQ}}$ and $\overline{\text{HACK}}$ have the same polarity.

2. In DMA mode, the HREQ output pin requests DMA activity and HACK can be selected as an acknowledge, as shown in **Section 20.5.2.1**. In non-DMA mode, the HACK pin is not used.

In addition, open-drain operation is optional through the HPCR[HROD] bit. When open-drain operation is selected, an external pull-up resistor is required on the appropriate pin(s). Take care with the pull-up resistor to ensure that HDSP is sampled properly at reset (see the design checklist

application note). For open-drain operation, these request signals must be configured as active low, that is, HPCF[HRP] must be cleared. You can also configure these pins as active only for receive operations, only for transmit operation, or for both receive and transmit operations. **Table 20-8** shows how to configure these pins via the ICR HDRQ, TREQ, and RREQ bits.

Table 20-8. Programming Request Signal(s) in Non-DMA Mode

HDRQ	TREQ	RREQ	HREQ/HTRQ Signal	HRRQ Signal
When Configured for HREQ Signal Functionality (HRRQ not used)				
0	**0**	**0**	Not used (polled operation only).	Not used.
0	0	1	**Receive-Only:** HREQ asserted on ISR[RXDF] request (interrupt driven operation)	Not used.
0	1	0	**Transmit-Only:** HREQ asserted on ISR[TXDE] request (interrupt driven operation)	Not used.
0	1	1	**Transmit and Receive:** HREQ asserted on ISR[RXDF] or ISR[TXDE] request (interrupt driven operation)	Not used.
When Configured for HTRQ and HRRQ Signal Functionality				
1	**0**	**0**	Not used (polled TX operation only).	Not used (polled RX operation only).
1	**0**	**1**	Not used (polled TX operation only).	**Receive-Only:** HRRQ asserted on ISR[RXDF] request (interrupt driven operation)
1	**1**	**0**	**Transmit-Only:** HREQ asserted on ISR[TXDE] request (interrupt driven operation)	Not used (polled RX operation only).
1	**1**	**1**	**Transmit:** HREQ asserted on ISR[TXDE] request (interrupt driven operation)	**Receive:** HRRQ asserted on ISR[RXDF] request (interrupt driven operation)

20.5.3 Host Data Bus Size Configuration (External Host Side)

External hosts are programmed for an 8- or 16-bit data bus. When the HPCR[H8BIT] bit or the H8BIT input pin is cleared, the 16-bit mode is enabled. The host processor reads and writes the entire 16 bits of the HDI16 host bank registers. Data transferred from the HOTX FIFO to RX[0–3] registers is written with all 64-bits of the HOTX FIFO. Data transferred from the TX[0–3] registers to the HORX FIFO is written with all 64-bits of the TX[0–3] registers.

When the HPCR[H8BIT] bit or the H8BIT input pin is set, the 8-bit mode is enabled. The host processor writes/reads only the eight LSBs of the HDI16 host bank registers. The eight MSBs of the CVR and ICR are always written with zeros. Data is transferred from the 32 LSBs of the HOTX FIFO to the eight LSBs of the RX[0–3] registers as shown in **Figure 20-4**.

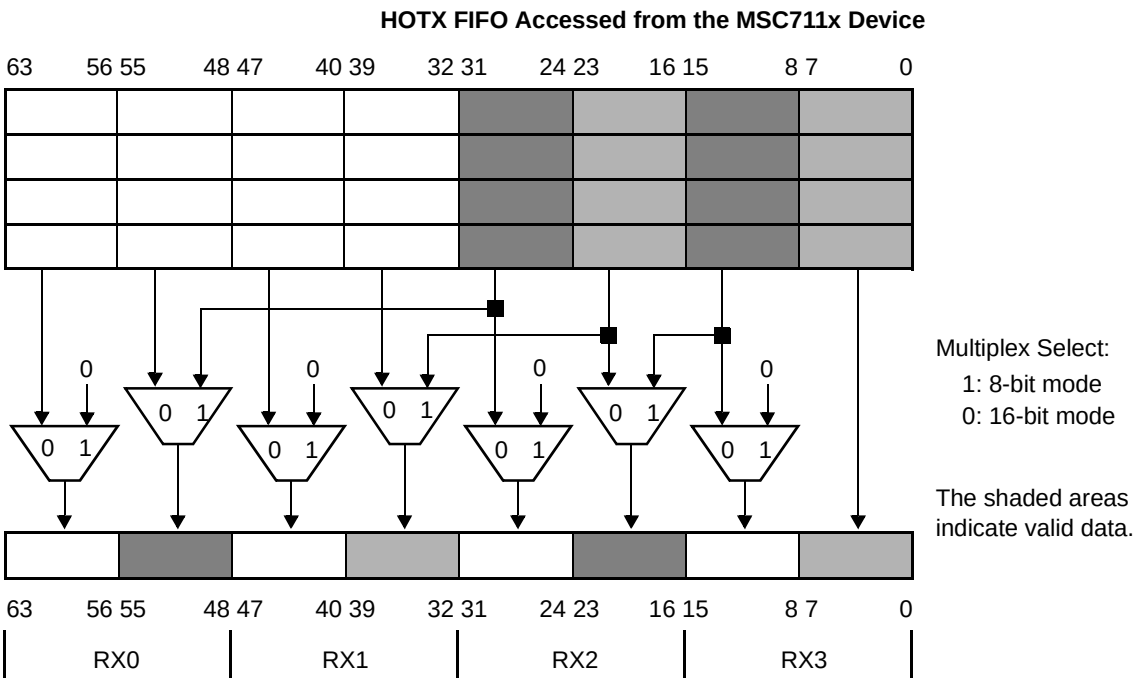


Figure 20-4. 8-Bit Mode: MSC711x-to-External Host Transfers

Data transferred from TX[0–3] to the HORX FIFO is written to the 32 LSBs of the HORX FIFO registers. See **Figure 20-5**.

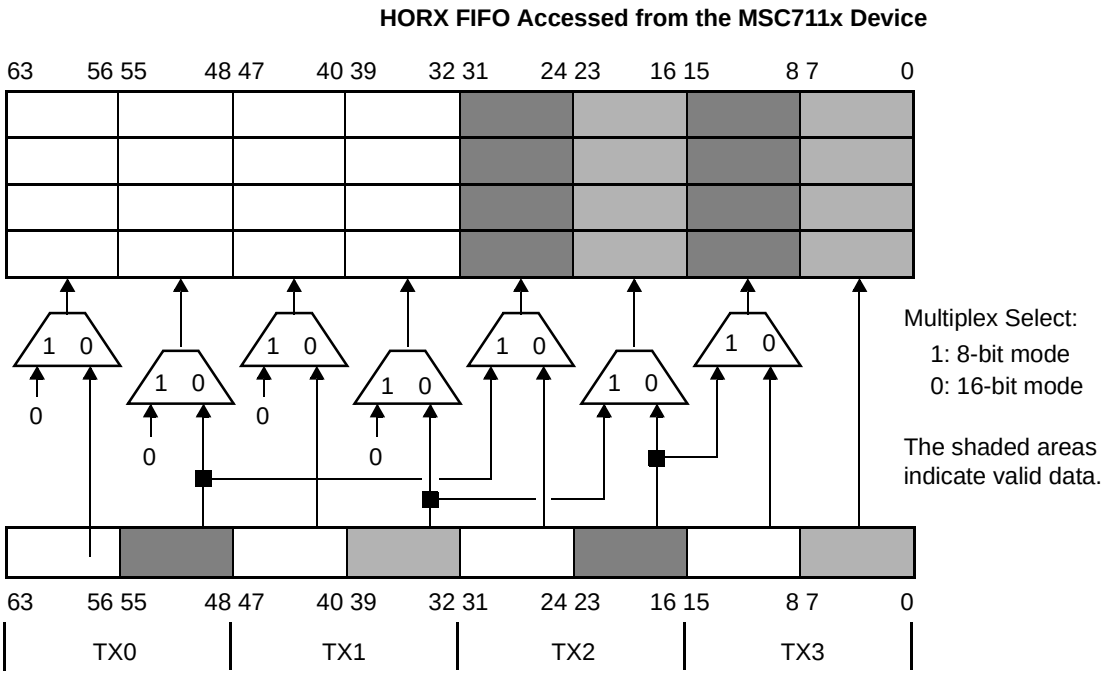


Figure 20-5. 8-Bit Mode Diagram (External Host-to-MSC711x)

20.6 HDI16 Data Transfer

HDI16 data transfers are separated into two categories, MSC711x side and external host side. This section considers each side independently.

20.6.1 Data Transfer on the MSC711x Side

To the MSC711x device, the HDI16 registers appears as a contiguous block of memory-mapped registers. The MSC711x device can write to the appropriate HDI16 register to configure the HDI16 for proper operation. Properly configured, the HDI16 port can be used to transfer data between the external host and the MSC711x device. Data is transferred using MSC711x polling, interrupts, or DMA data transfers.

20.6.1.1 Polling

The MSC711x device can service the HDI16 by polling the HSR status bits and accessing the transmit and receive FIFOs when appropriate. The HSR status bits polled are as follows:

- Host Transmit FIFO Not Full (HTFNF)
- Host Transmit FIFO Empty (HTFE)
- Host Receive FIFO Full (HRFF)
- Host Receive FIFO Not Empty (HRFNE)

These bits are discussed in detail in **Table 20-25**, *HSR Bit Descriptions*, on page 20-31.

20.6.1.2 Interrupt-Driven Operation

The HDI16 port can be configured to request interrupt service from the MSC711x device, the SC1400 interrupts do not require an external interrupt signal. When the appropriate HCR interrupt enable bit is set, an interrupt condition caused by the host processor sets the appropriate bit in the HSR, generating an interrupt request to the SC1400 core, which responds by jumping to the appropriate interrupt service routine. **Table 20-9** shows the different MSC711x-side HDI16 interrupts

Table 20-9. Interrupts and Associated Bit Settings

Interrupt	HCR Bit	HSR Bit	CVR	HCVR
Receive data FIFO Full	HRFIE	HRFF		
Receive data FIFO not empty	HREIE	HRFNE		
Transmit data FIFO empty	HTEIE	HTFE		
Transfer data FIFO not full	HTFIE	HTFNF		
Host command	HCIE			HCP
Host NMI			NMI	HCP

To clear the interrupt, the SC1400 core interrupt service routine must read or write the appropriate HDI16 register, for example, clearing HRFF or HTFE. For host command interrupts and NMI, the pending interrupt condition is cleared when the MSC711x program controller reads HCVR. **Figure 20-6** illustrates HDI16 interrupt generation.

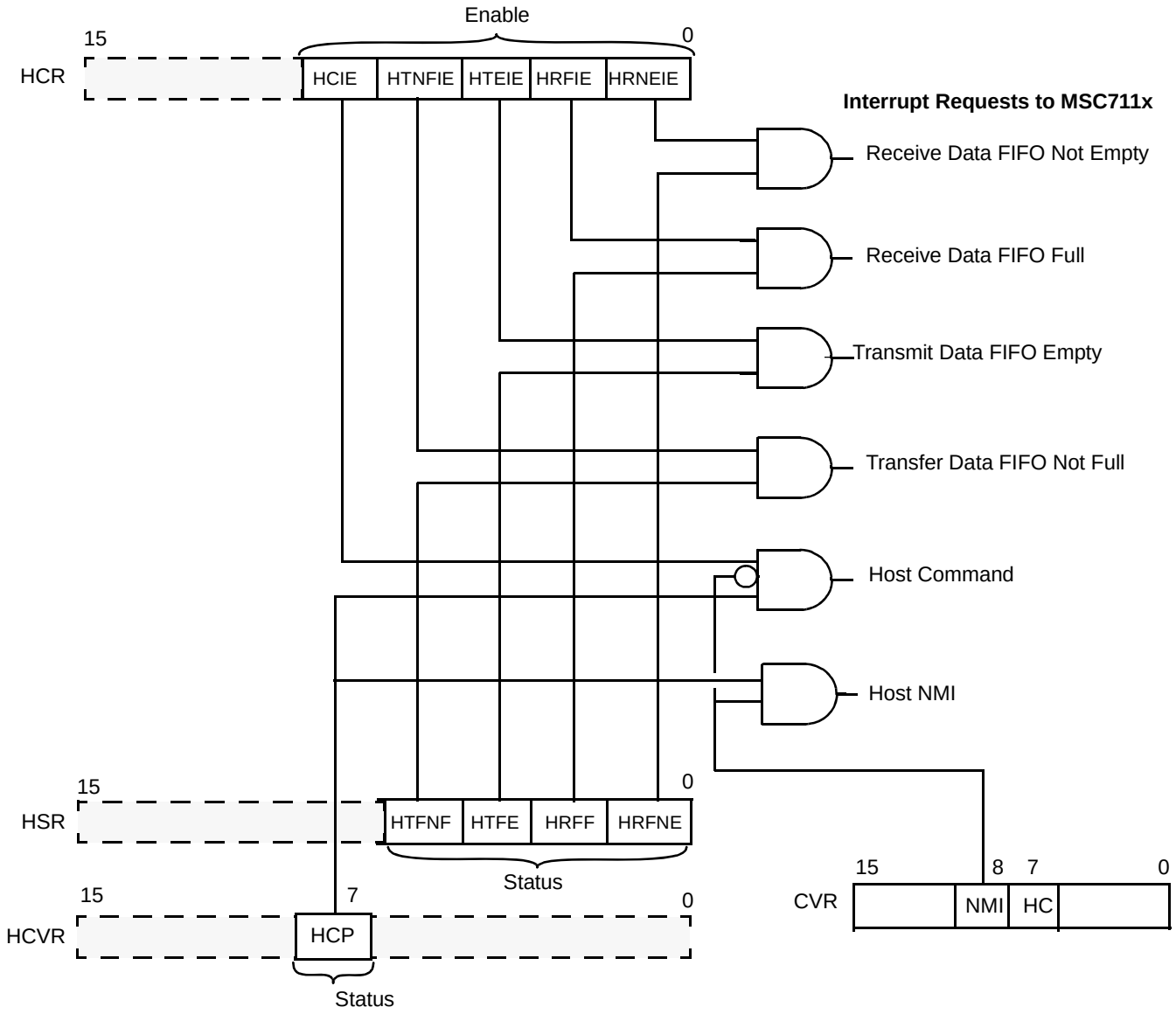


Figure 20-6. HDI16 Interrupt Generation

20.6.1.3 DMA Operation

The MSC711x DMA controller, described in **Chapter 8**, *DMA Controller*, can access the HOTX and HORX FIFOs through the ASTH bus. Both single and WRAP4 accesses are supported.

MSC711x DMA operation is set up through the following bits:

- HCR[DBRE], which selects hardware DMA read requests when HORX is full versus not empty.
- HCR[DBTE], which selects hardware DMA write requests when HOTX is empty versus not full.

The DMA controller can service the HDI16 port independently of the HDI16 mode, which is only for DMA accesses on the external host side.

20.6.2 Data Transfer on the External Host Side

To an external host processor, the HDI16 registers appears as a contiguous block of memory mapped registers. The external host processor can write to the appropriate HDI16 register to configure the HDI16 for proper operation. Once properly configured, the HDI16 port can transfer data between the external host and the MSC711x device using one of the following techniques:

- Non-DMA mode:
  - External host polling.
  - External host interrupts.
  - Host DMA can be level-triggered using $\overline{\text{HREQ}}$, $\overline{\text{HTRQ}}$, or $\overline{\text{HRRQ}}$.
- External host DMA mode
  - Acknowledge using the HACK pin.
  - Acknowledge upon access to host address 0x4.
  - Host DMA can be edge-triggered using $\overline{\text{HREQ}}$.

The HREQ/HTRQ and HACK/HRRQ handshake flags provide polled or interrupt-driven data transfers with the host processor or host DMA transfers. Because of the speed of the MSC711x interrupt response, most host microprocessors can load or store data at their maximum programmed I/O instruction rate without testing the handshake flags for each transfer. If full handshake is not needed, the host processor can treat the MSC711x as a fast device, and data can transfer between the host processor and the MSC711x device at the fastest host processor data rate. The host processor performs the following steps:

1. Asserts the HDI16 address to select the external host register to read or write.
2. Selects the direction of the data transfer (if the host processor is writing, it sources the data on the bus.)
3. Strobes the data transfer.

20.6.2.1 Polled Operation (Non-DMA Mode)

For polling, the HREQ signal connection to the host processor is not required. The host processor first performs a data read transfer to read the ISR. This read transfer allows the host processor to determine the status of the HDI16 and perform the appropriate actions:

- If ISR[RXDF] is set, the receive data register is full. The host processor can perform a data read.
- If ISR[TXDE] is set, the transmit data register is empty. The host processor can perform a data write.
- If ISR[TRDY] is set, the transmit data register is empty. This implies that the receive data FIFO on the core side is also empty. Data written by the host processor to the HDI16 is transferred directly to the core side.
- If any of ISR[HF(4–7)] is set, depending on how the host flags are used, this may indicate an application-specific state within the SC1400 core. Intervention by the host processor may be required.
- If ISR[HREQ] is set, the HREQ signal is asserted, and the SC1400 core requests the attention of the host processor. One of the previous four conditions exists.

After the appropriate data transfer occurs, the corresponding status bit is updated to reflect the transfer.

20.6.2.2 External Interrupt (Non-DMA Mode)

The HDI16 can request interrupt service from either the SC1400 or the host processor. When HREQ is connected to the host processor interrupt input, the HDI16 asserts HREQ to request service from the host processor. HREQ is asserted according to the programming of ICR[HTRA] and the status bits TXDE, TXDE16, and TXDE2 when ICR[TREQ] is set, or according to the programming of ICR[HRRA] and the status bits RXDF, RXDF16, and RXDF32 when ICR[RREQ] is set, as **Figure 20-7** shows. The programming of HRRA or HTRA chooses the status bit, and depending on its value, HREQ is asserted or deasserted.

The host processor acknowledges host interrupts by executing an interrupt service routine. The host processor tests the appropriate status bit (TXDE, TXDE16, TXDE32, RXDF, RXDF16, or RXDF32, depending on the application) to determine the interrupt source. The host processor interrupt service routine must read or write the appropriate HDI16 data register to clear the interrupt. HREQ is deasserted under the following conditions:

- The enabled request is cleared or masked.
- The SC1400 core is reset.

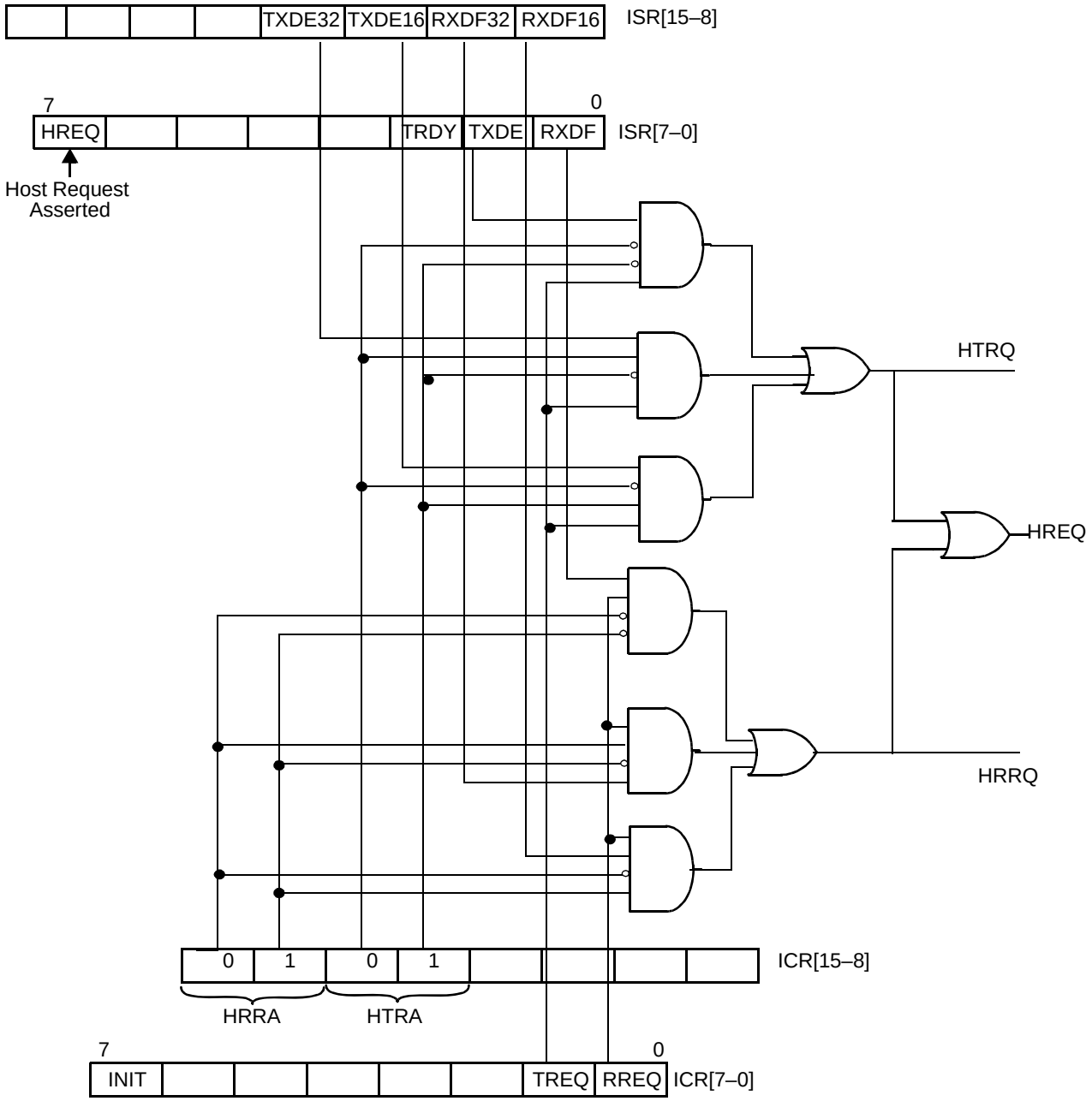


Figure 20-7. HDI16 Host Request Structure

20.6.2.3 Host DMA Mode

The HDI16 host DMA mode supports external DMA controller devices connected to the HDI16 on the external host side. The external DMA controllers should not be confused with the MSC711x DMA controllers. The HDI16 DMA mode has the following features:

- Mode can be programmed either from the MSC711x side or external host side
- Programmable transfer directions:
  - External host-to-MSC711x
  - MSC711x-to-external host

- Programmable DMA word size: 16, 32, 48, or 64 bits
- Programmable transfer acknowledge:
  - Data valid when the HACK input pin is asserted
  - Data valid on external host accesses to host address 0x4

The HDI16 module can be programmed either from the MSC711x side or external host side. This procedure is outlined in **Section 20.7**, *Setting Up the HDI16 Port*, on page 20-19. The HPCR[DMA] bit must also be set to enable the external host DMA mode.

Read transfers from the MSC711x device to the external host in DMA mode proceed as follows:

1. HREQ is asserted to the external device when the appropriate RX registers are full, depending on the programmed transfer size.
2. The external device reads the value on the host data bus (the acknowledge provides the data strobe as covered is **Section 20.5.2.1**, *Transfer Acknowledge Configuration*).
3. Additional transfers are performed until the programmed transfer size is met.

Write transfers from the external host to the MSC711x device in DMA mode proceed as follows:

1. HREQ is asserted to the external device when the appropriate TX registers are empty, depending on the programmed transfer size.
2. The external device writes a value onto the host data bus (the acknowledge provides the data strobe as covered is **Section 20.5.2.1**, *Transfer Acknowledge Configuration*).
3. Additional transfers are performed until the programmed transfer size is met.

The HDI16 data register (RX[0–3]/TX[0–3]) selected during a DMA transfer is not selected by the HA[3–0] pins but by an internal 2-bit internal address counter that is initialized via the INIT bit. After each DMA transfer on the host data bus, the address counter is automatically bumped as follows to access the next HDI16 data register:

- Decrements (little-endian mode)
- Increments (big-endian mode)

When the address counter (corresponds to the two LSBs of the address bus) reaches the last register, the address counter is loaded with the value of the HCR[HDM] or ICR[HM] bits. This arrangement allows a circular transfer of 16-bit, 32-bit, 48-bit, or 64-bit data and eliminates the need for the DMA controller to supply the HA[3–0] pins. For 32-bit, 48-bit, or 64-bit data transfers, the SC1400 CPU interrupt rate is reduced by a factor of 2, 3 or 4, respectively, from the host request rate. Thus, for every two, three, or four host processor data transfers of one word each, there is only one 64-bit core CPU interrupt.

20.7 Setting Up the HDI16 Port

The HDI16 port is configured either from the MSC711x side or from the external host side by writing to the HCR[HICR] bit from the MSC711x side, typically during program initialization. **Table 20-10** shows how this bit selects the source for programming the HDI16 port:

Table 20-10. Selecting HDI16 Programming Source

Value of HCR[HICR]	Side to Configure the HDI16
0	HDI16 is programmed from the *MSC711x side* through the HCR register.
1	HDI16 is programmed from the *external host side* through the ICR register.

The value of the HCR[HICR] bit changes the programming model for the HCR and ICR registers, as described for the HCR on **page 20-28** or the ICR on **page 20-39**. The ICR is accessible only to the host, and the HCR is accessible only to the MSC711x device.

20.7.1 Non-DMA Mode Programmed from MSC711x Side (HICR = 0)

To use the HDI16 non-DMA mode, perform the following steps:

1. Initialize the HDI16 by issuing an individual reset via clearing HPCR[HEN].
2. Select HDI16 programmability from the MSC711x side using the HCR[HICR] bit as outlined in **Section 20.7**, *Setting Up the HDI16 Port*. In this case, the HCR[HICR] bit is set to 0 so that the HDI16 port is programmed through the HCR register.
3. Program the HDI16 for non-DMA mode as shown in **Table 20-11**:

Table 20-11. Selecting Non-DMA Mode (from MSC711x Side)

HPCR[DMA]	DMA Mode
0	HDI16 port operates in non-DMA mode.
1	HDI16 port operates in DMA mode.

4. Program the data strobe functionality of the HDI16 pins as outlined in **Section 20.5.2**, *Data Strobe Pin Configuration*. This allows the HDI16 module to conform to the protocol and polarity found on the pins of the external host processor.
5. Program the host request functionality of the HDI16 pins as outlined in **Section 20.5.2.2**, *Host Request Pin Configuration*. This provides either a single interrupt request, HREQ, to the external host or separate requests for receive and transmit, HRRQ and HTRQ.

6. Select the address where data is transferred to/from the FIFOs as shown in **Table 20-12** (also see examples with different transfer sizes in **Section 20.7.5.1**, *Non-DMA External Host Accesses*):

Table 20-12. Non-DMA Trigger Addresses (from MSC711x Side)

HCR[HDM1]	HCR[HDM2*]	Trigger after Access to Host Address (Big-Endian Operation)
0	0	0x7 (TX3 or RX3)
0	1	0x6 (TX2 or RX2)
1	0	0x5 (TX1 or RX1)
1	1	0x4 (TX0 or RX0)

Notes: 1. HDM2 is the LSB in the HDM field.
2. **Section 20.7.5**, *Data Transfer Sizes Through the HDI16,* on page 20-23 shows how the trigger address relates to the size of the data transfer.

7. Enable the HDI16 for operation by setting HPCR[HEN].
8. The external host can now access the HDI16 module as desired. The direction of the transfers is correctly handled by the host address and data strobe signals provided by the external host processor.

20.7.2 Non-DMA Mode Programmed from External Host Side (HICR = 1)

To use the HDI16 non-DMA mode, perform the following steps:

1. Initialize the HDI16 by issuing an individual reset via clearing HPCR[HEN].
2. Program the HDI16 for Non-DMA mode as shown in **Table 20-13**:

Table 20-13. Selecting Non-DMA Mode (from MSC711x Side)

HPCR[DMA]	DMA Mode
0	HDI16 port operates in Non-DMA mode.
1	HDI16 port operates in DMA mode.

3. Select HDI16 programmability from the external host side by setting the HCR[HICR] bit to 1 so that the HDI6 is programmed through the ICR register.
4. Program the data strobe functionality of the HDI16 pins as outlined in **Section 20.5.2**, *Data Strobe Pin Configuration*. This allows the HDI16 module to conform to the protocol and polarity on the pins of the external host processor.
5. Program the host request functionality of the HDI16 pins as outlined in **Section 20.5.2.2**, *Host Request Pin Configuration* and **Table 20-8**. This provides either a single

interrupt request, HREQ, to the external host or separate requests for receive and transmit, HRRQ and HTRQ.

6. Select the size of the data transfers between the devices as shown in **Table 20-14** (also see examples with different transfer sizes in **Section 20.7.5.1**, *Non-DMA External Host Accesses*):

Table 20-14. Non-DMA Trigger Addresses (from External Host Side)

ICR[HM1]	ICR[HM0]	Trigger After Access to Host Address
0	0	0x7
0	1	0x6
1	0	0x5
1	1	0x4

Note: **Section 20.7.5**, *Data Transfer Sizes Through the HDI16,* on page 20-23 shows how the trigger address relates to the size of the data transfer.

7. Enable the HDI16 for operation by setting HPCR[HEN].

The external host can now access the HDI16 module as desired. The direction of the transfers is correctly handled by the host address and data strobe signals provided by the external host processor.

20.7.3 DMA Mode Programmed from MSC711x Side (HICR = 0)

To use the HDI16 DMA mode, perform the following steps:

1. Initialize the HDI16 by issuing an individual reset via clearing HPCR[HEN].
2. Select HDI16 programmability from the MSC711x side by clearing the HCR[HICR] bit to 0 so that the HDI16 port is programmed through the HCR register.
3. Select the size of the DMA transfer, as shown in **Table 20-15**:

Table 20-15. Selecting DMA Transfer Size from MSC711x Side

HCR[HDM1]	HCR[HDM2*]	Transfer Size
0	0	Transfer 64-bit DMA data.
0	1	Transfer 48-bit DMA data.
1	0	Transfer 32-bit DMA data.
1	1	Transfer 16-bit DMA data.

Note: HDM2 is the LSB in the HDM field.

4. Select the type of transfer acknowledge for DMA operations via the HPCR[OAD] bit.

5. Select the direction of the DMA transfer as shown in **Table 20-16**:

Table 20-16. Selecting DMA Transfer Direction from MSC711x Side

HCR[HDM0]	Transfer Direction
0	Input — Transfers are from External Host-to-MSC711x.
1	Output — Transfers are from MSC711x-to-External Host.

6. Program the HDI16 for DMA mode as shown in **Table 20-17**:

Table 20-17. Selecting DMA Mode from MSC711x Side

HPCR[DMA]	DMA Mode
0	HDI16 port operates in non-DMA mode.
1	HDI16 port operates in DMA mode.

7. Enable the HDI16 for operation by setting HPCR[HEN].

At this point, the HDI16 is ready. The HREQ pin requests transfers from the external host.

20.7.4 DMA Mode Programmed from External Host Side (HICR = 1)

To use the HDI16 DMA mode, perform the following steps:

1. Initialize the HDI16 by issuing an individual reset via clearing HPCR[HEN].
2. Select HDI16 programmability from the MSC711x side by setting the HCR[HICR] bit to one so that the HDI16 is programmed through the ICR register.
3. Select the size of the DMA transfer as shown in **Table 20-18**:

Table 20-18. Selecting DMA Transfer Size from External Host Side

ICR[HM1]	ICR[HM0]	Transfer Size
0	0	Transfer 64-bit DMA data.
0	1	Transfer 48-bit DMA data.
1	0	Transfer 32-bit DMA data.
1	1	Transfer 16-bit DMA data.

4. Select the direction of the DMA transfer as shown in **Table 20-19**:

Table 20-19. Selecting DMA Transfer Direction from External Host Side

ICR[TREQ]	ICR[RREQ]	**Table 20-20.** Direction of DMA Transfers
0	0	No transfer performed.
0	1	Output. Transfers are from MSC711x-to-external host.
1	0	Input. Transfers are from external host-to-MSC711x.
1	1	Reserved.

5. Select the type of transfer acknowledge used in DMA operations using the HPCR[OAD] bit as outlined in **Section 20.5.2.1**, *Transfer Acknowledge Configuration*.
6. Program the HDI16 for DMA mode as shown in **Table 20-21**.

Table 20-21. Selecting DMA Mode from MSC711x Side

HPCR[DMA]	DMA Mode
0	HDI16 port operates in non-DMA mode.
1	HDI16 port operates in DMA mode.

7. Enable the HDI16 for operation by setting HPCR[HEN].

At this point, the HDI16 is ready. The HREQ pin requests transfers from the host.

20.7.4.1 Host-Side Configuration Visible to MSC711x

Some information programmed into the external host-side ICR register is visible to the MSC711x-side programming model:

- The value of the ICR[HM] bits can be viewed on the MSC711x side as bits 9 and 8 in the HCR[HM]. This allows MSC711x-level resources to determine the DMA transfer size.
- The ICR[RREQ] bit is reflected in bit 10 of the HCR[HDM0]. This allows MSC711x-level resources to determine the direction of the DMA transfer set up by the external host.

The inverted value of the HCR[HDM0] bit is reflected in bit 1 of the ICR[TREQ] bit so that the external host can determine the direction of the DMA transfer set up by the MSC711x.

20.7.5 Data Transfer Sizes Through the HDI16

This section demonstrates proper use of the non-DMA and DMA modes for different transfer sizes. Only transmit operations from the external host are considered. The receive operation functions similarly. Also, only big-endian operation is covered.

20.7.5.1 Non-DMA External Host Accesses

This section shows how data passes through the transmit path of the HDI16 port in non-DMA mode. Non-DMA external host accesses for 16-bit data transfers proceed as follows:

1. The HDI16 is programmed to trigger on host address 0x7 in non-DMA mode.
2. The external host writes 16-bit data to the host-side TX3 register at host address 0x7.
3. The write triggers the transfer of data to the HORX FIFO.
4. The device reads data from the MSC711x-side HORX register in bits 15–0.

Non-DMA external host accesses for 32-bit data transfers proceed as follows:

1. The HDI16 is programmed to trigger on host address 0x7 in non-DMA mode.
2. The host writes the most significant 16 bits of data to the host-side TX2 register at host address 0x6.
3. The host writes the least significant 16 bits of data to the host-side TX3 register at host address 0x7.
4. The write triggers the transfer of data to the HORX FIFO.
5. The device reads data from the MSC711x-side HORX register in bits 31–0.

Non-DMA external host accesses for 32-bit data transfers using an alternate technique proceed as follows:

1. The HDI16 is programmed to trigger on host address 0x6 in non-DMA mode.
2. The host writes the least significant 16 bits of data to the host-side TX3 register at host address 0x7.
3. The host writes the most significant 16 bits of data to the host-side TX2 register at host address 0x6.
4. The write triggers the transfer of data to the HORX FIFO.
5. The device reads data from the MSC711x-side HORX register in bits 31–0.

Non-DMA external host accesses for 64-bit data transfers proceed as follows:

1. The HDI16 is programmed to trigger on host address 0x7 in non-DMA mode.
2. The host writes the most significant 16 bits of data to the host-side TX0 register at host address 0x4.
3. The host writes the next most significant 16 bits of data to the host-side TX1 register at host address 0x5.
4. The host writes the next most significant 16 bits of data to the host-side TX2 register at host address 0x6.

5. The host writes the least significant 16 bits of data to the host-side TX3 register at host address 0x7.
6. The write triggers the transfer of data to the HORX FIFO.
7. The device reads data from the MSC711x-side HORX register in bits 63–0.

An alternative technique for 64-bit non-DMA data transfers proceeds as follows:

1. The HDI16 is programmed to trigger on host address 0x4 in non-DMA mode.
2. The host writes the least significant 16 bits of data to the host-side TX3 register at host address 0x7.
3. The host writes the next most significant 16 bits of data to the host-side TX2 register at host address 0x6.
4. The host writes the next most significant 16 bits of data to the host-side TX1 register at host address 0x5.
5. The host writes the most significant 16 bits of data to the host-side TX0 register at host address 0x4.
6. The write triggers the transfer of data to the HORX FIFO.
7. The device reads data from the MSC711x-side HORX register in bits 63–0.

20.7.5.2 External Host DMA Accesses

This section shows how data passes through the transmit path of the HDI16 port. The 32-bit data transfers in host acknowledge mode proceed as follows:

1. The HDI16 is programmed for 32-bit transfers in DMA mode.
2. The HDI16 module asserts its HREQ signal pin when it is ready for an external DMA access.
3. The host writes the most significant 16-bits of data to the host-side TX2 register (the two-bit DMA counter points to the TX2 register) using HACK as a strobe to indicate valid DMA data on the host data bus.
4. The host writes the least significant 16 bits of data using HACK as a strobe to indicate valid DMA data on the host data bus. The internal DMA counter increments the address so that the write is performed to the host-side TX3 register (the two-bit DMA counter points to the TX3 register).
5. Completing the second access triggers the transfer of data (**Table 20-15**) to the HORX FIFO.
6. The two-bit DMA counter is loaded from the HCR[HDM] or ICR[HM] bits to point to the TX2 register, depending on the value of HCR[HICR] bit.
7. The device reads the data from the MSC711x-side HORX register in bits 31–0.

Note: In host acknowledge mode, the HRD/HWR pins are *not* used in read and write DMA accesses.

The 32-bit data transfers in OAD mode proceed as follows:

1. The HDI16 programmed for 32-bit transfers in DMA mode.
2. The HDI16 module asserts its HREQ pin when it is ready for an external DMA access.
3. The host writes the most significant 16-bits of data to the host-side TX2 register (the two-bit DMA counter points to the TX2 register) at host address 0x4.
4. The host writes the least significant 16-bits of data to the same address (host address 0x4). The internal DMA counter increments the address so that the write is performed to the host-side TX3 register (the two-bit DMA counter points to the TX3 register).
5. Completing the second access triggers the transfer of data (**Table 20-15**) to the HORX FIFO.
6. The two-bit DMA counter is loaded from the HCR[HDM] or ICR[HM] bits to point to the TX2 register, depending on the value of HCR[HICR] bit.
7. The device then reads the data from the MSC711x-side HORX register in bits 31–0.

Note: In OAD Mode, the HRD/HWR signal pins are used in read and write DMA accesses.

20.7.6 Forcing DMA Rx Servicing

When the HDI16 module is configured so that the internal DMA controller services it on the device side and the host has finished sending data, the HORX FIFO may not be completely filled. If this happens, the DMA controller does not receive a data transfer request. The external host can use the ICR[LWRT] bit to indicate the write of its last 8- or 16-bit data word to the TXx registers. The host must set the ICR[LWRT] bit immediately before writing its last data value. Then, when the host writes the last data value, the contents of the TXx registers are transferred into the receive FIFO and a DMA data transfer request is initiated.

20.7.7 Host Flags (HF[0–7])

The host flags are for general-purpose communication of additional system-level information between the external host and the MSC711x. There are two sets of host flags, as shown in **Table 20-22**.

Table 20-22. Host Flags

Transfer Direction	Host Flags	Writable by	Readable by
Host-to-MSC711x	HF[0–3]	Host-to-ICR	MSC711x from HSR
MSC711x-to-host	HF[4–7]	MSC711x-to-HCR	Host from ISR

20.7.8 Command Vector

One of the most innovative features of the HDI16 is the host command feature. An external host processor can issue an interrupt request to the SC1400 core, selecting any of 128 possible vectors for execution on an MSC711x device. For example, MSC711x host interrupts can allow the host processor to read or write core registers in data or program memory locations and perform control and debugging operations. The command vector is always requested by the external host and serviced by the MSC711x. The external host first writes the command vector register, CVR, with the CVR[HC] bit set. This value is transferred through the HDI16 to the MSC711x Host Command Vector Register (HCVR), where the SC1400 core reads it to identify the requested interrupt routine. This flexibility allows the host processor to execute up to 128 pre-programmed functions inside the SC1400 core.

The host processor can also issue *non-maskable* interrupts to the SC1400 core by setting the CVR[NMI] bit in addition to setting the CVR[HC] bit when writing the external host CVR. For details on this operation, see the NMI bit description on **page 20-46**. When the external host processor issues a command to the MSC711x device, it can read the CVR[HC] bit to determine whether the interrupt controller in the SC1400 core has accepted the command. When the command is accepted for execution, the interrupt controller clears this bit.

20.7.9 Initializing the HDI16 Module

The external host can force HDI16 initialization by setting ICR[INIT] as described in **Table 20-32**, *Effects of the INIT Command*, on page 20-44.

20.8 MSC711x-Side Programming Model

The SC1400 core treats the HDI16 as a memory-mapped peripheral, employing either standard polled or interrupt-driven programming techniques. Direct memory mapping allows the SC1400 core to access the HDI16 registers using standard instructions and addressing modes. The on-chip DMA controller or the SC1400 core services the transmit and receive FIFOs for data transfers. The SC1400 core accesses all the device-side registers, but the external host cannot access these registers. The value of the base address for the HDI16 register file, HDI16_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4.

Note: The HDI16 programming model addressable by the external host processor is described in **Section 20.9**, *External Host-Side Programming Model*, on page 20-37.

The HDI16 port registers are affected by two different types of reset:

- Peripheral module reset is caused by reset conditions on the MSC711x device (see **Table 13-2**, *Reset Actions for Each Reset Source*, on page 13-2.
- Individual reset is caused by clearing HPCR[HEN] via an APB bus access.

Table 20-23 shows the effects of the reset types on the registers accessible from an MSC711x device. The HDI16 is disabled on all types of reset.

Table 20-23. Device-Side Registers After Reset

Register Name	Page	Register Data	Reset Type	
			Peripheral Module Reset	Individual Reset
Host Control Register (HCR)	**page 20-28**	All bits	0	—[1]
Host Port Control Register (HPCR)	**page 20-33**	All bits	—[3]	—[3]
Host Status Register (HSR)	**page 20-31**	HF[0–3]	0	—[1]
		HTFE	1[4]	1[4]
		HTFNF	1[4]	1[4]
		HRFNE	0	0
		HRFF	0	0
Host Command Vector Register (HCVR)	**page 20-32**	HCP	0	0
		HV	0	—[1]
Host Receive Data Register (HORX)	**page 20-37**	All bits	Empty	Empty
Host Transmit Data Register (HOTX)	**page 20-36**	All bits	Empty	Empty

Notes:
1. A long dash (—) means that the bit value is indeterminate after reset.
2. *Empty* means that the data at this location is invalid (trash).
3. The HPCR register is indeterminate out of reset because the HDSP, H8BIT, and the HDDS bits can be asserted out of reset, depending on the values of their corresponding pins.
4. Internally, the register bit is reset to 1. However, its value is ANDed with that of the HPCR[HEN] bit, so if HEN is not set, this bit reads as a 0. When the HPCR[HEN] bit is set, the value of this bit then reads as 1.

Programmable from Device Side (Non-DMA and DMA Modes)

HCR Host Control Register (HCR) HICR = 0 0x7000

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HF4	HF5	HF6	HF7	HICR	HDM0	HDM1	HDM2	—	DBTE	DBRE	HCIE	HTNFIE	HTEIE	HRFIE	HRNEIE
TYPE	R/W															
RESET	See **Table 20-23**, *Device-Side Registers After Reset,* on page 20-28															

Programmable from External Host Side (DMA Mode Only)

HCR Host Control Register (HCR) HICR = 1 0x7000

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HF4	HF5	HF6	HF7	HICR	RREQ	HM		—	DBTE	DBRE	HCIE	HTNFIE	HTEIE	HRFIE	HRNEIE
TYPE	R/W					R	R		R/W							
RESET	See **Table 20-23**, *Device-Side Registers After Reset,* on page 20-28															

HCR is the register by which the SC1400 core controls the HDI16 operating mode. It is used for the following tasks:

- Configuring programmability from the device or external host side (HICR).
- Writing host flags to the external host (HF[4–7])
- Enabling host port interrupts to the MSC711x device (HCIE–HREIE)
- Enabling DMA bursts (DBTE, DBRE)

See also the HPCR register, which configures the HDI16 modes (**page 20-33**) and the ICR, which is a control register accessible to the external host side (**Section 20.9**, *External Host-Side Programming Model*, on page 20-37).

Note: The function of bits 10–8 in the HCR is determined by the value of HCR[HICR].

Table 20-24. HCR Bit Descriptions

Name	Description	Settings
HF[4–7] 15–12	**Host Flags 4–7** General-purpose flags for MSC711x-to-external host communication that are set or cleared by the SC1400 core. The values of HF[4–7] are reflected in the Interface Status Register (ISR); if MSC711x software modifies them, the host processor can read the modified values by reading the ISR. As general-purpose flags, HF[4–7] can be used individually or as encoded pairs in a simple device-to-external host communication protocol in both the MSC711x and host processor software.	
HICR 11	**ICR/HCR Priority for DMA/Last Address Mode** Selects which register defines DMA/last address mode, ICR or HCR. If HICR is cleared, the HDI16 DMA/last address mode is defined by the HCR[HDM] bits, as described in **Section 20.6.2.3**, *Host DMA Mode,* on page 20-17. If HICR is set, the DMA/last address mode is defined by the values of HM, RREQ, and TREQ in ICR.	0 DMA/last address mode defined in HCR. 1 DMA/last address mode defined in ICR.

Table 20-24. HCR Bit Descriptions (Continued)

Name	Description	Settings
HDM[0–2] (HICR = 0) 10–8	**Host DMA/Last Address Mode Control** When HDI16 DMA mode is enabled (HICR is cleared and the HPCR[DMA] bit is set), HDM defines the HDI16 DMA mode of operation. When the HDI16 is in non-DMA mode (both HICR and HPCR[DMA] are cleared), the HDM bits define the data transfer size, as described in **Section 20.7.2**, *Non-DMA Mode Programmed from External Host Side (HICR = 1),* on page 20-20.	**HDM0 (Bit 10):** 0 Input. Data transfers are from the external host to the MSC711x device. 1 Output. Data transfers are from the MSC711x device to the external host. **HDM[1–2] (Bits 9–8):** 00 Transfer 64-bit DMA data. 01 Transfer 48-bit DMA data. 10 Transfer 32-bit DMA data. 11 Transfer 16-bit DMA data.
RREQ (HICR = 1) 10	**RREQ Status** When HICR is set and HPCR[DMA] is also set, RREQ reflects the status of ICR[RREQ], which indicates the direction of the DMA transfer selected by the host. When HICR is set and the HPCR[DMA] is cleared, this bit is invalid.	
HM (HICR = 1) 9–8	**ICR[HM] Status** When HICR is set, HM reflects the status of the ICR[HM] bits, which indicate the transfer data size.	
— 7	Reserved. Write to zero for future compatibility.	
DBTE 6	**DMA Transmit Burst Enable** Enables/Disables burst mode for DMA transactions to the HDI16. If DBTE is cleared, the HDI16 transmits data in single (non-burst) DMA mode. In this mode, if HSR[HTFNF] is set, the HDI16 generates a transmit data FIFO DMA request. If DBTE is set, HDI16 transmits data in burst DMA mode. In this mode, if HSR[HTFE] is set, the HDI16 generates a transmit data FIFO DMA request.	0 DMA transmit burst mode disabled. 1 DMA transmit burst mode enabled.
DBRE 5	**DMA Receive Burst Enable** Enables/Disables burst mode for DMA transactions from the HDI16. If DBRE is cleared, the HDI16 receives data in single (non-burst) DMA mode. In this mode, if HSR[HRFNE] is set, the HDI16 generates a receive data FIFO DMA request. If DBRE is set, the HDI16 receives data in burst DMA mode. In this mode, if HRFF is set, the HDI16 generates a receive data FIFO DMA request.	0 DMA receive burst mode disabled. 1 DMA receive burst mode enabled.
HCIE 4	**Host Command Interrupt Enable** Generates a host command interrupt request if the host command pending status bit HCP is set in the Host Command Vector Register (HCVR).	0 Host command interrupt disabled. 1 Host command interrupt enabled.
HTNFIE 3	**Host Transmit Not Full Interrupt Enable** Enables an SC1400 interrupt when the host transmit FIFO not full status bit HTFNF is set in the HSR. If both HTNFIE and HTFNF are set, a host transmit data interrupt request is generated.	0 Host transmit not full interrupt disabled. 1 Host transmit not full interrupt enabled.

Table 20-24. HCR Bit Descriptions (Continued)

Name	Description	Settings
HTEIE 2	**Host Transmit Empty Interrupt Enable** Enables an SC1400 interrupt when the host transmit FIFO empty status bit HTFE is set in the HSR. If both HTEIE and HTFE are set, a host transmit data interrupt request is generated.	0 Host transmit empty interrupt disabled. 1 Host transmit empty interrupt enabled.
HRFIE 1	**Host Receive Full Interrupt Enable** Enables an SC1400 interrupt when the host receive FIFO full status bit HRFF is set in the HSR. If both HRFIE and HRFF are set, a host receive FIFO full interrupt request is generated.	0 Host receive full interrupt disabled. 1 Host receive full interrupt enabled.
HRNEIE 0	**Host Receive Not Empty Interrupt Enable** Enables an SC1400 interrupt when the host receive FIFO not empty status bit HRFNE is set in HSR. If both HRNEIE and HRFNE are set, a host receive FIFO not empty interrupt request is generated.	0 Host receive not empty interrupt disabled. 1 Host receive not empty interrupt enabled.

HSR Host Status Register 0x7040

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HF0	HF1	HF2	HF3	—								HTFNF	HTFE	HRFF	HRFNE
TYPE	R															
RESET	See **Table 20-23**, *Device-Side Registers After Reset,* on page 20-28															

HSR is a register that the SC1400 core can read to get the status and flags of the HDI16. The SC1400 core accesses HSR to read host flags from the external host (HF[0–3]) and to read host port FIFO status (HTFNF–HRFNE).

Table 20-25. HSR Bit Descriptions

Name	Description	Settings
HF[0–3] 15–12	**Host Flags 0–3** General-purpose flags for external host-to-MSC711x communication, which the host sets or clears. The values of HF[0–3] are reflected in the Interface Control Register (ICR) on the host side. The HF flags can be used individually or as encoded pairs in a simple external host-to-MSC711x protocol in both the SC1400 core and host processor software.	
— 11–4	Reserved. Write to zero for future compatibility.	

Table 20-25. HSR Bit Descriptions (Continued)

Name	Description	Settings
HTFNF 3	**Host Transmit Not Full** Indicates whether the host transmit data FIFO (HOTX) has room for the SC1400 core to write data. HTFNF is set when data is transferred from the HOTX FIFO to the RX0:RX1:RX2:RX3 registers. HTFNF is cleared when HOTX is full after the SC1400 core has written data. If HTFNF is set and DBTE is cleared, the HDI16 generates a transmit data DMA request for a single transfer. If both HTFNF and HTFIE are set, a host transmit FIFO not full interrupt request is generated. The host processor can also set HTFNF via the initialize function.	0 Host transmit FIFO is full. 1 Host transmit FIFO is not full.
HTFE 2	**Host Transmit Empty** Indicates whether the host transmit data FIFO (HOTX) is empty and the SC1400 core can write data. HTFE is set when the last 64-bit word of data is transferred from the HOTX FIFO to the RX0:RX1:RX2:RX3 registers. HTFE is cleared after the SC1400 core writes data, if HOTX was previously empty. If both HFTE and DBTE are set, the HDI16 generates a transmit data DMA request for a burst transfer. If both HFTE and HTEIE are set, a host transmit FIFO empty interrupt request is generated. The host processor can also set HTFE via the initialize function.	0 Host transmit FIFO is not empty. 1 Host transmit FIFO is empty.
HRFF 1	**Host Receive Full** Indicates whether the host receive data FIFO (HORX) contains four 64-bit words of data from the host processor. HRFF is set when data is transferred from the TX0:TX1:TX2:TX3 registers to the HORX FIFO, if HORX previously contained three 64-bit words of data. HRFF is cleared when the SC1400 core reads HORX, if the FIFO previously contained four 64-bit words of data. If both HRFF and DBRE are set, the HDI16 generates a receive data DMA request for burst transfer. If HRFF is set when HRFIE is set, a host receive data FIFO full interrupt request is generated. The host processor can also clear HRFF via the initialize function.	0 Host receive FIFO is not full. 1 Host receive FIFO is full.
HRFNE 0	**Host Receive Not Empty** Indicates whether the host receive data FIFO (HORX) contains data from the host processor. HRFNE is set when data is transferred from the TX0:TX1:TX2:TX3 registers to the HORX FIFO, if HORX previously contained no data. HRFNE is cleared when the SC1400 core reads HORX, if the FIFO previously contained only one 64-bit word of data. If HRFNE is set and DBRE is cleared, the HDI16 generates a receive data DMA request. If HRFNE is set when HREIE is set, a host receive data FIFO not empty interrupt request is generated. The host processor can also clear HRFNE via the initialize function.	0 Host receive FIFO is empty. 1 Host receive FIFO is not empty.

HCVR Host Command Vector Register 0x7060

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EVIN1	EVIN0	—						HCP	HV						
TYPE	R															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HCVR is a register that the SC1400 core reads to get the host command pending status and the vector value identifying one of 128 possible functions in an interrupt routine. The host command interrupt service routine must read HCVR to clear the interrupt.

Table 20-26. HCVR Bit Descriptions

Name	Reset	Description
EVIN1 15	0	**Host Event Port Input 1** Reflects the status of the EVIN1 bit in the Command Vector Register (CVR). If set, EVIN1 indicates that the host has set the EVIN1 bit. This bit is then used as an input to the event port.
EVIN0 14	0	**Host Event Port Input 0** Reflects the status of the EVIN0 bit in the Command Vector Register (CVR). If set, EVIN0 indicates that the host has set the EVIN0 bit. This bit is then used as an input to the event port.
— 13–8	0	Reserved. Write to zero for future compatibility.
HCP 7	0	**Host Command Pending** Reflects the status of the HC bit in the Command Vector Register (CVR). If set, HCP indicates that the host has set the HC bit and that a host command interrupt is pending. HC and HCP are cleared when the SC1400 core reads HCVR. The host must not clear HC (which also clears HCP) until HC is cleared by reading HCVR on the core side.
HV 6–0	0	**Host Vector** Reflects the status of the HV bit in the CVR. These seven bits indicate a value from 0 to 127 that represents an interrupt routine. The SC1400 core reads the HV value to identify the interrupt routine. The HV bits enable the host programmer to select and execute up to 128 pre-programmed functions inside the SC1400 core.

HPCR Host Port Control Register 0x7020

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HAP	HRP	HCSP	HDDS	—		HDSP	HROD	HEN	H8BIT	HLEND	—			DMA	OAD
TYPE	R/W															
RESET	See **Table 20-23**, *Device-Side Registers After Reset,* on page 20-28															

HPCR is a register by which the SC1400 core controls the HDI16 operating mode. It performs the following tasks:

- Configures HDI16 pin polarity and functionality (HAP–HDSP).
- Enables the HDI16 module (HEN).
- Configures HDI16 modes (H8BIT, HLEND, DMA).
- Configures the DMA (OAD).

To ensure the correct operation of MSC711x devices, the HPCR HAP, HRP, HDDS, HDSP, and H8BIT bits must be changed only if the host enable bit, HEN, is cleared. These bits must not be set when HEN is already set. Also, they must not be set simultaneously with HEN.

Table 20-27. HPCR Bit Descriptions

Name	Description	Settings
HAP 15	**Host Acknowledge Polarity** Determines whether the host acknowledge (HACK) signal is configured as active low or active high.	0 HACK signal is configured as an active low input. 1 HACK signal is configured as an active high input.
HRP 14	**Host Request Polarity** Controls the polarity of the host request signals.	Single host request mode (ICR[HDRQ] cleared): 0 $\overline{\text{HREQ}}$ signal is an active low output. 1 HREQ signal is an active high output. In double host request mode (ICR[HDRQ] set: 0 $\overline{\text{HTRQ}}$ and $\overline{\text{HRRQ}}$ signals are active low outputs. 1 HTRQ and HRRQ signals are active high outputs.
HCSP 13	**Host Chip Select Polarity** Inverts the polarity of both the HCS1 and HSC0 pins *before* the ORing operation occurs.	0 Invert polarity of both $\overline{\text{HCS1}}$ and $\overline{\text{HCS2}}$ pins. That is, the pins become $\overline{\text{HCS1}}$ and $\overline{\text{HCS2}}$. 1 No inversion of polarity for HCS1 and HCS2 pins. That is, the pins are HCS1 and HCS2.
HDDS 12	**Host Dual Data Strobe** Determines whether the HDI16 operates in single-strobe or dual-strobe bus mode. In single-strobe mode, the bus has a single data strobe signal for both reads and writes. When this bit is read, it reflects the OR value between this bit and the corresponding pin. In dual-strobe mode, the bus has two separate data strobes, one for data reads, the other for data writes. See **Figure 20-3**, *Dual-Strobe Bus,* on page 20-9 and **Figure 20-2**, *Single-Strobe Bus,* on page 20-9 for details on the two types of buses.	0 HDI16 operates in single-strobe bus mode. 1 HDI16 operates in dual-strobe bus mode.
— 11–10	Reserved. Write to zero for future compatibility.	
HDSP 9	**Host Data Strobe Polarity** Determines whether the data strobe signals are configured as active high or active low inputs. When this bit is read, it reflects the OR value between this bit and the corresponding pin. The data strobe signals are HDS or HRD or HWR.	0 Data strobe signals are configured as active low inputs, and data is transferred when the data strobe is low. 1 Data strobe signals are configured as active high inputs, and data is transferred when the data strobe is high.
HROD 8	**Host Request Open Drain** Configures HREQ/HTRQ and HRRQ as open drain.	0 Normal (driven) operation. 1 Open-drain operation.
HEN 7	**Host Enable** Determines whether the HDI16 operates as the host interface.	0 HDI16 internal clock is frozen. 1 HDI16 operates as the host interface.
H8BIT 6	**H8-Bit Mode** This bit is ORed with the H8BIT input pin and defines whether the HDI16 operates in 8-bit or 16-bit mode (the default). When this bit is read, it reflects the OR value between this bit and the corresponding pin. If the H8BIT pin is connected to the V_{DD}, the value of H8BIT is ignored. If the H8BIT pin is connected to the ground and H8BIT is set, HDI16 8-bit mode is enabled.	0 16-bit mode is enabled (default). 1 8-bit mode is enabled.

Table 20-27. HPCR Bit Descriptions (Continued)

Name	Description	Settings
HLEND 5	**Host Little-Endian Mode** Configures the host port for big- or little-endian configuration. When this bit is set correctly for endianness on the device, it should not be modified.	0 Big-endian configuration. 1 Little-endian configuration.
— 4–2	Reserved. Write to zero for future compatibility.	
DMA 1	**Host DMA Mode Enable** Enables/Disables HDI16 DMA mode, which supports external DMA controller devices connected to the HDI16 on the host side. When DMA is cleared, the TREQ and RREQ control bits are used for host processor interrupt control via the external HREQ output pin or the HRREQ and HTREQ output pins if HDREQ in ICR is set. When DMA is set, HDI16 operation is determined by the HCR[HDM] bits if the HCR[HICR] bit is cleared or by the HM1, HM0, RREQ and TREQ bits in ICR, if HICR is set. See **Section 20.6.2.3**, *Host DMA Mode,* on page 20-17 for details. **Note:** This mode should not be confused with the operation of the MSC711x DMA controller.	0 Host DMA mode is disabled. 1 Host DMA mode is enabled.
OAD 0	**One-Address Host DMA Mode Enable** Enables/Disables one-address host DMA mode operation. If OAD is cleared and the DMA direction is MSC711x-to-external host, the contents of the selected register are written to the host data bus when HACK is asserted. If the DMA direction is external host-to-MSC711x, the selected register is read from the host data bus when HACK is asserted. If OAD is set and the DMA direction is MSC711x-to-external host, the contents of the selected register are written to the host data bus when a read operation at host address 0x4 is asserted. If the DMA direction is external host-to-MSC711x, the selected register is read from the host data bus when a write operation at host address 0x4 is asserted.	0 HACK input pin is used as a DMA transfer acknowledge input. 1 Host address 0x4 is used as a host DMA transfer acknowledge input.

HOTX Host Transmit Data Register 0x7080

Bit	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	Data Value															
TYPE	W															
Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
	Data Value															
TYPE	W															
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Data Value															
TYPE	W															
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Data Value															
TYPE	W															

HOTX handles MSC711x-to-external host data transfers. The MSC711x devices treat this register as a write-only FIFO with a capacity of four 64-bit words. Writing to HOTX clears the Host Transfer Data Empty bit (HSR[HTFE]) if HOTX was previously empty, or it clears HTFNF if HOTX previously contained three 64-bit words. The contents of HOTX are transferred as 64-bit data to the receive data registers (RX0:RX1:RX2:RX3) when both the following bits are cleared:

- Receive data register full ISR[RXDF] on the host side.
- Host transmit data empty HSR[HTFE] on the core side.

This transfer operation sets the ISR[RXDF] bit and the HSR[HTFE] bit, if HOTX previously contained only one 64-bit word, or HSR[HTFNF], if HOTX was previously full. The SC1400 core can cause a host transmit data interrupt by setting either:

- HCR[HTFIE] when HSR[HTFNF] is set.
- HCR[HTEIE] when HSR[HTFE] is set.

To prevent data overwrites, do not write data to HOTX in burst mode unless the HSR[HTFE] bit is set or in single-transfer mode unless HSR[HTFNF] is set. In addition, a DMA channel can be programmed to write to HOTX in burst mode when HSR[HTFE] is set and DBTE is set, or in single-transfer mode when HSR[HTFNF] is set and DBTE is cleared.

HORX Host Receive Data Register 0x70A0

Bit	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
	Data Value															
TYPE	R															
Bit	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
	Data Value															
TYPE	R															
Bit	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
	Data Value															
TYPE	R															
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Data Value															
TYPE	R															

HORX handles external host-to-MSC711x data transfers. The MSC711x treats this register as a read-only FIFO that can receive four 64-bit words. HORX is loaded with 64-bit data from the transmit data registers on the host side (TX0:TX1:TX2:TX3) when both the following bits are cleared:

- Transmit data register empty TXDE on the host side (ISR).
- Host receive data full HRFF on the core side (HSR).

Reading HORX clears HSR[HRFF] if the FIFO was previously full, or it clears HSR[HRFNE] if the FIFO previously contained one 64-bit word. When data is transferred, the ISR[TXDE] bit is set and either HSR[HRFNE] is set if HORX was previously empty or HSR[HRFF] is set if HORX previously contained three 64-bit words. The MSC711x may cause a host receive data interrupt by setting either:

- HCR[HRFIE] when HSR[HRFF] is set.
- HCR[HREIE] when HSR[HRFNE] is set.

In addition, a DMA channel can be programmed to read HORX either when HSR[HRFNE] is set and DBRE is cleared or when HRFF is set and DBRE is set, meaning that HORX contains valid data.

20.9 External Host-Side Programming Model

Host processors use standard host processor instructions and addressing modes, for example, byte move, to communicate with the HDI16 registers. The HDI16 registers are aligned so that 16-bit host processors can use 16/32/48/64-bit load and store instructions for data transfers. The HDI16 external host-side programming model is affected by whether the module is configured for big-

or little-endian operation. **Table 20-28** shows the external host HDI16 registers configured as big-endian and **Table 20-29** shows the external host HDI16 registers configured as little-endian. These registers are *not* accessible as part of the MSC711x memory map. They are accessible only to an external host through the host port pins. The HDI16 programming model addressable by the devices master(s) is described in **Section 20.8**, *MSC711x-Side Programming Model*, on page 20-27.

Table 20-28. External Host-Side HDI16 Registers Configured as Big-Endian

Name	Host Address (HA[0–3])	Description	Page Number
Interface Control Register (ICR)	0x0	Control register for the HDI16 port.	**page 20-39**
Interface Status Register (ISR)	0x2	Status register for the HDI16 port.	**page 20-44**
Command Vector Register (CVR)	0x1	Allows an external host processor to select one of 128 possible interrupt routines to execute.	**page 20-46**
Transmit Word Data Registers (TX[0–3])	0: 0x4 1: 0x5 2: 0x6 3: 0x7	Write-only registers written by an external host processor for transmitting data to the MSC711x device. TX0 is the MS portion of the data.	**page 20-47**
Receive Word Data Registers (RX[0–3])	0: 0x4 1: 0x5 2: 0x6 3: 0x7	Read-only registers read by an external host processor for receiving data from an MSC711x device. RX0 is the MS portion of the data.	**page 20-47**

Table 20-29. External Host-Side Registers Configured as Little-Endian

Name	Host Address (HA[0–3])	Description	Page Number
Interface Control Register (ICR)	0x0	Control register for the HDI16 port.	**page 20-39**
Interface Status Register (ISR)	0x2	Status register for the HDI16 port.	**page 20-44**
Command Vector Register (CVR)	0x1	Allows an external host processor to select one of 128 possible interrupt routines to execute.	**page 20-46**
Transmit Word Data Registers (TX[0–3])	0: 0x7 1: 0x6 2: 0x5 3: 0x4	Write-only registers written by an external host processor for transmitting data to the MSC711x device. TX0 is the MS portion of the data.	**page 20-47**
Receive Word Data Registers (RX[0–3])	0: 0x7 1: 0x6 2: 0x5 3: 0x4	Read-only registers read by an external host processor for receiving data from the MSC711x device. RX0 is the MS portion of the data.	**page 20-47**

The HDI16 port registers are affected by two different types of reset:

- Peripheral module reset is caused by reset conditions on the MSC711x device (see **Table 13-2**, *Reset Actions for Each Reset Source*, on page 13-2).

- Individual reset is caused by clearing HPCR[HEN].

Table 20-30 shows the effects of the reset types on the registers accessible to the external host.

Table 20-30. External Host-Side Registers After Reset

Register Name	Register Data	Reset Type	
		Peripheral Module Reset	Individual Reset
ICR	**All Bits**	**0**	**—**
CVR	NMI	0	0
	HC	0	0
	HV[6–0]	0	—
ISR	TXDE32	1	1
	TXDE16	1	1
	RXDF32	0	0
	RXDF16	0	0
	HREQ	0	1 if TREQ is set; 0 otherwise
	RX[0–3]	Empty	Empty
	TX[0–3]	Empty	Empty

Programmable from External Host Side, Non-DMA Mode

ICR Interface Control Register (DMA = 0, HICR = 1) 0x0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HRRA		HTRA		—	HF0	HF1	LWRT	INIT	HM		HF2	HF3	HDRQ	TREQ	RREQ
TYPE	R/W															
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

Programmable from External Host Side, DMA Mode

ICR Interface Control Register (DMA = 1, HICR = 1) 0x0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HRRA		HTRA		—	HF0	HF1	LWRT	INIT	HM		HF2	HF3	—	TREQ	RREQ
TYPE	R/W															
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

Programmable from MSC711x Side, Non-DMA Mode

ICR Interface Control Register (DMA = 0, HICR = 0) 0x0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HRRA		HTRA		—	HF0	HF1	LWRT	INIT	HDM		HF2	HF3	HDRQ	TREQ	RREQ
TYPE	R/W									R		R/W				
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

Programmable from MSC711x Side, DMA Mode

ICR Interface Control Register (DMA = 1, HICR = 0) 0x0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HRRA		HTRA		—	HF0	HF1	LWRT	INIT	HDM		HF2	HF3	—	Note 1	Note 2
TYPE	R/W									R		R/W			R/W	
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

Notes: **1.** $\overline{\text{HDM0}}$ when read, TREQ when written

2. HDM0 when read, RREQ when written

ICR allows the use of bit manipulation instructions on control register bits. The host processor uses the ICR to control the HDI16 interrupts and flags. The SC1400 core cannot access the ICR. The HPCR[DMA] bit and the HCR[HICR] bit control the function of the ICR bits. See also the HPCR register, which configures the HDI16 modes (**page 20-33**) and the HCR, a control register accessed on the MSC711x side (**page 20-28**).

Table 20-31. ICR Bit Descriptions

Name	Description	Settings
HRRA 15–14	**HDI Receive Request Assertion** The receive request is asserted for a specified time based on the number of data bytes to receive.	00 HRRQ asserted for 8 bytes (RX full). 01 HRRQ asserted for 16 bytes (RX full + 1 HOTX entry full). 10 HRRQ asserted for 32 bytes (RX full + 3 HOTX entries full). 11 Reserved.
HTRA 13–12	**HDI Transmit Request Assertion** The transmit request is asserted a specified time based on the number of data bytes to transmit.	00 HTRQ/HREQ asserted for 8 bytes (TX empty). 01 HTRQ/HREQ asserted for 16 bytes (TX empty + 1 HOTX entry empty). 10 HTRQ/HREQ asserted for 32 bytes (TX empty + 3 HOTX entries empty). 11 Reserved.
— 11	Reserved. Write to zero for future compatibility.	

Table 20-31. ICR Bit Descriptions (Continued)

Name	Description	Settings
HF0 10	**Host Flag 0** A general-purpose flag for external host-to-MSC711x communication. The host processor can set or clear HF0. HF0 is reflected in the HSR on the core side of the HDI16.	
HF1 9	**Host Flag 1** A general-purpose flag for external host-to-MSC711x communication. The host processor can set or clear HF1. HF1 is reflected in the HSR on the core side of the HDI16.	
LWRT 8	**Last Write** The external host processor sets this bit to indicate to the HDI16 module that the end of a frame has been transmitted. In response, the HORX generates a DMA data transfer request, even if the FIFO is not full. LWRT is automatically cleared when the HORX FIFO empties. See	
INIT 7	**Force Initialization** The host processor uses this bit to force initialization of the HDI16 hardware (which may or may not be necessary, depending on the software design of the interface). During initialization, the HDI16 transmit and receive control bits are configured. The type of initialization performed when the INIT bit is set depends on the state of TREQ and RREQ in the HDI16. The INIT command, which is local to the HDI16, conveniently configures the HDI16 into the desired data transfer mode. See **Table 20-32**, *Effects of the INIT Command,* on page 20-44. When the host sets the INIT bit, the HDI16 hardware executes the INIT command. The interface hardware clears the INIT bit after the command executes.	

Table 20-31. ICR Bit Descriptions (Continued)

Name	Description	Settings
HM/HDM 6–5	**Host Mode/Host DMA Mode** When host DMA mode is enabled, if HCR[HICR] is set, the HREQ pin requests DMA transfers, the TREQ and RREQ bits select the direction of DMA transfers, and the HACK input pin is used as a DMA transfer acknowledge input, if OAD in HPCR is cleared. If the DMA direction is from SC1400 core-to-host, the contents of the selected register are written to the host data bus when HACK is asserted. If the DMA direction is from host-to-SC1400 core, the selected register is read from the host data bus when HACK is asserted. If HPCR[OAD] is set, a host read or write to host address 0x4 is used as a DMA transfer acknowledge. If the DMA direction is from SC1400 core-to-host, the contents of the selected register are written to the host data bus when the host reads from host address 0x4. If the DMA direction is from host-to-SC1400 core, the selected register is read from the host data bus when the host writes to host address 0x4. HM also controls the size of the DMA word to be transferred. The HDI16 data register selected during a host DMA transfer is determined by a 2-bit address counter, which is preloaded with the value in HM. The address counter replaces the HA[0–1] bits of the HDI16 during a host DMA transfer. The address counter can be initialized with the INIT bit feature. After each DMA transfer on the host data bus, the address counter decrements. When the address counter reaches the last register, the address counter is loaded with the value in HM. Thus, 16-bit, 32-bit, 48-bit, or 64-bit data can be transferred in a circular fashion, and the need is eliminated for the DMA controller to supply the HA[3–0] pins (HPCR bit OAD = 0) or to read/write at host address 0x4 (HPCR bit OAD = 1). For 32-, 48- or 64-bit data transfers, the core CPU interrupt rate is reduced by a factor of 2, 3, or 4, respectively, from the host request rate. That is, for every two or three host processor data transfers of one byte each, there is only one 64-bit core CPU interrupt. This bit is available only in ICR mode (HCR[HICR] = 1). When the HDI16 is in ICR priority non-DMA mode (the HPCR[DMA] bit is cleared and HCR[HICR] is set), data transfer size is defined by HM, as described in **Table 20-14**, *Non-DMA Trigger Addresses (from External Host Side),* on page 20-21. The transfer size causes the RX*x*/TX*x* register read/write at the last (trigger) address to clear the RXDF/TXDE bits, respectively.	
HF2 4	**Host Flag 2** A general-purpose flag for external host-to-MSC711x communication. The host processor can set or clear HF2. HF2 is reflected in the HSR on the core side of the HDI16.	
HF3 3	**Host Flag 3** A general-purpose flag for external host-to-MSC711x communication. The host processor can set or clear HF3. HF3 is reflected in the HSR on the core side of the HDI16.	

Table 20-31. ICR Bit Descriptions (Continued)

Name	Description	Settings
HDRQ 2	**HREQ/HTRQ and HACK/HRRQ Pin Control** Controls the HREQ/HTRQ and HACK/HRRQ pins. If HDRQ is cleared, the HREQ/HTRQ pin functions as a single HREQ. If HDRQ is set, the HREQ/HTRQ and HACK/HRRQ pins function as HTRQ and HRRQ, respectively. This bit is available only in non-DMA (interrupt) mode (HPCR[DMA] = 0).	
TREQ/ **HDM0** 1	**HREQ/HTREQ Pin Control** Controls the HREQ/HTREQ pin for host transmit data transfers. In non-DMA (interrupt) mode (DMA = 0 in the HPCR), TREQ enables host requests via the host request (HREQ or HTRQ) pin when the Transmit Data Register Empty (TXDE) status bit in the ISR is set. If TREQ is cleared, TXDE interrupts are disabled. If TREQ and TXDE are set, the host request pin is asserted as shown in **Table 20-8**. In DMA modes (HPCR[DMA] = 1 and HCR[HICR] = 1), software must set or clear TREQ to select the direction of DMA transfers. Setting TREQ sets the direction of the DMA transfers as external host-to-MSC711x and enables the HREQ pin to request data transfers. When HCR[HICR] is cleared and HPCR[DMA] is set, a TREQ read reflects the status of the inverted value of HCR[HDM0]. When written, TREQ affects the INIT mode. See **Table 20-32**.	
RREQ/ **HDM0** 0	**HREQ and HRREQ Pin Control** Controls the HREQ and HRREQ pins for host receive data transfers. In non-DMA (interrupt) mode (HPCR[DMA] = 0), RREQ enables host requests via the host request (HREQ or HRRQ) pin when the Receive Data Register Full (RXDF) status bit in the ISR is set. If RREQ is cleared, ISR[RXDF] interrupts are disabled. If RREQ is set, the host request pin (HREQ or HRRQ) is asserted if ISR[RXDF] is set. This is shown in **Table 20-8**. In host DMA mode (DMA = 1 in the HPCR and HICR = 1 in the HCR), RREQ must be set or cleared by software to select the direction of DMA transfers. Setting RREQ sets the direction of the host DMA transfers to MSC711x-to-external host and enables the HREQ pin to request data transfers. When HCR[HICR] is cleared and HPCR[DMA] is set, an RREQ read reflects the status of HCR[HDM0]. When written, RREQ affects the INIT mode. See **Table 20-32**.	

Table 20-32 shows the effects of the INIT command in relation to the values of TREQ and RREQ.

Table 20-32. Effects of the INIT Command

TREQ	RREQ	After INIT Execution	Transfer Direction Initialized
0	0	**INIT = 0**	**None**
0	1	INIT = 0; ISR[RXDF] = 0; HSR[HTFE/HTFNF] = 1	MSC711x to External host
1	0	INIT = 0; ISR[TXDE] = 1; HSR[HRFF/HRFNE] = 0	External host to MSC711x
1	1	INIT = 0; ISR[RXDF] = 0; HSR[HTFE/HTFNF] = 1; ISR[TXDE] = 1; HSR[HRFF/HRFNE] = 0	Reserved

The ICR HDRQ, TREQ, and RREQ bits are also used to set up the HREQ/HTRQ and HRRQ signals (see **Section 20.5.2.2**, *Host Request Pin Configuration*, on page 20-10).

ISR Interface Status Register 0x2

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EV1	EV0	—		TXDE32	TXDE16	RXDF32	RXDF16	HREQ	HF4	HF5	HF6	HF7	TRDY	TXDE	RXDF
TYPE	R															
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

ISR is a register by which the host processor interrogates the HDI16 status and flags. The host processor can write to this address without affecting the internal state of the HDI16. The SC1400 core cannot access ISR.

Table 20-33. ISR Bit Descriptions

Name	Description	Settings
EV1 15	**Event Port Output 1**	0 EVGP1 signal from the event port is deasserted. 1 EVGP1 signal from the event port is asserted.
EV0 14	**Event Port Output 0**	0 EVGP0 signal from the event port is deasserted. 1 EVGP0 signal from the event port is asserted.
— 13–12	Reserved. Write to zero for future compatibility.	
TXDE32 11	**Transmit Queue 32 Bytes Empty**	0 < 32 bytes are empty. 1 32 bytes are empty (24 bytes in HORX and 8 bytes in TX).
TXDE16 10	**Transmit Queue 16 Bytes Empty**	0 < 16 bytes are empty. 1 16 bytes are empty (8 bytes in HORX and 8 bytes in TX).
RXDF32 9	**Receive Queue 32 Bytes Full**	0 < 32 bytes are full. 1 32 bytes are full (24 bytes in HOTX and 8 bytes in RX).

Table 20-33. ISR Bit Descriptions (Continued)

Name	Description	Settings
RXDF16 8	**Receive Queue 16 Bytes Full**	0 < 16 bytes are full. 1 16 bytes are full (8 bytes in HOTX and 8 bytes in RX).
HREQ 7	**HREQ Status** Indicates the status of the external transmit and receive request output signals (HTRQ and HRRQ) if HDRQ is set. If HDRQ is cleared, HREQ indicates the status of the external host request output signal (HREQ). The HREQ bit can be set under either or both of two conditions: the Receive Word Registers (RX[0–3]) are full or the Transmit Word Registers (TX[0–3]) are empty. These conditions are indicated by the ISR RXDF and TXDE status bits, respectively. If the interrupt source is enabled by the associated request enable bit in the ICR, HREQ is set if one or more of the two enabled interrupt sources is set.	If HDRQ is cleared: 0 No host processor interrupts requested. 1 Interrupt requested. If HDRQ is set: 0 No host processor interrupts requested (HTRQ and HRRQ cleared). 1 Interrupt requested (HTRQ or HRRQ set).
HF[4–7] 6–3	**Host Flags 4–7** Indicates the state of host flags 4–7 in the HCR on the core side. Only the SC1400 core can change HF[4–7].	
TRDY 2	**TRDY Status** Indicates whether the Transmit Word Registers TX[0–3] and the HORX FIFO are empty. TRDY is set if TXDE is set and HRFNE is cleared. If TRDY is set, the data that the host processor writes to TX[0–3] is immediately transferred to the core side of the HDI16 and can be read by the SC1400 core. This feature has many applications. For example, if the host processor issues a host command that causes the SC1400 core to read HORX, the host processor can be certain that the data it just transferred to the HDI16 is the same data received by the SC1400 core.	0 TX[0–3] and the HORX FIFO are not empty. 1 TX[0–3] and the HORX FIFO are empty.
TXDE 1	**Transmit Data Empty** Indicates whether the Transmit Word Registers (TX[0–3]) are empty and can be written by the host processor. TXDE is set when the contents of the TX[0–3] registers are transferred to the HORX register. TXDE is cleared when the host processor writes to the transmit data registers (TX) and the HORX FIFO is full. TXDE is useful for polling and operates independently of the ICR[TREQ] bit. TXDE can also assert the external HREQ/HTRQ pin if the TREQ bit is set. The host processor sets TXDE using the initialize function.	0 The TX registers are not empty. 1 The TX registers are empty and can be written by the host processor.

Table 20-33. ISR Bit Descriptions (Continued)

Name	Description	Settings
RXDF 0	**Receive Data Full** Indicates whether the Receive Word Registers (RX[0–3]) contain data from the SC1400 and can be read by the host processor. RXDF is set when the HOTX is transferred to the RX[0–3] registers. RXDF is cleared when the host processor reads the receive data registers (RX) at the last address and the HOTX FIFO is empty. RXDF is useful for polling and operates independently of the ICR[RREQ] bit. RXDF can also assert the external HREQ or HRRQ pins if the RREQ bit is set. The host processor can clear RXDF using the initialize function.	0 The RX registers are not full. 1 The RX registers are full and can be read by the host processor.

CVR Command Vector Register 0x1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EVIN1	EVIN0	—					NMI	HC	HV						
TYPE	R/W															
RESET	See **Table 20-30**, *External Host-Side Registers After Reset,* on page 20-39															

CVR is a special register by which the host processor issues commands to the SC1400 core. Only the host processor can access this register, and its contents are reflected in the HCVR to the core side. The host processor uses the CVR to cause the SC1400 core to execute an interrupt. The host command feature is independent of any data transfer mechanisms in the HDI16. It can be used to invoke execution of any of the 128 possible indirect interrupt routines in the SC1400 core, identified by the value of HCVR[HV].

Table 20-34. CVR Bit Descriptions

Name	Description
EVIN1 15	**Host Event Port Input 1** The host processor uses this bit to generate a trigger for the device event port. For example, a host processor could write to this bit to generate a DMA data transfer.
EVIN0 14	**Host Event Port Input 0** The host processor uses this bit to generate a trigger for the device event port. For example, a host processor could write to this bit to generate a DMA data transfer.
— 13–9	Reserved. Write to zero for future compatibility..
NMI 8	**Non-Maskable Interrupt** Used by the host processor to handshake the execution of non-maskable interrupts (NMIs). Normally, the host processor sets both NMI and HC to request an NMI from the SC1400 core. When the SC1400 core acknowledges the NMI, the host software reading HCVR clears the NMI and HC bits. The host processor can read the state of NMI to determine when the NMI is accepted. After setting NMI and HC, the host must not write to the CVR again until the host software reading HCVR clears both HC and NMI. Setting NMI and HC causes the host command HC pending bit HCP to be set.

Table 20-34. CVR Bit Descriptions

Name	Description
HC 7	**Host Command** Used by the host processor to handshake the execution of host command interrupts. Normally, the host processor sets HC to request a host command interrupt from the SC1400 core. When the SC1400 core acknowledges the host command interrupt, the host software reading HCVR clears HC. The host processor can read the state of HC to determine when the host command is accepted. After setting HC, the host must not write to the CVR again until the host software reading HCVR clears HC. Setting HC causes the host command pending bit HCP in HSR to be set. The host can write to HC and HV in the same write cycle.
HV 6–0	**Host Vector** The seven HV bits are reflected in the core-side Host Command Vector Register (HCVR), which the SC1400 core can read via an interrupt routine, enabling the host programmer to execute any of 128 preprogramming functions inside the SC1400 core. The host can write HC and HV, or NMI and HV, in the same write cycle.

RX*x* Receive Word Registers (RX[0–3]) (RX0) 0x7
(RX1) 0x6
(RX2) 0x5
(RX3) 0x4

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Data Value															
TYPE	R															

RX[0–3] receive 64-bit data from the HOTX FIFO and are selected by the external host address inputs (HA[3–0]) during a host processor read operation. The receive word registers contain valid data when the receive data register full (ISR[RXDF]) bit is set. The host processor programs the RREQ bit to assert the external HREQ pin when ISR[RXDF] is set. This assertion informs the host processor that the receive word registers are full. When the HOTX FIFO is empty and ISR[RXDF] is set, reading the data register at the host last address clears the ISR[RXDF] bit.

TX*x* Transmit Word Registers (TX[0–3]) (TX0) 0x7
(TX1) 0x6
(TX2) 0x5
(TX3) 0x4

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Data Value															
TYPE	W															

TX[0–3] send 64-bit data to the HORX FIFO and are selected by the external host address inputs (HA[3–0]) during a host processor write operation. Data can be written into the transmit word registers when the Transmit Data Empty (ISR[TXDE]) bit is set. The host processor programs the TREQ bit to assert the external HREQ pin when ISR[TXDE] is set. This assertion informs the host

processor that the transmit word registers are empty. When the HORX FIFO is full and ISR[TXDE] is set, writing the data register at the host last address clears the ISR[TXDE] bit. The transmit registers are transferred as 64-bit data to the HORX FIFO when both the ISR[TXDE] and HSR[HRFF] bits are cleared.

Timers Module 21

There are two identical quad timer modules on MSC711x devices, timer module A and timer module B. The quad timer module contains four identical counter/timer groups that serve as frequency dividers, clock generators, and event counters. Each 16 bit counter/timer group contains a prescaler, a counter, a load register, a hold register, a capture register, two compare registers, and two Status and Control registers. All of the registers except the prescaler are read/writable. This document uses the terms *timer* and *counter* interchangeably because the counter/timers may perform either or both tasks. The counters are also called *channels*. The timers interface with the MSC711x architecture and with the device pins through the event port.

21.1 Features

Features of the timers are as follows:

- Counters are cascadable, can be preloaded, count once or repeatedly, share input pins, do capture and compare, count up or down.
- Count modulo is programmable.
- Maximum count rate is the input clock rate when the timer input signals are not in use.
- Maximum count rate is half the input clock rate when the timer input signals are in use.
- Each counter has a separate prescaler.

21.2 Timer Module Signals

The timer interfaces to the MSC711x architecture and to the pins of the device through the event port. The input clock shown in the block diagram in **Figure 21-1** connects to the TIMER CLOCK signal, which is generated by the MSC711x clock synthesis module. The generation of the TIMER CLOCK signal is shown in **Figure 11-1**, *MSC711x Timing System*, on page 11-2.

21.2.1 Timer Input Signals, TIN[0–3]

The four timer input signals, TIN[0–3] shown in **Figure 21-1** connect as follows to output signals from the event port event multiplexers:

- TIN0 connects to EVOUT0.
- TIN1 connects to EVOUT1.
- TIN2 connects to EVOUT2.

- TIN3 connects to EVOUT3.

The timer module uses these event port output signals to count or measure the width of external events. TIN[0–3] are sampled at the input clock rate before they are sent to the primary and secondary clock multiplexers. The maximum count rate on TIN[0–3] is one-half the input clock rate.

21.2.2 Timer Output Signals, TOUT[0–3]

The four output signals from timer module A, TOUT[0–3], connect as inputs to the event port event multiplexers. The four output signals from timer module B, TOUT[0–3], connect to the TDM modules for optional clocking.

21.3 Timer Module Architecture

Each quad timer module contains four channels, that is, four separate counters. The block diagram of one channel within a quad timer module is shown in **Figure 21-1**.

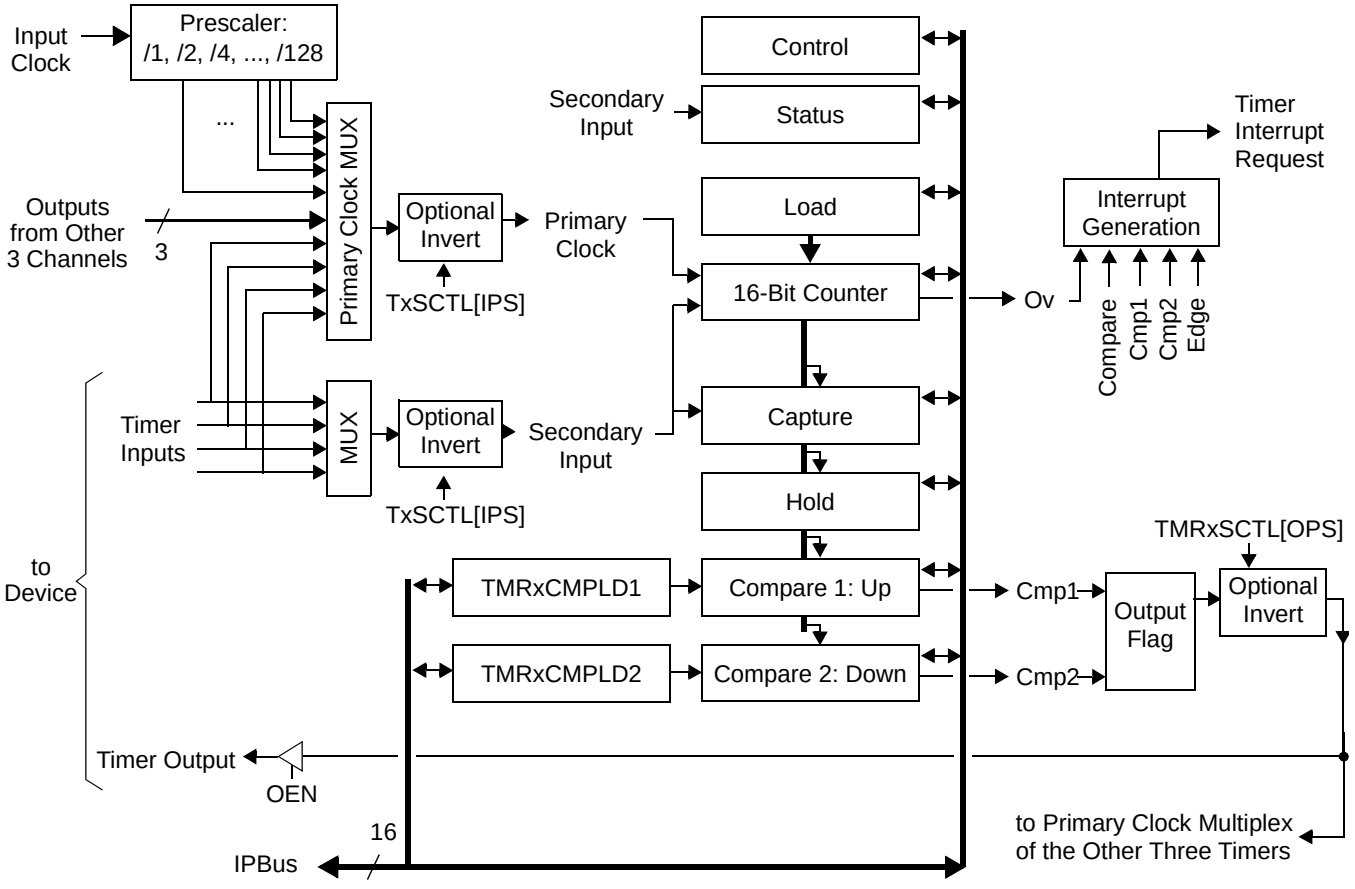


Figure 21-1. Timer Module Block Diagram, One of Four Channels

The channel architecture contains the following components:

- Primary clock selection
- Secondary input selection
- Counter
- Registers:
  - Control and status registers
  - Capture registers
  - Compare registers
- Compare unit
- Interrupt generation
- Output generation

21.3.1 Primary Clock Selection

The primary clock selection block contains a prescaler, a primary clock multiplexer, and an optional invert. It selects and optionally inverts a clock source for the primary clock. The primary clock can be selected from any of the following:

- Normal clocking:
  - Input clock.
  - Input clock /1, /2, /4, ..., /128 (using the prescaler block).
- Clocking from external events: any of the four timer input signals, TIN[0–3].
- Clocking in cascaded mode: an output from another channel (for cascaded operation).

The prescaler is used to generate the /2, /4, /8, ..., /128 function for the input clock.

21.3.2 Secondary Input Selection

The secondary input selection block contains a secondary input multiplexer and an optional invert. It selects and optionally inverts a secondary input, which can be selected from any of the four timer input signals, TIN[0–3].

21.3.3 Counter

A configurable up/down counter, TMRxCNTR, provides the counting capability for the channel. It is reinitialized when the terminal count value is reached. The terminal count value can be programmed as either a compare register or a counter rollover. The load register, TMRxLOAD, provides the initialization value to the counter when the counter terminal value is reached.

21.3.4 Control and Status Registers

Control registers (TMRxCTL and TMRxSCR) are used to configure the operation of the channel:

- Input source selection:
  - Primary count source
  - Secondary input source
- Counter modes:
  - Count mode
  - Count once or repeatedly
  - Selectively use compare value to reinitialize counter
  - Up/down counting
- Broadcast mode (one counter forces use of other counters)
- Capture mode
- Interrupt enables
- Input and output polarities
- Output mode selection
- Output enable

The TMRxSCR also provides readable status on the following:

- Timer compare flag
- Timer overflow flag
- Input edge flag
- Secondary input (allows reading of the secondary input signal)

21.3.5 Capture Registers

There are two mechanisms for capture on the quad timer module:

- *Normal capture function*, available via the capture register, TMRxCAP, which takes a snapshot of the current counter value when the secondary input signal is asserted. The capture unit functionality is covered in detail in **Section 21.6.1.1**, *Capture Register Use*, on page 21-14.
- Capture read of a cascaded counter, available via the hold registers, TMRxHOLD, which capture the counter value the instant any counter register is read so that cascaded counters can be read.

21.3.6 Compare Unit

The TMRCMP1 and TMRCMP2 registers provide the values with which the counter is compared. If a match occurs:

- The OFLAG signal can be set, cleared, or toggled.
- An interrupt is generated, if enabled.
- If enabled, the new compare value is loaded into the CMP1 or CMP2 registers from CMPLD1 and CMPLD2, respectively.
- If enabled, stops the counter.
- If enabled, reinitializes the counter.

The compare unit functionality is covered in detail in **Section 21.3.6**, *Compare Unit*, on page 21-5 and **Section 21.3.6**, *Compare Unit*, on page 21-5.

21.3.7 Interrupt Generation

Several conditions within a channel which can generate a quad timer module interrupt:

- Compare
- Compare 1
- Compare 2
- Overflow
- Input edge

These conditions are ORed together to create a single interrupt request for each timer channel within the quad timer module. The interrupt unit functionality is covered in detail in **Section 21.7**, *Resets and Interrupts*, on page 21-15.

21.3.8 Output Generation

The primary output of each timer/counter is the output signal, OFLAG, which can be programmed for any of the following:

- When the counter reaches the programmed compare value, OFLAG is:
  - set
  - cleared
  - toggled
- OFLAG is asserted while the counter is active.
- OFLAG is an enabled gated clock while the counter is active.
- OFLAG is toggled on alternating compare registers (used for Variable Frequency PWM mode)

- OFLAG is:
  - Set on a successful compare.
  - Reset on a secondary input signal edge.
- OFLAG is:
  - Set on a successful compare.
  - Reset on a counter rollover.

The polarity of the OFLAG output signal is programmable. This signal is sent outside the quad timer module via its corresponding TOUTx signal. See **Section 21.2**, *Timer Module Signals,* on page 21-1 for information on how the output signals are connected at the device level. The OFLAG output signal enables each counter to generate square waves (PWM) or pulse stream outputs.

Before a counter is enabled, you can initialize OFLAG by writing the desired initialization value to TMRxSCR[VAL] and then setting TMRxSCR[FORCE]. Read the descriptions of these bits for more information.

21.4 Setting up Counters for Cascaded Operation

To create a counter larger than 16 bits, the first counter is programmed for the desired configuration and count mode (it is *not* programmed in Cascade mode). This first counter also programs the TMRxCTL[PCS] field to select the desired primary clock. All other counters in the cascade are simply programmed with their count mode set to Cascade mode (TMRxCTL[CM] = 111). They also program the TMRxCTL[PCS] field using the appropriate Counter N output so that each can receive their clocks from the previous counter in the chain. The first counter in the chain must never be programmed in cascade mode. Also, the first counter in the chain must not choose one of the outputs from the timers for its primary clock.

All counters in the cascade follow the counting mode of the first counter in the chain. In Cascade mode, a special high-speed signal path is used, bypassing the timer output flag logic to ensure that the cascaded channels operate as a single synchronous counter.

You can connect counters using the other (non-cascade) counter modes and selecting the outputs of other counters as a clock source. In this case, the counters operate in a *ripple* mode, where higher-order counters transition a clock later than in a purely synchronous design. This is *not* the typical use for cascaded counters.

21.4.1 Operation of the Cascaded Counter

If the first counter in a cascaded chain is counting up and it encounters a compare event, the counter connected to it is incremented. If the first counter in the chain is counting down and it encounters a compare event, the counter is decremented. You can correctly read all 16-bit portions of a cascaded counter as follows using the TMRxHOLD registers:

1. Read any 16-bit portion of the cascaded timer from its TMRxCNTR register. You can do this at any time.
2. When any TMRxCNTR register in the module is read, all other counters simultaneously load their values into their hold registers.
3. Read the 16-bit portions of all other counters in the cascade from their TMRxHOLD registers.

21.4.2 Cascading Restrictions

To ensure that there are no feedback loops in a cascade, there are restrictions on which timers can be cascaded. The counter with the lowest number must always be the first in the cascade, the counter with the second lowest number must be second, and so on. The counter with the highest number must always be last in the cascade. **Table 21-1** summarizes the cascading restrictions.

Table 21-1. Restrictions On Cascading Timers

Counter Number	Valid Cascade Inputs	Legal Values for Cascading using TMRxCTL[PCS]	Description
Counter 0	None	None	Counter 0 can only be the first counter in a cascaded counter. It cannot receive another counter's output for cascaded operation. Counter 0 must always be the first counter in the cascade.
Counter 1	Counter 0 output	0100: Counter 0	Counter 1 can be cascaded with Counter 0, with Counter 0 as the first counter in the chain.
Counter 2	Counter 0 output Counter 1 output	0100: Counter 0 0101: Counter 1	Counter 2 can be cascaded with Counter 0 or Counter 1 when Counter 2 is not the first counter in the cascade.
Counter 3	Counter 0 output Counter 1 output Counter 2 output	0100: Counter 0 0101: Counter 1 0110: Counter 2	Counter 3 can be cascaded with Counter 0, Counter 1, or Counter 3. Counter 3 must always be the last counter in a cascade.

21.5 Timer Operating Modes

The counter/timer operates in two modes:

- Count one of the available MSC711x clock sources using the primary clock.
- Count one of the available MSC711x clock sources using the primary clock while a second input signal, the secondary clock, is asserted, thus timing the width of the secondary clock signal.

Each channel can be configured in the following ways:

- to count the rising, falling, or both edges of the selected input pin.
- to decode and count quadrature encoded input signals.
- to count up and down using dual inputs in a count with direction format.

- to program the counter terminal count value (modulo).
- to program the value loaded into the counter after it reaches its terminal count.
- to count repeatedly or to stop after completing one count cycle.
- to count to a programmed value (using the compare functionality) and then immediately reinitialize or to count through the compare value until the count rolls over to zero.

21.5.1 Counting Modes

The counting modes define the different modes for clocking the counters. The count mode is selected in the TMRxCTL[CM] field (**page 21-16**). If a counter is programmed to count to a specific value and then stop, the TMRCTL[CM] bit is cleared when the count terminates. **Table 21-2** summarizes the counting modes.

Table 21-2. Summary of Timer Counting Modes

Counting Mode	CM Bits	Description	Primary Clock	Secondary Clock
Disabled	000	Counter not active.	—	—
Count	001	Counts the rising edges of the selected clock source (falling edges if TxSCTL[IPS] is set). This mode is useful for generating periodic interrupts for timing purposes or for counting external events.	Clock*	—
Dual-Edge Count	010	Counts both edges of a timer Input signal. This mode is useful for counting the changes in the external environment. When this mode is selected, TMRxCTL[PCS] must not be set to any value between 1000 and 1111; that is, it must not set to the input clock or any scaled version of the input clock.	Clock	—
Gated Count	011	Counts primary clock edges while the secondary input is high (low if TxSCTL[IPS] is set). This mode is used to time the duration of external events when the primary clock is set to the input clock and the secondary input is set to use one of the timer input signals. It can also be used to count the number of external events that occur on one of the timer input signals, set as the primary clock, while a second timer input signal, connected to the secondary Input signal, is asserted.	Clock	Gate*
Quadrature Count	100	Uses quadrature encoded signals that are square waves, 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information. A timing diagram illustrating the basic operation of a quadrature incremental position encoder is provided in **Figure 21-2.**	Quadrature signal	Quadrature signal
Signed Count	101	Counts the primary clock source while a secondary input provides the count direction (up or down) for each recognized count.	Clock to count	Count direction

Table 21-2. Summary of Timer Counting Modes

Counting Mode	CM Bits	Description	Primary Clock	Secondary Clock
Triggered Count	110	Counts the primary clock source only after a rising edge is detected on the secondary input (falling edge if TxSCTL[IPS] is set). The counting continues until a compare event occurs or another positive input transition is detected. If a second input transition occurs before a terminal count is reached, counting stops. Subsequent odd-numbered edges of the secondary input restart the counting, and even numbered edges stop counting. This process continues until a compare event occurs.	Clock to count	Enable/disable counter*
Cascade Count	111	Cascades multiple counters. Cascade mode is used for creating counters larger than 16-bits. Up to four counters may be cascaded together to create a 64-bit wide counter. The Cascaded Counter mode is synchronous. See **Section 21.5.1.2**.	Clock to count	Triggers counter
Note: * This input can be inverted by the TxSCTL[IPS] bit.				

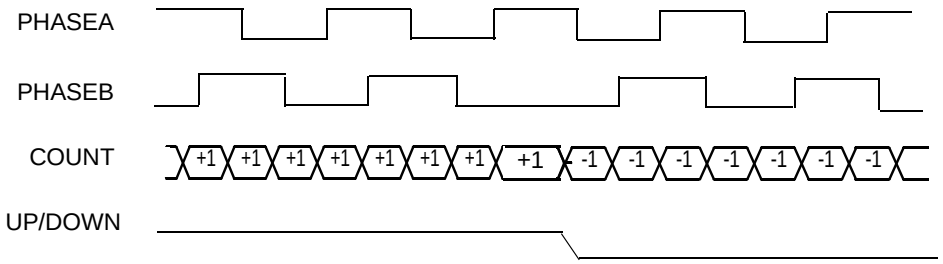


Figure 21-2. Quadrature Incremental Position Encoder

Other count modes derived as special cases of the modes described in **Table 21-2** are described in the remainder of this section.

21.5.1.1 One-Shot Mode

One-Shot mode is a variation on Triggered Count mode if the counter is set up as follows:

- TMRxCTL[CM] = 110 to count the rising and falling edges of the primary source (see **Table 21-4**, *TMRxSCTL Bit Descriptions*, on page 21-19).
- The Count Length bit, TMRxCTL[LEN] = 1.
- Output Flag mode, TMRxCTL[OFLM] = 101 to select set on compare, cleared on secondary input signal's edge.
- The Count Once bit, TMRxCTL[ONCE] = 1 to count till a compare and then stop.

An external event causes the counter to count. When terminal count is reached, the timer output flag is asserted. This delayed output assertion can be used to provide timing delays.

21.5.1.2 Pulse Output Mode

In Pulse Output mode, a variation on Count mode, the counter outputs a stream of pulses with the same frequency as the selected clock source (cannot be the input clock divided by 1) if the counter is set up as follows:

- TMRxCTL[CM] = 001 to count the rising edges of the primary source. (see **Table 21-4**, *TMRxSCTL Bit Descriptions*, on page 21-19).
- The Output Flag Mode, TMRxCTL[OFLM], = 111 to enable gated clock output while the counter is active.
- The Count Once bit, TMRxCTL[ONCE], = 1 to count till a compare and then stop.

The number of output pulses is equal to the compare value minus the initial value. The primary count source must be set to one of the counter outputs for gated clock output mode.

21.5.1.3 Fixed Frequency PWM Mode

Fixed Frequency Pulse Width Modulated (PWM) mode is a subset of Count mode. The counter is set up as follows:

- TMRxCTL[CM] = 001 to count the rising edges of the primary source.
- The Count Length bit, TMRxCTL[LEN], = 0 so that the counter continues counting past the compare value (binary roll-over).
- The Count Once bit, TMRxCTL[ONCE], = 0 to count repeatedly.
- The Output Flag Mode, TMRxCTL[OFLM], = 110 so that the output flag is set when a compare occurs.

The counter output yields a PWM signal with:

- a frequency equal to the count clock frequency divided by 65,536.
- a pulse width duty cycle equal to the compare value divided by 65,536.

21.5.1.4 Variable Frequency PWM Mode

The counter output yields a PWM signal with a frequency and pulse width determined by the values programmed into the TMRxCMP1 and TMRxCMP2 registers and the input clock frequency if the counter is set up as follows:

- TMRxCTL[CM] = 001 to count the rising edges of the primary source (see **Table 21-4**, *TMRxSCTL Bit Descriptions*, on page 21-19).
- The Count Length bit, TMRxCTL[LEN], = 1 so that the counter counts to the compare value and then reinitializes.
- The Count Once bit, TMRxCTL[ONCE], = 0 to count repeatedly.

- The Output Flag Mode, TMRxCTL[OFLM], = 100 to toggle the timer output flag using alternating compare registers.

This method of PWM generation has the advantage of allowing almost any desired PWM frequency and/or constant on or off periods. The TMRxCMPLD1 and TMRxCMPLD2 registers are especially useful for this mode because they give you time to calculate values for the next PWM cycle during the PWM current cycle.

To set up the timer to run in Variable Frequency PWM mode with compare preload, use the set-up described here for the desired counter. During set-up, update the TMRxCTL register last because the counter starts counting if the count mode changes to any value other than 000. Set up the Timer Control (TMRxCTL) register bits as follows:

- Count Mode (CM) = 001 to count the rising edges of the primary source.
- Primary Count Source (PCS) = 1000 to specify the IPBus clock to get the best granularity for waveform timing; prescaler input clock divided by 1.
- Secondary Count Source (SCS) = Any value because the bits are ignored in this mode.
- Count Once (ONCE) = 0 to count repeatedly.
- Count Length (LEN) = 1 so that the counter counts till it reaches a compare and then reinitializes the counter register.
- Direction (DIR) = Count up (0) or count down (1). The compare register values must be chosen carefully to account for roll-under and so on.
- External Initialization (EIN) = 0 so that external counters/timers cannot force a reinitialization of this timer. However, you can set this bit if you need the functionality.
- Output Mode (OFLM) = 100 to toggle the timer output flag using alternating compare registers.

Set up the Timer Status and Control Register (TMRxSCTL) bits as follows:

- Output Polarity Select (OPS) = Your choice, true (0) or inverted (1).
- Output Enable (OEN) = 1 to enable the timer output to be put on an external pin. Set this bit as needed.
- Ensure that the rest of the TMRxSCTL bits are cleared. Interrupts are enabled in the Timer Comparator Status and Control Register (TMRxCOMSC) instead of in this register.

Set up the Timer Comparator Status and Control Register (TMRxCOMSC) bits as follows:

- Timer Compare 2 Interrupt Enable (TCF2EN) = 1 to allow an interrupt to be issued when TCF2 is set).
- Timer Compare1 Interrupt Enable (TCF1EN) = 0 so that an interrupt cannot be issued when TCF1 is set.

- Timer Compare 1 Interrupt Source (TCF1) = 0 to clear the timer compare 1 interrupt source flag. This bit is set when a successful comparison of the timer and the TMRxCMP1 register occurs.
- Timer Compare 2 Interrupt Source (TCF2) = 0 to clear the timer compare 2 interrupt source flag. This bit is set when a successful comparison of the timer and the TMRxCMP2 register occurs.
- Compare Load Control 1 (CL1) = 10 to load the compare register when TCF2 is set.
- Compare Load Control 2 (CL2) = 01 to load the compare register when TCF1 is set.

To service the TCF2 interrupts generated by the Timer, the interrupt controller must be configured to enable the interrupts for the timer being used. Additionally, you must write an interrupt service routine to do at least the following:

- Clear the TCF2 and TCF1 flags.
- Calculate and write new values for TMRxCMPLD1[15–0] and TMRxCMPLD2[15–0].

Figure 21-3 shows the timing for the compare preload cycle, which begins when a compare event on TMRxCMP2 causes TCF2 to be set. TMRxCMP1 is loaded with the value in the TMRxCMPLD1 one IPBus clock later. In addition, the timer asserts an interrupt, and the interrupt service routine executes while both comparator load registers are updated with new values. When TCF1 is set, TMRxCMP2 is loaded with the value of the CLV2 bits in TMRxCMPLD2. During the subsequent TCF2 event, TMRxCMP1 is loaded with the value of the TMRxCMPLD1[CLV1] bits. The cycle starts over again as an interrupt is asserted and the interrupt service routine clears TCF1 and TCF2 and calculates new values for TMRxCMPLD1 and TMRxCMPLD2.

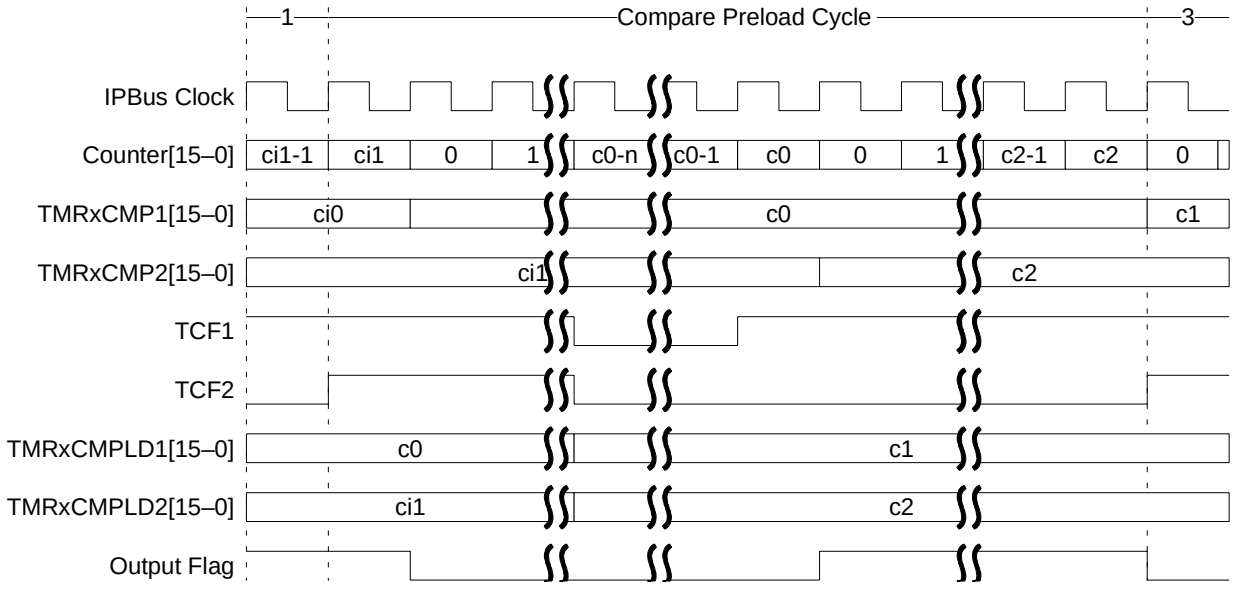


Figure 21-3. Compare Preload Timing

21.6 Timer Compare Functionality

The compare registers (TMRxCMP1 and TMRxCMP2) provide a bidirectional modulo count capability. TMRxCMP1 is used when the counter is counting *up*. Program it with the desired maximum count value or to 0xFFFF to indicate the maximum unsigned value prior to roll-over. TMRxCMP2 is used when the counter is counting *down*. Program it with the maximum negative count value or to 0x0000 to indicate the minimum unsigned value prior to roll-under. The only exception occurs when the counter operates with alternating compare registers.

When TMRxCTL[OFLM] = 100, alternating values of TMRxCMP1 and TMRxCMP2 are used to generate successful compares, and the output flag toggles while alternating compare registers are in use. For example, when TMRxCTL[OFLM] = 100, the counter is programmed to count upwards. It counts until the TMRxCMP1 value is reached, reinitializes, then counts until the TMRxCMP2 value is reached, reinitializes, then counts until the TMRxCMP1 value is reached, and so on. In this Variable Frequency PWM mode, the TMRxCMP2 value defines the desired pulse width of the *on-time*, and the TMRxCMP1 register defines the *off-time*. The Variable Frequency PWM mode is defined for positive counting only. See **Section 21.5.1.4**, *Variable Frequency PWM Mode,* on page 21-10.

Use caution when changing TMRxCMP1 and TMRxCMP2 while the counter is active. If the counter has already passed the new value, it counts to 0xFFFF or 0x0000, rolls over/under, and then begins counting toward the new value. The check is for Count = TMRxCMPx, not Count > = TMRxCMP1 or Count < = TMRxCMP2). Use of the preload registers addresses this problem.

21.6.1 Compare Preload Registers

The TMRxCMPLD1, TMRxCMPLD2, and TMRxCOMSC registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality, use the loading method described in this section.

The compare preload feature speeds updating of the compare registers. The compare preload feature allows you to calculate new compare values and store them into the comparator preload registers. When a compare event occurs, the new compare values in the comparator preload registers are directly written to the compare registers, eliminating the use of software to do this.

The compare preload feature is used in variable frequency PWM mode. See **Section 21.5.1.4**, *Variable Frequency PWM Mode,* on page 21-10. The TMRxCMP1 register determines the pulse width for the logic low part of the timer output, and TMRxCMP2 determines the pulse width for the logic high part of the timer output. The period of the waveform is determined by the TMRxCMP1 and TMRxCMP2 values and the frequency of the primary clock source.
See **Figure 21-4**.

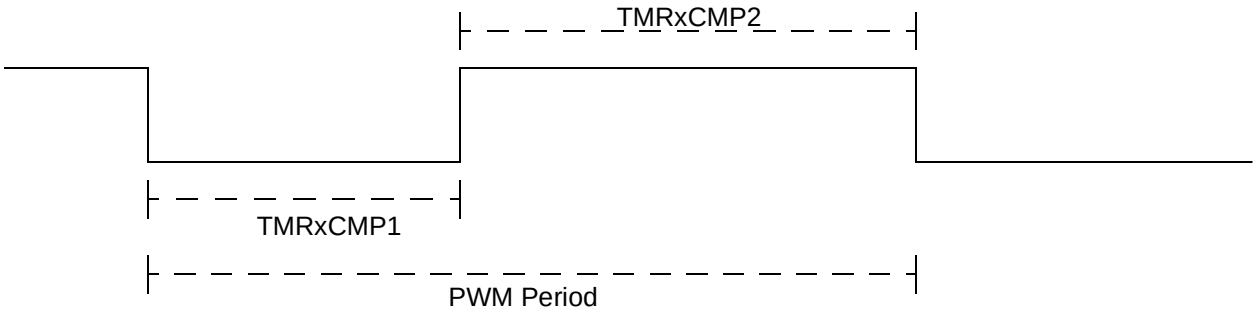


Figure 21-4. Variable PWM Waveform

To update the duty cycle or period of the waveform, update the TMRxCMP1 and TMRxCMP2 values using the compare preload feature.

21.6.1.1 Capture Register Use

The capture register, TMRxCAP (**page 21-23**), stores a copy of the counter value when an input edge (positive, negative, or both) on the secondary input signal is detected.

The capture mode, programmable via TMRxSCTL[CM] (**page 21-19**), is one of the following:

- CM = 00. Disabled.
- CM = 01. Load the capture register on the *rising* edge of the signal.
- CM = 10. Load the capture register on the *falling* edge of the signal.
- CM = 11. Load the capture register on *either* edge of the signal.

When a capture event occurs, there are no further updates of TMRxCAP until the input edge flag (IEF) is cleared by writing a value of 0 to the TMRxSCTL[IEF] bit (**page 21-19**).

21.6.1.2 Broadcast from a Master Counter

Any counter/timer can be assigned as a master. A master compare signal can be broadcast to the other counter/timers within the module. The other counters can be configured as follows to reinitialize their counters and/or force their output to predetermined values when a master counter/timer compare event occurs:

- Select one counter as the master counter by setting the TMRxSCTL[MSTR] bit.
- Program the other counters to perform an action when a compare event occurs on the master counter as follows:
  - The other counter is reinitialized if its TMRxCTL[EIN] bit is set.
  - The other counter forces its output flag signal if its TMRxSCTL[EEOF] bit is set.

21.7 Resets and Interrupts

A timer module is reset by a peripheral module reset, as shown in **Table 13-1**, *Reset Sources*, on page 13-1. This reset forces all registers to their reset state and clears the output flag signal if it is asserted. The counter is turned off until the settings in the control register are changed. Each channel in a quad timer module can be programmed for interrupts. The available types of interrupts are as follows:

- Timer compare
- Timer compare 1
- Timer compare 2
- Timer overflow
- Timer input edge

These different types are ORed together within each timer channel to generate a single interrupt request signal to the interrupt controller.

21.7.1 Timer Compare Interrupts

Interrupt requests are generated when a successful compare occurs between a counter and its compare registers while the Timer Compare Flag Interrupt Enable bit, TMRxSCTL[TCFIE], is set. These interrupt requests are cleared by writing a zero to the appropriate TMRxSCTL[TCF] bit. When a timer compare interrupt is set in the TMRxSCTL and the compare preload registers are available, one of the following two interrupts is also asserted:

- Timer compare 1 interrupt
- Timer compare 2 interrupt

Timer compare 1 interrupts are generated when a successful compare occurs between a counter and its TMRxCMP1 register while the Timer Compare 1 Interrupt Enable (TCF1EN) is set in the TMRxCOMSC register. These interrupts are cleared by writing a zero to the TMRxCOMSC[TCF1] bit. Timer compare 2 interrupts are generated when a successful compare occurs between a counter and its TMRxCMP2 register while the Timer Compare 2 Interrupt Enable (TCF2EN) bit is set in the TMRxCOMSC register. These interrupts are cleared by writing a zero to the TCF2 bit in the TMRxCOMSC.

21.7.2 Timer Overflow Interrupts

Timer overflow interrupts are generated when a counter rolls over its maximum value while the TCFIE bit is set in the TMRxSCTL register. These interrupts are cleared by writing a zero to the Timer Overflow Flag (TOF) bit of the appropriate TMRxSCTL.

21.7.3 Timer Input Edge Interrupts

Timer input edge interrupts are generated by a transition of the input signal (either positive or negative, depending on TMRxSCTL[IPS] setting) while the Input Edge Flag Interrupt Enable (IEFIE) bit is set in the TMRxSCTL. These interrupts are cleared by writing a zero to the appropriate TMRxSCTL[IEF] bit.

21.8 Timer Programming Model

This section presents only the registers for timer module A. The registers for timer module B are identical to those for module A. For a complete listing of all registers in both modules, consult **Table 5-2**, *MSC711x Detailed Memory Map*, on page 5-5.The value of the base address for the timer register file, TMRx_BASE, is provided in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. The timer registers are listed as follows, along with the number of the page on which each register is discussed:

- Timer Channel x Control Register, TMRxCTL, **page 21-16**.
- Timer Channel x Status and Control Register (TMRxSCTL), **page 21-19**.
- Timer Channel x Compare Register 1 (TMRxCMP1), **page 21-21**.
- Timer Channel x Compare Register 2 (TMRxCMP2), **page 21-21**.
- Timer Channel x Compare Load Register 1 (TMRxCMPLD1), **page 21-21**.
- Timer Channel x Compare Load Register 2 (TMRxCMPLD2), **page 21-22**.
- Timer Channel x Comparator Status and Control Registers (TMRxCOMSC), **page 21-22**.
- Timer Channel x Capture Register (TMRCAP), **page 21-22**.
- Timer Channel x Load Register (TMRxLOAD), **page 21-24**.
- Timer Channel x Hold Registers (TMRxHOLD), **page 21-24**.
- Timer Channel x Counter Register (TMRCNTR), **page 21-24**.

TMRxCTL	Timer A Channel x Control Register	
TMR0CTL	Timer A Channel 0 Control Register	BASE + 0x18
TMR1CTL	Timer A Channel 1 Control Register	BASE + 0x58
TMR2CTL	Timer A Channel 2 Control Register	BASE + 0x98
TMR3CTL	Timer A Channel 3 Control Register	BASE + 0xD8

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CM			PCS				SC		ONCE	LEN	DIR	EIN	OFLM		
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 21-3. TMRxCTL Bit Descriptions

Name	Reset	Description	Settings
CM 15–13	0	**Count Mode** Control the basic counting behavior of the counter. Rising edges are counted only when TxSCTL[IPS] = 0. Falling edges are counted only when TxSCTL[IPS] = 1. When count mode 010 is selected, the PCS bits must not be set to 1000. When count mode 111 is selected, the PCS bits must be set to one of the “Counter N output” selections.	000 No operation. Disabled. 001 Count rising edges of the primary source. 010 Count rising and falling edges of the primary source. 011 Count rising edges of the primary source while the secondary input is high active. 100 Quadrature count mode, uses primary clock and secondary input. 101 Count rising edges of the primary clock; secondary input specifies direction (1 = minus). 110 Edge of the secondary input triggers primary count until a compare occurs. 111 Cascaded counter mode (up/down).
PCS 12–9	0	**Primary Count Source** Select the primary count source. A timer selecting its own output for input is not a legal choice. The result is no counting.	0000 Counter 0 input signal. 0001 Counter 1 input signal. 0010 Counter 2 input signal. 0011 Counter 3 input signal. 0100 Counter 0 output for cascaded counter operation. 0101 Counter 1 output for cascaded counter operation. 0110 Counter 2 output for cascaded counter operation. 0111 Counter 3 output for cascaded counter operation. 1000 Prescaler (input clock divided by 1). 1001 Prescaler (input clock divided by 2). 1010 Prescaler (input clock divided by 4). 1011 Prescaler (input clock divided by 8). 1100 Prescaler (input clock divided by 16). 1101 Prescaler (input clock divided by 32). 1110 Prescaler (input clock divided by 64). 1111 Prescaler (input clock divided by 128).
SC 8–7	0	**Secondary Count Source** Provides additional information, such as direction, used for counting. These bits also define the source used by both the Capture mode bits and the input Edge Flag in the Channel Status and Control register.	00 TIN0 Timer input signal. 01 TIN1 Timer input signal. 10 TIN2 Timer input signal. 11 TIN3 Timer input signal.

Table 21-3. TMRxCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
ONCE 6	0	**Count Once** Selects continuous or one-shot counting. If counting up, a successful compare occurs when the counter reaches TMRxCMP1 value. If counting down, a successful compare occurs when a counter reaches TMRxCMP2 value.	0 Count repeatedly. 1 Count to the compare value and then stop.
LEN 5	0	**Count Length** Determines whether the counter counts to the compare value and then reinitializes itself, or the counter continues counting past the compare value (binary roll-over). If counting up, a successful compare occurs when the counter reaches the TMRxCMP1 value. If counting down, a successful compare occurs when the counter reaches the TMRxCMP2 value.	0 Roll-over. 1 Count to the compare value and then reinitialize.
DIR 4	0	**Count Direction** Selects either the normal count up direction, or the reverse down direction.	0 Count up. 1 Count down.
EIN 3	0	**External Initialization** Enables another counter/timer within the same module to force the reinitialization of this counter/timer when the other counter has an active compare event. Details on Broadcast mode are presented in **Section 21.6.1.2**, *Broadcast from a Master Counter,* on page 21-14.	0 External counter/timers can not force a reinitialization of this counter/timer. 1 External counter/timers may force a reinitialization of this counter/timer.
OFLM 2–0	0	**Output Mode** Determine the mode of operation for the timer output signal. For all of these modes except 000 and 111, the output flag is not toggled when the counter reaches the compare value but instead when the counter advances one value beyond. For example, for a compare value of 7, it toggles on the transition from 7 to 8, not 6 to 7. Unexpected results may occur if the Output mode field is set to use alternating compare registers (mode 100) and the ONCE bit is set.	000 Asserted while counter is active. 001 Clear timer output on successful compare. 010 Set timer output on a successful compare. 011 Toggle the timer output flag when a successful compare occurs. 100 Toggle the timer output flag using alternating compare registers. 101 Set on compare, cleared on secondary input signal's edge. 110 Set on compare, cleared on counter rollover. 111 Enable gated clock output while the counter is active.

TMRxSCTL	Timer A Channel x Status and Control Register	
TMR0SCTL	Timer A Channel 0 Status and Control Register	BASE + 0x1C
TMR1SCTL	Timer A Channel 1 Status and Control Register	BASE + 0x5C
TMR2SCTL	Timer A Channel 2 Status and Control Register	BASE + 0x9C
TMR3SCTL	Timer A Channel 3 Status and Control Register	BASE + 0xDC

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CM		MSTR	EEOF	VAL	FORC	OPS	OEN
TYPE	R/W							R	R/W					W	R/W	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 21-4. TMRxSCTL Bit Descriptions

Name	Reset	Description	Settings
TCF 15	0	**Timer Compare Flag** Set when a successful compare occurs. Clear the bit by writing 0 to it.	0 No successful compare. 1 Successful compare.
TCFIE 14	0	**Timer Compare Flag Interrupt Enable** Enables interrupts when the TCF bit is set.	0 No interrupt. 1 Interrupt.
TOF 13	0	**Timer Overflow Flag** Set when the counter rolls over its maximum value 0xFFFF or 0x0000, depending on count direction. Clear the bit by writing 0 to it.	0 No overflow. 1 Overflow. The timer has reached its maximum or minimum value.
TOFIE 12	0	**Timer Overflow Flag Interrupt Enable** Enables interrupts when the TOF bit is set.	0 No interrupt. 1 Interrupt.
IEF 11	0	**Input Edge Flag** Set when a positive input transition occurs while the counter is enabled. Clear the bit by writing a 0 to it. Setting the input polarity select (TxSCTL[IPS]) bit enables the detection of negative input edge transitions. Also, the control register secondary count source determines which external input pin is monitored by the detection circuitry.	0 No action. 1 Positive input transition while counter enabled.
IEFIE 10	0	**Input Edge Flag Interrupt Enable** Enables interrupts when the IEF bit is set.	0 No action. 1 Interrupts enabled.
IPS 9	0	**Input Polarity Select** Inverts the input signal polarity.	0 No action. 1 Invert signal polarity.
INPUT 8	0	**Secondary Input Signal** Reflects the current state of the secondary Input signal.	00 Capture function disabled. 01 Load capture register on rising edge of the secondary count source input. 10 Load capture register on falling edge of the secondary count source input. 11 Load capture register on any edge of the secondary count source input.

Table 21-4. TMRxSCTL Bit Descriptions (Continued)

Name	Reset	Description	Settings
CM 7–6	0	**Input Capture Mode** Specifies the operation of the capture register as well as the operation of the input edge flag.	00 Capture function is disabled. 01 Load capture register on the rising edge of the secondary count source input. 10 Load capture register on the falling edge of the secondary count source input. 11 Load capture register on any edge of the secondary count source input.
MSTR 5	0	**Master Mode** Enables the compare function output to broadcast to the other counter/timers in the module. This identifies a counter as the master counter in Broadcast mode. This signal is used to reinitialize the other counters and/or force their outputs. For details on Broadcast mode, see **Section 21.6.1.2**, *Broadcast from a Master Counter,* on page 21-14.	0 No action. 1 Broadcast mode.
EEOF 4	0	**Enable External Output Force** Enables the compare from another counter/timer configured as the master to force the state of this counter output signal. For details on Broadcast mode, see **Section 21.6.1.2**, *Broadcast from a Master Counter,* on page 21-14.	o No action. 1 Other counter can force this counter's output flag signal.
VAL 3	0	**Forced Output Flag Value** Determines the value of the timer output flag signal when a software-triggered FORCE command occurs.	
FORC 2	0	**Force Output** Forces the current value of the VAL bit to be written to the timer output. Always read this bit as a 0. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if the counter is disabled. Setting this bit while the counter is enabled may yield unpredictable results.	0 No action. 1 Forces the current value of the VAL bit to be written to timer output.
OPS 1	0	**Output Polarity Select** Determines the polarity of the output signal.	0 True polarity. 1 Inverted polarity.
OEN 0	0	**Output Enable** Enables the timer output. The OPS bit determines the polarity of the output.	0 Timer output not enabled. 1 Timer output enabled.

TMRxCMP1	Timer A Channel x Compare 1 Registers	
TxC0CMP1	Timer A Channel 0 Compare 1 Register	BASE + 0x00
TMR1CMP1	Timer A Channel 1 Compare 1 Register	BASE + 0x40
TMR2CMP1	Timer A Channel 2 Compare 1 Register	BASE + 0x80
TMR3CMP1	Timer A Channel 3 Compare 1 Register	BASE + 0xC0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TxCxCMP1 store the comparison value (CV) for comparison with the counter value.

TMRxCMP2	Timer A Channel x Compare 2 Registers	
TxC0CMP2	Timer A Channel 0 Compare 2 Register	BASE + 0x04
TMR1CMP2	Timer A Channel 1 Compare 2 Register	BASE + 0x44
TMR2CMP2	Timer A Channel 2 Compare 2 Register	BASE + 0x84
TMR3CMP2	Timer A Channel 3 Compare 2 Register	BASE + 0xC4

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TxCxCMP2 store the comparison value (CV) for comparison with the counter value.

TMRxCMPLD1	Timer A Channel x Compare Load 1 Registers	
TxC0CMPLD1	Timer A Channel 0 Compare Load 1 Register	BASE + 0x20
TMR1CMPLD1	Timer A Channel 1 Compare Load 1 Register	BASE + 0x60
TMR2CMPLD1	Timer A Channel 2 Compare Load 1 Register	BASE + 0xA0
TMR3CMPLD1	Timer A Channel 3 Compare Load 1 Register	BASE + 0xE0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CLV1															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxCMPLDx store the preload value for the TMRxCMP1.

TMRxCMPLD2 Timer A Channel x Compare Load 2 Registers

TxC0CMPLD2	Timer A Channel 0 Compare Load 2 Register	BASE + 0x24
TMR1CMPLD2	Timer A Channel 1 Compare Load 2 Register	BASE + 0x64
TMR2CMPLD2	Timer A Channel 2 Compare Load 2 Register	BASE + 0xA4
TMR3CMPLD2	Timer A Channel 3 Compare Load 2 Register	BASE + 0xE4

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CLV2															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxCMPLD2 store the preload value for the TMRxCMP2.

TMRxCOMSC Timer A Channel x Comparator Status and Control Registers

TxC0COMSC2	Timer A Channel 0 Comparator Status and Control Register	BASE + 0x28
TMR1COMSC2	Timer A Channel 1 Comparator Status and Control Register	BASE + 0x68
TMR2COMSC2	Timer A Channel 2 Comparator Status and Control Register	BASE + 0xA8
TMR3COMSC2	Timer A Channel 3 Comparator Status and Control Register	BASE + 0xE8

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—								TCF2EN	TCF1EN	TCF2	TCF1	CL2		CL1	
TYPE	R								R/W							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxCOMSC store the preload value used for the TMRxCMP2 register.

Table 21-5. TMRxCOMSC Bit Descriptions

Name	Reset	Description	Settings
— 15–8	0	Reserved. Write to zero for future compatibility.	
TCF2EN 7	0	**Timer Compare 2 Interrupt Enable** Enables the compare 2 interrupt. An interrupt is issued when both this bit and the TCF2 bit are set.	0 No action. 1 Enable compare 2 interrupt.
TCF1EN 6	0	**Timer Compare 1 Interrupt Enable** Enables the compare 1 interrupt. An interrupt is issued when both this bit and the TCF1 bit are set.	0 No action. 1 Enable compare 1 interrupt.
TCF2 5	0	**Timer Compare 2 Interrupt Source** Indicates a successful comparison of the timer and the TMRxCMP2. This bit is sticky and remains set until it is explicitly cleared by writing a zero to this bit location.	0 Normal operation. 1 Successful compare 2.

Table 21-5. TMRxCOMSC Bit Descriptions (Continued)

Name	Reset	Description	Settings
TCF1 4	0	**Timer Compare 1 Interrupt Source** Indicates a successful comparison of the timer and the TMRxCMP1. This bit is sticky and remains set until it is explicitly cleared by writing a zero to this bit location.	0 Normal operation. 1 Successful compare 1.
CL2 3–2	0	**Compare Load Control 2** Control when TMRxCMP2 is preloaded with the value from TMRxCMPLD2.	00 Never preload. 01 Load upon successful compare with the value in TMRxCMP1. 10 Load upon successful compare with the value in TMRxCMP2. 11 Reserved.
CL1 1–0	0	**Compare Load Control 2** Control when TMRxCMP1 is preloaded with the value from TMRxCMPLD1.	00 Never preload. 01 Load upon successful compare with the value in TMRxCMP1. 10 Load upon successful compare with the value in TMRxCMP2. 11 Reserved.

TMRxCAP	Timer A Channel x Capture Registers	
TxC0CAP	Timer A Channel 0 Capture Register	BASE + 0x08
TMR1CAP	Timer A Channel 1 Capture Register	BASE + 0x48
TMR2CAP	Timer A Channel 2 Capture Register	BASE + 0x88
TMR3CAP	Timer A Channel 3 Capture Register	BASE + 0xC8

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CAPV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxCAP store the values captured from the counters.

TMRxLOAD	Timer A Channel x Load Registers	
TxC0LOAD	Timer A Channel 0 Load Register	BASE + 0x0C
TMR1LOAD	Timer A Channel 1 Load Register	BASE + 0x4C
TMR2LOAD	Timer A Channel 2 Load Register	BASE + 0x8C
TMR3LOAD	Timer A Channel 3 Load Register	BASE + 0xCC

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LDV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxLOAD store the value used to load the counter.

TMRxHOLD	Timer A Channel x Hold Registers	
TxC0HOLD	Timer A Channel 0 Hold Register	BASE + 0x10
TMR1HOLD	Timer A Channel 1 Hold Register	BASE + 0x50
TMR2HOLD	Timer A Channel 2 Hold Register	BASE + 0x90
TMR3HOLD	Timer A Channel 3 Hold Register	BASE + 0xD0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HDV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxHOLD store the channel value whenever any counter is read.

TMRxCNTR	Timer A Channel x Counter Registers	
TxC0CNTR	Timer A Channel 0 Counter Register	BASE + 0x14
TMR1CNTR	Timer A Channel 1 Counter Register	BASE + 0x54
TMR2CNTR	Timer A Channel 2 Counter Register	BASE + 0x94
TMR3CNTR	Timer A Channel 3 Counter Register	BASE + 0xD4

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HDV															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxCNTR are counters.

I²C Software Module 22

The I²C module provides the functionality of a standard I²C slave and master and is designed to be compatible with the standard Phillips I²C bus protocol.[1] The I²C module is a two-wire, bidirectional serial bus for simple, efficient data exchange that minimizes the interconnection between devices. This bus accommodates applications requiring occasional communications over a short distance between many devices. I²C flexibility permits additional devices to connect to the bus for expansion and system development. I²C is a 16-bit IP module, so only 16-bit accesses should be performed to the module.

The interface operates at up to 400 kbps with maximum bus loading and timing. The I²C system is a true multiple-master bus with arbitration and collision detection to prevent data corruption if multiple devices attempt to control the bus simultaneously. This feature supports complex applications with multiprocessor control and is useful for rapid testing and alignment of end products through external connections to an assembly-line computer.

22.1 Features

Features of the I²C module are as follows:

- Compatibility with I²C bus standard
- Multiple-master operation
- Software-programmable for one of 64 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

1. For information on system configuration, protocol, and restrictions, see The *I²C Bus Specification*, Version 2.1.

22.2 Architecture

Figure 22-1 shows a functional block diagram of the I^{2}C module. Notice that The I^{2}C module operates via two pins:

- SCL, a bidirectional clock pin.
- SDA, a bidirectional data pin.

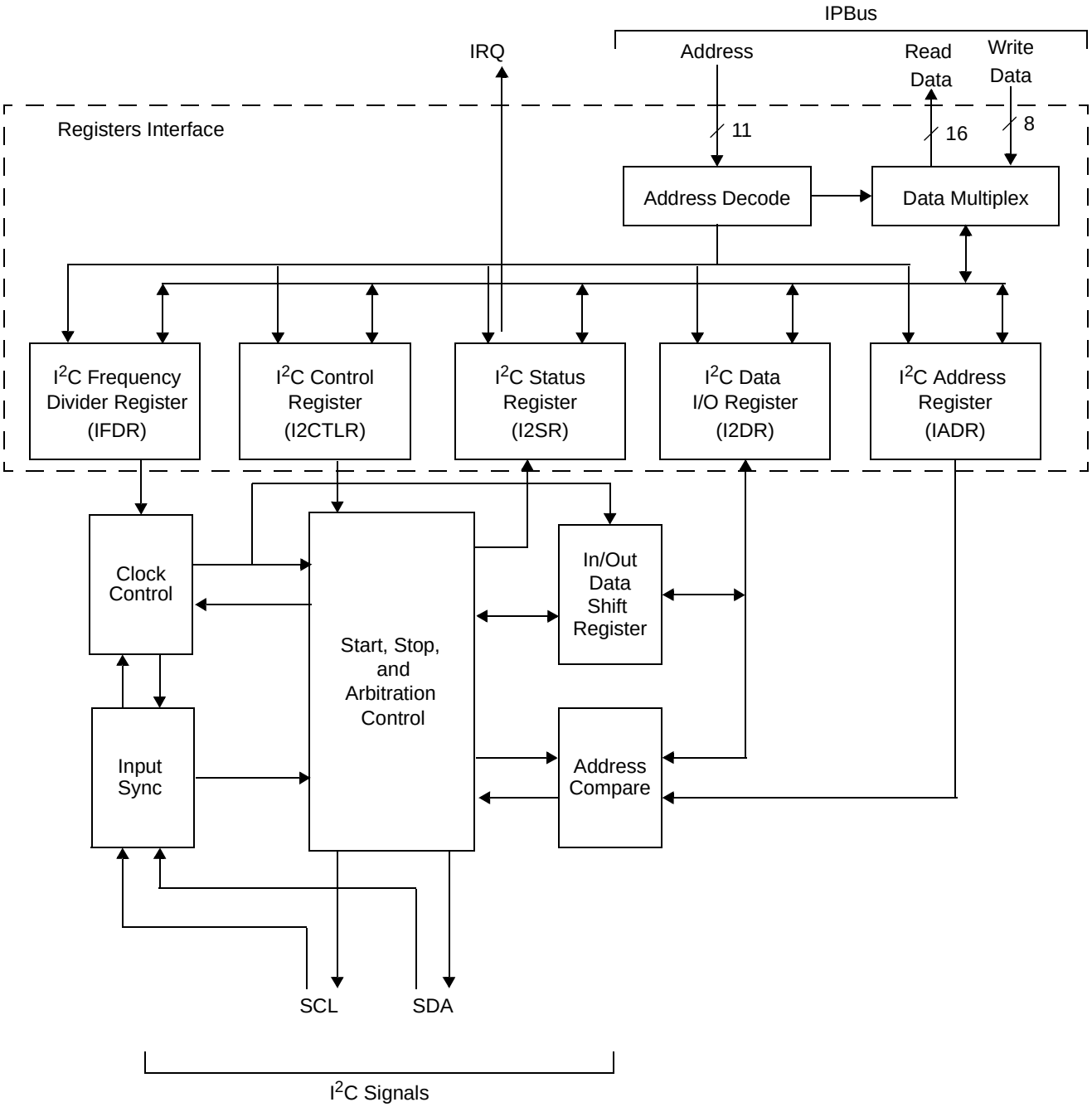


Figure 22-1. I^{2}C Block Diagram

22.3 I²C Operation

Out of reset, the I²C module is by default a slave receiver. When it is not programmed to be a master or when it is not responding to a slave transmit address, the I²C module should return to the default slave receiver state. The I²C communication protocol consists of six components: START, data source/recipient, data direction, slave acknowledge, data, data acknowledge, and STOP (see **Figure 22-3**).

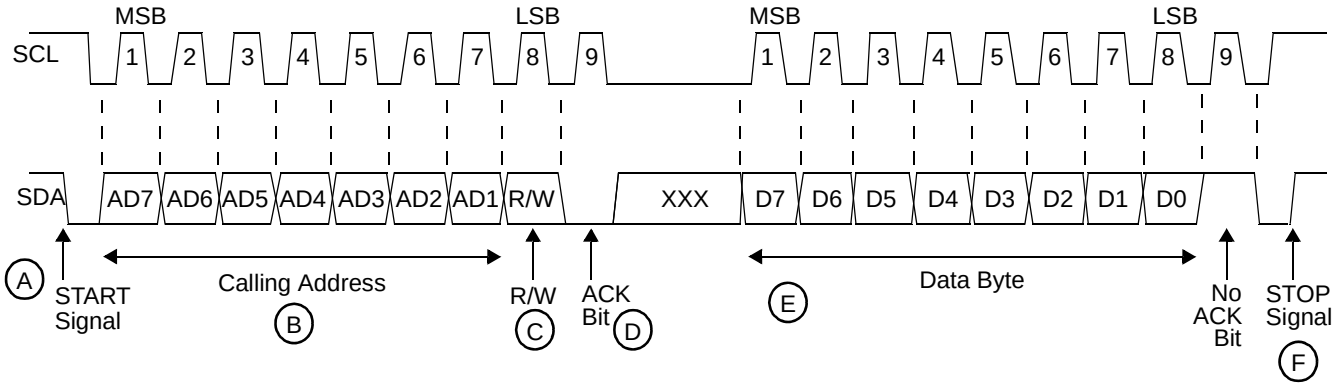


Figure 22-2. I²C Standard Communication Protocol

Table 22-1. Components of the I²C Standard Communication Protocol

	Protocol Component	Description
A	START signal	When no other device is bus master (both the SCL and SDA lines are at logic high), a device can initiate communication by sending a START signal, which is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a data transfer, which can be several bytes long, and awakens all slaves.
B	Calling address (slave address transmission)	The master sends the slave address in the first byte after the START signal. After the seven-bit calling address, it sends the R/W bit, which gives the slave the data transfer direction. Each slave must have a unique address. An I²C master must not transmit an address that is the same as its slave address; it cannot be master and slave at the same time. The slave with an address matching that sent by the master pulls SDA low at the ninth clock to return an acknowledge bit.
C	Data direction	When successful slave addressing is achieved, the data transfer can proceed byte-by-byte in the direction specified by the R/W bit sent by the calling master.
D	Acknowledge	If it does not acknowledge the master, the slave receiver must leave SDA high. The master can then generate a STOP signal to abort the data transfer or generate a START signal (repeated start) to start a new calling sequence. If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases SDA for the master to generate a STOP or START signal.
E	Data	Data can be changed only while SCL is low and must be held stable while SCL is high, as **Figure 22-2** shows. SCL is pulsed once for each data bit, and the MSB is sent first. The receiving device must acknowledge each byte by pulling SDA low at the ninth clock. Therefore, a data byte transfer requires nine clock pulses.

Table 22-1. Components of the I^{2}C Standard Communication Protocol (Continued)

	Protocol Component	Description
F	STOP signal or repeat start	The master terminates communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical high. A master can generate a STOP even if the slave has made an acknowledgment, and the slave must release the bus. Instead of signalling a STOP, the master can repeat the START signal, followed by a calling command, (A in **Figure 22-3**). A repeated START occurs when a START signal is generated without first generating a STOP signal to end the communication. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

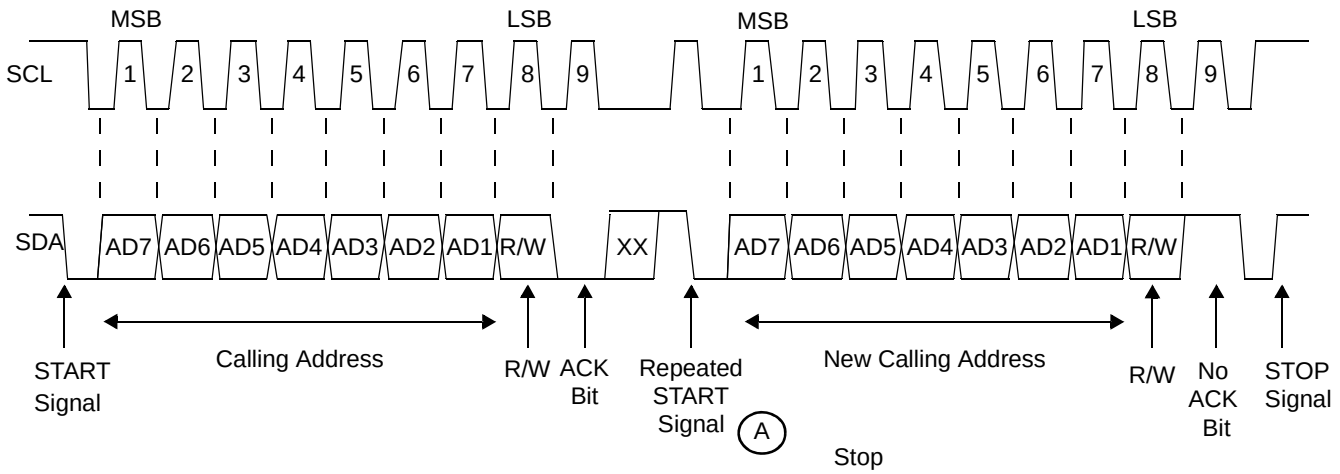


Figure 22-3. Repeated START

22.3.1 Arbitration

If multiple devices simultaneously request the bus, the bus clock is determined by a synchronization procedure in which the low period equals the longest clock-low period among the devices and the high period equals the shortest. A data arbitration procedure determines the relative priority of competing devices. A device loses arbitration if it sends logic high while another sends logic low; it immediately switches to slave-receive mode and stops driving SDA. The transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.

22.3.2 Clock Synchronization and Stretching

Because wire-AND logic is used, a high-to-low transition on SCL affects devices connected to the bus. Devices start counting their low period when the master drives SCL low. When a device clock goes low, it holds SCL low until the clock high state is reached. However, the low-to-high change in this device clock may not change the state of SCL if another device clock is still in its low period. Therefore, the device with the longest low period holds the synchronized clock SCL

low. Devices with shorter low periods enter a high wait state during this time (see **Figure 22-4**). When all affected devices have counted off their low period, the synchronized clock SCL is released and pulled high so that there is no difference between device clocks and the state of SCL. All devices start counting their high periods. The first device to complete its high period pulls SCL low again.

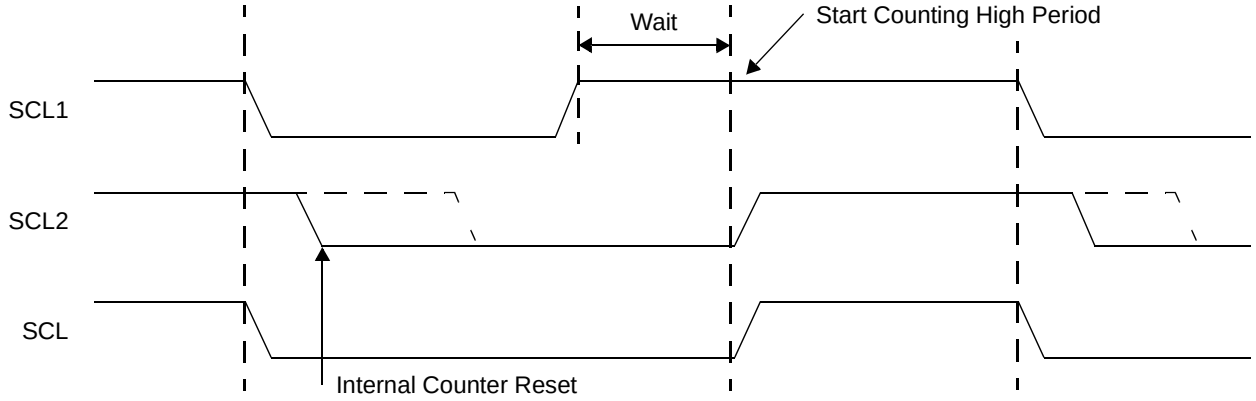


Figure 22-4. Synchronized Clock SCL

Clock synchronization can be used as a handshake in data transfers. Slave devices can hold SCL low after completing one byte transfer (9 bits). The bus clock halts, and the master clock goes into wait states until the slave releases SCL.

Slaves can use clock synchronization to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is longer than the master SCL low period, the resulting SCL bus signal low period is stretched.

22.4 Initialization/Application

This section presents programming examples for initialization, signalling START, post-transfer software response, signalling STOP, and generating a repeated START. Before the interface can transfer serial data, the registers must be initialized, as follows:

1. Configure the I^2C Frequency Divider Register (IFDR) clock rate (IC) bits to obtain the SCL frequency from the system bus clock.
2. Update I^2C Address Register (IADR) to define its slave address.
3. Set the control register I2CTLR[IEN] bit to enable the I^2C bus interface.
4. Modify the I^2C Control Register (I2CTLR) to select master/slave mode, transmit/receive mode, and interrupt enable or not.

22.4.1 Generation of START

After initialization completes, master transmitter mode is selected to transmit serial data. On a multiple-master bus system, I2SR[IBB] must be checked to determine whether the serial bus is free. If the bus is free (IBB = 0), the START signal and the first byte (the slave address) are sent. The data written to the data register comprises the address of the desired slave and the LSB to indicate the transfer direction. The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I^2C module is busy after the calling address is written to the I2DR and before data is written to the I^2C Data I/O Register (I2DR).

22.4.2 Post Transfer Software Response

Sending or receiving a byte sets the I^2C Status Register I2SR[ICF] bit, indicating that a one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization (I2CTLR[IIEN] = 1). Software must first clear IIF in the interrupt routine. ICF is cleared either by reading from I2DR in receive mode or by writing to I2DR in transmit mode.

Software can service the I^2C I/O in the main program by monitoring IIF if the interrupt function is disabled. Polling should monitor IIF rather than ICF because that operation is different when arbitration is lost. When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; that is, the address is sent. If master receive mode is required, (I2DR[R/W], I2CTLR[MTX] should be set.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

22.4.3 Generation of STOP or a Repeated START

A data transfer ends when the master signals a stop after all data is sent. To terminate a data transfer, the master receiver must inform the slave transmitter by not acknowledging the last data byte. That is, I2CTLR[TXAK] is set before the next-to-last byte is read. Before the last byte is read, a STOP signal must be generated. After the data transfer, if the master still wants the bus, it can signal another START followed by another slave address without signalling a STOP.

22.4.4 Slave Mode

In the slave interrupt service routine, the slave (IAAS) bit should be tested to check whether it has called its own address. If IAAS is set, software should set the transmit/receive mode select bit (I2CTLR[MTX]) according to the I2SR[SRW]. Writing to the I2CTLR automatically clears the IAAS bit. IAAS is read as set only from the interrupt at the end of the address cycle where an

address match occurred. Interrupts from subsequent data transfers must have IAAS cleared. A data transfer is now initiated by writing information to I2DR for slave transmit or reading from I2DR in slave receive mode. A dummy read of I2DR in slave/receive mode releases SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[RXAK] must be tested before the next byte of data is sent. RXAK = 1 means that the master receiver has sent an end-of-data signal and that software must switch from transmitter to receiver mode. Reading I2DR releases SCL so that the master can generate a STOP signal.

22.4.5 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes the master. Devices that lose arbitration are immediately switched to slave receive mode. Data output to SDA stops, but SCL is still generated until the end of the byte at which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR[IAL] = 1 and I2CTLR[MSTA] = 0.

If a device that is not a master tries to transmit or do a START, the system inhibits the transmission, clears MSTA without signalling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed attempt to engage the bus. Therefore, the slave service routine should first test IAL and software should clear it if it is set. **Figure 22-5** shows a diagram of an I2C interrupt service routine.

22.5 Halting and Starting the I2C Module

To halt the I2C module completely for lowest power consumption, complete the following steps:

1. To ensure that no data transfer is in progress, poll the I2SR[ICF] bit (**page 22-12**). Omitting this step can result in transmission or reception of invalid data.
2. To shut down the module, clear the I2CTLR[IEN] bit (**page 22-11**).
3. Set the HLTREQ[I2CCD] bit (**page 11-26**) to shut down the system clock to the module for lowest power consumption.

To restart the I2C module, complete the following steps:

1. Clear the HLTREQ[I2CCD] bit (**page 11-26**) to re-enable the system clock to the module.
2. Set the I2CTLR[IEN] bit (**page 22-11**) to re-enable the module.

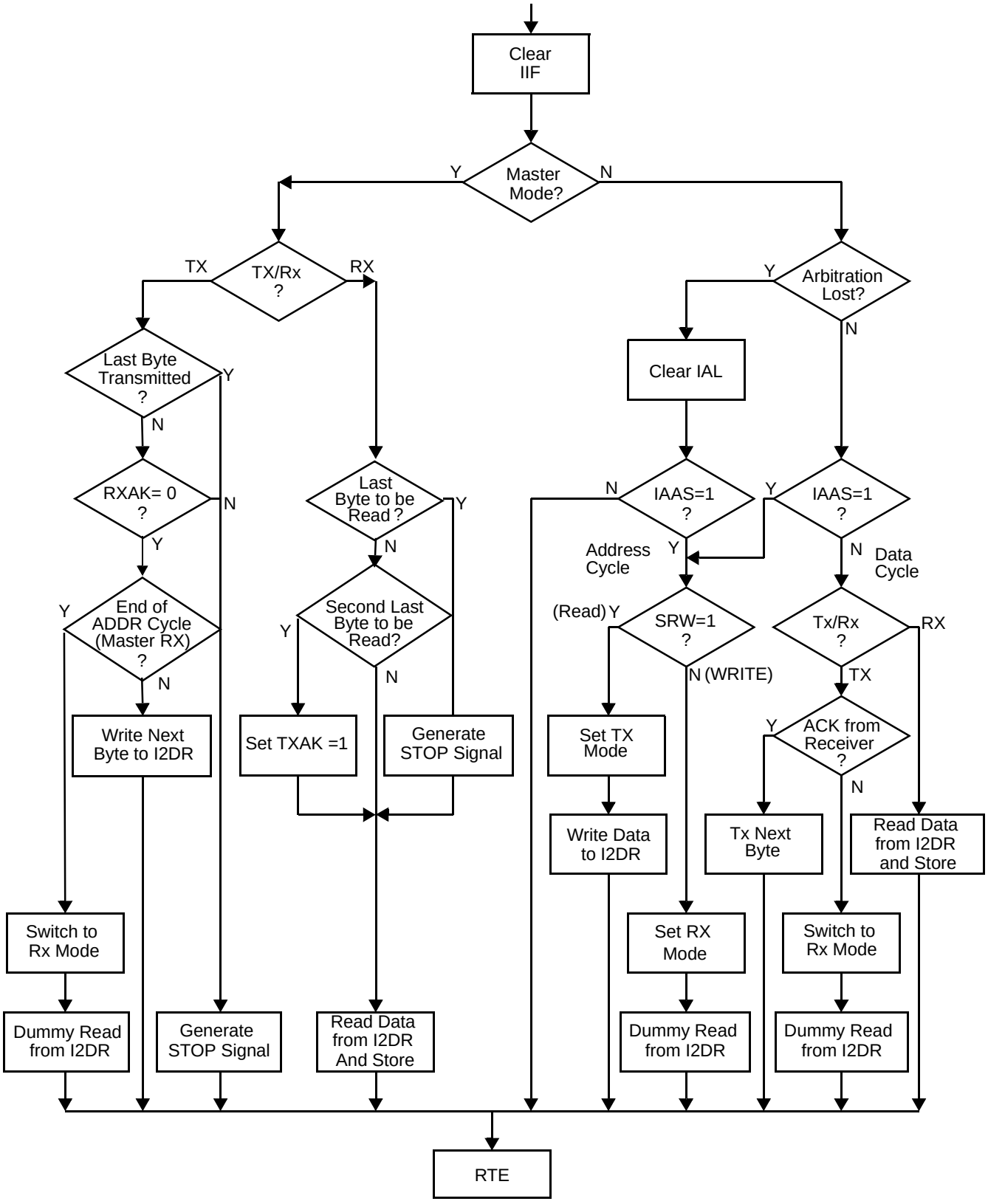


Figure 22-5. Flow-Chart of Typical I^{2}C Interrupt Routine

22.6 I2C Programming Model

The value of the base address for the I2C register file, I2C_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC7119 Register Files*, on page 5-4. Notice that the registers at offsets 0x02, 0x06, 0xa, 0x0e are reserved for future additions. Only 16-bit read or write accesses are allowed for the I2C registers. The I2C registers are listed as follows, along with the number of the page where each register is discussed:

- I2C Address Register (I2AR), **page 22-9**.
- I2C Frequency Register (I2FR), **page 22-10**.
- I2C Control Register (I2CTLR), **page 22-11**.
- I2C Status Register (I2SR), **page 22-12**.
- I2C Data Register (I2DR), **page 22-13**.

I2AR I2C Address Register IADR Base + 0x00

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—								ADR							—
TYPE	R								R/W							R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IADR holds the address for the I2C module when it is addressed as a slave. This is not the address sent on the bus during the address transfer. The register is not reset by a software reset.

Table 22-2. I2AR Bit Descriptions

Name	Reset	Description	Settings
— 15–8	0	Reserved. Write to zero for future compatibility.	
ADR 7–1	0	**Slave Address** Contains the slave address for the I2C module. Slave mode is the default mode for an address match on the bus.	
— 0	0	Reserved. Write to zero for future compatibility.	

I2FR I^2C Frequency Register IFDR Base + 0x04

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—										IC					
TYPE	R										R/W					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2FR provides a programmable prescaler to configure the clock for bit-rate selection. The register is not reset by software reset. The divider values above are *not* arranged in order from lowest to highest. As **Table 22-4** shows, 22 is the smallest divider and 3840 is the largest.

Table 22-3. I^2C Bit Descriptions

Name	Reset	Description	Settings
— 15–6	0	Reserved. Write to zero for future compatibility.	
IC 5–0	0	**I^2C Clock Rate** Prescales the clock for bit-rate selection. Because of potentially slow SCL and SDA rise and fall times, bus signals are sampled at the prescaler frequency. IC can be changed anywhere in a program.	See **Table 22-4**.

Table 22-4. I2C I2FR Field Values

IC	Divider	IC	Divider	IC	Divider	IC	Divider
0x00	30	0x10	288	0x20	22	0x30	160
0x01	32	0x11	320	0x21	24	0x31	192
0x02	36	0x12	384	0x22	26	0x32	224
0x03	42	0x13	480	0x23	28	0x33	256
0x04	48	0x14	576	0x24	32	0x34	320
0x05	52	0x15	640	0x25	36	0x35	384
0x06	60	0x16	768	0x26	40	0x36	448
0x07	72	0x17	960	0x27	44	0x37	512
0x08	80	0x18	1152	0x28	48	0x38	640
0x09	88	0x19	1280	0x29	56	0x39	768
0x0A	104	0x1A	1536	0x2A	64	0x3A	896
0x0B	128	0x1B	1920	0x2B	72	0x3B	1024
0x0C	144	0x1C	2304	0x2C	80	0x3C	1280
0x0D	160	0x1D	2560	0x2D	96	0x3D	1536
0x0E	192	0x1E	3072	0x2E	112	0x3E	1792
0x0F	240	0x1F	3840	0x2F	128	0x3F	2048

I2CTLR I²C Control Register IFDR Base + 0x08

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—								IEN	IIEN	MSTA	MTX	TXAK	RSTA	—	
TYPE	R								R/W						R	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2CTLR enables the I²C module and the I²C interrupt. It also contains bits that govern I²C operation as a slave or a master.

Table 22-5. I2CTLR Bit Descriptions

Name	Reset	Description	Settings
— 15–8	0	Reserved. Write to zero for future compatibility.	
IEN 7	0	**I²C Enable** Enables/disables the I²C module. Also controls the software reset of the entire I²C module. Resetting the bit generates an internal reset to the module. If the module is enabled and in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next start condition is detected. Master mode is not aware that the bus is busy, so initiating a start cycle may corrupt the current bus cycle, ultimately causing either the current master or the I²C module to lose arbitration, after which bus operation returns to normal.	0 I²C module disabled, but registers can still be accessed. 1 I²C module enabled. This bit must be set before any other I²C register bits have any effect.
IIEN 6	0	**I²C Interrupt Enable** Enables/disables I²C interrupts. When IIEN is set, an I²C interrupt occurs if I2SR[IIF] is also set. The interrupt remains asserted as long as I2SR[IIF] and IIEN both remain set.	0 I²C module interrupts are disabled, but currently pending interrupt conditions are not cleared. 1 I²C module interrupts are enabled.
MSTA 5	0	**Master/Slave Mode Select** Selects either master mode or slave mode. If the master loses arbitration, MSTA is cleared without generating a STOP signal. In master mode, changing MSTA from 0 to 1 signals a START on the bus and selects master mode. In slave mode, changing MSTA from 1 to 0 generates a STOP and selects slave mode. The module clock should be on for writing to the MSTA bit.	0 Slave mode. 1 Master mode.
MTX 4	0	**Transmit/Receive Mode Select** Selects the direction of master and slave data transfers. When a slave is addressed, software should set MTX according to I2SR[SRW]. In master mode, MTX should be set according to the type of transfer required. Therefore, for address cycles, MTX is always set to 1.	0 Receive. 1 Transmit.

Table 22-5. I2CTLR Bit Descriptions (Continued)

Name	Reset	Description	Settings
TXAK 3	0	**Transmit Acknowledge Enable** Specifies the value driven onto SDA during acknowledge cycles for both master and slave receivers. Writing TXAK applies only when the I^2C bus is a receiver.	0 An acknowledge signal is sent to the bus at the ninth clock bit after one byte of data is received. 1 No acknowledge signal response is sent (that is, acknowledge bit = 1).
RSTA 2	0	**Repeat Start** Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration.	0 No repeat start. 1 Generates a repeated START condition.
— 1–0	0	Reserved. Write to zero for future compatibility.	

I2SR I^2C Status Register IFDR Base + 0x0C

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									ICF	IAAS	IBB	IAL	—	SRW	IIF	RXAK
TYPE	R											R/W	R		R/W	R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2SR contains bits that indicate transaction direction and status.

Table 22-6. I2SR Bit Descriptions

Name	Reset	Description	Settings
— 15–8	0	Reserved. Write to zero for future compatibility.	
ICF 7	0	**Data Transfer** Indicates whether a data transfer is in progress. While one byte of data is transferred, ICF is cleared.	0 Transfer in progress. 1 Transfer complete. Set by the falling edge of the ninth clock of a byte transfer.
IAAS 6	0	**I^2C Address as Slave** Interrupts the CPU if I2CTLR[IIEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CTLR clears this bit.	0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
IBB 5	0	**I^2C Busy** Indicates the status of the bus. When START is detected, IBB is set. If a STOP signal is detected, IBB is cleared.	0 Bus is idle. 1 Bus is busy

Table 22-6. I2SR Bit Descriptions (Continued)

Name	Reset	Description	Settings
IAL 4	0	**Arbitration Lost** Set by hardware in the following circumstances: • SDA sampled low when the master drives high during an address or data transmit cycle. • SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it. For the first two cases, the bit is set at the falling edge of ninth SCL clock during an ACK cycle. IAL is cleared when software writes a zero to it. Software cannot set this bit.	0 No arbitration lost. 1 Arbitration lost.
— 3	0	Reserved. Write to zero for future compatibility.	
SRW 2	0	**Slave Read/Write** When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I²C module is a slave and has an address match.	0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
IIF 1	0	**I²C Interrupt** Indicates that an interrupt is pending, which causes a processor interrupt request (if IIEN = 1). IFF is set when one of the following occurs: • Complete one byte transfer (set at the falling edge of the ninth clock) • Reception of a calling address that matches its own specific address in slave-receive mode • Arbitration lost.	0 No I²C interrupt pending. 1 Interrupt pending.
RXAK 0	0	**Receive Acknowledge** The value of SDA during the acknowledge bit of a bus cycle.	0 Acknowledge signal received after the completion of the 8-bit data transmission on the bus. 1 No acknowledge signal detected at the ninth clock.

I2DR I²C Data Register I2DR Base + 0x10

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—								D							
TYPE	R								R/W							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In master-receive mode, reading the I2DR allows a read to occur and initiates next byte data receiving. In slave mode, the same function is available after it is addressed. Note that a value written by the SC1400 to the I2DR cannot be read back by the SC1400 core. Only data written on the I^2C bus side can be read.

Table 22-7. I2DR Bit Descriptions

Name	Reset	Description	Settings
— 15–8	0	Reserved. Write to zero for future compatibility.	
D 7–0	0	**Data Byte** Holds the last data byte received or the next data byte to be transferred. Software writes the next data byte to be transmitted or reads the data byte received.	

Universal Asynchronous Receiver/Transmitter (UART) 23

The UART, also known as the serial communication interface (SCI), provides a full-duplex port for serial communications with other MSC7119 devices, microprocessors, or DSPs. This interface uses two dedicated signals: transmit data (UTXD) and receive data (URXD) (see **Figure 23-1**). The extended core accesses the UART through the APB bus. The UART is modeled after the industry-standard 16550, with the register address space relocated to 32-bit data boundaries for APB bus implementation. It contains registers to control the character length, baud rate, parity generation/checking, and interrupt generation.

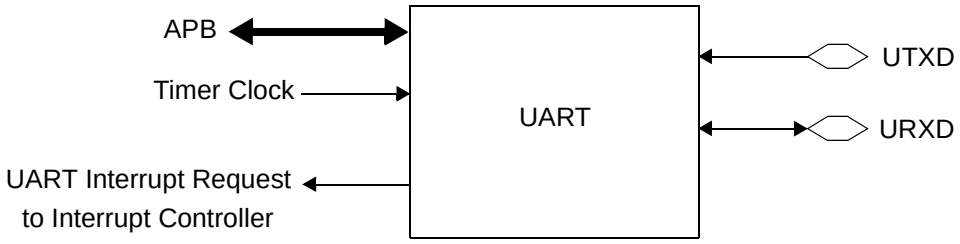


Figure 23-1. UART Interface

23.1 UART Basics

The main functional components of the UART are as follows (see **Figure 23-2**):

- *APB interface*. Standard AMBA 2.0-compliant APB interface with support for 8-, 16-, and 32-bit read and write bus widths.
- *Control and status registers*. Serial data control registers are stored and used for control and status generation. These registers control interrupt generation based on transmitter and receiver status and determine which interrupts are enabled.
- *Rx and Tx control*. Controls the data transfer between the Receive Buffer Register (RBR) and Rx block and the Transmit Holding Register (THR) and the Tx block.
- *Transmitter (Tx)*. Converts parallel data programmed into the THR into a serial data stream that is built according to conditions specified in the Line Control Register. The serial data is output on the UTXD pin.
- *Receiver (Rx)*. Converts serial data sent to the UART on the URXD pin to a parallel data character based on Line Control Register settings. When a complete character is received, it is sent to the RBR.

- *Baud clock generator*. Generates the clock for the Tx and Rx to serialize the data from THR or to receive the data into Rx. Serial clock frequency is based on the PCLK, DLL, and DLH registers.

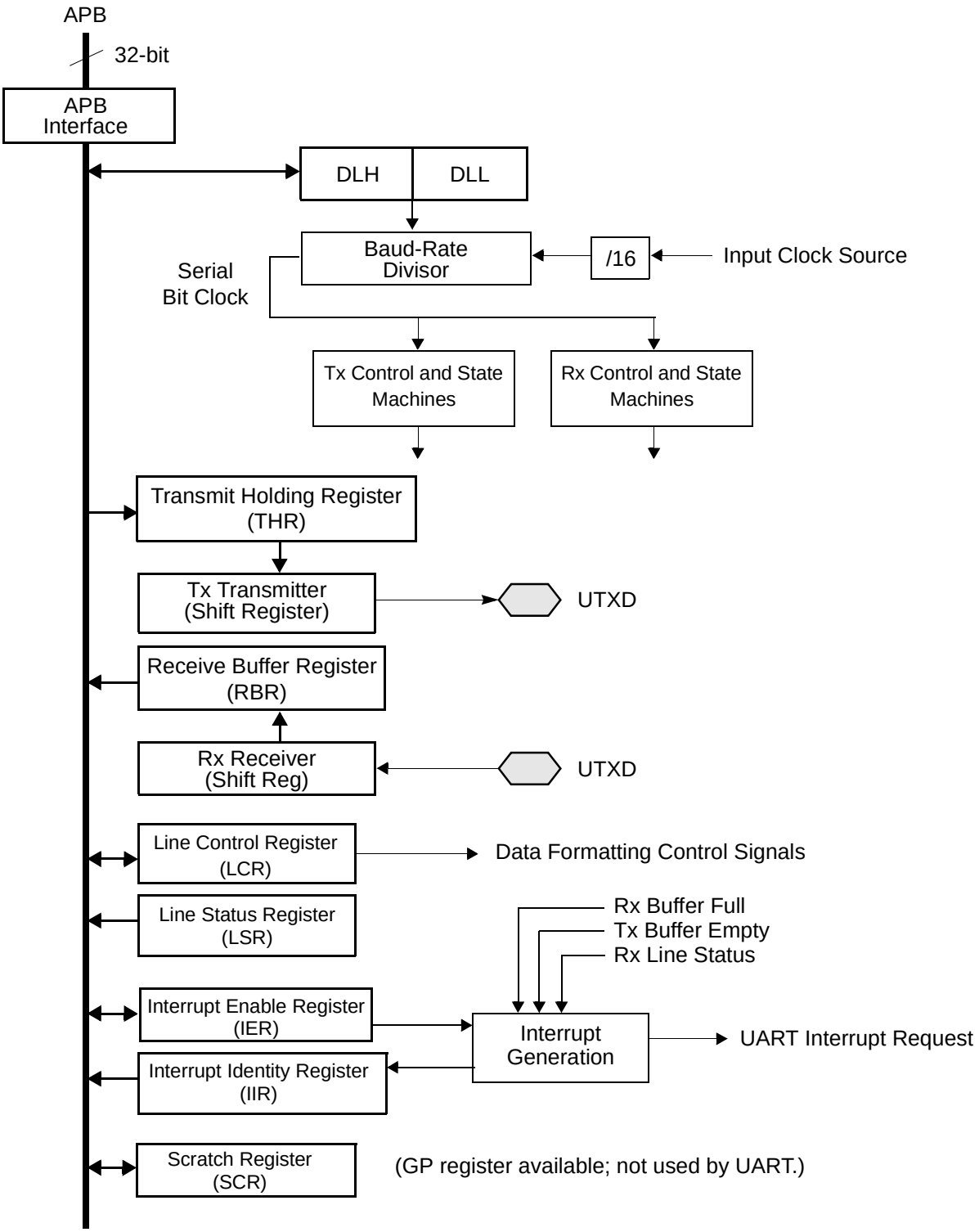


Figure 23-2. UART Block Diagram

Table 23-1 lists the serial interface signals.

Table 23-1. Serial Interface Signals

Signal Name	Type	Description
URXD	I	The serial data input from the modem, data set, or peripheral device (see **Figure 23-3**).
UTXD	O	The serial data output to the modem, data set, or peripheral device (see **Figure 23-3**). SOUT is driven high when the reset (preset) is applied.

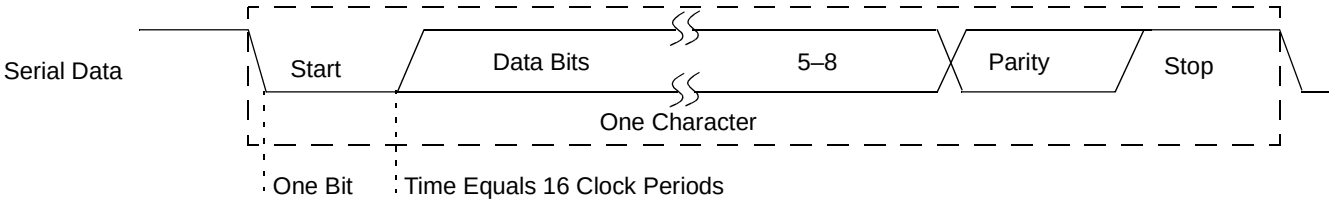


Figure 23-3. Serial Data Format

23.2 Halting and Restarting the UART

To halt the UART completely, use the following procedure:

1. Optionally, poll the LSR[TEMT] bit (**page 23-7**) to ensure that the UART is not transmitting data. Omitting this step can result in transmission of invalid data.
2. Set the HLTREQ[UARTCD] bit (**page 11-26**) to shut down the system clock to the module. Asserting this bit during transmission or reception can result in invalid data.

To restart the UART, clear the HLTREQ[UARTCD] bit to re-enable the system clock to the module.

23.3 UART Programming Model

This section describes the programmable features of the UART. The value of the base address for the `UART_BASE` register file is in **Table 5-1**, *Summary — Base Addresses for MSC7119 Register Files*, on page 5-4. Control registers should not be changed while the transmitter is busy, as indicated by the LSR[TEMT] bit. The UART registers are listed as follows, along with the number of the page where each register is discussed:

- Receive Buffer Register (RBR), **page 23-4**.
- Transmit Holding Register (THR), **page 23-4**.
- Baud-Rate Divider Register (BRDR), **page 23-4**.
- Interrupt Enable Register (IER), **page 23-5**.
- Interrupt Identity Register (IIR), **page 23-5**.
- Line Control Register (LCR), **page 23-7**.

- Line Status Register (LSR), **page 23-7**.
- Scratchpad Register (SCR), **page 23-8**.

RBR Receive Buffer Register 0x00

BIT	7	6	5	4	3	2	1	0
	RBR							
TYPE	R							
RESET	0	0	0	0	0	0	0	0

RBR contains the data byte received on the serial input port. The data in this register is valid only if the LSR[DR] (Data Ready) bit is set. This register is accessible only when LCR[DLAB] is cleared to 0. The data in the RBR must be read before the next data arrives. Otherwise, it is overwritten, resulting in an overrun error.

THR Transmit Holding Register 0x00

BIT	7	6	5	4	3	2	1	0
	THR							
TYPE	W							
RESET	0	0	0	0	0	0	0	0

THR contains data to be transmitted on the serial output port. Data can be written to the THR any time the LSR[THRE] bit is set.Writing a single character to the THR clears the THRE. Any additional writes to the THR before the THRE is set again causes the THR data to be overwritten. This register is accessible only when LCR[DLAB] is cleared to 0.

BRDR Baud-Rate Divider Register 0x00 0x04

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DLH								DLL							
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BRDR contains the baud-rate divisor for the UART. It is accessed by first setting the LCR[DLAB] bit (**page 23-7**). The output baud rate is baud rate = (PCLK) / (16 × divisor). When both DLH and DLL are set, at least (2 × divisor × 16) clock cycles of the UART should be

allowed to pass before data is transmitted or received. This 16-bit is accessible only by its 8-bit fields, DLH and DLL.

IER Interrupt Enable Register 0x04

Bit	7	6	5	4	3	2	1	0
	—	—	—	—	EDSSI	ELSI	ETBEI	ERBFI
TYPE	R/W							
RESET	0	0	0	0	0	0	0	0

IER contains four bits to enable interrupt generation. This register is accessible only when LCR[DLAB] is cleared to 0.

Table 23-2. IER Bit Descriptions

Name	Description	Settings
— 7–4	Reserved. Write to zero for future compatibility.	
EDSSI 3	**Enable Modem Status Interrupt** Enables/disables a modem status interrupt.	0 Interrupt disabled. 1 Interrupt enabled.
ELSI 2	**Enable Line Status Interrupt** Enables/disables a line status interrupt.	0 Interrupt disabled. 1 Interrupt enabled.
ETBEI 1	**Enable Transmit Buffer Empty Interrupt** Enables/disables a transmit buffer empty interrupt.	0 Interrupt disabled. 1 Interrupt enabled.
ERBFI 0	**Enable Receive Buffer Full Interrupt** Enables/disables a receive buffer full interrupt.	0 Interrupt disabled. 1 Interrupt enabled.

IIR Interrupt Identity Register 0x08

Bit	7	6	5	4	3	2	1	0
	—				INTID			
TYPE	R							
RESET	0	0	0	0	0	0	0	0

IIR identifies the source of an interrupt. The upper four bits of the register are reserved and have a value of 00. The lower four bits identify the pending interrupt with the highest priority, as shown in **Table 23-3**.

Table 23-3. IIR Bit Descriptions

Name	Description	Settings
— 7–4	Reserved. Write to zero for future compatibility.	
INTID 3–0	**Interrupt ID** Identifies the source of the interrupt.	

INTID[3–0]	Interrupt Identification
0000	Modem Status Changed.
0001	No Interrupt Pending.
0010	THR Empty.
0011	Reserved.
0100	Received Data Available.
0101	Reserved.
0110	Receiver Status.
0111	Reserved.
10xx	Reserved.
1100	Character Time Out.
1101	Reserved.
111x	Reserved.

Table 23-4. UART Interrupt Structure

Reserved	Interrupt Identification Register			Interrupt Set and Reset Functions			
Bit 3	**Bit 2**	**Bit 1**	**Bit 0**	**Priority Level**	**Interrupt Type**	**Interrupt Source**	**Interrupt Reset Control**
0	0	0	1	—	None	None	—
0	1	1	0	Highest	Receiver Line Status	Overrun/parity/framing errors or break interrupt	Reading the LSR
0	1	0	0	Second	Received data available	Receiver data available	Reading the RBR
0	0	1	0	Third	Transmitter holding register empty	Transmitter holding register	Reading the IIR or writing into THR
0	0	0	0	Fourth	Modem status	Clear to send or data set ready or ring indicator or data center detect	Reading the modem status register

LCR — Line Control Register — 0x0C

Bit	7	6	5	4	3	2	1	0
	DLAB	BRK	STKP	EPS	PEN	STOP	CLS	
TYPE	R							
RESET	0	0	0	0	0	0	0	0

LCR controls the format of the data that is transmitted and received by the UART.

Table 23-5. LCR Bit Descriptions

Name	Description	Settings
DLAB 7	**Divisor Register Access Bit** Provides an extra bit when the UART register file is addressed.	
BRK 6	**Break Control** Generates a break on the UTXD line.	0 No break generated. 1 Break generated.
STKP 5	**Stick Parity** Not used.	
EPS 4	**Even Parity Select** Parity must be enabled.	0 Odd parity is transmitted or checked. 1 Even parity is transmitted or checked.
PEN 4	**Parity Enable** Enables/disables parity.	0 Parity is disabled. 1 Parity is enabled.
STOP 4	**Number of Stop Bits Transmitted** Specifies the number of stop bits transmitted.	0 One stop bit is transmitted in serial data. 1 If data bits is set to 5, one and a half stop bits are generated. Otherwise, two stop bits are generated and transmitted in serial data out.
CLS 1–0	**CLS** Controls number of bits per character.	00 5 bits. 01 6 bits. 10 7 bits. 11 8 bits.

LSR — Line Status Register — 0x14

Bit	7	6	5	4	3	2	1	0
	—	TEMT	THRE	BI	FE	PE	OE	DR
TYPE	R							
RESET	0	1	0	0	0	0	0	0

LSR contains the status of the receiver and transmitter data transfers. You can read this status any time. The OE, PE and FE bits are reset when the LSR is read.

Table 23-6. LSR Bit Descriptions

Name	Description	Settings
— 7	Reserved. Write to zero for future compatibility.	
TEMT 6	**Transmitter Empty** Indicates that the transmitter is completely empty. TEMT is useful for ensuring that it is safe to change control registers. Changing control registers while the transmitter is busy can result in corrupted data.	0 THR and the Transmit Shift register are not both empty. 1 THR and Transmit Shift register are both empty.
THRE 5	**Transmitter Holding Register Empty** Indicates the UART can accept a new character for transmission. THRE is set when data is transferred from the THR to the transmit shift register and no new data is written to the THR.	0 THR is not empty. 1 THR is empty.
BI 4	**Break Interrupt** Set when the serial input (S_{in}) is held in a logic 0 state for longer than the sum of start time + data bits + parity + stop bits. When a break condition occurs on S_{in}, the UART receives one and only one character, consisting of all zeros. The BI indication occurs immediately and persists until the LSR is read	0 No break interrupt. 1 Break interrupt.
FE 3	**Framing Error** Indicates a framing error in the receiver. A framing error occurs when the receiver does not detect a valid stop bit in the received data.	0 No framing error. 1 Framing Error has occurred.
PE 2	**Parity Error** This status bit indicates parity errors only when parity is enabled in the LCR[PEN] bit.	0 No parity error. 1 Parity error.
OE 1	**Overrun Error** Indicates an overrun condition when a new character arrives in the receiver before the previous character is read from the RBR. The data in the RBR is overwritten.	0 No overrun. 1 Overrun.
DR 0	**Data Ready** Cleared when the RBR is read	0 Receiver does not contain data in the RBR. 1 Receiver contains data in the RBR.

SCR — Scratchpad Register — 0x1C

Bit	7	6	5	4	3	2	1	0
	SCR							
TYPE	R/W							
RESET	0	0	0	0	0	0	0	0

SCR is available for programmers to use as a temporary storage space. It has no defined purpose in the UART.

General-Purpose Input/Output (GPIO) 24

The MSC711x general-purpose I/O (GPIO) signal lines are multiplexed to serve as either GPIO signals or dedicated peripheral interface signals. Each GPIO signal is configured as an input/output, with a register for data output that is read or written at any time. GPIO signals do not have internal pull-up resistors. The dedicated MSC711x peripheral functions multiplexed with the GPIO signals are grouped to maximize their usefulness in the greatest number of MSC711x applications. All the GPIO pins are tri-stated in test mode when reset is asserted active low.

It is recommended that you read the following chapters on MSC711x peripherals before you read this one:

- **Chapter 18**, *Fast Ethernet Controller (FEC)*
- **Chapter 19**, *Time-Division Multiplexing (TDM) Interface*
- **Chapter 20**, *Host Interface (HDI16)*
- **Chapter 21**, *Timers Module*
- **Chapter 22**, *I2C Software Module*
- **Chapter 23**, *Universal Asynchronous Receiver/Transmitter (UART)*

24.1 GPIO Features

Features of the GPIO signals are as follows:

- Four ports, A to D, that are separately configurable for input or output or tri-state.
- Bits in each port can be independently configured for direction (input or output) and control:
  - Software control, with pin data and direction determined by GPIO registers.
  - Hardware control, with pin data and direction determined by a peripheral module.
- Separate data registers and data direction registers for each port, for use in software control mode.
- Data on port is always readable, regardless of pin mode or direction.

Port A of the GPIO block supports additional functionality, as follows:

- All signals that can generate independent interrupts.
- Configurable interrupt detection: edge detection, level-sensitive operation (unsynchronized), or level-sensitive operation (synchronized).
- Configurable interrupt detection polarity:
  - Active high or rising edge
  - Active low or falling edge
- Readable interrupt status before or after masking

24.2 Operating Modes

The GPIO signals operate either under software control as GPIO pins or under hardware control as dedicated peripheral pins.

24.2.1 Software Control Mode as GPIO Pins

When a port is configured for software control, the data and direction control for the port are sourced from the data register (GPxDR) (**page 24-18**) and the direction control register (GPxDDR) (**page 24-17**), where x designates port A, B, C, or D. The data written to GPxDDR is mapped onto an output port that controls the direction of an external I/O pad. The data written to the data register (GPxDR) drives the output buffer of the I/O pad when the direction is set to *output*. External data is input on the external data port. If the data direction is set to *input*, then a read of the external port register (GPxEXPRT) (**page 24-19**) shows the value on the port. This register is read-only, so it cannot be written from the APB software interface. Otherwise, if the direction is set to *output*, the value in the data register (GPxDR) is returned.

24.2.2 Hardware Control Mode as Peripheral Pins

If a port is configured for hardware control, its external auxiliary hardware signals control the data and the direction of ports A through D. In hardware control mode, the internal auxiliary data input signal and direction control signal are selected for the designated port. **Figure 24-2** shows how the GPIO peripheral controls the data and direction ports of an I/O PAD and the generation of data for the auxiliary source. The value on the external port is masked and returned to the auxiliary source.

24.2.3 Reading External Ports

A separate port is provided for reading the value on any port pin. Regardless of whether Software Control mode or Hardware Control mode is selected, the data on the external GPIO port can always be read via an APB read of the memory-mapped register GPxEXPRT (described on **page 24-19**). The data is first synchronized with teh AHB clock before it is read from the port. The maximum rate for reading data on this port is half the AHB clock frequency.

When the data direction is set to input, a read of the external port register (GPxEXPRT) produces the value on the port. When the data direction is set to output, a read returns the value of the memory-mapped data register GPxDR (described on **page 24-18**). Both the external GPIO port register and the data register (GPxDR) return the same value for both modes of operation.

Figure 24-2 shows the multiplexing of the hardware/software option with the control lines for the multiplexing coming from a memory mapped-register. It also shows the synchronization registers and the individual bit control of each data and data direction bit.

24.3 GPIO Architecture

Figure 24-1 shows the functional groupings of the main interfaces to the GPIO block:

- APB interface to/from the APB bridge
- External data interface to/from the I/O pads
- Auxiliary hardware data interface to/from auxiliary data sink/source
- Interrupt interface to/from the interrupt controller

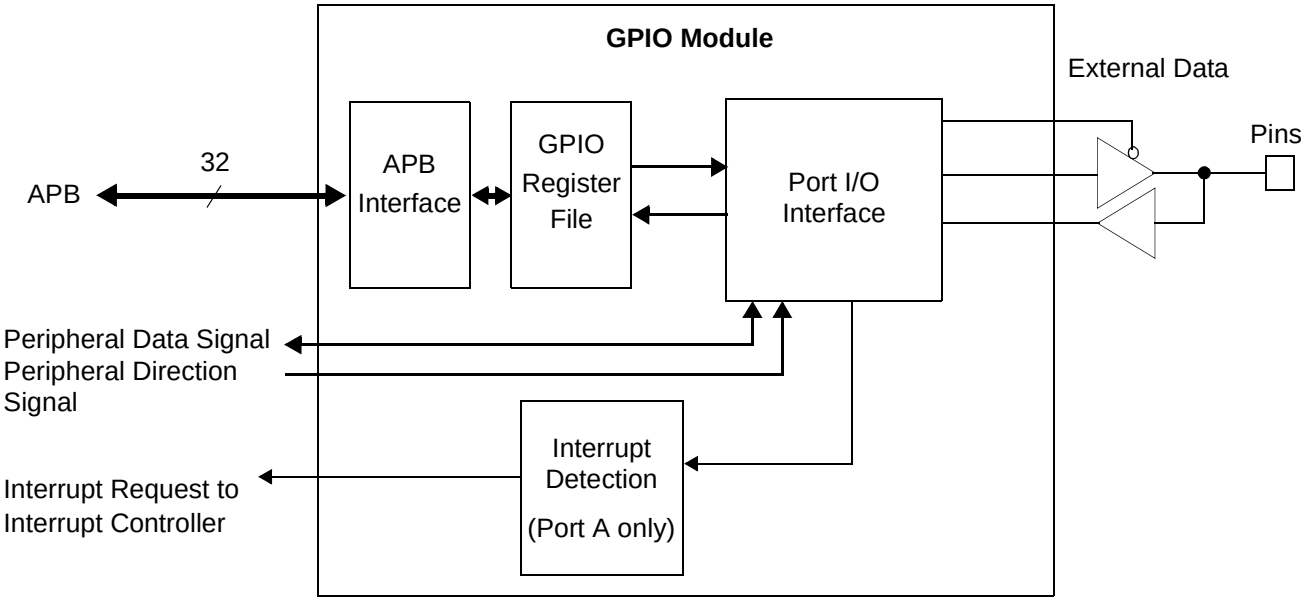


Figure 24-1. GPIO High-Level Functional Block Diagram

24.3.1 Data and Control Flow

The GPIO controls the output data and direction of external I/O pads. Software controls the input/output data and each port over the APB bus interface. Hardware control of the input/output data for each port is enabled through the control register. Each bit in each port is individually controllable, so each register can be regarded as *N* individual registers, where *N* is the port width. **Figure 24-2** shows the data and flow control for the ports. The default direction of any port is set at configuration time.

24.3.2 GPIO Port Assignments

This section describes how ports A, B, C, and D are allocated. Remember, only port A has interrupt capability. An MSC711x device comes out of reset with the primary functional pin enabled as an input. Signals with no GPIO for the primary function come out of reset as unused.

Multiple functionality is multiplexed to each port and is under software control. Peripheral functionality is not available until a port is configured to be under hardware control so that the internal peripheral can determine the direction of the signal (in/out/inout). When the signals are not configured for their primary function, bits in the DEVCFG register (**Section 7.4.3**, *Device Identification and Configuration*, on page 7-16) select between the secondary and additional functions. This is not the case for pins sampled out of reset, which are sampled independently of the state of DEVCFG bits.

When a port is configured for GPIO software control, the data register, GPxDR controls the data. The direction control register, GPxDDR, controls the direction of a data transfer. If the data direction is set to input, the value on the port is read in the external port register (GPxEXPRT). This register is read-only, so it cannot be written from the APB software interface. Otherwise, if the direction is set to output, the value in the Port X data register (GPxDR) is returned.

24.3.2.1 Port Configuration Out of Reset

An MSC711x device comes out of reset with the functionality shown in the “Software Control” column of **Table 24-1** through **Table 24-5** enabled as an input. For pins with no GPIO for the primary function, the pin comes out of reset as an unused input.

Note: To ensure compatibility with future devices, an application must not assign any of the reserved states in the port assignment tables presented in this section.

24.3.2.2 Port A

Port A contains all signals with interrupt capability, including the $\overline{\text{NMI}}$ signals and signals for the UART, I^2C, TDM1, and TDM0 modules, as well as the event port and Ethernet MAC signals with interrupt capability. The DEVCFG[PAS] bit (**page 7-17**) selects between the secondary function and the addition multiplexing function.

Table 24-1 shows the port A configurations for MSC711x devices containing an Ethernet MAC. **Table 24-2** shows the port A configurations for devices containing a third TDM.

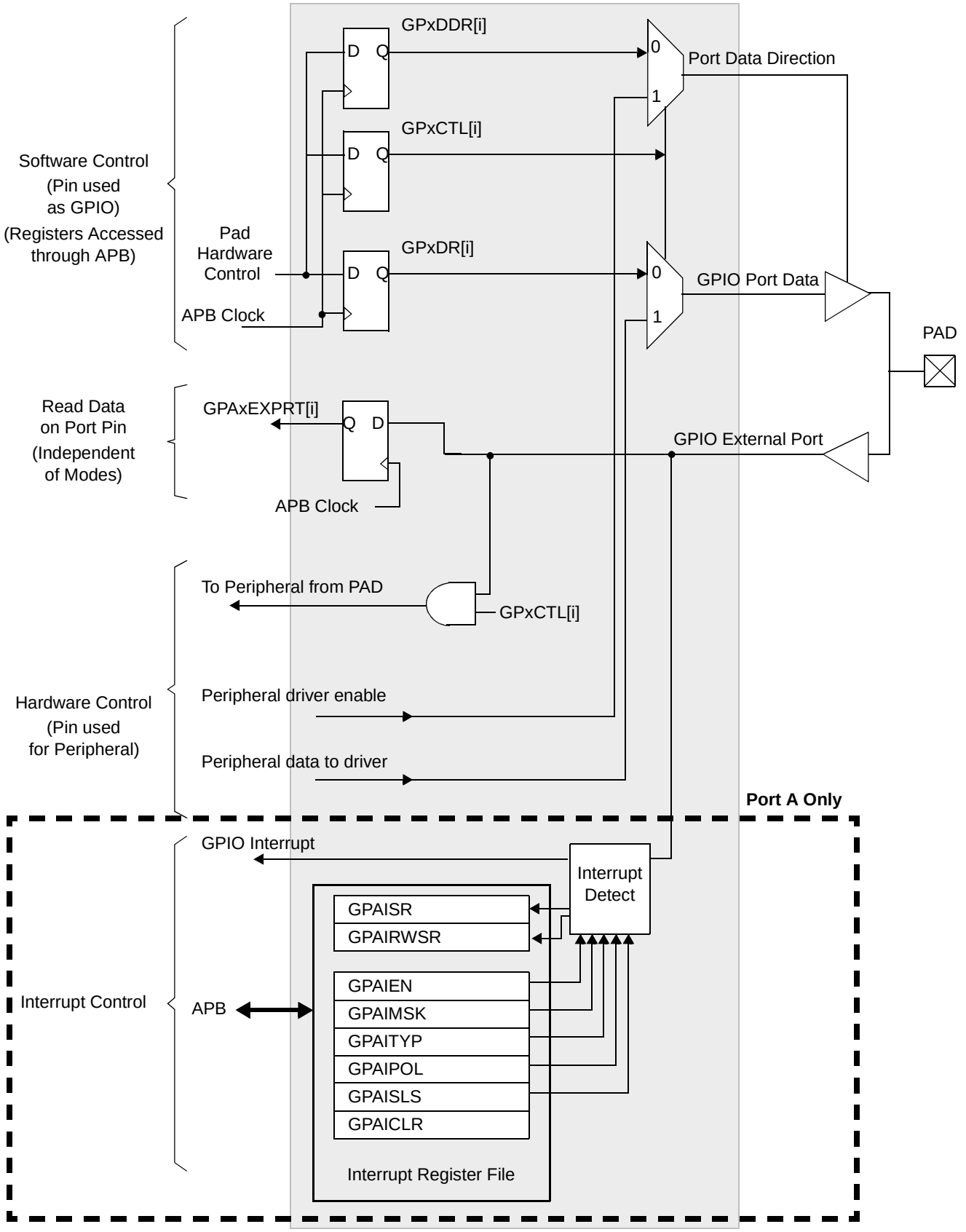


Figure 24-2. Block Diagram of one Bit (Ports A, B, C, D)

Table 24-1. Port A GPIO Signal Pin Assignments for Devices With Ethernet MAC

Bit	Software Control	Hardware Control		Interrupt Capability (Input)	Sampled on POR (Input)
	GPACTL[i]==0	GPACTL[i]==1 and DEVCFG[PAS]==0	GPACTL[i]==1 and DEVCFG[PAS]==1		
29	GPIO	RXD3 (I)	Reserved	$\overline{\text{IRQ18}}$	—
28	GPIO	TX_ER	Reserved	$\overline{\text{IRQ17}}$	—
27	GPIO	TXD3	Reserved	$\overline{\text{IRQ16}}$	—
26	GPIO	RX_ER		$\overline{\text{IRQ26}}$	—
25	GPIO	RX_DV		$\overline{\text{IRQ25}}$	—
24	GPIO	TX_EN		$\overline{\text{IRQ24}}$	—
23	GPIO	TXCLK		$\overline{\text{IRQ23}}$	—
22	GPIO	RXD0		$\overline{\text{IRQ22}}$	—
21	GPIO	RXD1		$\overline{\text{IRQ21}}$	—
20	GPIO	TXD0		$\overline{\text{IRQ20}}$	—
19	GPIO	TXD1		$\overline{\text{IRQ19}}$	—
18	Reserved for $\overline{\text{NMI}}$ signal (not configurable)				—
17	GPIO	EVNT1 or CLKO		$\overline{\text{IRQ13}}$	—
16	GPIO	EVNT4		$\overline{\text{IRQ12}}$	SWTE
15	GPIO	SCL		$\overline{\text{IRQ14}}$	—
14	GPIO	SDA		$\overline{\text{IRQ15}}$	—
13	GPIO	URXD		$\overline{\text{IRQ2}}$	—
12	GPIO	UTXD		$\overline{\text{IRQ3}}$	—
11	GPIO	T0RCK		$\overline{\text{IRQ4}}$	—
10	GPIO	T0RFS		$\overline{\text{IRQ5}}$	—
9	GPIO	T0RD		Reserved	—
8	GPIO	T0TCK		$\overline{\text{IRQ6}}$	—
7	GPIO	T0TFS		$\overline{\text{IRQ7}}$	—
6	GPIO	T0TD		Reserved	—
5	GPIO	T1RCK		$\overline{\text{IRQ0}}$	—
4	GPIO	T1RFS		$\overline{\text{IRQ1}}$	—
3	GPIO	T1RD		$\overline{\text{IRQ8}}$	—
2	GPIO	T1TCK		$\overline{\text{IRQ9}}$	—
1	GPIO	T1TFS		$\overline{\text{IRQ10}}$	—
0	GPIO	T1TD		$\overline{\text{IRQ11}}$	—

Notes:
1. The reset state of these pins is is the “Software Control” column. When this is reserved, the pin operates as an input.
2. When CLKCTL[CLKO] != 00, then CLKO is driven onto Port 17 in Hardware Control mode.
   When CLKCTL[CLKO] == 00, EVNT1 is driven onto Port 17 in Hardware Control mode.
   Bit 17 must be programmed in Hardware Control mode when CLKO is desired as an output.
3. When the Ethernet MAC is used:
   – For MII mode, DEVCFG[PAS] must be cleared for correct operation.
   – For RMII or 7-Wire Interface mode, DEVCFG[PAS] must be set to ensure future compatibility.
4. When bit 13 is in software control mode, the UART input is internally tied to 1, disabling it.

Table 24-2. Port A GPIO Signal Pin Assignments for Devices With a Third TDM (TDM2)

Bit	Software Control	Hardware Control		Interrupt Capability (Input)	Sampled on POR (Input)
	GPACTL[i]==0	GPACTL[i]==1 and DEVCFG[PAS]==0	GPACTL[i]==1 and DEVCFG[PAS]==1		
29	GPIO	Reserved	T2TFS	$\overline{\text{IRQ18}}$	—
28	GPIO	Reserved	T2RD	$\overline{\text{IRQ17}}$	—
27	GPIO	Reserved	T2RCK	$\overline{\text{IRQ16}}$	—
26	GPIO	Reserved		$\overline{\text{IRQ26}}$	—
25	GPIO	Reserved		$\overline{\text{IRQ25}}$	—
24	GPIO	Reserved		$\overline{\text{IRQ24}}$	—
23	GPIO	Reserved		$\overline{\text{IRQ23}}$	—
22	GPIO	Reserved		$\overline{\text{IRQ22}}$	—
21	GPIO	Reserved		$\overline{\text{IRQ21}}$	—
20	GPIO	Reserved		$\overline{\text{IRQ20}}$	—
19	GPIO	Reserved		$\overline{\text{IRQ19}}$	—
18	Reserved for $\overline{\text{NMI}}$ signal (not configurable)				—
17	GPIO	EVNT1 or CLKO		$\overline{\text{IRQ13}}$	—
16	GPIO	EVNT4		$\overline{\text{IRQ12}}$	SWTE
15	GPIO	SCL		$\overline{\text{IRQ14}}$	—
14	GPIO	SDA		$\overline{\text{IRQ15}}$	—
13	GPIO	URXD		$\overline{\text{IRQ2}}$	—
12	GPIO	UTXD		$\overline{\text{IRQ3}}$	—
11	GPIO	T0RCK		$\overline{\text{IRQ4}}$	—
10	GPIO	T0RFS		$\overline{\text{IRQ5}}$	—
9	GPIO	T0RD		Reserved	—
8	GPIO	T0TCK		$\overline{\text{IRQ6}}$	—
7	GPIO	T0TFS		$\overline{\text{IRQ7}}$	—
6	GPIO	T0TD		Reserved	—
5	GPIO	T1RCK		$\overline{\text{IRQ0}}$	—
4	GPIO	T1RFS		$\overline{\text{IRQ1}}$	—
3	GPIO	T1RD		$\overline{\text{IRQ8}}$	—
2	GPIO	T1TCK		$\overline{\text{IRQ9}}$	—
1	GPIO	T1TFS		$\overline{\text{IRQ10}}$	—
0	GPIO	T1TD		$\overline{\text{IRQ11}}$	—

Notes:
1. The reset state of these pins is is the “Software Control” column. When this is reserved, the pin operates as an input.
2. When CLKCTL[CLKO] != 00, CLKO is driven onto port 17 in Hardware Control mode.
   When CLKCTL[CLKO] == 00, EVNT1 is driven onto port 17 in Hardware Control mode.
   Bit 17 must be programmed in Hardware Control mode when CLKO is an output.
3. When TDM2 functionality is required, DEVCFG[PAS] must be set.
4. When bit 13 is in software control mode, the UART input is internally tied to 1, disabling it.

24.3.2.3 Port B

Port B contains a portion of the host port pins. HDSP is an input sampled only on power-on reset.

Table 24-3. Port B GPIO Signal Pin Assignments

Bit	Software Control	Hardware Control	Sampled on POR
	GPBCTL[i]==0	GPBCTL[i]==1	(Input)
15	Reserved	Reserved	—
14	Reserved	HDDS	—
13	Reserved	$\overline{\text{HDS}}$	—
12	Reserved	HRW	—
11	GPIO[2]	$\overline{\text{HCS2}}$	—
10	Reserved	$\overline{\text{HCS1}}$	—
9	Reserved	$\overline{\text{HACK}}$	—
8	Reserved	$\overline{\text{HREQ}}$	HDSP
7	Reserved	HD7	—
6	Reserved	HD6	—
5	Reserved	HD5	—
4	Reserved	HD4	—
3	Reserved	HD3	—
2	Reserved	HD2	—
1	Reserved	HD1	—
0	Reserved	HD0	—

Notes: **1.** The reset state of these pins is the “Software Control” column. For cases where this is reserved, the pin operates as an input.

2. GPIO functionality only available in mask set 1M88B.

24.3.2.4 Port C

Port C contains the remaining host port pins as well as non-interrupting pins for the debug and event ports (used also to specify the boot mode).

Table 24-4. Port C GPIO Signal Pin Assignments

Bit	Software Control	Hardware Control	Sampled on POR
	GPCCTL[i]==0	GPCCTL[i]==1	(Input)
15	GPIO	EVNT3	BM1
14	GPIO	EVNT2	BM0
13	Reserved	EVNT0	—
12	Reserved	Reserved	—
11	GPIO[2]	HA3	—
10	Reserved	HA2	—
9	Reserved	HA1	—
8	Reserved	HA0	—
7	GPIO	HD15	—
6	GPIO	HD14	—
5	GPIO	HD13	—
4	GPIO	HD12	—
3	GPIO	HD11	—
2	GPIO	HD10	—
1	GPIO	HD9	—
0	GPIO	HD8	—

Notes: **1.** The reset state of these pins is the “Software Control” column. For cases where this is reserved, the pin operates as an input.

2. GPIO functionality only available in mask set 1M88B.

24.3.2.5 Port D

Port D contains the Ethernet MAC pins, which do not have interrupt capability. H8BIT is an input sampled only during power-on reset. The operation of some of these port pins in Hardware Control mode is selected via the DEVCFG[PDS] bit (see **Section 7.4.3**, *Device Identification and Configuration*, on page 7-16). **Table 24-5** shows the port D signal pin configurations for MSC711x devices containing an Ethernet MAC.

Table 24-5. Port D GPIO Signal Pin Assignments for Devices With an Ethernet MAC

Bit	Software Control	Hardware Control		Sampled on POR (Input)
	GPDCTL[i]==0	GPDCTL[i]==1 and DEVCFG[PDS]==0	GPDCTL[i]==1 and DEVCFG[PDS]==1	
8	GPIO[3]	Reserved	Reserved	BM3
7	GPIO[3]	Reserved	Reserved	BM2
6	GPIO	RXD2	T2TD	—
5	GPIO	RXCLK	T2TCK	—
4	GPIO	TXD2	T2RFS	—
3	Reserved	MDIO		—
2	Reserved	MDC		H8BIT
1	Reserved	CRS		—
0	Reserved	COL		—

Notes: **1.** The reset state of these pins is stated in the “Software Control” column. For cases where this is reserved, the pin operates as an input.

2. When the Ethernet MAC is in use
- For MII mode, DEVCFG[PDS] must be cleared for correct operation.
- For RMII or 7-Wire Interface mode, DEVCFG[PDS] must be set to ensure future compatibility.

3. GPIO functionality only available in mask set 1M88B.

Table 24-6. Port D Signal Pin Assignments for Devices With a Third TDM (TDM2)

Bit	Software Control	Hardware Control		Sampled on POR (Input)
	GPDCTL[i]==0	GPDCTL[i]==1 and DEVCFG[PDS]==0	GPDCTL[i]==1 and DEVCFG[PDS]==1	
8	GPIO[3]	Reserved		BM3
7	GPIO[3]	Reserved		BM2
6	GPIO	RXD2	T2TD	—
5	GPIO	RXCLK	T2TCK	—
4	GPIO	TXD2	T2RFS	—
3	Reserved	MDIO		—
2	Reserved	MDC		H8BIT
1	Reserved	CRS		—
0	Reserved	COL		—

Notes: **1.** The reset state of these pins is is the “Software Control” column. For cases where this is reserved, the pin operates as an input.

2. When TDM2 functionality is required, DEVCFG[PDS] must be set.

3. GPIO functionality only available in mask set 1M88B.

24.4 Interrupts

Port A can be programmed to accept external signals as interrupt sources on any bits of the port. The type of interrupt is programmable, with the following selections:

- Active-high and level
- Active-low and level
- Rising edge
- Falling edge

Interrupts are masked through the GPAIMSK register. The interrupt status can be read before and after masking. When port A is configured for interrupts, the data direction must be set to input, and the mode must be set to software control for interrupts to be latched. If the data direction register is reprogrammed to output or the mode register is programmed to enable hardware mode, any pending interrupts are not lost. However, no new interrupts are generated. **Figure 24-3** illustrates how the interrupts are generated and how the data flows. The signal names in the diagram correspond to either I/O signals or memory mapped registers.

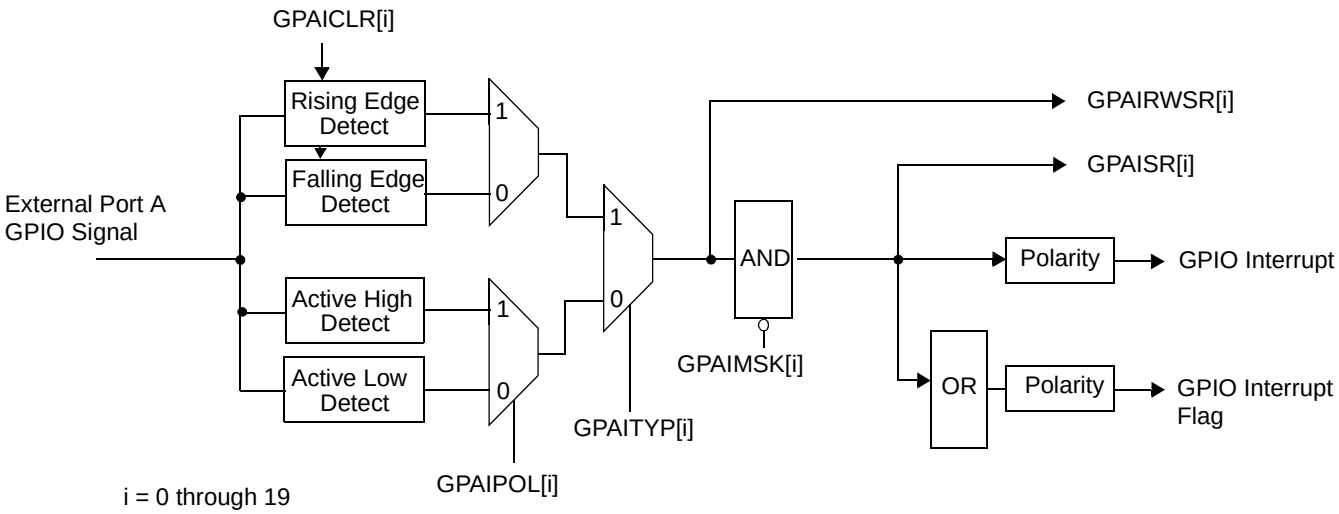


Figure 24-3. GPIO Interrupt Block Diagram

24.4.1 Clearing Interrupts

The interrupt service routine (ISR) can clear edge-detected interrupts by writing to the GPAICLR register to disable the interrupt. This write also clears the interrupt status and raw status registers. Writing to the GPAICLR register has no effect on level-sensitive interrupts. If level-sensitive interrupts are causing the processor to interrupt, the ISR can poll the GPAIRWSR register until the interrupt source disappears, or it can write to the GPAIMSK register to mask the interrupt before exiting the ISR.

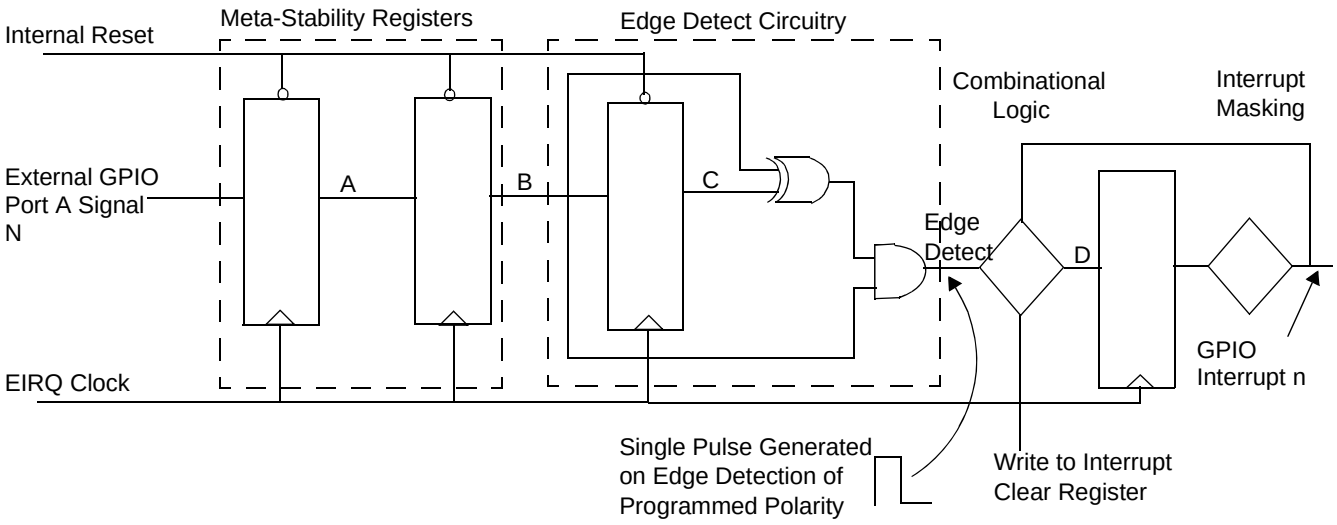
If the ISR exits without masking or disabling the interrupt, the level-sensitive interrupt repeatedly requests an interrupt until the interrupt is cleared at the source. There are no restrictions on the number of edge-detected interrupts that can be cleared simultaneously by writing to the GPAICLR register.

24.4.2 Synchronizing Interrupt Signals with the System Clock

Interrupt signals are synchronized internally with a free running system clock, AHB clock. The EIRQ clock signal is used to detect edge-triggered interrupts while the SC1400 core is in sleep or stop mode. The EIRQ clock is a version of the AHB clock that is enabled by the HLTREQ[EIRQHR] bit (**page 11-28**).

24.4.2.1 Interrupt Edge Detection

Figure 24-4 shows an RTL diagram of the synchronization and edge detection of interrupt sources on signals from the pin.



EIRQ clock is AHB clock gated with HLTREQ[EIRQHR].

Figure 24-4. Synchronization and Edge Detect Interrupt Generation

Figure 24-5 shows a timing diagram for an interrupt generated on the rising edge of an input on port a where the de-bounce logic is disabled. It also shows the clearing of such an interrupt by a write to the interrupt clear register. Note that signals A, B, C, and D in **Figure 24-5** refer to the points labelled A, B, C, and D in **Figure 24-4**, that is, points within the synchronizer and edge detection circuitry.

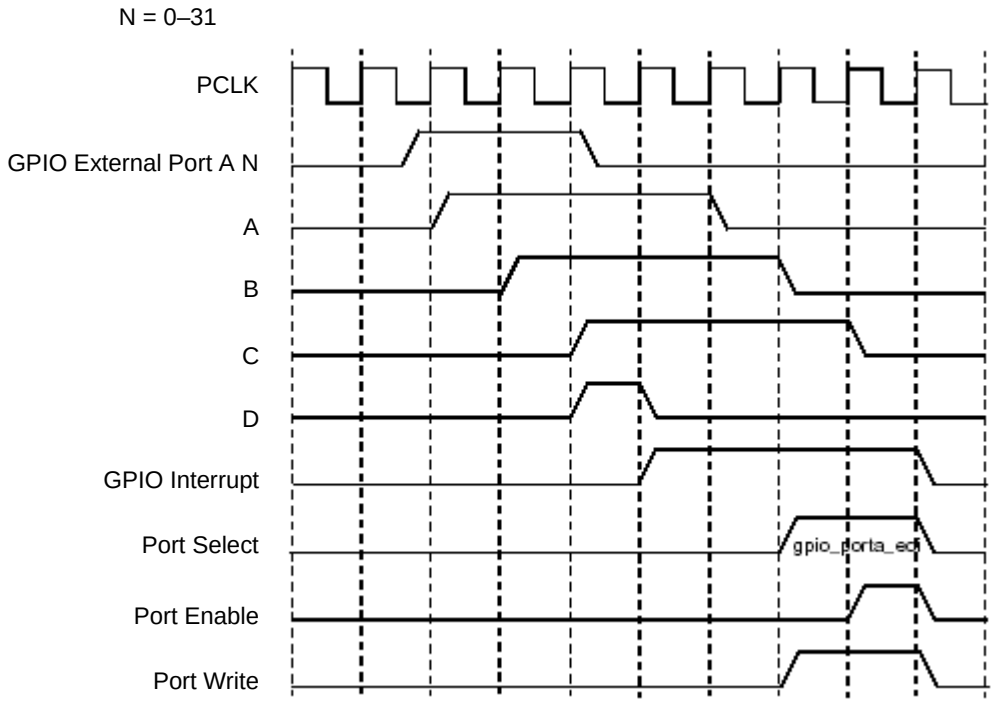


Figure 24-5. Interrupt Edge Detection and Interrupt Clear Timing

If the interrupt service routine is writing to the interrupt clear register to clear an interrupt on the same clock cycle as a new interrupt is detected, the write to the interrupt clear register clears only the first interrupt, and the second interrupt is not lost. **Figure 24-6** shows such a case. In this timing diagram, meta_out is the output of the second meta-stability register and edge_detect_out is the output of the edge detect logic. The second edge detection occurs on the same cycle as the write to the interrupt clear register. In this example, the write to the interrupt clear register does not clear the second interrupt, and the internal GPIO interrupt signal is not deasserted.

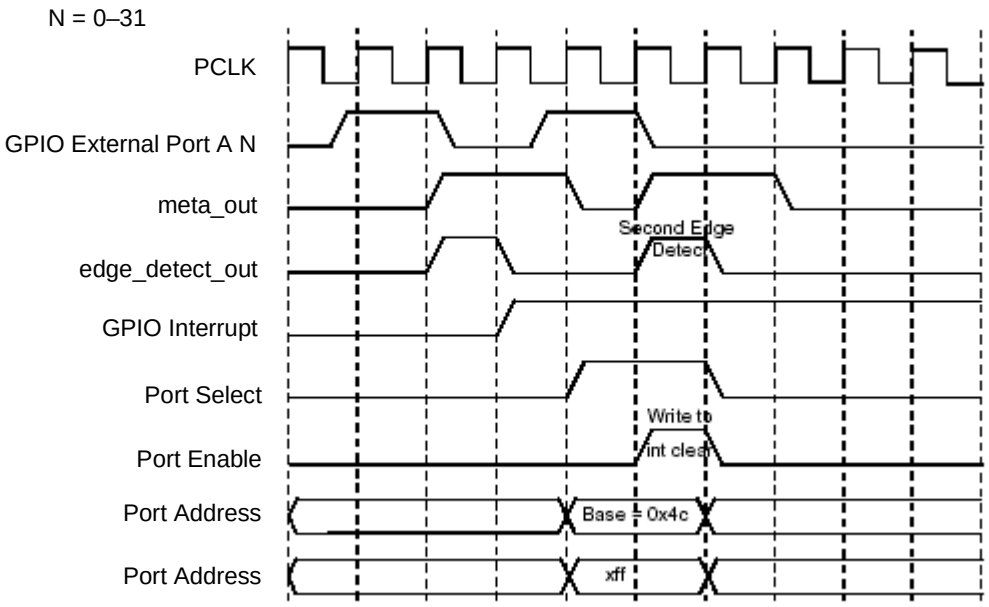


Figure 24-6. Write to Interrupt Clear Register, Coincident with Detection of New Interrupt

24.4.2.2 Level-Sensitive Interrupts

Figure 24-7 shows the generation of level-sensitive interrupts. As the figure shows, the LSSYN bit in the Port A Synchronous Level-Sensitive Register (GPAISLS), which is discussed on **page 24-22**, specifies whether the interrupt is synchronized with the EIRQ clock or is entirely combinational. LSSYN is a memory-mapped bit that inserts two meta-stability registers clocked off the EIRQ clock to synchronize the level-sensitive interrupts with the EIRQ clock. When the GPAISLS[LSSYN] bit is not set, there is no guarantee that the interrupt lines are synchronous to the EIRQ clock. The processor status register may need to be programmed to indicate asynchronous interrupts. When the GPAISLS[LSSYN] bit is set, the EIRQ clock must be present to pass the interrupt to the interrupt controller. The internal GPIO interrupt clock enable output signal is asserted when level-sensitive interrupts synchronized with EIRQ clock are active.

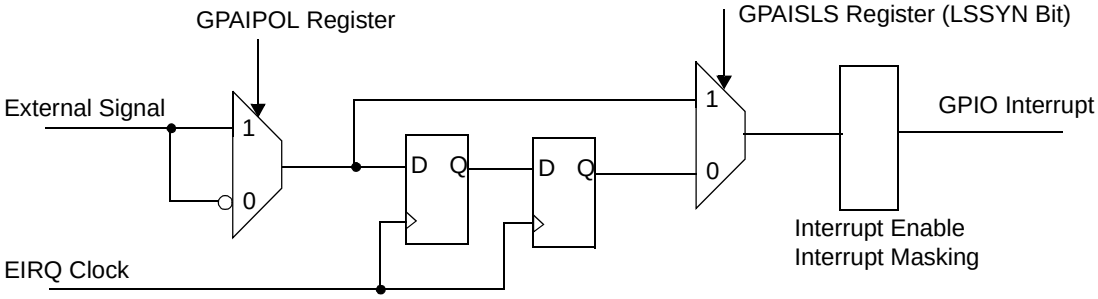


Figure 24-7. Level-Sensitive Interrupt Diagram

The input signal is inverted for active low level-sensitive interrupts. The same detection logic is used here as for active high level-sensitive interrupts. **Figure 24-8** shows the generation of an active low level-sensitive interrupt.

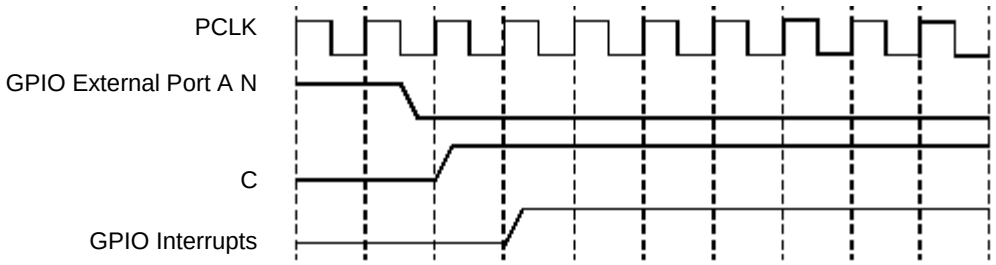


Figure 24-8. Active Low Level-Sensitive Interrupt Generation Timing

When level-sensitive operation is desired, the device must always be programmed for synchronous level-sensitive operation to ensure that external signals are synchronized before they enter the interrupt controller.

24.5 GPIO Programming Model

This section describes the GPIO registers. When you are programming the GPIO registers for interrupt capability, you must configure the edge-sensitive or level-sensitive interrupts and interrupt polarity before you enable the interrupts on port A to prevent spurious glitches on the lines to the interrupt controller. Writing to the interrupt clear register clears an edge-detected interrupt and has no effect on a level-sensitive interrupt. When the port width is less than the APB bus width of 32, for example port A, the undefined bits are read as zero.

The value of the base address for the GPIO register file, GPIO_BASE, is listed in **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4. The GPIO registers are listed as follows, along with the number of the page on which each register is discussed:

- Port *x* Control Register (GP*x*CTL), **page 24-16**.
- Port *x* Data Direction Register (GP*x*DDR), **page 24-17**.
- Port *x* Data Register (GP*x*DR), **page 24-18**.
- Port *x* External Port Register (GPxEXPRT), **page 24-19**.
- Port A Interrupt Enable Register (GPAIEN), **page 24-19**.
- Port A Interrupt Mask Register (GPAIMSK), **page 24-20**.
- Port A Interrupt Detection Type Register (GPAITYP), **page 24-21**.
- Port A Interrupt Polarity Register (GPAIPOL), **page 24-21.**
- Port A Interrupt Synchronous Level-Sensitive Register (GPAISLS), **page 24-22**.
- Port A Interrupt Clear Register (GPAICLR), **page 24-23**.
- Port A Interrupt Status Register (GPAISR), **page 24-23**.
- Port A Interrupt Raw Status Register (GPAIRSR), **page 24-24**.

GPxCTL Port *x* Control Register

(Port A) GPIO_BASE + 0x08
(Port B) GPIO_BASE + 0x14
(Port C) GPIO_BASE + 0x20
(Port D) GPIO_BASE + 0x2C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]	CTL[i]
TYPE	R/W															
RESET																

GPxCTL specifies whether a signal pin is configured as a GPIO pin under software control or as a peripheral-dedicated pin under hardware control.

Table 24-7. GPxCTL Bit Descriptions

Name	Reset	Description	Settings
CTL[i] 31–0	0	**Control Bit i** Each bit configures a port signal pin for either GPIO or peripheral functionality.	0 Signal pin is GPIO. 1 Signal pin is peripheral-dedicated.

Notes:
1. The number of bits in the GPIO ports, A, B, C, and D, differs from port to port, as shown in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.
2. Certain pins may also depend on DEVCFG[PAS] or DEVCFG[PDS] bit settings. See **Section 24.3.2**, *GPIO Port Assignments*, on page 24-4.

GPxDDR Port *x* Data Direction Register (Port A) GPIO_BASE + 0x04
(Port B) GPIO_BASE + 0x10
(Port C) GPIO_BASE + 0x1C
(Port D) GPIO_BASE + 0x28

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]	DD[i]
TYPE	R/W															
RESET																

GP*x*DDR is for reading or writing values to a GPIO signal pin when software configures the bit.

Table 24-8. GP*x*DDR Bit Descriptions

Name	Reset	Description	Settings
DD[i] 31–0	0	**Data Direction Bit i** Each bit corresponds to a GPIO pin on the device and specifies the direction of the corresponding pin when software programs the bit via the corresponding bit in the port's GPxCTL register. These bits are unaffected when the pins are configured as peripheral-dedicated rather than as GPIO.	0 GPIO pin is configured as an input pin. 1 GPIO pin is configured as an output pin
Note: The number of bits in the GPIO ports, A, B, C, and D, differs from port to port,, as shown in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (the uppermost bits of each register) are reserved. These bits read as 0 and must be written with 0 for future compatibility.			

GP*x*DR Port *x* Data Register (Port A) GPIO_BASE + 0x00
(Port B) GPIO_BASE + 0x0C
(Port C) GPIO_BASE + 0x18
(Port D) GPIO_BASE + 0x24

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]	D[i]
TYPE	R/W															
RESET																

GPxDR is for reading or writing values to a GPIO signal pin when software configures the bit.

Table 24-9. GPxDR Bit Descriptions

Name	Reset	Description	Settings
D[i] 31–0	0	**Data Bit i** Each bit corresponds to a GPIO signal pin on the device. Software reads or writes each bit via the corresponding bit in the port's GPxCTL register. These bits are unaffected when the pins are configured as peripheral-dedicated rather than as GPIO.	0 Write a 0 to the corresponding GPIO pin. 1 Write a 1 to the corresponding GPIO pin
Note: The number of bits in the GPIO ports, A, B, C, and D, differs from port to port,, as shown in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits read as 0 and must be written with 0 for future compatibility.			

GPxEXPRT Port x External Port Register

(Port A) GPIO_BASE + 0x50
(Port B) GPIO_BASE + 0x54
(Port C) GPIO_BASE + 0x58
(Port D) GPIO_BASE + 0x5C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]
TYPE	R															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]	PRT[i]
TYPE	R															
RESET																

GPxEXPRT allows the value on the port to be read, regardless of whether the signal pin is configured as a GPIO signal or a peripheral-dedicated signal.

Table 24-10. GPxEXPRT Bit Descriptions

Name	Reset	Description	Settings
PRT[i] 31–0	0	**Port Bit i** Each bit corresponds to a GPIO signal pin on the device. When the pin is configured as an input, the bit reads the value on the pin. When the pin is configured as an output, the bit reads the value of the corresponding bit in the GPxDR register. Notice that all of these bits are read-only.	0 A value of 0 is read. 1 A value of 1 is read.
Note: The number of bits in the GPIO ports, A, B, C, and D, differs from port to port,, as shown in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits read as 0 and must be written with 0 for future compatibility.			

GPAIEN Port A Interrupt Enable Register GPIO_BASE + 0x30

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]	EN[i]
TYPE	R/W															
RESET																

GPAIEN enables individual interrupt pins.

Table 24-11. GPAIEN Bit Descriptions

Name	Reset	Description	Settings
EN[i] 31–0	0	**Enable Bit i** Each bit corresponds to an interrupt signal pin on the device. Interrupts on the corresponding bit are disabled if the port is configured as peripheral-dedicated under hardware control or if the pin is configured as an output.	0 Interrupt source not enabled. 1 Interrupt source enabled.
Note: The number of bits in this register is shown in the Port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

GPAIMSK Port A Interrupt Mask Register GPIO_BASE + 0x34

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]	MSK[i]
TYPE	R/W															
RESET																

GPAIMSK masks individual interrupt pins.

Table 24-12. GPAIMSK Bit Descriptions

Name	Reset	Description	Settings
MSK[i] 31–0	0	**Mask Bit i** Each bit corresponds to an interrupt pin on the device.	0 Interrupt source is not masked. 1 Interrupt source is masked.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits read as 0 and must be written with 0 for future compatibility.			

GPAITYP Port A Interrupt Detection Type Register GPIO_BASE + 0x38

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]	TYP[i]
TYPE	R/W															
RESET																

GPAITYP specifies the type of the interrupt.

Table 24-13. GPAITYP Bit Descriptions

Name	Reset	Description	Settings
TYP[i] 31–0	0	**Type Bit i** Each bit corresponds to an interrupt pin on the device and configures it for either edge-triggered or level-sensitive operation.	0 Interrupt source is level-sensitive. 1 Interrupt source is edge-triggered.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

GPAIPOL Port A Interrupt Polarity Register GPIO_BASE + 0x3C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]
TYPE	R/W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]	POL[i]
TYPE	R/W															
RESET																

GPAIPOL configures the polarity of the interrupt pins.

Table 24-14. GPAIPOL Bit Descriptions

Name	Reset	Description	Settings
POL[i] 31–0	0	**Polarity Bit i** Each bit corresponds to an interrupt pin on the device and configures the polarity for both edge-triggered and level-sensitive operation.	0 Interrupt polarity is falling edge or active low. 1 Interrupt polarity is rising edge or active high.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

GPAISLS Port A Synchronous Level-Sensitive Register GPIO_BASE + 0x60

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—															
TYPE	R/W															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	—															LSSYN
TYPE	R/W															R
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

GPAISLS specifies whether pins programmed for level-sensitive operation are synchronized or not.

Table 24-15. GPAISLS Bit Descriptions

Name	Reset	Description	Settings
— 31–1	0	Reserved. Write to zero for future compatibility.	
LSSYN 0	1	**Level-Sensitive Synchronous** Corresponds to an interrupt pin on the device. LSSYN specifies whether level-sensitive interrupts are synchronized with the EIRQ clock. This bit always reads as a value of 1 for synchronized level-sensitive operation. It is read-only and cannot be modified by the user.	0 Reserved. 1 Level-sensitive interrupt sources are synchronized.

GPAICLR Port A Interrupt Clear Register GPIO_BASE + 0x4C

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]
TYPE	W															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]	CLR[i]
TYPE	W															
RESET																

GPAICLR clears pending interrupts.

Table 24-16. GPAICLR Bit Descriptions

Name	Reset	Description	Settings
CLR[i] 31–0	0	**Clear Bit i** Each bit corresponds to an interrupt pin on the device. When a 1 is written to one of these bits, the corresponding interrupt is cleared. Notice that all of these bits are write-only.	0 Pending Interrupt is not cleared. 1 Pending Interrupt is cleared.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

GPAISR Port A Interrupt Status Register GPIO_BASE + 0x40

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]
TYPE	R															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]	ST[i]
TYPE	R															
RESET																

GPAISR allows the value on the interrupt signal pins to be read after the masking operation is performed.

Table 24-17. GPAISR Bit Descriptions

Name	Reset	Description	Settings
ST[i] 31–0	0	**Status Bit i** Each bit corresponds to an interrupt pin on the device and reads the value of the processed interrupt signal after interrupt masking. Notice that all of these bits are read-only.	0 A masked value of 0 is read. 1 A masked value of 1 is read.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (the uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

GPAIRSR Port A Interrupt Raw Status Register GPIO_BASE + 0x44

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]
TYPE	R															
RESET																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]	RS[i]
TYPE	R															
RESET																

GPAIRSR allows the value on the interrupt signal pins to be read before the masking operation is performed.

Table 24-18. GPAIRSR Bit Descriptions

Name	Reset	Description	Settings
RS[i] 31–0	0	**Raw Status Bit i** Each bit corresponds to an interrupt pin on the device and reads the value of the processed interrupt signal before interrupt masking. Notice that all of these bits are read-only.	0 An unmasked value of 0. 1 An unmasked value of 1.
Note: The number of bits in this register is shown in the port A description in **Section 24.3.2**, *GPIO Port Assignments,* on page 24-4. Unused bits (uppermost bits of each register) are reserved. These bits are read as 0 and must be written with 0 for future compatibility.			

System Usage and Tuning/ Programming Reference A

The chapters of this manual discuss the MSC711x modules and describe the configuration options for set up and performance tuning. This appendix discusses the best way to configure these modules within the context of the entire system. It also covers trade-offs in tuning different parts of the system. The last section provided programming sheets to assist you in configuring key MSC711x registers. It is highly recommended that you read this appendix carefully so that you know how to get the best performance from your MSC711x device.

A.1 Best Use of the System

Many MSC711x modules can be tuned with different arbitration schemes and different priorities. Time-out monitors also detect incorrect system operation. This section discusses the required system settings for correct operation and the recommended settings for best operation of the modules within the system.

A.1.1 Critical Settings

The following task checklist helps to ensure best performance through the system:

- Correct placement of program and data as described in **Section A.1.2**, *M1 and M2 Memories*, on page A-2. Of special importance is the correct allocation of memory to reduce memory contention.
- Enabling the ICache and programming the burst parameters as described in **Section A.1.4**, *Instruction Fetch Unit*, on page A-3
- Set-up as described in **Section A.1.10**, *DDR Memory Controller Interface*, on page A-17.
- Correct settings for the ASEMI slave port as shown in **Table A-2**, *Arbitration Settings at Each Slave Port in the Crossbar Switch*, on page A-10. The ASM2 slave port parameters are also important.
- Priority elevation for the crossbar switch as described in **Section A.1.7.5**, *High-Priority Enable Bits*, on page A-14.
- Use of Page mode instead of Auto Precharge mode, as described in **Section A.1.10**, *DDR Memory Controller Interface*, on page A-17.
- Enabling the ECI write buffer for all writes to DDR and M2 memory as described in **Section A.1.5**, *Write Buffer and Write Buffer Data Areas*, on page A-3.

A.1.2 M1 and M2 Memories

Recommendations for placement of program code and data reflect the different ways the M1 and M2 memories are used to best advantage:

- Use M1 memory for data accesses. Up to two 64-bit data accesses can occur in one core clock.
- Use M1 memory for all important data that needs efficient access, such as DSP algorithms.
- Use M1 or M2 memory to store program code:
  - Program code in M1 memory does not use the ICache.
  - Program code in M2 memory uses the ICache very efficiently with little penalty in overall performance.
- Use of M2 memory for data accesses is not as efficient as use of M1 memory. One 64-bit data segment requires 7–8 core clocks. This can be reduced to ~1 core clock if the address range for the M2 memory is configured for writes through the write buffer or immediate writes with no core freeze (see **Section 4.7.1**, *Write Buffer Data Areas*, on page 4-35 and **Section A.1.5**, *Write Buffer and Write Buffer Data Areas*, on page A-3).

The following practices are recommended for most efficient system operation:

- Carefully read **Section 4.3.1**, *Memory Contention*, on page 4-6 to understand how best to organize data in M1 memory. DMA traffic to M1 memory must operate in a different half memory group than the half memory group heavily accessed by the SC1400 core when DSP algorithms are executing.
- To speed up writes to the SC1400 core, program one of the WBDAR regions covering the address space of M2 memory for normal write buffer operation.
- Place program code into M2 memory for highest efficiency.
- Any leftover space in M1 memory can be used for additional program code.
- Any leftover space in M2 memory can be used for additional data.
- Use the last 64 bytes of M2 memory for data only. Because of system pipelining, SC1400 core code fetches from this area can result in an attempt to access the reserved areas beyond the end of the M2 memory. Such fetches may cause the system to stop operating. To prevent this situation, do not store instruction code in the last 64 bytes of M2 memory. For example, for MSC711x devices with 192 KB M2 memory, the memory range 0x0102FFC0 should be reserved for data only.

Note: Initialization code is typically placed into DDR memory.

A.1.3 M1 Memory: Two Different Address Ranges

There are two different address ranges to access M1 memory:

- For access by the SC1400 core over its P, XA, or XB buses: 0x00000000–0x0003FFFF.

- For access by the DMA controller or Ethernet MAC as master ports on the crossbar switch: 0x01800000–0x0183FFFF. The DMA controller performs accesses over the AMDMA bus and the Ethernet MAC performs accesses over the AMENT bus.

For example, a 32-bit SC1400 core access to address 0x0000001C occurs at the same location in M1 memory as a 32-bit access from the DMA or Ethernet MAC at address 0x0180001C.

A.1.4 Instruction Fetch Unit

Usually M2 memory is used to store program code, which is efficiently burst into the ICache. The following burst parameters, programmable in the Instruction Region Configuration Registers (IRCR[0–3]) (**page 4-48**), are recommended for the address space that accesses M2:

- Primary set size of 4
- Burst size of 4
- Prefetch enabled.

DDR memory is also useful for storing program code. The following settings are useful when the address space that accesses DDR is configured:

- Primary set size of 4
- Burst size of 4
- Prefetch enabled

To attain the highest performance, you may need to adjust the parameters for accessing DDR. For example, in systems with high DMA traffic to DDR memory, the prefetch can be disabled, and so on.

A.1.5 Write Buffer and Write Buffer Data Areas

The write buffer data areas are programmed in the WB Data Area Registers (WBDAR[0–3] (see **page 4-42**). Recommended settings are as follows:

- Assign no write buffer data areas to the address space of M1 memory because the SC1400 core directly accesses M1 memory and does not use the write buffer.
- Assign one write buffer data area to the address space of both the IPBus peripherals and APB peripherals (see **Table 5-1**, *Summary — Base Addresses for MSC711x Register Files*, on page 5-4). Program the IMM field for this address range as *write immediate* (WBDARx[IMM] = 01) (see **Table 4-11**, *WBDARx Bit Descriptions*, on page 4-42).
- Assign one write buffer data area to the portion of the memory map for performing data accesses on the ASTH bus to MSC711x peripherals (HDI16 and TDMs). The IMM field for this address range is typically programmed as *regular write through write buffer* (WBDARx[IMM] = 00).

- Assign one or more write buffer data areas to the address space of M2 memory. The IMM field for this address range is typically programmed as *regular write through write buffer* (WBDARx[IMM] = 00).
- Assign one or more write buffer data areas to the address space of the DDR memory. The IMM field for this address range is typically programmed as *regular write through write buffer* (WBDARx[IMM] = 00).

Note: One data area can be used to cover both the M1 and M2 memories, if desired. One data area can be used to cover both the peripherals and external memory, if desired.

Following is an example write buffer data area programmed for the peripheral space.

1. Select a WBDAR area for the portion of the memory map containing MSC711x peripherals, 0x0400 0000–0x07FF FFFF.
2. Size = 64 MB → Line 18 in **Table 4-8**, *Programming the Write Buffer Data Area Base and Size,* on page 4-36
3. Upper 24-bits of base = 0x040000
4. Value placed into WBDARx[BASE] = 0x040000 || 020000 = 0x0600000
5. To program as write immediate: WBDARx[IMM] = 01.

A.1.6 DMA Controller

This section discusses the proper use of DMA preemption with fixed-priority arbitration, how to prevent master port time-outs, and overall recommended settings. For more information, see **Section 8.3**, *Data Transfer Overview,* on page 8-5 to get a better understanding of the time required for DMA bursting on the device between different sources and destinations.

A.1.6.1 Preemption with Fixed-Priority Arbitration

The DMA controller allows a DMA channel to be preempted if the DMA is configured for fixed-priority arbitration. For example, preemption can be enabled for long, low-priority DMA transfers. Because nested preemption within the DMA is not supported, there can be long latencies before a high-priority channel is serviced, as demonstrated in the following scenario:

1. A DMA channel with a large byte count and low priority is in progress.
2. A second DMA channel with a slightly higher priority and a large byte count preempts the first channel.
3. A peripheral such as a TDM with a priority higher than the first and the second channels issues a third DMA request, but it must wait until the preempting DMA channel (the second channel) is processed before it can preempt.

To avoid such a situation, use one of the following techniques to ensure that *only one* lower-priority channel allowing preemption can be active at a time:

- Link all low-priority channels together so that when one preemptable channel completes, processing continues with the next channel in the chain.
- Normally long, low-priority DMA transfers are performed in a single minor loop with a large byte count. Break this loop into several minor loops using the DMA major-minor looping structure. The DMA can then rearbitrate as each minor loop is processed, so highest priority channels must wait less time if preempted. For example, a 2048 byte DMA transfer can be broken into sixteen 128-byte transfers. This solution is not as effective as the preceding solution.

You can link all preemptable DMA channels or carefully schedule them so that they never preempt each other and there is no issue with preemption.

A.1.6.2 Preventing Master Port Time-Outs

For DMA data transfers in which both the source and destination address access the same memory (that is, M1→ M1, M2 → M2, or DDR→ DDR), use one of the following strategies to prevent time-outs in the bus error detection units on the other AHB master buses:

- Program the DMA controller for the corresponding crossbar slave port SGPCR as the lowest-priority master, and do not elevate the channel performing this memory-to-memory transfer (TCDx-7[BWC] != 01. This is a recommended solution.
- Set the bandwidth control for this channel for DMA controller stalls of either 4 or 8 cycles in the TCDx-7[BWC] field (see **Table 8-32**, *TCDx-7 Bit Descriptions*, on page 8-49). This is a recommended solution.
- Limit the maximum byte count for the minor loop. This solution is not as effective as the previous two solutions.

The maximum byte count permitted depends on how the DMA channel bandwidth control is programmed. If maximum byte count is an issue, use the bandwidth control options that introduce DMA stalls between each read-write sequence. If this option is not used, certain maximum byte count restrictions must be in effect. If the bandwidth control in TCDx-7[BWC] is set to *dynamic priority elevation* (TCDx-7[BWC] = 01), the DMA priority through the crossbar switch elevates, possibly blocking out other masters (AMIC, AMEC, or AMDMA). This is significant because other important AHB masters may be delayed. The minor loop byte count in DMA channels with this elevation must be further restricted to prevent bus time-outs. This limit ensures that other masters are not locked out from a particular slave port in the crossbar, resulting in a bus time-out. The maximum allowed byte count is determined in **Table A-1**.

Table A-1. DMA Minor Loop Byte Count Restrictions

Source	Destination	Source Transfer Size: TCD1[SSIZE]	Destination Transfer Size: TCDx-1[DSIZE]	TCDx-7[BWC]	Maximum Minor Loop Byte Count	Comments
M1	M1	8-bits	64-bits	00	2,370	Not recommended[3]
		16-bits	64-bits	00	4,250	Not recommended[3]
		32-bits	64-bits	00	7,100	Not recommended[3]
		64-bits	8-bits	00	2,370	Not recommended[3]
		64-bits	16-bits	00	4,250	Not recommended[3]
		64-bits	32-bits	00	7,100	Not recommended[3]
		64-bits	64-bits	00	10,600	See Note 1.
		64-bits	64-bits	01	900	Priority elevated channel.
		Any	Any	10 or 11	No Restriction	See Note 2.
	M2, DDR, or Peripheral	Any	Any	Any	No Restriction	
M2	M2	8-bits	64-bits	00	2,900	Not recommended[3]
		16-bits	64-bits	00	5,240	Not recommended[3]
		32-bits	64-bits	00	8,700	Not recommended[3]
		64-bits	8-bits	00	2,900	Not recommended[3]
		64-bits	16-bits	00	5,240	Not recommended[3]
		64-bits	32-bits	00	8,700	Not recommended[3]
		64-bits	64-bits	00	13,100	See Note 1.
		64-bits	64-bits	01	1,300	Priority elevated channel.
		Any	Any	10 or 11	No Restriction	See Note 2.
	M1, DDR, or Peripheral	Any	Any	Any	No Restriction	

Table A-1. DMA Minor Loop Byte Count Restrictions (Continued)

Source	Destination	Source Transfer Size: TCD1[SSIZE]	Destination Transfer Size: TCDx-1[DSIZE]	TCDx-7[BWC]	Maximum Minor Loop Byte Count	Comments
DDR (16-pin)	DDR (16-pin)	8-bits	64-bits	00	—	Not recommended[3]
		16-bits	64-bits	00	—	Not recommended[3]
		32-bits	64-bits	00	—	Not recommended[3]
		64-bits	8-bits	00	—	Not recommended[3]
		64-bits	16-bits	00	—	Not recommended[3]
		64-bits	32-bits	00	—	Not recommended[3]
		64-bits	64-bits	00	512	See Note 1.
		64-bits	64-bits	01	160	Priority elevated DMA channel.
		Any	Any	10 or 11	No Restriction	See Note 2.
	M1, M2, or Peripheral	Any	Any	Any	No Restriction	
DDR (32-pin)	DDR (32-pin)	8-bits	64-bits	00	—	Not recommended[3]
		16-bits	64-bits	00	—	Not recommended[3]
		32-bits	64-bits	00	—	Not recommended[3]
		64-bits	8-bits	00	—	Not recommended[3]
		64-bits	16-bits	00	—	Not recommended[3]
		64-bits	32-bits	00	—	Not recommended[3]
		64-bits	64-bits	00	1024	See Note 1.
		64-bits	64-bits	01	320	Priority elevated channel.
		Any	Any	10 or 11	No Restriction	See Note 2.
	M1, M2, or Peripheral	Any	Any	Any	No Restriction	
Peripheral	Any	Any	Any	Any	No Restriction	

Notes:
1. A transfer size of 64-bits is used when the size fields are programmed with a value of 0b101 to specify 32-byte transfers.
2. DMA bandwidth control ensures that the slave port buses used by the DMA are periodically freed after each read-write sequence. Better performance is attained when the DMA channel is also programmed for 32-byte bursts to both its source and destination.
3. During large memory to memory transfers, use the largest possible transfer size, which occurs when the source and destination are both configured for 32-byte bursts.

A.1.6.3 Recommended DMA Settings

The following DMA settings are recommended for most efficient system operation:

- For DMA data transfers greater than or equal to 128 bytes, program the TCDx-1[SSIZE] and TCDx-1[DSIZE] fields with a value of 101, which corresponds to a 32-byte AHB burst.
- To ensure compatibility with future devices, the largest value used in the 32-bit TCDx-2[NBYTES] field must be 0x1FFFFFFF.
- In applications with high DMA bandwidth requirements, all channels with the source and destination going to *different slave buses* on the crossbar switch should not use the bandwidth control. That is, do not configure TCDx-7[BWC] to *DMA engine stalls for 4 cycles* or *DMA engine stalls for 8 cycles*. Examples of this case are channels between DDR and M1 memory, DDR and M2 memory, M1 and M2 memory, and so on.
- In applications with high DMA bandwidth requirements, all channels with the source and destination going to the *same slave bus* on the crossbar switch should use bandwidth control. That is, configure TCDx-7[BWC] to *DMA engine stalls for 4 cycles* or *DMA engine stalls for 8 cycles*. Examples are channels between DDR and DDR memory, M2 and M2 memory, and M1 and M1 memory.
- For DDR-to-DDR transfers, ensure that 32-byte bursting is used in conjunction with one of the bandwidth control options (4 or 8 cycle stalls after each read-write sequence) to remove the restrictions on maximum byte count (see **Table A-1** on page A-6).

A.1.7 Crossbar Switch

For proper and most efficient use of the crossbar switch, each slave port must be configured correctly, as described in this section

A.1.7.1 Priority Elevation by the Masters

Different masters can elevate the priority of their accesses to important data, as described in. **Section 6.2.2**, *Priority Assignment*, on page 6-7. Hardware automatically handles priority elevation for the instruction fetch unit (IFU) and the extended core interface (ECI).

DMA channels should be programmed as follows:

- Long low-priority transfers with high byte counts have no priority elevation.
- DMA channels assigned to service the TDMs or HDI16 have priority elevation.

The Ethernet does not elevate any of its accesses, so Ethernet accesses normally have a lower fixed priority than priority-elevated accesses from the IFU, the ECI, and the DMA controller. However, you can configure the DEVCFG[ENTP] bit to define all Ethernet accesses as always elevated or always non-elevated. This bit is cleared in normal operation.

A.1.7.2 Crossbar Slave Port Capabilities

Figure A-1 shows a detailed view of a slave port. The correct set-up is described in the following subsections.

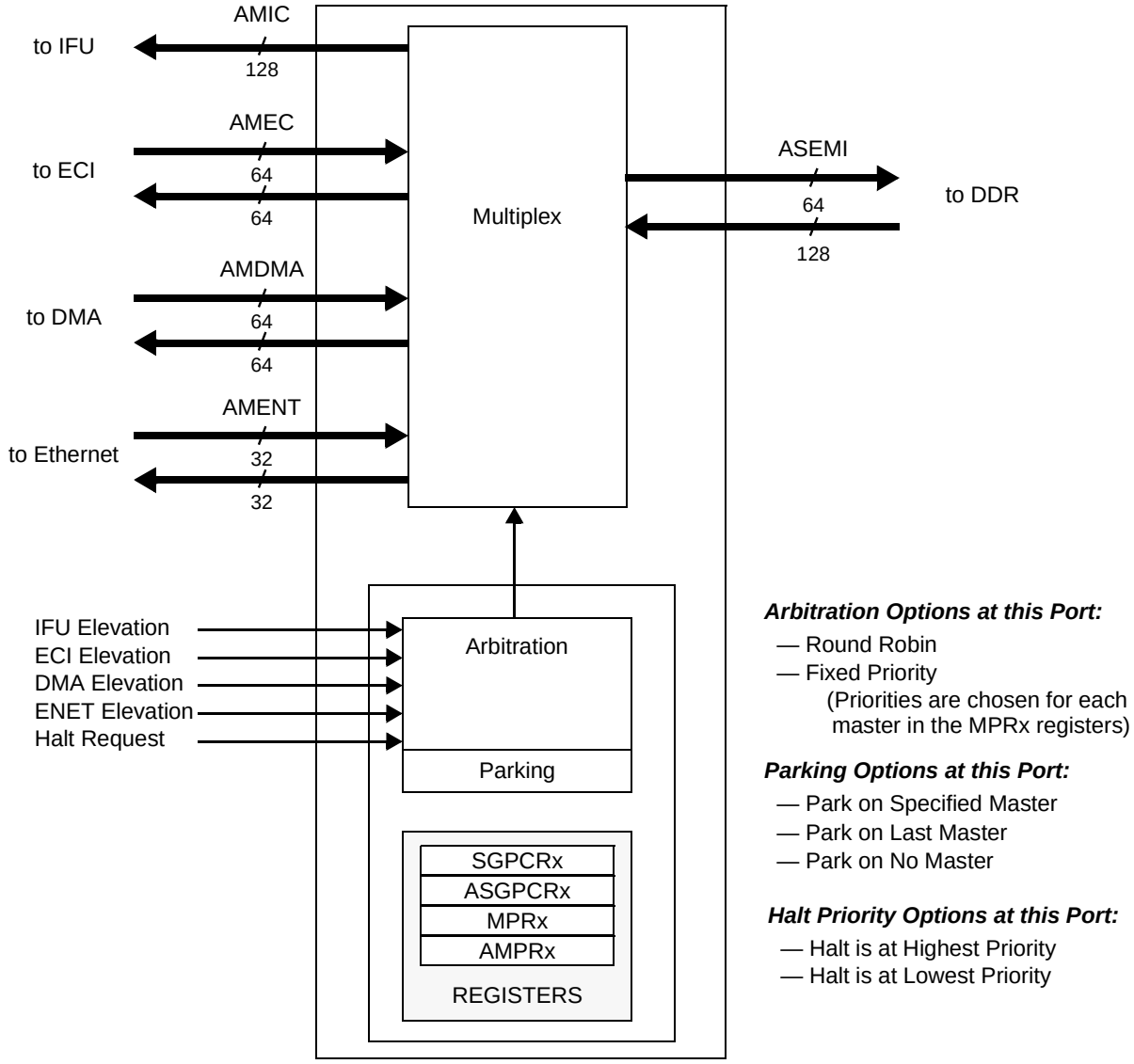


Figure A-1. View of One Crossbar Slave Port (ASEMI)

A.1.7.3 Arbitration at Crossbar Slave Ports

The arbitration scheme at each slave port in the crossbar switch are programmed in each slave port through its registers, as follows:

- SGPCRx[ARB] bit to set arbitration mode (see **page 6-22**).
- MPRx register when configured for fixed-priority operation (see **page 6-20**).

Table A-2 presents the different arbitration settings. Correctly setting the arbitration for the ASEMI port is important for attaining desired performance. Incorrect settings can reduce device performance.

Table A-2. Arbitration Settings at Each Slave Port in the Crossbar Switch

Scenario	Master Port			Comments
	Arbitration Type	Programmed Master Port Priorities	Resulting Priorities	
For ASM1 Ports — Does Not Support Program (IFU) Accesses				
ASM1-1	Fixed Priority	**1.** AMDMA **2.** AMENT	**1.** AMDMA with elevation **2.** AMDMA typical usage **3.** AMENT	All DMA transfers have a higher priority than Ethernet MAC DMA transfers.
ASM1-2	Fixed Priority	**1.** AMENT **2.** AMDMA	**1.** AMDMA with elevation **2.** AMENT **3.** AMDMA typical usage	Only DMA transfers with priority elevation have a higher priority than Ethernet MAC DMA transfers.
ASM1-3	Round Robin	—	**1.** Master accesses with elevated priorities **2.** Master accesses without elevated priorities	Masters with elevated priorities: • DMA with elevation (AMDMA) Masters without elevated priorities: • DMA (AMDMA) typical usage • Ethernet MAC (AMENT)
For ASM2 and ASEMI Ports				
ASEMI-1	Fixed Priority	**1.** AMIC **2.** AMEC **3.** AMENT **4.** AMDMA	**1.** AMIC with elevation **2.** AMEC with elevation **3.** AMENT with elevation **4.** AMDMA with elevation **5.** AMIC prefetch accesses **6.** AMEC WB Accesses **7.** AMENT no elevation **8.** AMDMA typical usage	Recommended setting: Favors IFU accesses for servicing cache misses that stall the SC1400 core. Ethernet MAC must be selected as either always or never elevating access priority. Long DMA bursts must not elevate priority.
ASEMI-2	Fixed Priority	**1.** AMEC **2.** AMIC **3.** AMENT **4.** AMDMA	**1.** AMEC with elevation **2.** AMIC with elevation **3.** AMENT with elevation **4.** AMDMA with elevation **5.** AMEC WB accesses **6.** AMIC prefetch accesses **7.** AMENT no elevation **8.** AMDMA typical usage	Alternate setting: Favors SC1400 core data accesses that can otherwise stall the core. Ethernet MAC must be selected as either always or never elevating access priority. Long DMA bursts must not elevate priority.

Table A-2. Arbitration Settings at Each Slave Port in the Crossbar Switch (Continued)

Scenario	Master Port			Comments
	Arbitration Type	Programmed Master Port Priorities	Resulting Priorities	
ASEMI-3	Fixed Priority	**1.** AMENT **2.** AMIC **3.** AMEC **4.** AMDMA	**1.** AMENT with elevation **2.** AMIC with elevation **3.** AMEC with elevation **4.** AMDMA with elevation **5.** AMENT no elevation **6.** AMIC prefetch accesses **7.** AMEC: WB accesses **8.** AMDMA typical usage	Alternate setting: Ethernet MAC has highest priority when elevated. Favors IFU accesses for servicing cache misses that stall the SC1400 core. Ethernet MAC must be selected as either always or never elevating access priority. Long DMA bursts must not elevate priority.
ASEMI-4	Round Robin	—	**1.** Master accesses with elevated priorities: **2.** Master accesses without elevated priorities.	Masters with elevated priorities: • ECI with elevation (AMEC) • DMA with elevation (AMDMA) • IFU (AMIC): primary set • Ethernet MAC (AMENT) when DEVCFG[ENTP] = 1 Masters without elevated priorities: • ECI (AMEC) • DMA (AMDMA): Typical usage • IFU (AMIC): Prefetch • Ethernet MAC (AMENT) when DEVCFG[ENTP] = 0
For ASTH Port — Does Not Support Program (IFU) or Ethernet MAC Accesses				
ASTH-1	Fixed Priority	**1.** AMDMA **2.** AMEC	**1.** AMDMA with elevation **2.** AMECI with elevation **3.** AMDMA typical usage **4.** AMEC	Recommended configuration when HDI16 and TDM registers are accessed via the DMA controller.
ASTH-2	Fixed Priority	**1.** AMEC **2.** AMDMA	**1.** AMEC with elevation **2.** AMDMA with elevation **3.** AMEC **4.** AMDMA typical usage	Recommended configuration when the HDI16 and TDM registers are accessed via the SC1400 core.
ASTH-3	Round Robin	—	**1.** Master accesses with elevated priorities: **2.** Master accesses without elevated priorities.	Masters with elevated priorities: • ECI with elevation (AMEC) • DMA with elevation (AMDMA) Masters without elevated priorities: • ECI (AMEC) • DMA (AMDMA): Typical usage
For ASSB, and ASAPB Ports — Do Not Support Program or Ethernet MAC Accesses				
ASAPB-1	Fixed Priority	**1.** AMEC **2.** AMDMA	**1.** AMEC: with elevation **2.** AMDMA: with elevation **3.** AMEC **4.** AMDMA (typical usage)	Recommended configuration because DMA transfers do not typically occur on these AHB slave buses.

Table A-2. Arbitration Settings at Each Slave Port in the Crossbar Switch (Continued)

Scenario	Master Port			Comments
	Arbitration Type	Programmed Master Port Priorities	Resulting Priorities	
ASAPB-2	Fixed Priority	**1.** AMDMA **2.** AMEC	**1.** AMDMA: with elevation **2.** AMECI: with elevation **3.** AMDMA (typical usage) **4.** AMEC	Alternate configuration, not typically used.
ASAPB-3	Round Robin	—	**1.** Master accesses with elevated priorities: **2.** Master accesses without elevated priorities.	Masters with elevated priorities: • ECI with elevation (AMEC) • DMA with elevation (AMDMA) Masters without elevated priorities: • ECI (AMEC) • DMA (AMDMA): Typical usage DMA transfers do not typically occur on these buses.

In the settings listed in **Table A-2**, it is critical that the slave ports be set up correctly for the ASEMI slave port because these settings directly affect DDR performance. It is also important to set up the M2 memory correctly. Most users choose the fixed-priority settings instead of round robin.

A.1.7.4 Slave Port Parking

Parking is programmed in each slave port via the following bits:

- SGPCRn[PCTL] to select the type of parking.
- SGPCRn[PARK] to determine which master is to be parked on a slave port when SGPCRn[PCTL] is programmed to park on a particular master port.

See **Table 6-6**, *SGPCRx and ASGPCRx Bit Descriptions*, on page 6-23 for details on the SGPCR bits. The following settings are recommended:

- For applications with a significant amount of code in DDR memory, the ASEMI slave port should park on the AMIC master, as selected in the SGPCRx[PCTL] and SGPCRx[PARK] fields. Alternatively, the slave port can park on the last master.
- For applications with very high DMA bandwidth through the DDR port and/or to M1 memory, the ASEMI/ASM1 slave ports should park on the AMDMA master, as selected in the SGPCRx[PCTL] and SGPCRx[PARK] fields. For applications with large DMA transfers between DDR memory and M1 memory, this is recommended for the ASEMI and ASM1 ports, especially if the DDR port is configured for 16-pin operation. Alternatively, these slave ports can park on the last master.
- For applications using M2 memory primarily to hold program code that is loaded on ICache misses, the ASM2 slave port should park on the AMIC master, as selected in the

SGPCRx[PCTL] and SGPCRx[PARK] fields. Alternatively, these slave ports can park on the last master.

- If the DMA controller services the TDMs and HDI16, the ASTH slave port should park on the AMDMA master. If the SC1400 core services the TDMs and HDI16, the ASTH slave port should park on the AMEC master, as selected in the SGPCRx[PCTL] and SGPCRx[PARK] fields.

Table A-3 summarizes the recommended park settings for the crossbar switch slave ports.

Table A-3. Park Settings for Each Crossbar Switch Slave Port

Slave Port	Parking Technique	Master to Park On	Comments
ASM1	Park on specified master	AMDMA	For systems with high DMA bandwidth between M1 memory and DDR or between M1 and M2 memories. Use this approach if DDR usage is high.
	Park on last master	—	Alternate approach, not as optimal for the DMA or DDR controller.
	Park on no master	—	Reduces power consumption. Suitable only for systems that do not require high DMA bandwidth to M1 memory.
ASM2	Park on specified master	AMIC	For systems using M2 memory primarily to hold program code for the ICache. If M2 memory is used for code overlays, this setting is also recommended.
	Park on specified master	AMDMA	For systems using M2 memory primarily for data loaded through the DMA controller.
	Park on last master	—	Alternate approach, not as optimal for the IFU.
	Park on no master	—	Reduces power consumption. Suitable only for systems with low MIPs requirements that can afford slower cache bursting.
ASEMI	Park on specified master	AMIC	For systems with large amounts of code in DDR.
	Park on specified master	AMDMA	For systems with very high DMA bandwidth requirements between M1 and DDR memories or between M2 memory and DDR.
	Park on last master	—	Alternate approach, more appropriate if there is high usage from DMA, Instruction Cache, and/or Ethernet MAC.
	Park on no master	—	Reduces power consumption. Only suitable for systems which do not require high DMA bandwidth to DDR memory.
ASTH	Park on specified master	AMDMA	For when the DMA controller services the HDI16 and TDMs.
	Park on specified master	AMEC	For when the SC1400 core services the HDI16 and TDMs.
	Park on last master	—	Alternate approach, not as optimal for transfers.
	Park on no master	—	Only for masters with low bandwidth requirements on the DMA and low bandwidth requirements to the HDI16 and TDMs.

Table A-3. Park Settings for Each Crossbar Switch Slave Port (Continued)

Slave Port	Parking Technique	Master to Park On	Comments
ASSB	Park on specified master	AMEC	For servicing by the SC1400 core.
	Park on no master	—	Only for systems with low bandwidth requirements.
ASAPB	Park on specified master	AMEC	For servicing by the SC1400 core.
	Park on no master	—	Only for systems with low bandwidth requirements.

A.1.7.5 High-Priority Enable Bits

The high-priority enable bits are normally set to ensure that the crossbar switch works with the masters as desired. **Table A-4** describes how to set these crossbar switch bits correctly:

- SGPCRx[HPE7–4] must be cleared to 0 on all slave ports; they correspond to unimplemented master ports on the crossbar.
- SGPCRx[HPE3] corresponds to the Ethernet MAC and must be set to 1 on all slave ports.
- SGPCRx[HPE2] corresponds to the ICache IFU and must be cleared to 0 for the following slave ports that do not support program accesses:
  - ASM1
  - ASTH
  - ASSB
  - ASAPB
- SGPCRx[HPE2] corresponds to the IFU and must be set to 1 for the following slave ports that support program accesses:
  - ASM2
  - ASEMI

  This bit setting ensures that the IFU has a higher priority than the DMA controller when its priority is elevated for primary set accesses (see **Section 4.6.1**, *Cache Bursting Parameters*, on page 4-31). This is important because DMA bursts may occur for long intervals, possibly locking out accesses from the IFU. This does not affect prefetch accesses from the IFU when the IFU priority is not elevated.
- SGPCRn[HPE0] corresponds to the extended core interface (ECI) and should be cleared to 0 for the following slave ports that do not support ECI accesses:
  - ASM1
- SGPCRn[HPE0] corresponds to the ECI and should be set to 1 for the following slave ports:
  - ASM2
  - ASEMI

  This bit setting ensures that the ECI has higher priority than the DMA controller when its priority is elevated (see **Section 6.2.2**, *Priority Assignment*, on page 6-7). This is

important because DMA bursts may occur for long intervals, possibly locking out accesses from the ECI. This does not affect ECI accesses when the priority is not elevated.

Table A-4 summarizes the correct settings for the HPE bits.

Table A-4. Required Settings for the HPE bits for each Slave Port in the Crossbar

Crossbar Slave Port	Master Port				
	Unused Masters: HPE[7–4]	Ethernet MAC: HPE3	IFU: HPE2	DMA: HPE1	ECI: HPE0[2]
ASM1	0x0	1	0	As desired[1]	0
ASM2	0x0	1	1	As desired	1
ASEMI	0x0	1	1	As desired	1
ASTH	0x0	1	0	As desired	As desired
ASSB	0x0	1	0	As desired	As desired
ASAPB	0x0	1	0	As desired	As desired

Notes: **1.** If you are unsure how to set the entries labeled “as desired,” then set them to 1.
2. The HPE bits for the ECI master, HPE0, are typically set for the crossbar ASSB and ASAPB slave ports.

A.1.7.6 Alternate Priorities

There is an alternate priority register set for each crossbar slave port. Use the alternate priorities only for unusual situations on the device or for recovery from an application error. See **Section A.6.5**, *Configurable Priority Modification During a Chip Event,* on page A-24.

Alternate priorities for a slave port are set up in the following registers:

- AMPRx (see **page 6-20**)
- ASGPCRx (see **page 6-22**)

This capability is provided for advanced error recovery if an application needs it.

A.1.8 Programmable Bus Time-Out Monitors on Master Buses

Closely connected with the crossbar switch, there is a dedicated programmable time-out monitor on each master port of the crossbar switch (see **Section 7.1.2**, *Bus Time-Out and Error Detection (Master Buses),* on page 7-2). This section describes the correct usage of these monitors.

Note: It is highly recommended that the bus monitors are enabled and used in the system. They will greatly reduce the time spent debugging the system.

These time-out monitors can be independently programmed for accesses where priority is elevated and for normal accesses which are not elevated. For correct system operation, it is recommended that the bus error monitors are programmed with any of the values in **Table A-5**:

Table A-5. Recommended Settings for the Bus Error Detection

Master Port Priority	Crossbar Master Port	Recommended Bus Error Time-Out Values			
		256 AHB clocks	1024 AHB clocks	4096 AHB clocks	Disabled
Elevated	AMIC	Highly Recommended	Yes	Yes	Not Recommended
	AMEC	Highly Recommended	Yes	Yes	Not Recommended
	AMDMA	Highly Recommended	Yes	Yes	Not Recommended
	AMENT	Highly Recommended	Highly Recommended	Yes	Not Recommended
Normal	AMIC	—	Highly Recommended	Yes	Not Recommended
	AMEC	—	Highly Recommended	Yes	Not Recommended
	AMDMA	—	Highly Recommended	Yes	Not Recommended
	AMENT	—	Highly Recommended	Yes	Not Recommended

Note: When initially debugging a system, use the lowest recommended value to help reveal any incorrect system set-up.

A.1.9 Programmable Bus Time-Out Monitors on Slave Buses

A dedicated programmable time-out monitor resides on each slave port of the crossbar switch. For significant savings in the time spent debugging the system, these bus monitors should be enabled and used. Program the time-out monitors only with of the values in **Table A-6**.

Table A-6. Settings for Bus Time-Out Monitors

Crossbar Slave Port	Corresponding Time-Out Monitor Values				
	31 AHB clocks	127 AHB clocks	511 AHB clocks	2047 AHB clocks	Disabled
ASM1	—	—	Highly Recommended	Yes	Not Recommended
ASM2	Highly Recommended	Yes	Yes	Yes	Not Recommended
ASEMI	—	—	Highly Recommended	Yes	Not Recommended
ASTH	Highly Recommended	Yes	Yes	Yes	Not Recommended
ASSB	Highly Recommended	Yes	Yes	Yes	Not Recommended
ASAPB	Highly Recommended	Yes	Yes	Yes	Not Recommended

When debugging system problems, you can reduce some of the values listed in **Table A-6**. Although it is not recommended, you can reduce these values to constrain the size of DMA transfers to a very low value, using the DMA bandwidth control capability (TCDx-7[BWC]) or ensure that there are no DMA transfers to and from the same slave (DDR to DDR, M2 to M2, and so on). If these constraints are not possible, you should use only the settings listed in **Table A-6**. The minimum value specified for ASEMI time-out ensures proper operation during refresh. It is preferable to enable the time-out monitor and to use the smallest recommended value.

A.1.10 DDR Memory Controller Interface

The following settings are recommended for optimal system operation:

- Set ICache predictive reads to always enabled (MCIFCTRL[IPRE] = 01).
- Enable DMA predictive reads (MCIFCTRL[DPRE] = 1).
- Enable ECI predictive reads (MCIFCTRL[EPRE] = 1).

For more information, see **Section 10.3**, *Programming the MCIF*, on page 10-5.

A.1.11 DDR Memory Controller

Following are recommendations for best use of the DDR memory controller:

- Use Page mode instead of Auto Precharge mode:
  - Page mode is optimal for DMA bursting.
  - Auto Precharge mode is appropriate only when most DDR accesses are random accesses and not DMA bursts, as programmed in the SICFG[BSTOPRE] field (see **page 9-42**).
- If DDR is used in a power sensitive application, it is useful to read **Section 9.5.4**, *Low-Power Modes*, on page 9-15

A.1.12 Event Port

The event port can greatly assist in debugging difficult system-level problems. **Section 16.3**, *System-Level Debugging*, on page 16-4 shows how the event port can be integrated with the debug port for advanced debugging. Software can examine the values of signals or combinations of signals in the system using an event multiplexer in the event port to combine the desired signals (see **Figure 15-3**, *Block Diagram of One Event Multiplexer*, on page 15-4 and **Figure 15-6**, *Combining Logic in an Event Multiplexer*, on page 15-11). The output trigger from the event multiplexer can be viewed after latching via the EVCTL[EMUX] field, as shown in the block diagram. The values can also be viewed without latching by sending the output of an event multiplexer to a timer (that is, programming a timer's secondary input as TIN[0–3]) and reading its associated TMRxSCR[INPUT] bit, which reflects the value on the secondary input signal.

A.2 Access Times from the SC1400 Core to Device Components

Table A-7 summarizes the access time from the SC1400 core to different locations on an MSC711x device.

Table A-7. MSC711x Access Times from SC1400 Core

Destination	Description of SC1400 Access	R/W	Core Clocks	Conditions	Comments
Access to Memory Elements					
M1 memory	Single 32-bit access	Read	1	No M1 memory contention.	See **Section 4.3.1**, *Memory Contention*, on page 4-6
		Writes	1	No M1 memory contention.	
	8 consecutive 32-bit accesses	Reads	8	No M1 memory contention.	
		Writes	8	No M1 memory contention.	
M2 memory	Single 32-bit access	Read	9	No other accesses on ASM2.	
		Wr-Imm	8	No other accesses on ASM2.	
		Wr-WB	1	Write buffer empty and no other accesses on AMEC or ASM2.	Write Immediate with No Freeze performs similarly.
	4 consecutive 32-bit accesses	Wr-WB	4	Write buffer empty and no other accesses on AMEC or ASM2.	
	8 consecutive 32-bit Accesses	Reads	40	No other accesses on ASM2.	
		Wr-Imm	40	No other accesses on ASM2.	
		Wr-WB	12	Write buffer empty and no other accesses on AMEC or ASM2.	Write Immediate with No Freeze performs similarly.
External DDR memory 16-Pin	Single 32-bit access	Read	23	No other accesses on ASEMI.	More cycles if $\overline{\text{RAS}}$ is issued.
		Wr-Imm	8	No other accesses on ASEMI.	
		Wr-WB	1	Write buffer empty and no other accesses on AMEC or ASEMI.	Write Immediate with No Freeze performs similarly.
	4 consecutive 32-bit accesses	Wr-WB	4	Write buffer empty and no other accesses on AMEC or ASM2.	
	8 consecutive 32-bit accesses	Reads	84	No other accesses on ASEMI.	MCIF Predictive Read is on. If MCIF ECI Predictive Read is off, the access takes 104 cycles.
		Wr-Imm	40	No other accesses on ASEMI.	
		Wr-WB	13	Write buffer empty and no other accesses on AMEC or ASEMI.	Write Immediate with No Freeze performs similarly.

Table A-7. MSC711x Access Times from SC1400 Core (Continued)

Destination	Description of SC1400 Access	R/W	Core Clocks	Conditions	Comments
External DDR memory 32-Pin	Single 32-bit access	Read	23	No other accesses on ASEMI.	More cycles if RAS is issued.
		Wr-Imm	8	No other accesses on ASEMI.	
		Wr-WB	1	Write buffer empty and no other accesses on AMEC or ASEMI.	Write Immediate with No Freeze performs similarly.
	4 consecutive 32-bit accesses	Wr-WB	4	Write buffer empty and no other accesses on AMEC or ASM2.	
	8 consecutive 32-bit accesses	Reads	72	No other accesses on ASEMI.	MCIF Predictive Read is on.Otherwise, the accesses take 104 cycles.
		Wr-Imm	40	No other accesses on ASEMI.	
		Wr-WB	13	Write buffer empty and no other accesses on AMEC or ASEMI.	Write Immediate with No Freeze performs similarly.
Accesses to Registers					
IPBus Peripheral Registers	Single Access	Read	8	No other accesses on ASSB.	
		Wr-Imm	10	No other accesses on ASSB.	
		BMSET	18	No other accesses on ASSB.	Performs read and write.
APB Peripheral Registers	Single Access	Read	8	No other accesses on ASAPB.	
		Wr-Imm	6	No other accesses on ASAPB.	
		BMSET	13	No other accesses on ASAPB.	Performs read and write.
ECore Registers	Single Access	Read	3		
		Wr-Imm	2		
		BMSET	5		Performs read and write.

A.3 DMA Burst Times

Table A-8 summarizes the burst time through the DMA controller between different locations on an MSC711x device. Both the source and destination in the DMA transfer are programmed for 32-byte bursts.

Table A-8. MSC711x DMA Burst Times

Source	Destination	64-bit Transfers	Bytes	AHB Clocks	Conditions	Comments
M1 to Destination						
M1	M1	64	512	159	—	DMA bandwidth control not used.
	M2	64	512	144	—	
	DDR	64	512	144	Open Page mode, 16-pin	MCIF Write Buffer never full.
	DDR	64	512	144	Open Page mode, 32-pin	MCIF Write Buffer never full.
M2 to Destination						
M2	M1	64	512	128	—	
	M2	64	512	128	—	DMA bandwidth control not used.
	DDR	64	512	128	Open Page mode, 16-pin	MCIF write buffer never full.
	DDR	64	512	128	Open Page mode, 32-pin	MCIF write buffer never full.
DDR (16-Pin Mode) to Destination						
DDR 16-Pin (Page open)	M1	64	512	175	—	MCIF DMA Predictive read enabled.
	M2	64	512	168	—	MCIF DMA Predictive read enabled.
	DDR	64	512	436	Open Page mode, 16-pin	MCIF DMA Predictive read enabled. DMA bandwidth control not used.
DDR (32-Pin Mode) to Destination						
DDR 32-Pin (Page open)	M1	64	512	144	—	MCIF DMA Predictive read enabled.
	M2	64	512	135	—	MCIF DMA Predictive read enabled.
	DDR	64	512	275	Open Page mode, 32-pin	MCIF DMA Predictive read enabled. DMA bandwidth control not used.

Notes: **1.** One AHB clock is equivalent to two core clocks

2. In all cases, the ASM1, ASM2, and ASEMI buses are programmed to park on the AMDMA master when no transactions are occurring.

A.4 DMA Burst Efficiency

It is useful to understand how efficient are long DMA bursts on one channel between a source memory and destination memory, compared to the theoretical bandwidth available through the DDR. This section provides measured results for two DMA transfers of different size: 0.5 KB

and 32 KB. Measurements include the time required to open a new bank of the DDR. **Table A-10** summarizes the efficiencies attained by the DMA controller during bursts through the DDR. No other masters were accessing the DDR when these measurements were collected.

Table A-9. DMA Burst Efficiency Between DDR and M1 Memory

Source	Destination	512 Byte DMA Transfer	32 KB DMA Transfer	Measured BW (300 MHz, 32 KB)
M1	DDR: 16-pin	89%	87.75%	526.5 MBps
	DDR: 32-pin	44%	44.13%	529.6 MBps
M2	DDR: 16-pin	98%	87.75%	526.5 MBps
	DDR: 32-pin	49%	44.13%	529.6 MBps
DDR: 16-pin	M1	73%	74.25%	445.5 MBps
	M2	76%	71.0%	426.0 MBps
	DDR: 16-pin	29%	50.0%	300.0 MBps
DDR: 32-pin	M1	45%	46.0%	552.0 MBps
	M2	47%	43.5%	522.0 MBps
	DDR: 32-pin	23%	33.26%	399.1 MBps

Notes: **1.** Theoretical bandwidth at 300 MHz is calculated as follows:
16-pin DDR: 2 edges × 2 bytes × 150 MHz = 600 MBps
32-pin DDR: 2 edges × 4 bytes × 150 MHz = 1200 MBps

2. These efficiencies require the DMA predictive read capability in the MCIF to be enabled. In the 32 KB transfer, the DMA channel is programmed for WRAP4 bursts with NBYTES = 32 KB and the major loop counter equal to 1. The result is a minor loop count of 1024.

The data indicates that the measured bandwidth through the DMA is approximately the same for 32-pin DDR as for 16-pin. However, in a fully loaded system, the 32-pin option maintains high DMA transfer rates while also servicing other masters such as ICache misses to DDR and Ethernet bursts to DDR.

A.5 ICache Efficiency

Running the ICache can result in small performance degradations compared to running the same code directly out of M1 memory. Since M1 memory is best suited for storing data, it important to understand the ICache impact when an application runs with program code in M2 or DDR memory. This section describes the measured performance impact when code runs from the different memories. Real applications are more complex, but these measurements provide a useful frame of reference. In all measurements listed in **Table A-10**, the cache is flushed before the application runs to emulate the fact that in a real application the ICache may not yet contain the desired entries. No other masters on the device are accessing the DDR when the measurements listed in **Table A-10** were collected.

Table A-10. Impact of Running Programs from M2 or DDR Memory

Application	Ratio: Code in M1 memory	Ratio: Code in M2 memory	Ratio: Code in DDR memory (32-pin DDR)	Ratio: Code in DDR memory (16-pin DDR)
G.729ab encode	1.0	1.01	1.12	1.28
G.729ab decode	1.0	1.03	1.19	1.47
G.729ab encode + decode	1.0	1.02	1.14	1.32
G.723.1 encode	1.0	1.00	1.02	1.05
G.723.1 decode	1.0	1.03	1.10	1.22
G.723.1 encode + decode	1.0	1.01	1.03	1.07

Notes: **1.** These measurements were made with the ICache predictive read feature enabled. The IFU IRCR is programmed with a value of 0x0405.
2. When code is stored in M1 memory, the ICache is not used.
3. Total code size for G.729AB is 39 KB. Total code size for G.723.1 is 40 KB.

A.6 Handling Access Errors

Memory access errors are signalled by non-maskable interrupt requests, and the source of the request is readable in the interrupt controller NMIPR.

A.6.1 Extended Core

Table A-11 summarizes the errors that can occur during SC1400 memory accesses to addresses in the extended core:

Table A-11. SC1400 Access Error Types Within the Extended Core

Error Type	Description	Detection Location	Bit(s) in NMIPR	Comments
SC1400 Program Accesses				
Address Out-of-Range	SC1400 initiated a program access to an invalid address in the ECore.	ECore	19	
Misaligned Access	SC1400 generated a misaligned program access	ECore	25	All SC1400 program accesses must be aligned on 16-bit boundaries.
SC1400 Data Accesses				
Address Out-of-Range	SC1400 initiated a data access to an invalid address in the ECore.	ECore	18	
Misaligned Access	SC1400 generated a misaligned data access	ECore	28	All SC1400 data accesses must be aligned to the size of the access.

A.6.2 AHB Subsystem

Table A-12 summarizes the errors that can occur during MSC711x memory accesses in the AHB subsystem.

Table A-12. Access Error Types to AHB Subsystem

Access Error Type	Description	Detection Location	Bit(s) in NMIPR	Comments
Address Out-of-Range	Address from an AHB Master is not in an allowed address range.	AHB master initiating access	4–1	Only the master for the access can be known. Status bits indicate which AHB master initiated the access.
Misaligned Access	AHB master has generated a misaligned access.	AHB master initiating access	24–21	Only the master for the access can be known. Status bits indicate which AHB master initiated the access.
Bus Error	AHB master receives an AHB error response for an access.	AHB master initiating access	14–11	Receives response from either an AHB slave or the crossbar switch. See **Section A.6.3**, *Error Detection on Both Ends of the Transfer*.
Bus Time-Out	Access on an AHB slave has timed out.	AHB slave bus with late/no response	10–5	Both the master and slave for the access can be known. The slave is indicated by a bus time-out. The master is indicated by a bus error (delayed).
Write to ROM Location	AHB master initiated a write access to a location in ROM.	Crossbar switch	0	Destination is known, ROM location. The source is indicated by a bus error (delayed).
DDR Memory Select Error	AHB master initiated an access to an address outside the regions defined by the DDR controller chip selects.	DDR controller	26	Destination is known, - DDR address space. The source is indicated by a bus error (delayed).
Peripheral Size Error	A 64-bit AHB access to an AHB or IPBus peripheral.	Crossbar switch	20	Destination is known, MSC711x peripheral. The source is indicated by a bus error (delayed).

A.6.3 Error Detection on Both Ends of the Transfer

For some access errors listed in **Table A-12**, you can detect both the master that generated the access and the slave bus for which the access was targeted. The first non-maskable interrupt request arrives and is followed after a short delay by a second non-maskable request.

Table A-13. Detecting Both Ends of a Failed Access

First Access Error (Slave Bus)	Second Access Error (Master Bus)
Bus time-out	Bus error
Write to ROM location	Bus error
DDR memory select error	Bus error

Table A-13. Detecting Both Ends of a Failed Access

First Access Error (Slave Bus)	Second Access Error (Master Bus)
M1 access time-out	Bus error
Peripheral size error	Bus error

When a chip-level non-maskable interrupt is serviced, which includes any first access error in **Table A-13**, the service routine reads the NMIPR. When the register is read, the status bits for both sources are set. Both interrupt sources should then be cleared by servicing these two bits in the NMIPR.

Reading the NMIPR after the proper delay, you can determine both ends of the access. If the NMIPR register is read immediately, you can miss the second access error, which then generates a subsequent non-maskable interrupt request when exiting the original interrupt service handler.

A.6.4 Automatic State Recovery During Error Detection

An access error can be detected when the device is in a state in which it cannot respond correctly. Therefore, the device automatically reconfigures itself so that it can respond. Specifically, the device is designed to recover from the following cases:

- Accidental IFU or ECI access when the crossbar switch is powered down and a time-out occurs.
- Non-maskable interrupt when a master other than the SC1400 core has a desired crossbar slave port.

The following capabilities ensure that deadlock does not occur and the SC1400 can respond to the non-maskable interrupt:

- Automatic crossbar priority change on non-maskable interrupt
  (see CNMI bit in **Table 7-10**, *DEVCFG Bit Descriptions*, on page 7-17)
- Automatic crossbar wake-up on non-maskable interrupt
  (see XHRQ bit in **Table 4-12**, *GPSCTL Bit Descriptions*, on page 4-43)

A.6.5 Configurable Priority Modification During a Chip Event

Another programmable feature for error recovery from a particular state is the crossbar switch alternate priority register set. A particular event detectable in the event port switches the crossbar switch from its normal set of priority registers to an alternate set. Priority switching is programmed not only in crossbar switch alternate priority registers but also in the event port EVOUTx[ACT1–0] register bits. Program these registers when the crossbar slaves are configured for fixed-priority arbitration and the AMEC has the highest priority. This ensures that the SC1400 core can recover from a system error when the device is not programmed correctly and a resource is locked out of the crossbar switch, thus providing an additional mechanism for

assisting in recovery from an application error. If you do not want this feature, do not program the event port to generate this action and configure the alternate priority registers with the same values as their corresponding normal slave port registers.

A.7 Best Use of the Development Tools

You can disassemble the boot ROM using the development tools. When you are using the compiler to access peripheral registers in the MSC711x memory map, be aware of the restrictions on access sizes on many of these registers. The following tables summarize the permitted access sizes:

- **Table 5-11**, *Permitted Accesses to MSC711x Blocks via Device-Level Buses*, on page 5-38
- **Table 5-12**, *Permitted Accesses to MSC711x Blocks via System-Level Buses*, on page 5-39

When using indirection in compiled code, be careful with MSC711x registers that do not support 16-bit accesses as summarized in **Table 5-11**, *Permitted Accesses to MSC711x Blocks via Device-Level Buses*, on page 5-38. If an AND or OR operation with immediate data is performed on one of these restricted registers using indirection, incorrect operation results:

```
*crossbar_reg1 |= 0x8000; // Careful with this C code
*crossbar_reg2 &= 0x8000; // Careful with this C code
*crossbar_reg3 ^= 0x8000; // Careful with this C code
```

The compiler may generate a BMSET or BMCLR instruction, but these instructions do not work correctly because of access restrictions. This applies only to peripheral registers on the MSC711x device that do not support 16-bit accesses. For example, the crossbar registers do not support 16-bit accesses.

Take care when you are debugging with the watchdog timer on. Often, it is easier to debug without the watchdog timer, which is disabled in two different ways:

- The program never enables the watchdog. In this case, the watchdog timer comes out of reset in a disabled state.
- Use the SWTE pin for applications programmed with the watchdog timer enabled. The SWTE pin is sampled only power-on reset is deasserted.

You can also configure the operation of the device when it enters Debug mode:

- Maskable and Non-maskable interrupts can optionally be masked when the device is in Debug mode. Configure this feature with the interrupt controller MIPR[DDBG] bit.
- The DMA controller can optionally be configured not to start new channels when the device enters Debug mode. Configure this feature with the DMA controller DMACR[EDBG] bit. When this bit is set, the DMA stalls when the current minor loop completes.

MSC711x Boot Code

B

```
; ###############################################################################
; ###
; ### MSC711x Boot Program
; ###
; ### >>>>>>
; ### >>>>>> NOTE: Before using this boot program, it is *** CRITICAL ***
; ### >>>>>>       that the section below in boot code entitled
; ### >>>>>>       "Configuring the Boot Program" is correctly set up.
; ### >>>>>>
; ### >>>>>>       This section is used for modifying the boot program
; ### >>>>>>       for verification in the lab or on the tester.
; ### >>>>>>
; ### >>>>>>       The FINAL VERSION OF THE BOOT PROGRAM must set all these
; ### >>>>>>       parameters up correctly.
; ### >>>>>>
; ### >>>>>>
; ###
; ### Assembled as follows:
; ###    $asmsc100 -a -b -l -mex -obe <filename>.asm
; ###
; ### Description:
; ###    Allows boot from:
; ###       - HDI16
; ###       - I2C
; ###       - SPI (Implemented in software using GPIO pins)
; ###       - Test Mode
; ###
; ### Revision History:
; ### -----------------
; ###                      01/20/2004   First Revision
; ###                      09/28/2004   First Modifications for Rev A
; ###                      10/25/2004   1st Release for Rev A
; ###                      11/01/2004   2nd Release for Rev A
; ###                      11/02/2004   3rd Release for Rev A
; ###                      11/03/2004   4th Release for Rev A
; ###                      11/22/2004   5th Release for Rev A
; ###                                      - passes all tests
; ###                      12/17/2004   6th Release for Rev A
; ###                                      - updated comments
; ###                                      - fixed SR[DI] issue - added "ei"
; ###                                      - Test Boot now turns on CLKO pin
; ###                                      - No longer setting CHPCFG[PAS]
; ###                                        since not reqd on Rev A for CLKO
; ###                      12/20/2004   Next Release for Rev A
; ###                                      - writes to SR, EMR due to SC issue
; ###                      01/06/2004   Next Release for Rev A
; ###                                      - commenting improved
```

```
; ###                       01/26/2004   Final set of commenting on I2C code.
; ###                       04/22/2004   Removed internal use only info
; ###                       09/16/2005   Removed internal use only info
; ###
; ### Layout of Code in the Boot ROM:
; ### ------------------------------
; ###    BASE+$0000:   Reset Vectors
; ###    BASE+$____:   Boot Parameter Table (located at end of reset vectors)
; ###    BASE+$____:   Main Program
; ###    BASE+$____:   HDI16 Boot   (using HDI8 or HDI16 functionality)
; ###    BASE+$____:   I2C   Boot
; ###    BASE+$____:   SPI   Boot   (main pin set)
; ###    BASE+$____:   SPI   Boot   (alternate pin set)
; ###    BASE+$1000:   Interrupt Vectors (non-maskable interrupts)
; ###    BASE+$____:   Special Code - CodeWarrior
; ###    BASE+$____:   Special Code - Tester
; ###
; ###    Note: The interrupt vector table is located right at the middle
; ###          of the boot code at 4KB because it must be located
; ###          on 4KB boundaries.
; ###
; ### Reserved M1 Memory Used by Boot Program:
; ### ----------------------------------------
; ###    The boot code also requires locations in M1 memory to operate
; ###    correctly.  As a result, the last 512 bytes of M1 memory are reserved.
; ###
; ###    >>> See detailed comment below for the exact usage of these locations.
; ###
; ###
; ###
; ### Chip Configuration after Power-On Reset:
; ### ----------------------------------------
; ###    - clocked w/ bypass clock
; ###    - PLL is disabled
; ###    - Interrupts are disabled   (non-maskable are still recognized though)
; ###    - jumps to "power-on reset vector"
; ###
; ### Chip Configuration after Hard Reset:
; ### ------------------------------------
; ###    - clocked w/ same clock as before hardware reset
; ###    - PLL configuration not modified
; ###    - Interrupts are disabled  (non-maskable are still recognized though)
; ###    - jumps to "hard reset vector"
; ###
; ###
; ### Register Usage - Over ENTIRE Program:
; ### -------------------------------------
; ###    r8      - stores address of 1st location in the section of M1 memory
; ###              reserved for use by the boot program (ie, last 512 locats)
; ###
; ### Register Usage - HDI
; ### --------------------
; ###    (see HDI section)
; ###
; ### Register Usage - SPI
; ### --------------------
; ###    (see SPI section)
; ###
```

```
; ### Additional Notes:
; ### -----------------
; ###     - Almost all of the code for Test Boot modes and HDI (16pin) boot
; ###       is identical.  A sizeable reduction in code size can be achieved
; ###       by having this common code only present once.
; ###
; ##############################################################################

; --- M1 Memory Parameters

BASE_M1_MEMORYequ     $00000000; (typically not modified)

BASE_M2_MEMORYequ     $01000000; (typically not modified)
BASE_BOOT_ROM equ     $01400000; (typically not modified)

; --- Memory Sizes for Different Devices

M1_MEMSZ_7110 equ           $00010000     ; Size of M1 for this device

M1_MEMSZ_7112 equ           $00030000     ; Size of M1 for this device
M1_MEMSZ_7113 equ           M1_MEMSZ_7112 ; Size of M1 for this device
M1_MEMSZ_7115 equ           M1_MEMSZ_7112 ; Size of M1 for this device
M1_MEMSZ_7116 equ           M1_MEMSZ_7112 ; Size of M1 for this device

M1_MEMSZ_7118 equ           $00040000     ; Size of M1 for this device
M1_MEMSZ_7119 equ           M1_MEMSZ_7118 ; Size of M1 for this device

; --- Device IDs (corresponds to DEVID[DEVNBR] field)

ID_FINAL      equ           -1            ; Device ID for Final Boot Code
ID_7110                     equ           $3            ; Device ID for 7110 device
ID_7112                     equ           $7            ; Device ID for 7112 device
ID_7113                     equ           $6            ; Device ID for 7113 device
ID_7115                     equ           $1            ; Device ID for 7115 device
ID_7116                     equ           $2            ; Device ID for 7116 device
ID_7118                     equ           $9            ; Device ID for 7118 device
ID_7119                     equ           $A            ; Device ID for 7119 device


; ##############################################################################
; ###
; ### Configuring the Boot Program:
; ###
; ###    The Boot Program can be configured with different parameters
; ###    in this section.  These are set correctly for the final configuration
; ###    required for boot on the device.
; ###
; ###
; ###
; ###

OVERRIDE_BM_PINSequ         -1            ; Configuration used in final boot code
OVERRIDE_DEVIDequ           ID_FINAL      ; Configuration used in final boot code
BASE_BOOT_CODEequ           BASE_BOOT_ROM ; Base address used in final boot code
BURN_SERIAL                 equ           0             ; Configuration used in final boot code
LABTEST_SPI                 equ           0             ; Configuration used in final boot code
JMP_AT_LOC_00000000equ      0             ; No  instruction at p:$00000000
```

```
; ###
; ### End - Configuring the Boot Program
; ###
; ##############################################################################

            page         132                                        ; useful for list files



; ###
; ### Include Files
; ###

      IF LABTEST_SPI
            nolist
            include 'spidata.asm'
            list
      ENDIF

; ##############################################################################
; ###
; ### Usage - Last 512 Locations of M1 Memory
; ### ---------------------------------------
; ###    $000 to $0FF   Overlay Region   (256 bytes)
; ###    $100 to $13F   (64 byte reserved area)
; ###    $140 to $17F   (64 byte area for Boot Variables - see below)
; ###    $180 to $1FF   System Stack used by Boot Program   (128 bytes)
; ###                      - each bsr pushes 8 bytes
; ###                      - thus, maximum depth of 16 subroutines supported
; ###
; ###    $140 to $17f   64 bytes reserved for Boot Variables:
; ###    ------------   -----------------------------------
; ###    $160           Boot Pgm Var  (8-bit): Reserved
; ###    $161           Boot Pgm Var  (8-bit): Reserved
; ###    $162           Boot Pgm Var  (8-bit): Reserved
; ###    $163           Boot Pgm Var  (8-bit): Reserved
; ###    $164           Boot Pgm Var  (8-bit): Reserved
; ###    $165           Boot Pgm Var  (8-bit): Reserved
; ###    $166           Boot Pgm Var  (8-bit): NMITYPE:
; ###                                             - -1: No nonmaskable ints
; ###                                                   occurred.
; ###                                             - $0: TRAP     occurred
; ###                                             - $1: (reserved)
; ###                                             - $2: ILLEGAL  occurred
; ###                                             - $3: DEBUG occurred
; ###                                             - $4: OVERFLOW occurred
; ###                                             - $5: (reserved)
; ###                                             - $6: Auto-NMI occurred
; ###                                             - $7: Auto-maskable occurred
; ###    $167           Boot Pgm Var  (8-bit): RSTSRC - Reset Source:
; ###                                             - $0: POR
; ###                                             - $1: External Hard Reset
; ###                                             - $2: Software Watchdog Tmr
; ###                                             - $3: Bus Timeout Reset
; ###                                             - $4: JTAG Int Reset
; ###                                             - $5: (reserved)
; ###                                             - $6: (reserved)
; ###                                             - $7: (reserved)
```

```
; ###
; ###    $168           Boot Pgm Var (16-bit): Reserved
; ###    $16A           Boot Pgm Var (16-bit): Reserved
; ###    $16C           Boot Pgm Var (16-bit): Reserved
; ###    $16E           Boot Pgm Var (16-bit): BTERR- Boot Error:
; ###                                             - $FFFF: Boot in Progress
; ###                                             - $0000: Boot Successful
; ###                                             - $BAD0: NMI occurred
; ###                                                      (see locat $17C)
; ###                                             - $BAD1: HDI checksum
; ###                                             - $BAD2: SPI checksum
; ###                                             - $BAD3: I2C checksum
; ###                                             - $BAD4: (unused)
; ###                                             - $BAD5: (unused)
; ###                                             - $BAD6: (unused)
; ###
; ###    $170           Boot Pgm Var (32-bit): Reserved
; ###    $174           Boot Pgm Var (32-bit): Reserved
; ###    $178           Boot Pgm Var (32-bit): EXTBTMD - Extracted Boot Mode
; ###                                           - extracted from BM pins by the
; ###                                             routine "determine_boot_mode"
; ###    $17C           Boot Pgm Var (32-bit): CNMIPR - Captured NMIPR
; ###                                           - Value of NMIPR if a
; ###                                             non-maskable int occurs
; ###
; ###
; ###
; ### Offset Addresses - For Addressing the Last 512 Locations of M1 Memory
; ### ----------------
; ###    - Overlay Area
; ###    - Stack
; ###    - Boot Code Variables
; ###
; ### Note: Each of these is an offset from the r8 pointer
; ###       which points to the first reserved location
; ###       at the end of M1 memory.
; ###
; ##############################################################################

OFFSET_M1_OVERLAYequ     $0000; Region reserved for Overlays
OFFSET_M1_VARS             equ     $0140 ; Region reserved for Boot Variables
OFFSET_M1_STACK            equ     $0180 ; Region reserved for Stack

OFFSET_M1VAR_NMITYPEequ     $0166; Offset to  8-bit Var: Type of NMI
OFFSET_M1VAR_RSTSRCequ     $0167; Offset to  8-bit Var: Reset Source
OFFSET_M1VAR_BTERRequ     $016E; Offset to 16-bit Var: Boot Error
OFFSET_M1VAR_EXTBTMDequ     $0178; Offset to 32-bit Var: Extr Boot Mode
OFFSET_M1VAR_CNMIPRequ     $017C; Offset to 32-bit Var: NMIPR register


; ###
; ### Interrupt Vector Table
; ###

BASE_EXCEPTION_TABLE equ  BASE_BOOT_CODE+$1000
```

```
; ###
; ### Base Addresses - Register Banks
; ###

HDIAPB_BASE   equ     $06007000; For ASAPB: HCR,HPCR etc
HDIAHB_BASE   equ     $01F87000; For ASTH:  HORX and HOTX
CLK_BASE      equ     $0400C000; Clock Control and Reset Registers base

GPIO_BASE     equ     $06009000
BTM_BASE        equ     $0400A000        ; Bus Timeout Monitor Register
I2C_BASE        equ     $04009000


; ###
; ### Register Addresses
; ###

; ---- SYSTEM CONTROL ----

CHIPCFG_REG   equ    BTM_BASE+$00000080
DEVID_REG     equ    BTM_BASE+$00000088


; ---- GPIO ----

GPIO_PortA_CTLequ          GPIO_BASE+$00000008
GPIO_PortB_CTLequ          GPIO_BASE+$00000014
GPIO_PortC_CTLequ          GPIO_BASE+$00000020

GPIO_PortA_DATAequ         GPIO_BASE+$00
GPIO_PortB_DATAequ         GPIO_BASE+$0c
GPIO_PortC_DATAequ         GPIO_BASE+$18


GPIO_PortC_EXPRTequ        GPIO_BASE+$58

; ---- Reset and Clock ----

RSR                        equ          CLK_BASE+$00000040; Reset Status Register
CLKCTRL                    equ          CLK_BASE+$00000000      ; Clock Control Register

; ---- HDI16 (Chip Side) ----

HCR                        equ     HDIAPB_BASE+$00000000
HPCR                       equ     HDIAPB_BASE+$00000020
HSR                        equ     HDIAPB_BASE+$00000040
HCVR                       equ     HDIAPB_BASE+$00000060

HOTX                       equ     HDIAHB_BASE+$00000080
HORX                       equ     HDIAHB_BASE+$000000a0

; ---- I2C Registers ----
IADR            equ     I2C_BASE+$00
IFDR            equ     I2C_BASE+$04
I2CR            equ     I2C_BASE+$08
I2SR            equ     I2C_BASE+$0C
I2DR            equ     I2C_BASE+$10
```

```
EONCE_BASE      equ $00EFF000
ECI_BASE        equ $00F00000
IC_BASE         equ $00F00100
ICARRAY_BASE    equ $00F20000
SKYBLUE_BASE    equ $04000000
SBG_BASE        equ $04000000
TMRA_BASE       equ $04001000
TMRB_BASE       equ $04002000
XBAR_BASE       equ $04003000
DMA_BASE        equ $04004000
ENET_BASE       equ $04006000
EV_BASE         equ $0400B000
APB_BASE        equ $06000000
SWT_BASE        equ $06001000
TDM0_BASE       equ $06004000
TDM1_BASE       equ $06005000
TDM2_BASE       equ $06006000
HDI16_BASE      equ $06007000
UART_BASE       equ $06008000
ICTL_BASE       equ $0600A000
IPL_BASE        equ $0600B000
AHB_BASE        equ $01F84000
TDM0AHB_BASE    equ $01F84000
TDM1AHB_BASE    equ $01F85000
TDM2AHB_BASE    equ $01F86000
HDI16AHB_BASE   equ $01F87000


GPA_DR       equ          GPIO_BASE+$00
GPB_DR       equ          GPIO_BASE+$0C
GPC_DR       equ          GPIO_BASE+$18
GPD_DR       equ          GPIO_BASE+$24

GPA_DDR      equ          GPIO_BASE+$04
GPB_DDR      equ          GPIO_BASE+$10
GPC_DDR      equ          GPIO_BASE+$1C
GPD_DDR      equ          GPIO_BASE+$28

GPA_EXPRT    equ          GPIO_BASE+$50
GPB_EXPRT    equ          GPIO_BASE+$54
GPC_EXPRT    equ          GPIO_BASE+$58
GPD_EXPRT    equ          GPIO_BASE+$5C

PACTL        equ          GPIO_BASE+$08
PBCTL        equ          GPIO_BASE+$14
PCCTL        equ          GPIO_BASE+$20
PDCTL        equ          GPIO_BASE+$2C

ICCR         equ     IC_BASE
ICCMR        equ     IC_BASE+$04
LRUSR        equ     IC_BASE+$10
TASR         equ     IC_BASE+$14
VBASR        equ     IC_BASE+$18

IRBSR0       equ     IC_BASE+$82
IRBSR1       equ     IC_BASE+$86
IRBSR2       equ     IC_BASE+$8A
IRBSR3       equ     IC_BASE+$8E
```

```
IRCR0       equ     IC_BASE+$80
IRCR1       equ     IC_BASE+$84
IRCR2       equ     IC_BASE+$88
IRCR3       equ     IC_BASE+$8c

SZ_Perif   equ $04000000    ; size of the peripheral address space
B_Perif    equ $04000000    ; beginning address for all peripherals
ECI_WBDAR0 equ $00F00040    ; address of WB's WBDAR0 register
ECI_WBCR   equ $00F00004    ; address of WB control register



; ###
; ### Macro Definitions - Used across entire program
; ###

INSNOPS macro NBRNOPS                   ; Argument is number of NOPs desired (decimal)
              dup           ?NBRNOPS
              nop
              endm
              endm

; ================================================================
; ================================================================
; ================================================================

; ###
; ### Location P:$00000000
; ###

      IF JMP_AT_LOC_00000000
              org                       p:$00000000
              jmp                       power_on_reset
      ENDIF


;#########################################################
;#########################################################
;##
;##   Reset Vectors
;##      - see "Reset Vector" in the "Reset" Chapter of the Manual
;##      - Boot ROM Base = BASE_BOOT_CODE
;##      - Valid Reset Sources:
;##           0x000: Power On Reset
;##           0x040: External Hard Reset
;##           0x080: Software Watchdog Timer Reset
;##           0x0C0: Bus Timeout Reset
;##           0x100: JTAG Reset
;##
;#########################################################
;#########################################################

; ###
; ### Power On Reset
; ###

              org             p:BASE_BOOT_CODE+$0000
power_on_reset
```

```
                ;
                ; Add code here unique to Power-On Reset
                ;

; --- Get Value to Place (later) in RSTSRC Boot Variable
                move.w          #$0,d7

                bra             >common_start
end_power_on_reset

                INSNOPS         ((power_on_reset+$40-end_power_on_reset)/2)

; ###
; ### External Hard Reset
; ###

                org               p:BASE_BOOT_CODE+$0040
ext_hard_reset
                ;
                ; Add code here unique to External Hard Reset
                ;

; --- Get Value to Place (later) in RSTSRC Boot Variable
                move.w          #$1,d7

                bra             >common_start
end_ext_hard_reset

                INSNOPS         ((ext_hard_reset+$40-end_ext_hard_reset)/2)

; ###
; ### Watchdog Reset
; ###

                org               p:BASE_BOOT_CODE+$0080
watchdog_reset
                ;
                ; Add code here unique to Watchdog Reset
                ;

; --- Get Value to Place (later) in RSTSRC Boot Variable
                move.w          #$2,d7

                bra             >common_start
end_watchdog_reset

                INSNOPS         ((watchdog_reset+$40-end_watchdog_reset)/2)

; ###
; ### Bus Timeout Reset
; ###

                org               p:BASE_BOOT_CODE+$00C0
bus_timeout_reset
                ;
                ; Add code here unique to Bus Timeout Reset
                ;
```

```
; --- Get Value to Place (later) in RSTSRC Boot Variable
              move.w        #$3,d7

              bra           >common_start
end_bus_timeout_reset

              INSNOPS       ((bus_timeout_reset+$40-end_bus_timeout_reset)/2)

; ###
; ### JTAG Reset
; ###

              org             p:BASE_BOOT_CODE+$0100
jtag_reset
              ;
              ; Add code here unique to JTAG Reset
              ;

; --- Get Value to Place (later) in RSTSRC Boot Variable
              move.w        #$4,d7

              bra           >common_start
end_jtag_reset

              ;;;INSNOPS    ((jtag_reset+$40-end_jtag_reset)/2)

; ###
; ### Last Reserved Reset Vectors
; ###

last_reset
              ; (currently unused)
              ; (space instead used for code below)
end_last_reset



; ###
; ### Routine: Determine Boot Mode
; ###
; ###    - Examines BM pins to correctly set the following:
; ###         - CLKCTRL[PLLMLTF]
; ###         - CLKCTRL[PLLDVF]
; ###         - CLKCTRL[RNG]
; ###         - CLKCTRL[CKSEL]
; ###
; ###    - Currently written to support up to 64 boot modes (via loop)
; ###
; ###    - Return Value:
; ###         - D0 = Value of Boot Parameters returned
; ###
; ###    - Register Usage:
; ###         - D2 = Value on BM pins
; ###         - D3 = Loop Index
; ###         - D0 = Value of Boot Parameters returned
; ###
; ###         - R0 = Decrementing Pointer which accesses Boot Parameter Table
; ###
```

```
; ###    - NOTE: Algorithm written to take equal time indep of boot mode.
; ###            This can be very useful for a tester!
; ###

NBR_BTMDS equ 64                        ; Total Number of Boot Modes

             align        16

determine_boot_mode

; --- Get BM Pins
             move.l       RSR,d2                    ; D2 = Value of BM pins (in bits
[15:12])
             asrr         #12,d2                    ; then shifted to bits [5:0]

;      Handle Override of BM Pins
             move.w       #OVERRIDE_BM_PINS,d3
             cmpeq.w      #-1,d3
             nop                                    ; reqd by pipeline: see dependency T.1
             iff          move.l       d3,d2

;      Continue
             and          #$3f,d2,d2    ;    (6-bit value)

             move.l       #Param_Tbl+(NBR_BTMDS-1)*2,r0
                                                    ; R0 = Ptr to last entry in Update Value
Table

; --- Determine Boot Mode

   ; for (i=63; i=0; i--)   {

             move.w       #NBR_BTMDS-1,d3; D3 = Loop Index

find_bm_lp
             cmpeq        d3,d2                     ; if loop index == BM[n:0]
             nop                                    ; reqd by pipeline: see dependency T.1
             ift   move.w   (r0),d0; load CLKCTRL fields for this boot index
             suba         #2,r0

             deceq        d3
             cmpeq.w      #-1,d3
             bf           <find_bm_lp

   ; }

; --- Return to Main Program
             rts

; ###
; ### Parameter Table
; ###
; ### This table specifies parameters for a boot mode (clocking and boot src).
; ### Although the CLKCTRL register is 32-bits, less than 16-bits can be
; ### configured by the boot pins.  As a result, this table stores a
; ### 16-bit entry for each boot mode, allocated as follows:
; ###    - PLLMLTF field: [05:00]   (6-bits)
; ###    - PLLDVF  field: [08:06]   (3-bits)
```

```
; ###    - RNG     field: [09]      (1-bit)
; ###    - CKSEL   field: [11:10]   (2-bits)
; ###
; ###    - BSRC    field: [15:12]   (4-bits)   ("BSRC" is "Boot Source")
; ###

RNG____                  equ          ($1<<9)                    ; CLKCTRL[RNG] unused
RNG_DV1                  equ          ($1<<9)                    ; CLKCTRL[RNG] set for /1
RNG_DV2                  equ          ($0<<9)                    ; CLKCTRL[RNG] set for /2

CKSEL_BYPASS  equ        ($0<<10)     ; CLKCTRL[CKSEL] set for /1
CKSEL_PLLDV1  equ        ($3<<10)     ; CLKCTRL[CKSEL] set for /1
CKSEL_PLLDV2  equ        ($1<<10)     ; CLKCTRL[CKSEL] set for /2

DVF__                    equ          (0<<6)                     ; CLKCTRL[PLLDVF] unused
DVF_1                    equ          (0<<6)                     ; CLKCTRL[PLLDVF] set for
/1
DVF_2                    equ          (1<<6)                     ; CLKCTRL[PLLDVF] set for
/2
DVF_3                    equ          (2<<6)                     ; CLKCTRL[PLLDVF] set for
/3

MLTF___                  equ          00                         ; CLKCTRL[PLLMLTF] unused
MLTF_01                  equ          00                         ; CLKCTRL[PLLMLTF] set for
*01
MLTF_12                  equ          11                         ; CLKCTRL[PLLMLTF] set for
*12
MLTF_13                  equ          12                         ; CLKCTRL[PLLMLTF] set for
*13
MLTF_14                  equ          13                         ; CLKCTRL[PLLMLTF] set for
*14
MLTF_15                  equ          14                         ; CLKCTRL[PLLMLTF] set for
*15
MLTF_16                  equ          15                         ; CLKCTRL[PLLMLTF] set for
*16
MLTF_17                  equ          16                         ; CLKCTRL[PLLMLTF] set for
*17
MLTF_18                  equ          17                         ; CLKCTRL[PLLMLTF] set for
*18
MLTF_32                  equ          31                         ; CLKCTRL[PLLMLTF] set for
*32

BT_HDI0                  equ          ($0<<12)      ; Boot Source: HDI port
BT_SPI0                  equ          ($1<<12)      ; Boot Source: SPI - main pins
BT_SPI1                  equ          ($2<<12)      ; Boot Source: SPI - backup pins
BT_I2C0                  equ          ($3<<12)      ; Boot Source: I2C
BT_TST0                  equ          ($4<<12)      ; Boot Source: Test Mode (0011)
BT_TST1                  equ          ($5<<12)      ; Boot Source: Test Mode (1111)

             align       16

Param_Tbl
             dcw         BT_HDI0+DVF__+MLTF___+RNG____+CKSEL_BYPASS; Boot Mode  0
             dcw         BT_I2C0+DVF__+MLTF___+RNG____+CKSEL_BYPASS; Boot Mode  1
             dcw         BT_HDI0+DVF_2+MLTF_32+RNG_DV1+CKSEL_PLLDV2; Boot Mode  2
             dcw         BT_TST0+DVF_2+MLTF_32+RNG_DV2+CKSEL_PLLDV1; Boot Mode  3

             dcw         BT_HDI0+DVF_2+MLTF_12+RNG_DV1+CKSEL_PLLDV1; Boot Mode  4
```

```
                dcw             BT_HDI0+DVF_1+MLTF_12+RNG_DV1+CKSEL_PLLDV1; Boot Mode  5
                dcw             $0000                                                   ; Boot Mode
6
                dcw             BT_HDI0+DVF_3+MLTF_12+RNG_DV1+CKSEL_PLLDV1; Boot Mode  7

                dcw             BT_SPI0+DVF__+MLTF___+RNG____+CKSEL_BYPASS; Boot Mode  8
                dcw             BT_SPI0+DVF_1+MLTF_17+RNG_DV2+CKSEL_PLLDV1; Boot Mode  9
                dcw             BT_SPI0+DVF_2+MLTF_16+RNG_DV2+CKSEL_PLLDV1; Boot Mode 10
                dcw             BT_SPI0+DVF_3+MLTF_18+RNG_DV2+CKSEL_PLLDV1; Boot Mode 11

                dcw             BT_SPI1+DVF__+MLTF___+RNG____+CKSEL_BYPASS; Boot Mode 12
                dcw             $0000                                                   ; Boot Mode
13
                dcw             $0000                                                   ; Boot Mode
14
                dcw             BT_TST1+DVF_2+MLTF_16+RNG_DV1+CKSEL_PLLDV1; Boot Mode 15

                dcw             $0000           ; Boot Mode 16
                dcw             $0000           ; Boot Mode 17
                dcw             $0000           ; Boot Mode 18
                dcw             $0000           ; Boot Mode 19
                dcw             $0000           ; Boot Mode 20
                dcw             $0000           ; Boot Mode 21
                dcw             $0000           ; Boot Mode 22
                dcw             $0000           ; Boot Mode 23

                dcw             $0000           ; Boot Mode 24
                dcw             $0000           ; Boot Mode 25
                dcw             $0000           ; Boot Mode 26
                dcw             $0000           ; Boot Mode 27
                dcw             $0000           ; Boot Mode 28
                dcw             $0000           ; Boot Mode 29
                dcw             $0000           ; Boot Mode 30
                dcw             $0000           ; Boot Mode 31

                dcw             $0000           ; Boot Mode 32
                dcw             $0000           ; Boot Mode 33
                dcw             $0000           ; Boot Mode 34
                dcw             $0000           ; Boot Mode 35
                dcw             $0000           ; Boot Mode 36
                dcw             $0000           ; Boot Mode 37
                dcw             $0000           ; Boot Mode 38
                dcw             $0000           ; Boot Mode 39

                dcw             $0000           ; Boot Mode 40
                dcw             $0000           ; Boot Mode 41
                dcw             $0000           ; Boot Mode 42
                dcw             $0000           ; Boot Mode 43
                dcw             $0000           ; Boot Mode 44
                dcw             $0000           ; Boot Mode 45
                dcw             $0000           ; Boot Mode 46
                dcw             $0000           ; Boot Mode 47

                dcw             $0000           ; Boot Mode 48
                dcw             $0000           ; Boot Mode 49
                dcw             $0000           ; Boot Mode 50
                dcw             $0000           ; Boot Mode 51
                dcw             $0000           ; Boot Mode 52
```

```
            dcw         $0000       ; Boot Mode 53
            dcw         $0000       ; Boot Mode 54
            dcw         $0000       ; Boot Mode 55

            dcw         $0000       ; Boot Mode 56
            dcw         $0000       ; Boot Mode 57
            dcw         $0000       ; Boot Mode 58
            dcw         $0000       ; Boot Mode 59
            dcw         $0000       ; Boot Mode 60
            dcw         $0000       ; Boot Mode 61
            dcw         $0000       ; Boot Mode 62
            dcw         $0000       ; Boot Mode 63

end_determine_boot_mode


            ;;;INSNOPS    ((determine_boot_mode+$40-end_determine_boot_mode)/2)


; ###
; ### ------ End of Reset Vectors ------
; ###


;#############################################################
;#############################################################
;##
;##   Main Program
;##      - Initialize:
;##           - VBA, vector base address register for interrupts
;##           - Stack Pointer
;##      - Place where Boot Port is selected based on BM pins
;##      - Program up chip clocks correctly
;##
;#############################################################
;#############################################################


            align       16

common_start

;      At this point, the value in d7 indicates which reset vector was taken.
;      Will be stored in a variable in M1 memory once r8 is calculated below.

; --- SW fix for problem - SC1400's SR and EMR come up in unknown state:
; ---    (should be reset, i.e., interrupts enabled)
            move.l      #$00E40000,sr ; software fix
            bmclr       #$000F,emr.l  ; software fix

;      Handle Override of Device ID: Override if not equal to -1
            move.w      #OVERRIDE_DEVID,d3
            cmpeq.w     #ID_FINAL,d3
            nop                                     ; reqd by pipeline: see dependency T.1
            iff         move.l      d3,d2
            bf          <mask_id

; --- SW fix for problem - SC1400's SR[DI] bit comes up in unknown state
```

```
; ---    (should be reset, i.e., interrupts enabled)
            ;ei                                     ; software fix

; --- Get Device ID (Can be overridden)
            move.l      DEVID_REG,d2  ; D2 = Value of DEVNBR pins (in bits [15:12])
            asrr        #12,d2                      ; then shifted to bits [3:0]


;      Continue
mask_id
            and         #$0f,d2,d2    ;     (4-bit value)

; --- Calculate Address - Reserved Section at end of M1 memory
            move.l      #BASE_M1_MEMORY,r8; Initially r8 points to FIRST locat'n.
                                                        ; Later modified to point
to the
                                                        ; first RESERVED location.
            move.l      #M1_MEMSZ_7110,r0; Initialize r0 w/ smallest value.

            cmpeq.w     #ID_7110,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7110,r0

            cmpeq.w     #ID_7112,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7112,r0

            cmpeq.w     #ID_7113,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7113,r0

            cmpeq.w     #ID_7115,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7115,r0

            cmpeq.w     #ID_7116,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7116,r0

            cmpeq.w     #ID_7118,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7118,r0

            cmpeq.w     #ID_7119,d2
            nop                                     ; reqd by pipeline: see dependency T.1
            ift         move.l      #M1_MEMSZ_7119,r0

            nop                                     ; reqd by pipeline
            adda        #-512,r0,r8

; --- Initialize Stack and Vector Base Address(VBA) register

            move.l #BASE_EXCEPTION_TABLE,vba;  init vba


            adda        #OFFSET_M1_STACK,r8,r0
            nop
            tfra        r0,sp                                   ; init ESP
```

```
      ;; stack initialization
              move.l #0,d3
              move.l d3,(r0)

; --- Load Boot Variable RSTSRC
              ; d7 loaded in reset vectors
              move.b       d7,(r8+OFFSET_M1VAR_RSTSRC)

; --- Clear Boot Variable BTERR
              move.w       #$FFFF,d0
              move.w       d0,(r8+OFFSET_M1VAR_BTERR)

; --- Reset Boot Variable NMITYPE
              move.w       #-1,d0
              move.b       d0,(r8+OFFSET_M1VAR_NMITYPE)

; --- Region of NOPs - allows for later patching
              INSNOPS      (32)

; ###
; ### Extract Boot Parameters
; ###
; ### The boot parameters are packed into a 16-bit word.
; ### This is where the different boot info is extracted.
; ###


; --- Read the values of the BM pins and Load BTMD variable
              bsr          determine_boot_mode; Return Value is in d0
              move.w       d0,(r8+OFFSET_M1VAR_EXTBTMD)

; --- Set Up PLL (if not one of the Test Boot modes)
              ; d0 contains boot parameters for routine

      IF 0

      ; Skip PLL if Test Boot Mode
              extractu #4,#12,d0,d7      ; #sz,#offset,src,dst
              asll         #12,d7

              cmpeq.w      #BT_TST0,d7
              nop
              bt           skip_PLL

              cmpeq.w      #BT_TST1,d7
              nop
              bt           skip_PLL

      ENDIF

      ; Setup PLL Otherwise
              bsr          setup_PLL                  ; Programs up PLL and waits for lock

skip_PLL
```

```
; --- Determine Which Boot Loader to Use

; --- NOTE: Algorithm written to take equal time indep of boot mode.
; ---       This is useful for the tester.

              extractu #4,#12,d0,d7      ; #sz,#offset,src,dst
              asll          #12,d7

;       Boot Source = HDI16
              cmpeq.w       #BT_HDI0,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_HDI_0,r0

;       Boot Source = SPI
              cmpeq.w       #BT_SPI0,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_SPI_0,r0

              cmpeq.w       #BT_SPI1,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_SPI_1,r0

;       Force Alternate SPI Pins if LABTEST_SPI is set
              move.w        #LABTEST_SPI,d2
              cmpeq.w       #0,d2
              nop                                     ; reqd by pipeline: see dependency T.1
              iff           move.l    #LOADER_SPI_1,r0

;       Boot Source = I2C
              cmpeq.w       #BT_I2C0,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_I2C_0,r0

;       Boot Source = Test Mode
              cmpeq.w       #BT_TST0,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_TST_0,r0

              cmpeq.w       #BT_TST1,d7
              nop                                     ; reqd by pipeline: see dependency T.1
              ift           move.l    #LOADER_TST_1,r0

              INSNOPS       (20)

; ###
; ### Jump to appropriate Boot Loader
; ###

              jmp           r0
; ###################################
; ###
; ### Setup PLL
; ###
; ###    - Extracts fields from 16-bit boot parameter
; ###
; ###    NOTE: Must NOT destroy the d0 register!
; ###
; ###################################
```

```
setup_PLL

; --- Extract CLKCTRL Fields
              move.l        CLKCTRL,d1

      ; Clear all desired clock fields
              bmclr         #$10ff,d1.h
              bmclr         #$3f30,d1.l

      ; PLLDVF: BootParam[8:6] => CLKCTRL[10:8]
              extractu #3,#6,d0,d7       ; #sz,#offset,src,dst
              insert   #3,#8,d7,d1       ; #sz,#offset,src,dst

      ; PLLMLTF: BootParam[5:0] => CLKCTRL[21:16]
              extractu #6,#0,d0,d7       ; #sz,#offset,src,dst
              insert   #6,#16,d7,d1      ; #sz,#offset,src,dst

      ; RNG: BootParam[9] => CLKCTRL[28]
              extractu #1,#9,d0,d7       ; #sz,#offset,src,dst
              insert   #1,#28,d7,d1      ; #sz,#offset,src,dst

      ; Get CKSEL Field: BootParam[11:10] => d7[1:0]
              extractu #2,#10,d0,d7      ; #sz,#offset,src,dst
              ; not yet placed into CLKCTRL[5:4] ...

      ; These nops allow the addition of future fields.
              nop
              nop
              nop
              nop
              nop
              nop
              nop
              nop
              nop

; --- Skip PLL section if (CKSEL == Bypass)
; ---    (uses result from prev "extractu")
              tsteq         d7
              nop                                   ; reqd by pipeline: see dependency T.1

      ; If CKSEL==00, then rts
              ift           rts

      ; Else continue with code below which turns on the PLL


; --- Enable PLL if Selected

      ; PLLEN: 1 => CLKCTRL[7]
      ; RSTRT: 1 => CLKCTRL[6]
              move.w        #3,d2
              insert        #2,#6,d2,d1             ; #sz,#offset,src,dst

      ; Write value to CLKCTRL
              move.l        d1,CLKCTRL
```

```
; --- Wait for Lock:
; ---    (the wait loop is written in a manner to remove timing variation)

; --- Outer Loop - Polling Loop

Wait_for_pll_lck
;      do   {

      ; Inner Loop - Fixed Delay Loop
             doensh0      #256
               nop
             loopstart0
               nop                      ; in body of loop
               nop                      ; in body of loop
             loopend0

;      } while (not locked);
             move.l CLKCTRL,d2
             bmtsts #$0200,d2.h
             bf  <Wait_for_pll_lck

; --- Change from PLL Bypass to Clocking with the PLL
      ; CKSEL: BootParam[11:10] (already in d7[1:0]) => CLKCTRL[5:4]
             insert   #2,#4,d7,d1       ; #sz,#offset,src,dst

      ; RSTRT: 0 => CLKCTRL[6]
             bmclr        #$0040,d1.l

             move.l       d1,CLKCTRL

             rts


;#############################################################
;#############################################################
;##
;##   HDI Boot
;##   --------
;##      - Examines value of sampled H8BIT pin
;##      - Selects between HDI8 vs HDI16 boot
;##
;##   HDI16 Configuration Used by Boot Program:
;##   -----------------------------------------
;##      - Boot program configures the HDI16 for:
;##           - Non-DMA mode   (cannot use DMA mode)
;##           - HCR[HICR] = 0   ==> part is programmed from StarLite
;##           - Triggers on Access to Host Address 0x7
;##           - to operate in Polled Mode on Chip Side
;##      - Boot program sets up the Host Flags as follows:
;##           - HF4: "finished bit",
;##                  used by SC1400 to tell Ext Host boot is done
;##           - HF7: "sticky bit",
;##                  used by SC1400 to tell Ext Host error occurred
;##           - HF3: "checksum",
;##                  used by Ext Host to tell SC1400 to calculate
;##      - NOTE: Revisions 0 and 0.1 also modified the HF5 bit.
;##                HF5: "Core ready bit",
;##                     used by SC1400 to tell Ext Host boot that it is ready
```

```
;##                      to accept data from external host.
;##              This has been removed from Revision A onwards.
;##      - operates in ______ Mode on External Host
;##      - Via Pins (sampled only at power-on reset):
;##           - HDDS: Single or Dual Data Strobe
;##           - HDSP: Data Strobe Polarity
;##           - xxxx
;##
;##   How the External Host Processor Should Act During the Boot Process
;##   ------------------------------------------------------------------
;##      - User must perform 64-bit transfers in 4 parts a
;##      - User perform 16-bit writes from external host using TX0-3
;##        as follows:
;##            - write Word 1 from Block Structure to TX0 (HA=0x4)
;##            - write Word 2 from Block Structure to TX1 (HA=0x5)
;##            - write Word 2 from Block Structure to TX2 (HA=0x6)
;##            - write Word 3 from Block Structure to TX3 (HA=0x7)
;##
;##            - write Word 4 from Block Structure to TX0 (HA=0x4)
;##            - write Word 5 from Block Structure to TX1 (HA=0x5)
;##            - write Word 6 from Block Structure to TX2 (HA=0x6)
;##            - write Word 7 from Block Structure to TX3 (HA=0x7)
;##              .
;##      - Host flags are available for use as described above
;##           (it is not necessary to use these)
;##
;##   Register Usage - HDI16:
;##   -----------------------
;##      d6      - used to hold the block size
;##      d7      - used for calculating checksum
;##
;##      r3      - used to store data from data blocks to Memories
;##
;#############################################################
;#############################################################


; ###########################################################################
; ###
; ### HDI16 Boot Block Structure:
; ###    - Defn: Entire Block contains:
; ###               - 32-bit Block Size
; ###               - 32-bit Dest Address
; ###               - Set of 16-bit Data Records
; ###               - 32-bits for: Checksum and !Checksum
; ###    - Defn: Block Size:
; ###               - Number of 16-bit Data Records
; ###               - Must satisfy: Block Size 4N+2   (N=integer from 0 to ...)
; ###               - A Block Size of "0" indicates the End of Boot data.
; ###    - Size of entire block is a multiple of 8 bytes (i.e. 64-bits)
; ###
; ###########################################################################

            align        16
            ;org                      p:HDI_BOOT_START
```

```
LOADER_HDI_0
;HDI16_BOOT


; --------------------------Basic HDI configuration-------------------------

; --- Go to HDI16 16-bit Loader



; --- GPIO Setup - GPIO pins used for HDI16 functionality, not GPIO
    ; Set GPIO PortB pins 0-14 into hardware mode
            move.l #$00007fff,d0
            move.l d0,GPIO_PortB_CTL

    ; Set GPIO PortC pins 0-11 into hardware mode
            move.l #$00000fff,d1
            move.l d1,GPIO_PortC_CTL


;-----------BEGIN: INITIAL TEST SECTION-------------

 ; ---Basic configuration of HOST port-------
      ; Enabling of HOST by setting HEN bit in HPCR

            move.w HPCR,d3
            or #$0080,d3.l
            move.w d3,HPCR

      ; Set the HICR bit in HCR so that ICR can define the HM bits
            move.w HCR,d3
            or #$0800,d3.l
            move.w d3,HCR


      ;; ############################

      IF 0
      ; Set the HF5 flag in HCR to let the host know that HDI is ready to accept data
            move.w HCR,d3
            or #$4000,d3.l
            move.w d3,HCR
      ENDIF

      ;; ############################

hdi_loader_after_setup

 ; --- Go to HDI16 or HDI8 Loader (depending on HPCR[8])
      ; get the 8bit bit from hpcr ( which is in d3 )
            move.w       HPCR,d3
            bmtsts       #$0040,d3.l

      ; if (8bit) goto load_8bit
            bt           <load_8bit

      ; else goto load_16bit
            bra          load_16bit
```

```
;#########################################################
;##
;##   HDI8 Boot - 8-bit Host Data Bus
;##      - Executes code specific to HDI8 boot
;##
;#########################################################


load_8bit

; --- Clear Checksum (stored in d7)
              move.l #0,d7

; --- Get first 2 words from Block Structure:
;        - Word 1: Block_Size[31:16]
;        - Word 2: Block_Size[15:00]

              bsr load_from_fifo       ; (returns 64-bits into d4:d5)
                                       ; (d4=0,d5=32 bits)

      ;; check if the finished bit is set ,if it is set clear it
      ;; and the sticky bit .( it means there was an error and
      ;; the blocks are being loaded for the second time
              move.w HCR,d6
              bmtsts.w #$8000,d6.l
              bf <continue_loading_8

              ; Clear Finished Bit, HCR[HF4], and Sticky Bit, HCR[HF7]
              move.w HCR,d6
              and #$6fff,d6.l          ; clears HCR[HF4] and HCR[HF7]
              move.w d6,HCR

continue_loading_8
              eor d5,d7   ;; checksum is calculated  on size as d5 contains block size
              move.l d5,d6
      ; if (size == 0) it means the last block was loaded
      ; so goto end_of_loading
              tsteq  d6
              bt     end_of_loading
      ; load the address
              bsr load_from_fifo
              move.l d5,r3
              eor d5,d7 ;checksum is calculated on address
load_loop_8
      ; load data word
              bsr load_from_fifo
              move.l d5,(r3)
      ; add 4 bytes to the address
              adda #$4,r3

      ; CALCULATE_CHECKSUM d7 = d7 ^ d5
              eor d5,d7

      ; substruct 2 words from the size
              sub #$2,d6
      ; if ( size | = 0) go to load_loop_8
              tsteq  d6
              bf     <load_loop_8
```

```
;;------Calculating the checksum & ~checksum for the block

      ; get the checksum into d7.l
      ; d2 = (0xffff0000 & d7)>>16
             extractu #16,#16,d7,d2

      ; d2 = d2 & 0x0000ffff
             and #$0000ffff,d2,d2

      ; d7 = d7 & 0x0000ffff
             and #$0000ffff,d7,d7

      ; d7 = d7 ^ d2
             eor d2,d7

      ; d2 = (~d7 & 0x0000ffff) = ~checksum
             not d7,d2
             and #$0000ffff,d2,d2

;;------Getting the Loaded checksum & ~checksum

      ; load the checksum ,~checksum
             bsr load_from_fifo

      ; get ~checksum into d4
             extractu #16,#16,d5,d4

      ; delete the ~checksum from d5
             and #$0000ffff,d5,d5

      ; if ( checksum_loaded |= Checksum_calculated ) goto set sticky bit
             cmpeq d5,d7
             nop                          ; Inserted due to T.1 from StarCore spec:
                                          ;    IFc not allowed to follow a group
                                          ;    containing a T bit modification
             iff bsr set_sticky_bit
      ; if ( ~checksum_loaded |= ~Checksum_calculated ) goto set sticky bit
             and #$0000ffff,d4,d4
             cmpeq d4,d2                  ; d4,d7 contain checksum
             nop                          ; Inserted due to T.1 from StarCore spec:
                                          ;    IFc not allowed to follow a group
                                          ;    containing a T bit modification
             iff bsr set_sticky_bit
      ; goto load the next block
             bra load_8bit


end_of_loading

      ; load the destination address into r3
             bsr load_from_fifo
             move.l d5,r3

             eor d5,d7

             bsr load_from_fifo  ; reads checksum to d5
```

```
        ; get the checksum into d7.l
        ; d2 = (0xffff0000 & d7)>>16
               extractu #16,#16,d7,d2
        ; d2 = d2 & 0x0000ffff
               and #$0000ffff,d2,d2
        ; d7 = d7 & 0x0000ffff
               and #$0000ffff,d7,d7
        ; d7 = d7 ^ d2
               eor d2,d7
        ; d2 = (~d7 & 0x0000ffff) = ~checksum
               not d7,d2
               and #$0000ffff,d2,d2

        ; get ~checksum into d4
               extractu #16,#16,d5,d4
        ; delete the ~checksum from d5
               and #$0000ffff,d5,d5
        ; if ( checksum_loaded |= Checksum_calculated ) goto set sticky bit
               cmpeq d5,d7
               nop                          ; Inserted due to T.1 from StarCore spec:
                                            ;    IFc not allowed to follow a group
                                            ;    containing a T bit modification
               iff bsr set_sticky_bit
        ; clean d4.h
               and #$0000ffff,d4,d4
        ; if ( ~checksum_loaded |= ~Checksum_calculated ) goto set sticky bit
               cmpeq d4,d2
               nop                          ; Inserted due to T.1 from StarCore spec:
                                            ;    IFc not allowed to follow a group
                                            ;    containing a T bit modification
               iff bsr set_sticky_bit


        ; set hf4 bit in HCR to show that loading is finished
               move.w HCR,d6
               or #$8000,d6.l
               move.w d6,HCR

        ; check that the "check checksum" bit is set ( hf3 in HSR  (HSR[12) )
               move.w HSR,d6
               and #$00001000,d6,d6

        ; check if the sticky bit is set ( hf7 in HCR(HCR[12]) )
               move.w HCR,d4
               and #$00001000,d4,d4

        ; if both of the flags are set start the loading again
               and d4,d6
               tsteq d6
               bf hdi_loader_after_setup
               nop

; --- Jump to Starting Address of Booted User Code (stored in r3)
; ---    (exits boot program)

               jmp  r3
```

```
;#########################################################
;##
;##   HDI16 Boot - 16-bit Host Data Bus
;##      - Executes code specific to HDI16 boot
;##
;#########################################################


; ###
; ### Receive and Store:
; ###    - Receives and stores last two Boot Data Entries
; ###    - Receives Expected Checksums and perform final checksum calculations
; ###
; ###    ***************************************
; ###    **  Not the Entry Point.  See Below.  **
; ###    ***************************************
; ###

load_last_2_16bit_data

             ; If the size left to load is only 2 words ( 4 bytes)
             ;    and the mode is 16bit so every load action is of 4 words,
             ;
             ; Then the load action will load the last two word of the data and checksum
             ;      and the ~checksum which are the very last two word of
             ;      a block.so it is a special case .

; --- Get LAST 4 words from Record:
;        Word 1: Boot_Data_Entry[15:00]       => d4[31:16]
;        Word 2: Boot_Data_Entry[15:00]       => d4[15:00]
;        Word 3: Expected_Checksum_Inv[15:00] => d5[31:16]
;        Word 4: Expected_Checksum[15:00]     => d5[15:00]

             bsr load_from_fifo; (returns 64-bits into d4:d5)

      ; move the last 2 words to Load address (found in r3)
             move.l d4,(r3)             ; Two 16 bit words are moved to the memory.
             adda #$4,r3


; --- Last Two Boot Data Entries of Block are combined w/ Running Checksum (d7)
; ---    (note that Running_Checksum is calculated using 32-bit XORs)
             eor d4,d7

; --- At this point, the Running Checksum is complete.

; ---
; --- Calculate Two Final Checksums:
; ---    - Final_Checksum[15:0]
; ---    - Final_Checksum_Inv[15:0]
; ---
; ---   Final_Checksum[15:0] = Running_Checksum[31:16] ^ Running_Checksum[15:0]
; ---   Final_Checksum_Inv[15:0] = One's complement of Final_Checksum[15:0]
; ---
; ---    During the loading process the checksum is calculated by 32-bit XORs.
; ---    At the end of the block, the upper and lower 16-bit words in the
; ---    Running checksum are xored with each other to get the Final checksum.
; ---
```

```
      ; Running_Checksum[31:16] => d2[15:0]    (upper 16-bits are cleared)
             extractu #16,#16,d7,d2
      ; d2 = d2 & 0x0000ffff
             and #$0000ffff,d2,d2

      ; Running_Checksum[15:00] => d7[15:0]    (upper 16-bits are cleared)
      ; d7 = d7 & 0x0000ffff
             and #$0000ffff,d7,d7

      ; Running_Checksum[31:16] ^ Running_Checksum[15:0] => Final_Checksum[15:0]
      ; d7 = d7 ^ d2 = checksum
             eor d2,d7

      ; One's Complement of Final_Checksum[15:0] => Final_Checksum_Inv[15:0]
      ; d2 = (~d7 & 0x0000ffff) = ~checksum
             not d7,d2
             and #$0000ffff,d2,d2

; ---
; --- Checksum Calculation:
; ---    - Compare Expected Checksums against Final Checksums
; ---

      ; Expected_Checksum_Inv[15:00] => d4[15:0]    (upper 16-bits are cleared)
             extractu #16,#16,d5,d4
             and #$0000ffff,d4,d4

      ; Expected_Checksum[15:00] => d5[15:0]    (upper 16-bits are cleared)
             and #$0000ffff,d5,d5

; ---  If ( Expected_Checksum != Final_Checksum )
; ---     Set sticky bit

             cmpeq d5,d7
             nop                                        ; reqd by pipeline: see dependency T.1
             iff bsr set_sticky_bit

; ---  If ( Expected_Checksum_Inv != Final_Checksum_Inv )
; ---     Set sticky bit

             cmpeq d4,d2
             nop                                        ; reqd by pipeline: see dependency T.1
             iff bsr set_sticky_bit

; ---  Fall through into code below

      ; goto loading the next block
      ; goto load_16bit

; ###
; ### HDI16's 16-Bit Loader (Start of Loader)
; ###
; ###    *****************
; ###    ** ENTRY POINT **
; ###    *****************
; ###
```

```
; ###    Reached this point either by:
; ###       - Entry into the HDI16 loader or
; ###       - When finishing processing of the previous Record and about to
; ###         begin processing of a new Record.
; ###

load_16bit

; --- Clear Checksum (stored in d7)
             move.l #0,d7

; --- Get First 4 words from Record:
;        Word 1: Block_Size[31:16] => d4[31:16]
;        Word 2: Block_Size[15:00] => d4[15:00]
;        Word 3: Dest_Addr[31:16]  => d5[31:16]
;        Word 4: Dest_Addr[15:00]  => d5[15:00]

             bsr load_from_fifo; (returns 64-bits into d4:d5)

; --- Block Size (32-bits) => "d6"
      ; move the size into d6
             move.l d4,d6

; --- Dest Address (32-bits) => "r3"
      ; move the address into r3
             move.l d5,r3

; --- First Four Words of the Block are combined w/ Running Checksum (d7)
      ; Checksum should be calculated on data and address also
             eor d4,d7
             eor d5,d7

; --- If (Finished Bit is Set)   {
      ; Clear Finished Bit and Sticky Bit.
      ; (It means there was an error and blocks are being loaded a second time)

      ; Check Finished Bit
             move.w HCR,d4
             bmtsts.w #$8000,d4.l
             bf <continue_loading_16    ; skip over clearing of HF4 and HF7

      ; Clear Finished Bit, HCR[HF4], and Sticky Bit, HCR[HF7]
             move.w HCR,d4
             and #$6fff,d4.l                         ; clears HCR[HF4] and HCR[HF7]
             move.w d4,HCR
; --- }


continue_loading_16

; ###
; ### Check if Final Record
; ###
; ###    The Last Record is found when its Record Size is 0.
; ###    If so, then there are no more Boot Data Entries to process.
; ###    There are only four more 16-bit values in the Final Record
; ###    which are handled in the exit code.
; ###
```

```
; ---
; --- If (size == 0)
; ---    Then goto end_of_loading_16

             tsteq  d6
             bt <end_of_loading_16

; ###
; ### Otherwise,
; ###    Receive and Store all but Last Two Boot Data Entries
; ###


; --- while (Block_Size > 2)   {

load_loop_16

; --- Special Case: Block Size = 2   (last 2 words of data before 2 wd checksum)
; ---
; ---  If (size ==2 ) it is a special case so
; ---     Goto load_last_2_word

             move.l #$00000002,d4
             cmpeq d4,d6
             bt   load_last_2_16bit_data

; --- Normal Case: Process 4 words (64-bits, 1 HORX FIFO entry)
; ---


; --- Get 4 Boot Data Entries from Record:
;        Word 1: Boot_Data_Entry[15:00]  => d4[31:16]
;        Word 2: Boot_Data_Entry[15:00]  => d4[15:00]
;        Word 3: Boot_Data_Entry[15:00]  => d5[31:16]
;        Word 4: Boot_Data_Entry[15:00]  => d5[15:00]

             bsr load_from_fifo; (returns 64-bits into d4:d5)

; --- Store First Two Boot Data Entries:
      ; load the first 2 data words (4 bytes) to the address
             move.l d4,(r3)
      ; increment the address by 4 bytes
             adda #$4,r3               ; 4 is added because of 32 bit is loaded
                                     ; from d4 and the memory is byte addresable.

; --- Store Next Two Boot Data Entries:
      ; load the second 2 data words (4 bytes) to the address
             move.l d5,(r3)
      ; increment the address by 4 bytes
             adda #$4,r3

      ; CALCULATE_CHECKSUM on all 4 Boot Data Entries
             eor d4,d7
             eor d5,d7

; --- Decrease Block Size by 4 words
             sub #$4,d6                ; 4 is deducted because 64 bit is loaded
                                                  ; from HORX at a
```

```
                                        ; and the block size is given in 16 bits.
; --- Back to Top of Loop
             bra load_loop_16

; --- }

; --- Upon exiting this While Loop
; ---    goto load_last_2_16bit_data (located above).


; ###
; ### Process Last 4 Words of Final HDI Record
; ###

end_of_loading_16

; --- Get LAST 4 words from Record:
;        Word 1: Expected_Checksum_Inv[15:00] => d4[31:16]
;        Word 2: Expected_Checksum[15:00]     => d4[15:00]
;        Word 3: $0000                        => d5[31:16]
;        Word 4: $0000                        => d5[15:00]

             bsr load_from_fifo; (returns 64-bits into d4:d5)

; --- Checksum Calculations on Final Boot Record
      ; get the checksum into d7.l
      ; d2 = (0xffff0000 & d7)>>16
             extractu #16,#16,d7,d2

      ; d2 = d2 & 0x0000ffff
             and #$0000ffff,d2,d2

      ; d7 = d7 & 0x0000ffff
             and #$0000ffff,d7,d7

      ; d7 = d7 ^ d2
             eor d2,d7

      ; d2 = (~d7 & 0x0000ffff) = ~checksum
             not d7,d2
             and #$0000ffff,d2,d2



      ; get ~checksum into d5
             extractu #16,#16,d4,d5

      ; delete the ~checksum from d4 so that it only contains the checksum
             and #$0000ffff,d4,d4

      ; if ( Expected_Checksum != Checksum_calculated ) Set sticky bit
             cmpeq d4,d7
             nop                        ; Inserted due to T.1 from StarCore spec:
                                        ;    IFc not allowed to follow a group
                                        ;    containing a T bit modification
             iff bsr set_sticky_bit
      ; clean d5.h
             and #$0000ffff,d5,d5
```

```
      ; if ( Expected_Checksum_Inv != Checksum_calculated_Inv ) Set sticky bit
             cmpeq d5,d2
             nop                          ; Inserted due to T.1 from StarCore spec:
                                          ;    IFc not allowed to follow a group
                                          ;    containing a T bit modification
             iff bsr set_sticky_bit

; ###
; ### Loading of All Boot Records Completed:
; ###    if (no errors)
; ###       Exit Boot Loader and go to User's Starting Address
; ###    if (errors but no reload on error)
; ###       Exit Boot Loader and go to User's Starting Address
; ###    if (errors and reload on error)
; ###       Go back to the beginning of the HDI Boot Loader
; ###

      ; set HCR[HF4] to show that loading is finished
             move.w HCR,d6
             or #$00008000,d6.l
             move.w d6,HCR

      ; check if the user data should be reloaded if checksums don't match
      ; ( HSR[HF3] which is bit 12 )
             move.w HSR,d6
             and #$00001000,d6,d6

      ; check if the sticky bit is set ( HCR[HF7] which is bit 12 )
             move.w HCR,d4
             and #$00001000,d4,d4

      ; if both of the flags are set start the loading again
             and d4,d6
             tsteq d6
             bf hdi_loader_after_setup



; --- Jump to Starting Address of Booted User Code (stored in r3)
; ---    (exits boot program)

             jmp  r3


;#########################################################
;#########################################################
;##
;##   Supporting Routines - HDI Boot:
;##      - Called when booting through HDI port for:
;##           - Normal HDI  boot modes
;##           - Test boot modes:
;##                - Test boot modes use the HDI16 port (16-pin data bus)
;##                - previously the Test boot modes had their own routines
;##
;#########################################################
;#########################################################
```

```
; ###
; ### Load From FIFO
; ###    - Polls until HDI16 Receive FIFO is not empty
; ###    - Load 64-bit value into d4:d5
; ###

load_from_fifo

; --- Poll until Receive FIFO Not Empty (HRFNE)

load_no_wd
      ; if the host is empty wait for it to fill
              move.w HSR,d4
              bmtsts.w #$0001,d4.l;  Checking HRFNE bit
              bf    <load_no_wd

; --- Get 64-bit Entry from Receive FIFO => d4:d5
      ;; the receive FIFO is not empty so load 64bit  from it

              move.l #HORX,r0
              nop
              move.2l (r0),d4:d5

              rts

; ###
; ### Set Sticky Bit
; ###    - xxx
; ###    - xxx
; ###

set_sticky_bit
      ; Set HCR[HF7]
              move.w HCR,d6
              or #$1000,d6.l
              move.w d6,HCR

              rts

;#########################################################
;#########################################################
;##
;##   I2C Boot
;##
;##
;#########################################################
;#########################################################

; ###
; ### Defines
; ###

I2C_SLAVE_ADDRequ   $00A0

I2C_WR                    equ   $0      ; Placed into R/W bit of 1st transmitted byte
I2C_RD                    equ   $1      ; Placed into R/W bit of 1st transmitted byte
```

```
            align       16
            ;org        p:I2C_BOOT_START


LOADER_I2C_0
;I2C_BOOT


; ###
; ### Initial I2C Setup
; ###
; ###    - Program GPIO's Port A for I2C pin functionality
; ###    - Set Clock Divider
; ###    - Enable I2C
; ###


; --- Setup GPIO's Port A for SDA and SCL functionality

    ; Set GPIO PortA pin 14(SDA),15(SCL) into hardware mode for CLKO pin
            move.l #$0000c000,d0
            move.l d0,GPIO_PortA_CTL

; --- Set Clock Divider to 128
; ---    - IFDR=$0B => Divider of 128
; ---    - For CLKIN = 100 MHZ during booting,
; ---         => IPBus Clock = 50.0 MHz
; ---         => I2C   Clock = 50.0 MHz / 128 = 390.625 KHZ
; ---    - For CLKIN = 25 MHZ during booting,
; ---         => IPBus Clock = 12.5 MHz
; ---         => I2C   Clock = 12.5 MHz / 128 = 97.66 KHZ

            move.w #$000B,d0
            move.w d0,IFDR

;------ Enable I2C Peripheral
      ; Set I2CR[IEN] bit

            move.w I2CR,d3
            or #$0080,d3.l
            move.w d3,I2CR

; ###
; ### Generate First Start
; ###

;------ Test that Bus is not busy  before generating a start.

bus_busy
            move.w I2SR,d10
            bmtsts.w #$0020,d10.l;  waiting for  IBB bit to be 0
            bt <bus_busy

; --- Generating the very first start of the I2C
      ; --Setting other bits of I2C
      ;      (IEN is already set)
      ; ## IIEN=1
      ; ## MSTA=1, 711x devices is in I2C's master mode
```

```
      ; ## MTX=1,transmiit,for initial slave address transmitting
      ; ## TXAK=0,Acknowledge transfer
      ; ## RSTA=0,No repeat start.Since first data read and the address

              move.w #$00f0,d3
              move.w d3,I2CR

;------ wait for the start to "kick in"

bus_not_busy
              move.w I2SR,d10
              bmtsts.w #$0020,d10.l; waiting for  IBB bit to be 1
              bf <bus_not_busy

; ###
; ### First Dummy Write
; ###
; ###    - Set the EEPROM address counter to 0
; ###

; ### I2C Write Access:

; --- Send Calling Address in first byte:
; ---    Master => Slave
; ---    [7:1]   = 7-bit Calling Address of I2C Slave ($A0)
; ---    [0]     = R/W   (0=WR, 1=RD)

              move.w #((I2C_SLAVE_ADDR)+I2C_WR),d3
              move.w d3,I2DR

              bsr   wait_for_interrupt

; --- Send 2 Data Bytes:
; ---    These 2 bytes contain the Starting Address of the
; ---    First Record in the I2C EEPROM:
; ---       1st Byte = [15:8] = $00
; ---       2nd Byte = [07:0] = $00

              move.l #$0,d0                          ; I2C EEPROM's Starting Address

              move.l d0,I2DR
              bsr   wait_for_interrupt

              move.l d0,I2DR
              bsr   wait_for_interrupt

; ###
; ### Initiate Reads from EEPROM Device
; ###

;------Repeat start

;----Initiate the repeat start for sending EEPROM
     ;Device ID with read bit turned on

              move.w I2CR,d3
              or   #$0004,d3.l; repeat starting RSTA=1
              move.w d3,I2CR             ;
```

```
            nop
            nop
            nop

; ### I2C Read Access:

; --- Send Calling Address in first byte:
; ---    Master => Slave
; ---    [7:1]   = 7-bit Calling Address of I2C Slave ($A0)
; ---    [0]     = R/W   (0=WR, 1=RD)

            move.w #((I2C_SLAVE_ADDR)+I2C_RD),d3
            move.w d3,I2DR

            bsr   wait_for_interrupt


; --- Turn off the repeat start & put the I2C master in receive mode
; --- to receive data from EEPROM slave
      ;;  I2CR[MTX]=0 I2CR[RSTA]=0
            move.w I2CR,d3
            and #$ffeb,d3.l
            move.w d3,I2CR

; --- Receive 1 Data Byte:
; ---    This is a dummy read.

            move.w I2DR,d7

; ###
; ### Block Read Loop   (Outer Loop)
; ###
; ###    Each iteration processes one Block (Record).
; ###    (the word Record and Block are used with the same meaning).
; ###
; ###    There are two different locations at the top of the Read Loop:
; ###       - read_next_block
; ###       - reload_block
; ###
; ###    These are almost identical except that read_next_block:
; ###       - resets the error counter (d9)
; ###       - provides a new starting address for data (d6)
; ###    In contrast, reload_block does neither of these two tasks
; ###

; --- Initialize Loop Variables before entering loop:
; ---
; ---    d6  <= $0      Record Start Address in I2C EEPROM
; ---                      - the value $0 indicates "sequential loading"
; ---    d14 <= $0      Next Record Address  in I2C EEPROM
; ---    ---------      --------------------------------
; ---    d12 <= $0      Record Size            --unchged in loop
; ---    d15 <= $0      Number of Remaining Bytes in Record
; ---                       Used as Number of Remaining Bytes in the
; ---                       loops which read the Boot Data Entries
; ---    d13 <= $0      Number of bytes read so far
; ---                       It will be used during reloading
; ---                       of a consecutive block by
; ---                       subtracting d12 from d13.
```

```
; ---
; ---                        Reload Start Address
; ---                           = d13 - (Record Size + first four word)
; ---                           = d13 - (d12 + 4*2)
; ---
             move.l #$0,d6
             move.l #$0,d14
             move.l #$0,d15
             move.l #$0,d12
             move.l #$0,d13

; --- Other Loop Variables before entering loop:
; ---
; ---    d8      Checksum Calculation Enable
; ---               (extracted from bit 15 of Record Size)
; ---    r3      Load Address:
; ---               Used for storing 16-bit Boot Data Entries received
; ---               through the I2C.  Updated by 2 for each entry.


; --- Top of Loop (if no error)

read_next_block

; --- Update Loop Variables:
; ---
; ---    d9  <= $0      Number of Checksum errors found = 0
; ---    d6  <= d14     Record Start Address <= Next Record Address

             move.l #$0,d9
             move.l d14,d6

; --- Top of Loop (if error occurred)

reload_block

; --- Update Loop Variables:
; ---
; ---    d7  <= $0      Running Checksum

             move.l #$0,d7

; ###
; ### Select which path to process:
; ###    - Sequential Reload
; ###    - NonSequential Load or Reload
; ###    - Sequential Load
; ###
; ###    Decision based on values in d6, d9 registers:
; ###
; ###             d6 |  d9 | Action Performed
; ###           -----------------------------
; ###      1)     0  |  0  | Sequential    Load
; ###      2)     0  |  1  | Sequential    ReLoad   (i.e. one checksum error)
; ###      3)     1  |  0  | NonSequential Load
; ###      4)     1  |  1  | NonSequential ReLoad   (i.e. one checksum error)
; ###           -----------------------------
; ###
```

```
; --- If d6=0, sequential loading or reloading.
; --- If d6=1, nonsequential loading or reloading
             tsteq d6
             bf <nonsequential_load_or_reload

sequential_load_or_reload

; --- If d9=0 at this point, sequential loading
; --- If d9=1 at this point, sequential reloading

             tsteq d9
             bt sequential_loading

; ###
; ### Reload Record - One Checksum Error occurred during Sequential Record
; ###
; ### Executed on:
; ###    - Sequential Reload Only
; ###

sequential_reloading

             tsteq d12                  ; if last block d12=0

             move.l #$10,d0             ; d0=size of the last block

             move.l #$8,d2
             add d2,d12,d1         ; d1 =the the size of any non last block

; --- If (Last Record Size == 0)
; ---    // Then reloading "Last Record" - Special Case
; ---       Record_Start_Addr = Number_of_Bytes - d0;

              ift sub d0,d13,d6

; --- Else
; ---    // Normal Reload
; ---       Record_Start_Addr = Number_of_Bytes - d1;

              iff sub d1,d13,d6

; ###
; ### Executed on:
; ###    - Sequential    Reload
; ###    - NonSequential Load
; ###    - NonSequential Reload
; ###

nonsequential_load_or_reload

; ---
; --- Dummy write to EEPROM to update address counter to block address
; ---

; --- Initiate the repeat start and transmit mode for sending
; --- EEPROM Device ID followed by two byte of read memory address

             move.w I2CR,d3
```

```
              or   #$0014,d3.l; repeat starting RSTA=1 & MTX=1
              move.w d3,I2CR             ;

              nop
              nop
              nop

; ### I2C Write Access:

; --- Send Calling Address in first byte:
; ---    Master => Slave
; ---    [7:1]   = 7-bit Calling Address of I2C Slave ($A0)
; ---    [0]     = R/W   (0=WR, 1=RD)

              move.w #((I2C_SLAVE_ADDR)+I2C_WR),d3
              move.w d3,I2DR

              bsr   wait_for_interrupt

; --- stop repeat start by putting RSTA=0

              move.w I2CR,d3
              and   #$fffb,d3.l; stop repeat starting
              move.w d3,I2CR

              nop
              nop
              nop

; --- Send 2 Data Bytes:
; ---    These 2 bytes contain the Record's Starting Address in the EEPROM:
; ---       1st Byte = d6[15:8]
; ---       2nd Byte = d6[07:0]
; ---    This can be one of two values:
; ---       - Record Start Address for Current Record when boot failure occurred
; ---       - Record Start Address for Next Record, obtained from the
; ---            Last Record's "Next Record Address" field

              move.l d6,d11
              lsrr #8,d11

              move.w d11,I2DR
              bsr   wait_for_interrupt

              extractu #$8,#$0,d6,d5

              move.w d5,I2DR
              bsr   wait_for_interrupt

; ---
; --- End of Dummy write to EEPROM
; ---

; --- Initiate the repeat start for sending EEPROM Device ID again

              move.w I2CR,d3
              or   #$0004,d3.l; repeat starting
              move.w d3,I2CR
```

```
            nop
            nop
            nop

; ### I2C Read Access:

; --- Send Calling Address in first byte:
; ---    Master => Slave
; ---    [7:1]   = 7-bit Calling Address of I2C Slave ($A0)
; ---    [0]     = R/W   (0=WR, 1=RD)

            move.w #((I2C_SLAVE_ADDR)+I2C_RD),d3
            move.w d3,I2DR

            bsr   wait_for_interrupt

; --- stop repeat start and put I2C in receive mode again for read
            move.w I2CR,d3
            and   #$ffeb,d3.l; RSTA=0;MTX=0;
            move.w d3,I2CR

            nop
            nop
            nop

; --- Receive 1+N Data Bytes:
; ---    - First byte is a dummy read.
; ---    - All other bytes in the record are then read using the code below:
; ---         - Block Address
; ---         - Next Block Address
; ---         - Load Address
; ---         - Boot Data Entries

            move.w I2DR,d7

; ###
; ### Get First 4 Record Fields:
; ###    - Block Size            15-bits (MSB is Checksum Comparison Enable)
; ###    - Next Block Address    16-bits
; ###    - Load Address          32-bits
; ###
; ### This code is executed for all of the following:
; ###    - Sequential    Load
; ###    - Sequential    Reload
; ###    - NonSequential Load
; ###    - NonSequential Reload
; ###

sequential_loading

; ---
; --- Read Block Size (15 bits, where MSB is checksum comparison enable)
; ---

; --- Get 16-bit Value
            bsr   wait_for_interrupt    ; bit[15:8] of block size
            move.w I2DR,d0
            asll #8,d0
```

```
            bsr   wait_for_interrupt    ; bit[7:0] of block size
            move.w I2DR,d1

            add d0,d1,d15                  ; bit[15:0] of block size

            eor d15,d7

; --- Extract MSB which is Checksum Calculation Enable
            extractu #$1,#$f,d15,d8     ; Isolate the checksum enable bit
                                        ; from d15 and put it into d8
                                        ; to be used later

            and #$7fff,d15,d15          ; set the CSE bit to zero in d15
                                        ; to get the real block size


; --- Record Size stored in d12
            move.l d15,d12                 ; (will be subtracted during reload)


;---- Tracking how many bytes loaded so far
            move.l #$8,d0
            move.l #$10,d1

            tsteq d15                  ; Testing if block size=0 implying last block
            nop
            nop                                        ; reqd by pipeline: see dependency T.1

            ift add d1,d15,d13; if last block(d15=0) add 16
                                                       ; (Total number bytes in last block) to
d13

            iff add d0,d15,d13; else add block size+8 to d13

; ---
; --- Read Next Block Address (16-bits)
; ---

            bsr   wait_for_interrupt; bit[15:8] of next block addr
            move.w I2DR,d0
            asll #8,d0

            bsr   wait_for_interrupt; bit[7:0] of next block addr
            move.w I2DR,d1

            add d0,d1,d14                   ; bit[15:0] of next block addr

            eor d14,d7

; ---
; --- Read Load Address (32-bits)
; ---

            bsr   wait_for_interrupt;Reading Bit[31:24] of source
            move.w I2DR,d0             ;program destination address
            asll   #24,d0

            bsr   wait_for_interrupt;Reading Bit[23:16] of source
```

```
              move.w I2DR,d1            ;program destination address
              asll   #16,d1


              add d0,d1,d2                              ;Put the Bit[31:16] in a data
                                                                ;register. Bit[15:0] of
data reg=0


              add d0,d1,d3
              lsrr #16,d3                               ;Right align bit [31:16] for checksum
              eor d3,d7

              bsr   wait_for_interrupt;Reading Bit[15:8] of source
              move.w I2DR,d0            ;program destination address
              asll   #8,d0


              bsr   wait_for_interrupt;Reading Bit[7:0] of source
              move.w I2DR,d1            ;program destination address



              add d0,d1,d3              ;Put the Bit[15:0] in a data
                                                ;register. Bit[31:16] of data reg=0

              eor d3,d7                ;xoring bit[15:0]



              add d2,d3,d4       ;Create the 32 bit source program
                                 ;destination address


              move.l d4,r3       ; Final 32-bit Load Address saved in "r3" and
                                 ;    will be used for storing all Boot Data Entries

; ###
; ### Determine if this is the "End Block"
; ###    - i.e., the Final Boot Record
; ###    - indicated by a Block with a size of "0"
; ###

 ; --- If (Block_Size == 0)
 ; ---    Then   {
 ; ---       // Final I2C Boot Record
 ; ---          goto finish_final_record
 ; ---    }
 ; ---    Else   {
 ; ---       Read in Block Data via one of two different loops:
 ; ---          Read Block Data with Checksum Calculation
 ; ---          Read Block Data w/o  Checksum Calculation
 ; ---    }

              tsteq  d15                ; Block Size in d15
              bt  finish_final_record

; ---  For above "Else" case, Goto One of Two "Read Data Byte" Loops:
```

```
; ---     If Checksum enabled
; ---        goto keep_reading_db_Csum
; ---     If Checksum not enabled
; ---        goto keep_reading_db_NoCsum

             tsteq d8
             bt keep_reading_db_NoCsum

; ###
; ### Keep Reading Data Bytes Loop (w/ Checksum Calculation Enabled)
; ###
; ###    - This loop reads in all the Boot Data Entries in the Block
; ###

; --- while (FOREVER)   {

keep_reading_db_Csum

; --- if (Number_of_Remaining_Bytes == 4)   {
; ---    // Loop Exit found in this code

             move.l #$0004,d0          ;Testing if only chksum &
             cmpeq d0,d15                          ;~chksum left to be loaded
             bf   <load_source_program

; --- Get Expected Checksum Fields from Record

; ---  load checksum in d2
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1


             add d0,d1,d2

; ---  load ~checksum in d3
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1

             add d0,d1,d3

; --- Perform Checksum Calculation
      ; d11 = (~d7 & 0x0000ffff) = calculated ~checksum
             not d7,d11
             and #$0000ffff,d11,d11

             cmpeq d7,d2
             bt <check_Csumbar

; --- Error Detected on Checksum field
             add #1,d9                 ; Number_Checksum_Errors++;
```

```
                move.l #$2,d5               ; If (Number_Checksum_Errors == 2)
                cmpeq d5,d9                 ;    goto I2C_Stop_Boot
                bt I2C_Stop_Boot

                bra reload_block; Else goto reload_block

; --- Perform Checksum_Inv Calculation

check_Csumbar
                cmpeq d11,d3
                bt read_next_block; No Error Detected on either calculation,
                                                        ;    goto Top of Loop

; --- Error Detected on Checksum_Inv field
                add #1,d9                   ; Number_Checksum_Errors++;

                move.l #$2,d5               ; If (Number_Checksum_Errors == 2)
                cmpeq d5,d9                 ;    goto I2C_Stop_Boot
                bt I2C_Stop_Boot

                bra reload_block; Else goto reload_block

; --- }      // End If


load_source_program

; --- Get Boot Data Entry from Record

;     Get upper 8-bits
                bsr   wait_for_interrupt
                move.w I2DR,d0
                asll #8,d0

;     Get lower 8-bits
                bsr   wait_for_interrupt
                move.w I2DR,d1

;     Combine into 16-bit value and XOR into checksum
                add d0,d1,d2
                eor d2,d7

;     Write 16-bit Boot Data Entry to current value of the Load Address
                move.w d2,(r3)+

;     Update Number of Remaining Bytes
                sub #$2,d15

                bra keep_reading_db_Csum

; --- }      // End While FOREVER Loop


; ###
; ### Keep Reading Data Bytes Loop (w/o Checksum Calculation Enabled)
; ###
; ###    - This loop reads in all the Boot Data Entries in the Block
; ###
```

```
; --- while (FOREVER)   {

keep_reading_db_NoCsum

; --- if (Number_of_Remaining_Bytes == 4)   {
; ---    // Loop Exit found in this code

             move.l #$0004,d0
             cmpeq d0,d15
             bf   Load_source_program

Load_Csum

; --- Read Expected Checksum and Expected Checksum Inv but unused

             bsr   wait_for_interrupt
             move.w I2DR,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1


             bsr   wait_for_interrupt
             move.w I2DR,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1

             bra read_next_block

; --- }      // End If


Load_source_program

      ; Get Boot_Data_Entry[15:8]
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

      ; Get Boot_Data_Entry[7:0]
             bsr   wait_for_interrupt
             move.w I2DR,d1

      ; Combine into 16-bit value
             add d0,d1,d2

;     Write 16-bit Boot Data Entry to current value of the Load Address
             move.w d2,(r3)+

;     Update Number of Remaining Bytes
             sub #$2,d15

             bra keep_reading_db_NoCsum

; --- }      // End While FOREVER Loop
```

```
; ###
; ### Finish Final Record
; ###
; ### Get Remaining Records in Last Boot Record:
; ###    - $0000
; ###    - $0000
; ###    - Expected Checksum
; ###    - Expected Checksum Inv
; ###

finish_final_record


; ---  Loading 0x0000
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1

             add d0,d1,d2

             eor d2,d7

; ---  Loading 0x0000
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1

             add d0,d1,d2

             eor d2,d7

; ---  load checksum in d2
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1

             add d0,d1,d2

; ---  load ~checksum in d3
             bsr   wait_for_interrupt
             move.w I2DR,d0
             asll #8,d0

             bsr   wait_for_interrupt
             move.w I2DR,d1


             add d0,d1,d3
```

```
; --- Perform Checksum Calculation
      ; d11 = (~d7 & 0x0000ffff) = calculated ~checksum
              not d7,d11
              and #$0000ffff,d11,d11

              cmpeq d7,d2
              bt <CHeck_Csumbar

; --- Error Detected on Checksum field
              add #1,d9                  ; Number_Checksum_Errors++;

              move.l #$2,d5              ; If (Number_Checksum_Errors == 2)
              cmpeq d5,d9                ;    goto I2C_Stop_Boot
              bt I2C_Stop_Boot

              bra reload_block; Else goto reload_block

; --- Perform Checksum_Inv Calculation

CHeck_Csumbar

              cmpeq d11,d3
              bt  <stop_loading; No Error Detected on either calculation,
                                                    ;    goto stop_loading since last block

; --- Error Detected on Checksum_Inv field
              add #1,d9                  ; Number_Checksum_Errors++;

              move.l #$2,d5              ; If (Number_Checksum_Errors == 2)
              cmpeq d5,d9                ;    goto I2C_Stop_Boot
              bt I2C_Stop_Boot

              bra reload_block; Else goto reload_block


; ###
; ### Exit I2C Boot
; ###
; ### This point is reached when all records loaded successfully
; ###

stop_loading

;------ penultimate Read

              bsr   wait_for_interrupt

              move.w I2CR,d3             ; Set TXAK=1 before reading the penultimate
              or #$0008,d3.l             ; data to generate stop in master receive mode
              move.w d3,I2CR

              move.w I2DR,d0

;------ last Read

              bsr   wait_for_interrupt

              move.w I2CR,d3             ; Set msta=0 before  the last
```

```
            and #$ffdf,d3.l          ; will put the I2C in slave mode and will
            move.w d3,I2CR           ; generate stop


            move.w I2DR,d0

;###
;###  Jump to Starting Address of User's Booted Code (stored in r3)
;###     (exits boot program)
;###

     jmp r3


;#########################################################
;#########################################################
;##
;##   Supporting Routines - I2C Boot:
;##      - Called when booting through I2C port
;##
;#########################################################
;#########################################################

; #####
; ###Poll I2SR to see if the interrupt bit has,meaning data is
; ###ready in I2DR
; #####

wait_for_interrupt


data_not_ready
            move.w I2SR,d10
            bmtsts.w #$0002,d10.l;  Checking IIF bit
            bf <data_not_ready

      ;; Clearing the IIF bit in I2SR
            move.w I2SR,d10
            and #$fffd,d10.l
            move.w d10,I2SR

            rts

;#########################################################
;#########################################################
;##
;#########################################################
;#########################################################

; (code for internal use has been removed from this file)


            align        16

LOADER_TST_0                                    ; both test modes start at same point
LOADER_TST_1                                    ; both test modes start at same point
                                                        ; (difference is in PLL
setup
```

```
                                                                ;  AFTER booting has
completed.)


;#########################################################
;#########################################################
;##
;##   Interrupt Vectors
;##      - The boot code is executed in response to one of the
;##        different resets which can occur on the device.
;##        As a result, all maskable interrupts are disabled.
;##      - This vector table only contains non-maskable interrupts.
;##
;##      - Base Address = BASE_EXCEPTION_TABLE
;##
;##      - NOTE: The vector table MUST be allocated on a 4 KB Boundary
;##              due to the format of the address driven onto the VAB
;##              (as shown in the User's Manual in the Interrupt Processing
;##              chapter).  This is reflected in having a 20-bit VBA register.
;##
;##      - NOTE: Because only non-maskable interrupts are possible while
;##              booting, it is possible to use ROM locations above the
;##              last maskable interrupt vector for general usage:
;##                 BASE+1000 to BASE+11BF: Used by non-maskable vectors
;##                                         (uses  448 bytes of 4096)
;##                 BASE+11C0 to BASE+1FFF: Available for more boot code
;##                                         (     3648 bytes of 4096)
;##
;#########################################################
;#########################################################


;---------- TRAP   (0x000) ----------
             org             p:BASE_EXCEPTION_TABLE+$000

non_msk_Trap
             ;---
             ;--- This should not occur because
             ;--- there aren't any TRAP instrs in the boot code.
             ;---

;     Mark that non-maskable interrupt has occurred
             move.w        #0,d0
             move.b        d0,(r8+OFFSET_M1VAR_NMITYPE)

             rte


;---------- (Reserved)   (0x040) ----------
             org             p:BASE_EXCEPTION_TABLE+$040

non_msk_Reserved1
             ;---
             ;--- This should never occur because
             ;--- this is a reserved vector.
             ;---

;     Mark that non-maskable interrupt has occurred
```

```
                move.w        #1,d0
                move.b        d0,(r8+OFFSET_M1VAR_NMITYPE)

                rte


;---------- ILLEGAL    (0x080) ----------
                org             p:BASE_EXCEPTION_TABLE+$080

non_msk_Illegal
                ;---
                ;--- This occurs if an ILLEGAL opcode is encountered.
                ;---

;       Mark that non-maskable interrupt has occurred
                move.w        #2,d0
                move.b        d0,(r8+OFFSET_M1VAR_NMITYPE)

                rte


;---------- DEBUG PORT    (0x0C0) ----------
                org             p:BASE_EXCEPTION_TABLE+$0C0

non_msk_DebugPort
                ;---
                ;--- This occurs if an EOnCE exception is generated.
                ;---

;       Mark that non-maskable interrupt has occurred
                move.w        #3,d0
                move.b        d0,(r8+OFFSET_M1VAR_NMITYPE)

                rte


;---------- OVERFLOW    (0x100) ----------
                org             p:BASE_EXCEPTION_TABLE+$100

non_msk_Overflow
                ;---
                ;--- This occurs if an overflow occurs while booting.
                ;---

;       Mark that non-maskable interrupt has occurred
                move.w        #4,d0
                move.b        d0,(r8+OFFSET_M1VAR_NMITYPE)

                rte


;---------- (Reserved)    (0x140) ----------
                org             p:BASE_EXCEPTION_TABLE+$140

non_msk_Reserved2
                ;---
                ;--- This should never occur because
                ;--- this is a reserved vector.
```

```
                ;---

;       Mark that non-maskable interrupt has occurred
                move.w          #5,d0
                move.b          d0,(r8+OFFSET_M1VAR_NMITYPE)

                rte


;---------- Auto-NMI    (0x180) ----------
                org             p:BASE_EXCEPTION_TABLE+$180

non_msk_AutoNMI
                ;---
                ;--- This occurs if ANY non-maskable interrupt occurs.
                ;---
                ;--- NOTE: If this occurs, examine the interrupt controller's
                ;---       NMIPR register to determine the cause of the failure,
                ;---       i.e., what generated the non-maskable interrupt.
                ;---
                ;---       NMIPR register located at $0600A000:
                ;---          NMIPR[31] = _______
                ;---          NMIPR[30] = _______
                ;---          NMIPR[29] = _______
                ;---          NMIPR[28] = _______
                ;---          NMIPR[27] = _______
                ;---              .         .
                ;---              .         .

;       Mark that non-maskable interrupt has occurred
                move.w          #6,d0
                move.b          d0,(r8+OFFSET_M1VAR_NMITYPE)

; --- Report Error in Boot Variable BTERR
                move.w          #$BAD0,d0
                move.w          d0,(r8+OFFSET_M1VAR_BTERR)

; --- Capture NMIPR Register in Boot Variable NMIPR
                move.l          $0600A000,d0
                move.l          d0,(r8+OFFSET_M1VAR_CNMIPR)

                ;;debug                      ; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                rte


;---------- Auto-Maskable    (0x1C0) ----------
                org              p:BASE_EXCEPTION_TABLE+$1C0

non_msk_AutoMask
                ;---
                ;--- This should never occur because
                ;--- maskable interrupts are disabled out of reset
                ;--- and are not enabled in the boot code.
                ;---

;       Mark that interrupt has occurred
                move.w          #7,d0
                move.b          d0,(r8+OFFSET_M1VAR_NMITYPE)
```

```
            rte


;#########################################################
;#########################################################
;##
;##   Additional Boot Code
;##      - This section of the boot program is located in unused space
;##        above the Interrupt Vector Table
;##      - It uses the space reserved for Maskable interrupts
;##        which do not occur when booting.
;##
;#########################################################
;#########################################################

            align       16


;#########################################################
;#########################################################
;##
;##
;#########################################################
;#########################################################

; (code for internal use has been removed from this file)


;#########################################################
;#########################################################
;##
;##   SPI Boot (through GPIO pins in software)
;##
;##
;#########################################################
;#########################################################

;###
;### DEFINEs and EQUs
;###

WREN                                    equ         $06
WRDI                                    equ         $04
RDSR                                    equ         $05
WRSR                                    equ         $01
READ                                    equ         $03
WRITE                                   equ         $02
SECERASE                    equ         $D8
BULKERASE                   equ         $C7
READID_0                    equ         $90
READID_1                    equ         $AB
DUMMY                                   equ         $FF

FLASH_BYTES_PER_PAGEequ     256
EEPROM_BYTES_PER_PAGEequ    64
BASEADDR                    equ         $40

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; These bits are updated in STATUS_reg
;
; Status bits - low word

OddByte                                 equ          $0001
DataByte                    equ         $0002
ChecksumEnable              equ         $0004
FirstError                  equ         $0008
ErrorReport                 equ         $0010
StartBlock                  equ         $0020

; Status bits - high word

EEPROMFlash                 equ         $0001
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

             define                     BASEADDR_reg   'd15'
             define                     STATUS_reg     'd13'
             define                     CALCCS_reg     'd14'
             define                     TWOBYTES_reg   'd12'
             define                     MISOSHIFT_reg  'd4'

             define                     SPICLK_reg     'r1'
             define                     MOSI_reg       'r2'
             define                     MISO_reg       'r3'
             define                     SPISEL_reg     'r4'

             define                     MISO_data      'd2'
             define                     MOSI_data      'd7'
             define                     SPICLK_data    'd11'
             define                     SPISEL_data    'd6'

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;MSC711x                    EEPROM/FLASH
;-------                    ------------
;SPISEL                     Pin 1
;MISO                                    Pin 2
;3.3V                                    Pin 3 (/WP)
;GND                                     Pin 4 (Vss)
;MOSI                                    Pin          5
;SPICLK                                  Pin 6
;3.3V                                    Pin 7 (/Hold)
;3.3V                       Pin 8 (Vcc)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Offset from Port Data Reg
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DR_offset                   equ         $00
PxCTL_offset                equ         $08
DDR_offset                  equ         $04
EXPRT_offset                equ         $50

;###
;### SPI Macro Definitions
;###
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ClearBit   MACROmask,reg,offset
             not          mask,d9      move.l       (reg+offset),d10   ;1 cycle
             and          d9,d10             ;1 cycle
             move.l       d10,(reg+offset)   ;1 cycle

             ENDM
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SetBit          MACRO     mask,reg,offset
             move.l       (reg+offset),d10   ;1 cycle
             or                        mask,d10   ;1 cycle
             move.l       d10,(reg+offset)   ;1 cycle
             ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;#######################################
;###
;### SPI Loader Code:
;###    - for main pins
;###    - for alternate pins
;###
;#######################################

             align        16

; ###
; ### Loader for Main SPI Pins
; ###
; ###    ****************************
; ###    **  Entry Point: Boot Modes w/ Main Pins
; ###    ****************************
; ###

LOADER_SPI_0
             bsr                       InitWB                     ; common to both setups

                                                     ; Bit Locat in GPIO register for each
pin

;----------------------------------------
                                                     ; Port         Pin   ADS    EEPROM/Flash

;----------------------------------------
MAIN_MISO     equ         $00000800    ;Port B [11]   /HCS2   2
MAIN_MOSI     equ         $00000100    ;Port D [08]   BM3     5
MAIN_SPICLK   equ         $00000080    ;Port D [07]   BM2     6
MAIN_SPISEL   equ         $00000800    ;Port C [11]   HA3     1

             move.l       #MAIN_MISO,MISO_data
             move.l       #MAIN_MOSI,MOSI_data
             move.l       #MAIN_SPICLK,SPICLK_data
             move.l       #MAIN_SPISEL,SPISEL_data

      ; Pointers to GPIO registers for each pin
             move.l       #GPB_DR,MISO_reg
             move.l       #GPD_DR,MOSI_reg
```

```
              move.l        #GPD_DR,SPICLK_reg
              move.l        #GPC_DR,SPISEL_reg


              bra                         <LOADER_SPI_COMMON

; ###
; ### Loader for Alternate SPI Pins
; ###
; ###     ****************************
; ###     **  Entry Point: Boot Modes w/ Alt Pins
; ###     ****************************
; ###

LOADER_SPI_1
              bsr                         InitWB                      ; common to both setups

                                                                ; Bit Locat in GPIO register for each
pin

;----------------------------------------
                                                                ; Port          Pin    ADS     EEPROM/Flash

;----------------------------------------
ALT_MISO      equ           $00008000     ;Port A [15]    SCL    J5.B21       2
ALT_MOSI      equ           $00001000     ;Port A [12]    UTXD   J5.E22       5
ALT_SPICLK    equ           $00002000     ;Port A [13]    URXD   J5.D22       6
ALT_SPISEL    equ           $00004000     ;Port A [19]    SDA    J5.B19       1

              move.l        #ALT_MISO,MISO_data
              move.l        #ALT_MOSI,MOSI_data
              move.l        #ALT_SPICLK,SPICLK_data
              move.l        #ALT_SPISEL,SPISEL_data

      ; Pointers to GPIO registers for each pin
              move.l        #GPA_DR,MISO_reg
              move.l        #GPA_DR,MOSI_reg
              move.l        #GPA_DR,SPICLK_reg
              move.l        #GPA_DR,SPISEL_reg

              bra                         <LOADER_SPI_COMMON

              nop
              nop
              nop
              nop
              nop
              nop
              nop
              nop


LOADER_SPI_COMMON

SPIERR                      equ           $00008000     ;Port C bit 15 EVNT3          N/A
```

```
; --- Initialize GPIO
            bsr                         InitGPIO

; --- Initialize SPI
            bsr                         InitSPIRegs

; --- Calculate Shift Amount
            bsr                         CalculateMISOShift

; --- Determine Type of SPI Device:
; ---    - Serial Flash or Serial EEPROM
            bsr                         Check_EEPROM_Flash

; -------------------
; Only for burning serial memory
      IF BURN_SERIAL
            bsr                         WriteTest
            debug
      ENDIF
; -------------------

; ###
; ### Process First SPI Record
; ###

FirstBlock
            bmset       #StartBlock,STATUS_reg.l; Set for first block
            bmclr       #FirstError,STATUS_reg.l; d13  = initialize CS error found
            bmset       #ChecksumEnable,STATUS_reg.l; start with CS on
            bmclr       #OddByte,STATUS_reg.l      ; start with odd byte
            bsr                         ReadBlock                   ; read first block
            bmclr       #StartBlock,STATUS_reg.l; Clear after first block



; ###
; ### Process Remaining SPI Records
; ###
; ###    Runs as an infinite loop.
; ###    Exit from this loop in "ReadBlock" when Final Record found.
; ###

MoreBlocks

; --- while (FOREVER)   {

            bsr                         Save_BLOCK_ADDR

      ; Get next data block address
            move.l      n1,BASEADDR_reg
            bsr                         ReadBlock

            jmp                         MoreBlocks

; --- }
```

```
; --- (This statement should NEVER be reached.) ---
              debug

;#########################################################
;#########################################################
;##
;##   Supporting Routines - SPI Boot
;##
;#########################################################
;#########################################################

; ###
; ### Initialize GPIO Pins
; ###

InitGPIO

; --- MISO mask d2, reg r3
      ; Clear PxCTL bit to set as GPIO
              ClearBit      MISO_data,MISO_reg,PxCTL_offset

      ; Clear GPx_DDR bit to set as input
              ClearBit      MISO_data,MISO_reg,DDR_offset

; --- MOSI mask d7, reg r2
      ; Clear PxCTL bit to set as GPIO
              ClearBit      MOSI_data,MOSI_reg,PxCTL_offset

      ; Set GPx_DDR bit to set as output
              SetBit                    MOSI_data,MOSI_reg,DDR_offset

      ; Clear GPx_DR bit to set logic high
              ClearBit      MOSI_data,MOSI_reg,DR_offset

; --- SPICLK mask d11, reg r1
      ; Clear PxCTL bit to set as GPIO
              ClearBit      SPICLK_data,SPICLK_reg,PxCTL_offset

      ; Set GPx_DDR bit to set as output
              SetBit                    SPICLK_data,SPICLK_reg,DDR_offset

      ; Clear GPx_DR bit to set logic low
              ClearBit      SPICLK_data,SPICLK_reg,DR_offset

; --- SPISEL mask d6, reg r4
      ; Clear PxCTL bit to set as GPIO
              ClearBit      SPISEL_data,SPISEL_reg,PxCTL_offset

      ; Set GPx_DDR bit to set as output
              SetBit                    SPISEL_data,SPISEL_reg,DDR_offset

      ; Set GPx_DR bit to set logic high
              SetBit                    SPISEL_data,SPISEL_reg,DR_offset


TestPortA
              move.l        #GPA_DR,r0
              nop
```

```
                cmpeqa          MISO_reg,r0
                bf                              TestPortB

                move.l          #GPA_EXPRT,MISO_reg

                bra                             DoneTestPort

TestPortB
                move.l          #GPB_DR,r0
                nop
                cmpeqa          MISO_reg,r0
                bf                              TestPortC

                move.l          #GPB_EXPRT,MISO_reg

                bra                             DoneTestPort

TestPortC
                move.l          #GPC_DR,r0
                nop
                cmpeqa          MISO_reg,r0
                bf                              PortD

                move.l          #GPC_EXPRT,MISO_reg

                bra                             DoneTestPort

PortD
                move.l          #GPD_EXPRT,MISO_reg

DoneTestPort
                rts

; ###
; ### Initialize Write Buffer
; ###

InitWB
                ; Enable ECI's Write Buffer
                ; d0 = WB Timeout value
                move.l          #$3ff,d0
                bsr                             Cwb_timeout

                ; Setup WB DARs

    ; WBDAR[-] - M1 Memory: Not Required
    ; WBDAR[0] - M2 Memory: (done below)
    ; WBDAR[1] - SB and APB Peripherals: Write Immediate
                move.l          #B_Perif,d0    ; buffer's base
                move.l          #SZ_Perif,d1 ; buffer's size
                move.w          #1,d2                           ; Select WBDAR[1]
                move.w          #0,d3                           ; IMM field: 0=WB, 1=WImm, 2=WImm no
frz

                bsr                             Cwbdar_cfg

                rts
```

```
; ###
; ### Set Write Buffer Timeout
; ###

Cwb_timeout
      ; d0 contains desired timeout value
              push          d0

              bmclr         #$fc00,d0.l                ; Clear uppermost 6 bits
              bmclr         #$1000,d0.l                ; Clear turns on the WB
              move.w        d0,ECI_WBCR

              pop           d0
              rts

; ###
; ### Configure Write Buffer Data Area Registers
; ###

Cwbdar_cfg
              push          r1
              push          r0
              push          d1
              push          d0

    ; Calculate Register Address
              move.l        d2,r1
              move.l        #ECI_WBDAR0,r0
              asla          r1
              asla          r1
              adda          r1,r0

    ; Load Base Address into Area Base field
              ; already in d0
              ; (8 LSBs must all be zeros -currently no err chk)
              asr           d1,d1
              add           d1,d0,d0

    ; Load IMM field
              insert        #2,#5,d3,d0                ; insert D3[1:0] -> D0[6:5]

    ; Enable via EN bit
              bmset         #$0004,d0.l

    ; Save WBDAR[0-3] reg
              move.l        d0,(r0)

              pop           d0
              pop           d1
              pop           r0
              pop           r1
              rts

; ###
; ### Initialize SPI Registers
; ###

InitSPIRegs
```

```
      ; Clear d1 = MISO data
              clr                       d1

      ; First 64 bytes reserved in serial memory
      ; d15 = base address is 0x40
              move.l        #BASEADDR,BASEADDR_reg

      ; Clear d14 = calculated CS
              clr                       CALCCS_reg


      ; Clear d12 = 2 bytes of received data
              clr                       TWOBYTES_reg                          ; d12 = 2
bytes of data

      ; Reset m1 = current block address
              move.l        #0,m1

      ; Enable error reporting on EVNT3
              bmset         #ErrorReport,STATUS_reg.l
              rts


; ###
; ### Calculate MISO Shift
; ###

CalculateMISOShift
      ; MISO mask affects number of bits to shift MISO data
      ; so this needs to be calculated
              move.l        MISO_data,MISOSHIFT_reg
              ;move.l       #MISO,MISOSHIFT_reg
              clb                       MISOSHIFT_reg,MISOSHIFT_reg
              neg                       MISOSHIFT_reg
              add                       #$A,MISOSHIFT_reg
              rts


; ###
; ### SPI Transmit / Receive
; ###
; ###    Inputs:
; ###       d0    - Byte to send on MOSI
; ###       d4    - Numb bytes to shift MISO mask to read data
; ###    Output:
; ###       d1    - Byte read on MISO
; ###

SPI_TxRx
              move.w                    #0,d1                     ; Reqd so that "tsteq"
executed
                                                                  ; immed after returning
from routine
                                                                  ; has valid set of upper
bits.
              push                      d5

              dosetup2      TxRx_Byte
```

```
              doen2                     #8
              loopstart2

TxRx_Byte
              bmtsts                    #$80,d0.l
              bt                                    MOSI_set

MOSI_clr

      ; Force MOSI_bit low
              ClearBit      MOSI_data,MOSI_reg,DR_offset
              bra                                   >SPICLK_set

MOSI_set
              SetBit                    MOSI_data,MOSI_reg,DR_offset


SPICLK_set
              SetBit                    SPICLK_data,SPICLK_reg,DR_offset

      ; Read MISO bit
      ; Prepare next MOSI bit
              asl           d0,d0       move.l (MISO_reg),d5

      ; Put MISO bit in carry bit
              lsll          MISOSHIFT_reg,d5

      ; Roll carry bit into d1
              rol                       d1

SPICLK_clr
              ClearBit      SPICLK_data,SPICLK_reg,DR_offset
              loopend2

              pop           d5
              rts

; ---
; --- The "equ" directives below are used to measure size of prev routine
; ---

end_SPI_TxRx

SPI_TxRx_SZ   equ           (end_SPI_TxRx-SPI_TxRx+004)
;SPI_TxRx_COPYequ           (BASE_M1_MEMORY+M1_MEMORY_SIZE-$100)

; ###
; ### Sector Erase
; ###
; ###    Inputs:
; ###       n2    - Must contain the value $DEADBEEF
; ###
; ### Protection against accidental erase:
; ###    To protect against inadvertent call to this routine,
; ###    user must write correct data to register n2.
; ###    If n2 is not 0xDEADBEEF, then routine is not executed.
; ###
```

```
Sector_Erase
             push         n1

; --- Verify key is correct in n2 register
             move.l       #$DEADBEEF,n1
             move.w       #SECERASE,d0
             cmpeqa       n2,n1
             bf                        Done_Sector_Erase

; --- Continue with routine
             ClearBit SPISEL_data,SPISEL_reg,DR_offset
             bsr                       SPI_TxRx
             move.w       #00,d0
             bsr                       SPI_TxRx
             move.w       #00,d0
             bsr                       SPI_TxRx
             move.w       #00,d0
             bsr                       SPI_TxRx
             SetBit       SPISEL_data,SPISEL_reg,DR_offset

Done_Sector_Erase

; --- Overwrite key
             move.l       #0,n2

             pop                       n1
             rts

; ###
; ### Bulk Erase
; ###
; ###    Inputs:
; ###       n2    - Must contain the value $DEADBEEF
; ###
; ### Protection against accidental erase:
; ###    To protect against inadvertent call to this routine,
; ###    user must write correct data to register n2.
; ###    If n2 is not 0xDEADBEEF, then routine is not executed.
; ###

Bulk_Erase
             push         n1

; --- Verify key is correct in n2 register
             move.l       #$DEADBEEF,n1
             move.w       #BULKERASE,d0
             cmpeqa       n2,n1
             bf                        Done_Bulk_Erase

; --- Continue with routine
             ClearBit SPISEL_data,SPISEL_reg,DR_offset
             bsr                       SPI_TxRx
             SetBit       SPISEL_data,SPISEL_reg,DR_offset

Done_Bulk_Erase

; --- Overwrite key
             move.l       #0,n2
```

```
            pop                         n1
            rts


; ###
; ### Write Enable
; ###

Write_Enable
            ClearBit SPISEL_data,SPISEL_reg,DR_offset
            move.w      #WREN,d0
            bsr                         SPI_TxRx
            SetBit      SPISEL_data,SPISEL_reg,DR_offset

            rts

; ###
; ### Write Disable
; ###

Write_Disable
            ClearBit SPISEL_data,SPISEL_reg,DR_offset
            move.w      #WRDI,d0
            bsr                         SPI_TxRx
            SetBit      SPISEL_data,SPISEL_reg,DR_offset

            rts

; ###
; ### Enable Write
; ###

EnableWrite
            bsr                         Write_Enable
            bsr                         WaitforWriteEnabled

            rts
; ###
; ### Erase Flash
; ###

EraseFlash
            bsr                         EnableWrite
            bsr                         Bulk_Erase
            bsr                         WaitforNotBusy

            rts
; ###
; ### Read Status
; ###
; ###    Outputs:
; ###       d1    - contains status value with bits as follows:
; ###                   [7] = WPEN
; ###                   [6] = (reserved)
; ###                   [5] = (reserved)
; ###                   [4] = (reserved)
; ###                   [3] = BP1
; ###                   [2] = BP0
```

```
; ###                 [1] = WEL
; ###                 [0] = /RDY
; ###
; ###

Read_Status
             ClearBit SPISEL_data,SPISEL_reg,DR_offset
             move.w       #RDSR,d0
             bsr                        SPI_TxRx

             move.w       #DUMMY,d0
             bsr                        SPI_TxRx
             SetBit       SPISEL_data,SPISEL_reg,DR_offset
             rts
; ###
; ### Wait for Not Busy
; ###

WaitforNotBusy
             bsr                        Read_Status

      ; Check if WIP bit is cleared
             bmtstc       #$01,d1.l
             bt                         NotBusy
             jmp                        WaitforNotBusy

NotBusy
             rts

; ###
; ### Wait for Write Enabled
; ###

WaitforWriteEnabled
             bsr                        Read_Status

      ; Check if WEL bit is set in MISO
             bmtsts       #$02,d1.l
             bt                         WriteEnabled
             jmp                        WaitforWriteEnabled

WriteEnabled
             rts

; ###
; ### Wait for Read Done
; ###

WaitforReadDone
             bsr                        Read_Status

      ; Check if WIP and WEL bits is cleared
             bmtstc       #$03,d1.l
             bt                         ReadDone
             jmp                        WaitforReadDone

ReadDone
             rts
```

```
; ###
; ### Write Data
; ###
; ###    Inputs:
; ###       d0 - EEPROM address
; ###       d5 - max number bytes per page
; ###       r0 - pointer where write data is stored
; ###       r5 - number of bytes
; ###

WriteData
              push          n1
              move.l        #$DEADBEEF,n1
              push          d8
              cmpeqa        n2,n1
              bf                          Done_WriteData

              bmtstc        #EEPROMFlash,STATUS_reg.h
              tfr                         d0,d3                                       ;d3 = EEPROM
address
              ift                         move.l        #EEPROM_BYTES_PER_PAGE,d5
              iff                         move.l        #FLASH_BYTES_PER_PAGE,d5

              move.l        r5,d8                                                     ;d8 = actual
compare
              cmpgt         d8,d5                                                     ;if
actual<max
              jt                          belowmax

              bsr                         Write_Enable
              bsr                         WaitforWriteEnabled

              ClearBit SPISEL_data,SPISEL_reg,DR_offset

      ; Send WRITE command
              move.w        #WRITE,d0
              bsr                         SPI_TxRx

      ; Send 3 address bytes for Flash
              bmtsts        #EEPROMFlash,STATUS_reg.h
              tfr                         d3,d0
              ift           bsr                         SendExtraByteAddress_Slow

      ; Send 2 address bytes for EEPROM
              tfr                         d3,d0
              asrr          #8,d0
              bsr                         SPI_TxRx

              tfr                         d3,d0
              bsr                         SPI_TxRx

writeloop
      ; Write data
              move.b        (r0)+,d0
              bsr                         SPI_TxRx                                  ;write max
bytes
              deceq         d5
              bf                          writeloop
```

```
              bmtstc        #EEPROMFlash,STATUS_reg.h
              nop                                     ; reqd by pipeline: see dependency T.1
              ift                           move.l    #EEPROM_BYTES_PER_PAGE,d5
              iff                           move.l    #FLASH_BYTES_PER_PAGE,d5

              sub                           d5,d8,d8
;actual=actual-max
              add                           d5,d3,d3
              SetBit        SPISEL_data,SPISEL_reg,DR_offset
              jmp                           compare

belowmax

              bsr                           Write_Enable
              bsr                           WaitforWriteEnabled

              move.l        d8,r5
              ClearBit SPISEL_data,SPISEL_reg,DR_offset

      ; Send WRITE command
              move.w        #WRITE,d0
              bsr                           SPI_TxRx

      ; Send 3 address bytes for Flash
              bmtsts        #EEPROMFlash,STATUS_reg.h
              tfr                           d3,d0
              ift           bsr                         SendExtraByteAddress_Slow

      ; Send 2 address bytes for EEPROM
              tfr                           d3,d0
              asrr          #8,d0
              bsr                           SPI_TxRx

              tfr                           d3,d0
              bsr                           SPI_TxRx

      ; Write r5 bytes
              dosetup0 WriteBytes
              doen0          r5
              nop

              loopstart0
WriteBytes
              move.b        (r0)+,d0
              bsr                           SPI_TxRx
              nop
              nop
              nop
              loopend0
              SetBit        SPISEL_data,SPISEL_reg,DR_offset
Done_WriteData
              move.l        #0,n2
              pop                           d8
              pop                           n1
              rts
```

```
; ###
; ### Read Field
; ###
; ###    Inputs:
; ###       d0   - EEPROM address
; ###       r6   - pointer where read data is stored
; ###                  used only if d13.l bit 1 = 1
; ###       r5   - number of bytes
; ###       d13.l - bit 1 determines data or field
; ###
; ###    Outputs:
; ###       d12 - Concatenated 2 bytes data

ReadField
             push         d3
      ; Copy EEPROM address to d11
             tfr                        d0,d3
             ClearBit     SPISEL_data,SPISEL_reg,DR_offset

      ; Send READ command
             move.w       #READ,d0
             bsr                        SPITxRx

      ; Send 3 address bytes for Flash
             bmtsts       #EEPROMFlash,STATUS_reg.h
             tfr                        d3,d0
             ift          bsr                        SendExtraByteAddress_Fast

             tfr                        d3,d0
             asrr         #8,d0
             bsr                        SPITxRx

             tfr                        d3,d0
             bsr                        SPITxRx

             pop                        d3

      ; Receive r5 bytes
             dosetup0 ReadBytes
             doen0          r5
             loopstart0
ReadBytes
             bsr                        SPITxRx


Check_Data
             bmtsts       #DataByte,STATUS_reg.l
             nop                                    ; reqd by pipeline: see dependency T.1
             ift                        move.b      d1,(r6)+

Check_Odd_Even_Byte
             bmtstc       #OddByte,STATUS_reg.l
             bf                         Even_Byte
Odd_Byte
             asll         #8,d1
             tfr                        d1,TWOBYTES_reg
             jmp                        Toggle
Even_Byte
```

```
                or                      d1,TWOBYTES_reg
                eor                     TWOBYTES_reg,CALCCS_reg
Toggle
                bmchg           #OddByte,STATUS_reg.l
                nop
                nop
                nop
                loopend0
                and                     #0,TWOBYTES_reg.h
                SetBit          SPISEL_data,SPISEL_reg,DR_offset
                bsr                     WaitforReadDone
                rts


; ###
; ### Get CS Size
; ###

Get_CS_SIZE
                push            r5

                tfr                     BASEADDR_reg,d0      ; d0 = base + 0 for cs/size
                bmclr           #DataByte,STATUS_reg.l       ; not data
                move.w          #2,r5                        ; r5 = 2 bytes for cs/size
                bsr                     ReadField
                move.l          TWOBYTES_reg,n0              ; n0 = CS_SIZE

Check_CS_Enabled
                tfr                     TWOBYTES_reg,CALCCS_reg      ;d14 = calculated CS
                bmtsts          #$8000,CALCCS_reg.l
                bf                      CS_OFF
CS_ON
                bmset           #ChecksumEnable,STATUS_reg.l
                jmp                     CS_SIZE_done
CS_OFF
                bmclr           #ChecksumEnable,STATUS_reg.l
CS_SIZE_done

                pop                     r5
                rts

; ###
; ### Get Next Block Address (Upper 16 bits)
; ###

Get_NEXT_BLOCK_ADDR_HIGH
                push            r5
                tfr                     BASEADDR_reg,d0
                add                     #2,d0                                        ;d0 =
base + 2 bytes for next addr
                bmclr           #DataByte,STATUS_reg.l     ;not data
                move.w          #2,r5                                                ;r5 =
2 bytes for next addr
                bsr                     ReadField

                asll            #16,TWOBYTES_reg
                move.l          TWOBYTES_reg,n1
                pop                     r5
                rts
```

```
; ###
; ### Get Next Block Address (Lower 16 bits)
; ###

Get_NEXT_BLOCK_ADDR_LOW
              push          r5
              push          d5
              tfr                           BASEADDR_reg,d0
              add                           #4,d0               ; d0 = base + 4 bytes for next
addr
              bmclr         #DataByte,STATUS_reg.l      ; not data
              move.w        #2,r5                       ; r5 = 2 bytes for next addr
              bsr                           ReadField
              and                           #0,TWOBYTES_reg.h move.l   n1,d5
              or                            d5,TWOBYTES_reg
              move.l        TWOBYTES_reg,n1             ; n1 = next block address
              pop                           d5
              pop                           r5
              rts

; ###
; ### Check Sequence Order
; ###

Check_SEQ_ORDER
              move.l        n1,d0
              tsteq         d0
              bf                            SEQ_ORDER_Done

SEQ_ORDER
              move.l        n0,d0
              bmclr         #$8000,d0.l
              add                           BASEADDR_reg,d0,d0
              add                           #$a,d0
              move.l        d0,n1

SEQ_ORDER_Done
              rts

; ###
; ### Check End Block
; ###

Check_END_BLOCK
              move.l        n0,d0                                   ; test first two bytes to
see if 0
              tsteq         d0
              bf                            NOT_END_BLOCK

              tfr                           BASEADDR_reg,d0; test next two bytes to see if 0
              add                           #2,d0
              move.w        #2,r5
              bsr                           ReadField

      ; Check two bytes are 0
              bmtstc        #$ffff,TWOBYTES_reg.l
              bf                            NOT_END_BLOCK
```

```
END_BLOCK

      ;Next 4 bytes are jump addr
              tfr                       BASEADDR_reg,d0
              add                       #4,d0                                   ; d0 = base +
4 bytes for jump addr
              move.w        #2,r5                                   ; r5 = 2 bytes for load
addr
              bsr                       ReadField
              asll          #16,TWOBYTES_reg
              move.l        TWOBYTES_reg,r7

              tfr                       BASEADDR_reg,d0
              add                       #6,d0                                   ; d0 = base +
6 bytes for next addr
              move.w        #2,r5                                   ; r5 = 2 bytes for next
addr
              bsr                       ReadField
              move.l        r7,d5
              or                        d5,TWOBYTES_reg
              move.l        TWOBYTES_reg,r7; r7 = jump address

              bsr                       JumpAddr

NOT_END_BLOCK
              rts

; ###
; ### Set Load Address (Upper 16-bits)
; ###

Get_LOAD_ADDR_HIGH
              push          r5
              tfr                       BASEADDR_reg,d0
              add                       #6,d0                                   ; d0 = base +
6 bytes for load addr
              bmclr         #DataByte,STATUS_reg.l; not data
              move.w        #2,r5                                   ; r5 = 2
bytes for load addr
              bsr                       ReadField
              asll          #16,TWOBYTES_reg
              move.l        TWOBYTES_reg,r6
              pop                       r5
              rts

; ###
; ### Set Load Address (Lower 16-bits)
; ###

Get_LOAD_ADDR_LOW
              push          r5
              tfr                       BASEADDR_reg,d0
              add                       #8,d0                                   ; d0 = base +
8 bytes for load addr
              bmclr         #DataByte,STATUS_reg.l; not data
              move.w        #2,r5                                   ; r5 = 2
bytes for load addr
              bsr                       ReadField
```

```
                and                     #0,TWOBYTES_reg.hmove.l r6,d5
                or                      d5,TWOBYTES_reg
                move.l      TWOBYTES_reg,r6
                pop                     r5
                rts


; ###
; ### Get Data
; ###

Get_DATA
                push        r5
                tfr                     BASEADDR_reg,d0
                add                     #$A,d0                                  ; d0 = base +
10 for data
                move.l      n0,d8
                bmclr       #$8000,d8.l
                move.l      d8,r5                                               ; d8 =
program sz including CS and /CS
                bmset       #DataByte,STATUS_reg.l; indicate data
                suba        #4,r5                                               ; r5 =
subtract 4 bytes for CS and /CS
                bsr                     ReadField
                bmclr       #DataByte,STATUS_reg.l; indicate not data

                pop                     r5
                rts

; ###
; ### Save Calculated Checksum
; ###

Save_CALC_CS
                move.l      CALCCS_reg,d0                       ; save calculated CS
                not                     CALCCS_reg,d1                           ; save
calculated /CS
                move.w      #0,d1.h                 move.w #0,d0.h
                asll        #16,d0
                or                      d1,d0
                move.l      d0,m3
                rts

; ###
; ### Get Checksum
; ###

Get_CS
                push        r5
                tfr                     BASEADDR_reg,d0
                add                     #$6,d8
                add                     d8,d0,d0                                ; d0 = CS
address is program size + 5 bytes
                bmclr       #DataByte,STATUS_reg.l; not data
                move.w      #2,r5                                               ; r5 = 2
bytes CS
                bsr                     ReadField
                asll        #16,TWOBYTES_reg
```

```
              move.l        TWOBYTES_reg,n3

              pop                         r5
              rts


; ###
; ### Get Checksum Inv
; ###

Get_NotCS
              push          r5
              tfr                         BASEADDR_reg,d0
              add                         #$2,d8
              add                         d8,d0,d0                                    ; d0 = /CS
address is program size + 7 bytes
              bmclr         #DataByte,STATUS_reg.l; not data
              move.w        #2,r5                                                     ; r5 = 2
bytes /CS
              bsr                         ReadField
              and                         #0,TWOBYTES_reg.hmove.l    n3,d5
              or                          d5,TWOBYTES_reg
              move.l        TWOBYTES_reg,n3
              pop                         r5
              rts

; ###
; ### Compare Checksum
; ###
; ###    Inputs:
; ###       n3  = Expected   16-bit Checksum concated w/ 16-bit Checksum Inv
; ###                - Upper 16-bits contains ____________
; ###                - Lower 16-bits contains ____________
; ###       m3  = Calculated 16-bit Checksum concated w/ 16-bit Checksum Inv
; ###                - Upper 16-bits contains ____________
; ###                - Lower 16-bits contains ____________
; ###

Compare_CS

; --- if (Checksum Compare is Enabled for this Record)   {
              bmtstc        #ChecksumEnable,STATUS_reg.l
              bt                          Compare_CS_Done

; --- Compare Calculated vs Expected
              move.l        n3,d0                                          ; d0 = provided CS and /CS
              move.l        m3,d1                                          ; d1 = calculated CS and
/CS
              cmpeq         d0,d1
              bt                          No_CS_Error

; --- Determine Error Handling

CS_Error
      ; Check if first error
              bmtsts        #FirstError,STATUS_reg.l; check if first error
              bt                          First_Error
```

```
; --- Process - More than one Checksum Error for this Record
              bsr                         SPI_Stop_Boot

; --- Process - Only one Checksum Error for this Record

First_Error
              bmset         #FirstError,STATUS_reg.l; d13 = CS error found
              bsr                         Restore_BLOCK_ADDR
              jmp                         MoreBlocks                ; Reload same block

; --- }

; --- Process - No Checksum Errors for this Record

No_CS_Error
              bmclr         #FirstError,STATUS_reg.l; d13 = reset CS errors

Compare_CS_Done
              rts

; ###
; ### Save Block Address
; ###

Save_BLOCK_ADDR
              move.l        n1,d0
              move.l        d0,m1
              rts


; ###
; ### Restore Block Address
; ###

Restore_BLOCK_ADDR
              move.l        m1,d0
              move.l        d0,n1
              rts
; ###
; ### Jump Address
; ###

JumpAddr
      ; Cannot jump to jump address if this is first block
              bmtsts        #StartBlock,STATUS_reg.l
              bt                          SPI_Stop_Boot

      ; At end of boot process make all SPI pins input
      ; Clear GPx_DDR bit to set as input
              ClearBit MOSI_data,MOSI_reg,DDR_offset
              ClearBit SPICLK_data,SPICLK_reg,DDR_offset
              ClearBit SPISEL_data,SPISEL_reg,DDR_offset
              jmp                          r7
              rts



; ###
```

```
; ### SPI Stop Boot
; ###
; ### On SPI Boot Error:
; ###
; ###    If (SPI Boot Error occurred)   {
; ###       Write BTERR Variable with $BAD2
; ###       If Checksum Comparison Enabled
; ###          Then   {
; ###             Place   GPIO's PortC[15] in Software Mode: BM1/GPIO/EVNT3 pin
; ###             Program GPIO's PortC[15] as GPIO Output
; ###             Toggle  BM1/GPIO/EVNT3 pin (which is used as GP Output)
; ###                (continues toggling pin in an infinite loop)
; ###          }
; ###       Else
; ###          Simply Loop on a single jmp instruction (at end of this routine)
; ###                (continues in an infinite loop)
; ###    }
; ###
; ### On I2C Boot Error:
; ###
; ###    If (I2C Boot Error occurred)   {
; ###       Write BTERR Variable with $BAD3
; ###
; ###       Place   GPIO's PortC[15] in Software Mode: BM1/GPIO/EVNT3 pin
; ###       Program GPIO's PortC[15] as GPIO Output
; ###       Toggle  BM1/GPIO/EVNT3 pin (which is used as GP Output)
; ###          (continues toggling pin in an infinite loop)
; ###    }
; ###
; ### This routine is called if:
; ###    - SPI Error: A second checksum error happened in a record.
; ###    - SPI Error: The very first Record is also the Final record.
; ###    - I2C Error: A second checksum error happened in a record.
; ###
; ### NOTE: Upon entry to this routine, the BM1/GPIO/EVNT3 pin has
; ###       not yet been programmed by the Boot ROM so it is a GPIO Input.
; ###       Because it is muxed with BM1, there will already be a
; ###       pullup/pulldown on this pin.
; ###
; ### NOTE: No exit from this routine - infinite looping!
; ###

SPI_Stop_Boot

; --- Flag SPI Boot Error in Boot Error Variable
             move.w       #$BAD2,d0
             move.w       d0,(r8+OFFSET_M1VAR_BTERR)

; --- If ErrorReport is disabled, just loop continuously
             bmtstc       #ErrorReport,STATUS_reg.l
             bt                        LoopForever

Signal_Boot_Error

; --- Else   {
; ---    Set pin in Software Control Mode (should already be in this from reset)
; ---    Set pin as a GP output
; ---    Toggle BM1/GPIO/EVNT3 pin in an infinite loop
```

```
      ; Clear PxCTL bit to set as GPIO (i.e., in Software Control Mode)
              move.l        #PCCTL,r0
              move.l        #SPIERR,d9    ; d9 = $0000 8000 for BM1/GPIO/EVNT3 pin

              move.l        (r0),d10
              not           d9,d9                       ; d9 = $FFFF 7FFF
              and           d9,d10
              move.l        d10,(r0)

      ; Set GPx_DDR bit to set as output
              move.l        #GPC_DDR,r0
              move.l        #SPIERR,d9

              move.l        (r0),d10
              or            d9,d10
              move.l        d10,(r0)


      ; Program r0, d9 regs to point to correct bit in GPx_DR bit reg
              move.l        #GPC_DR,r0    ; Address      of GPx_DR reg
              move.l        #SPIERR,d9    ; Bit position in GPx_DR reg

; --- while (FOREVER)   {

toggle_pin
      ; Clear BM1/GPIO/EVNT3 Pin
              move.l        (r0),d10
              not           d9,d9                       ; d9 = $FFFF 7FFF
              and           d9,d10
              move.l        d10,(r0)

      ; Delay
              doensh0       #100
              nop
              loopstart0
                 nop
                 nop
              loopend0

      ; Set BM1/GPIO/EVNT3 Pin
              move.l        (r0),d10
              not           d9,d9                       ; d9 = $0000 8000
              or            d9,d10
              move.l        d10,(r0)

      ; Delay
              doensh0       #100
              nop
              loopstart0
                 nop
                 nop
              loopend0

              bra           <toggle_pin

; --- }
```

```
; --- }


LoopForever
             jmp                        *
             rts

; ###
; ### I2C Stop Boot
; ###
; ###    (see comment under SPI Stop Boot)
; ###

I2C_Stop_Boot

; --- Flag I2C Boot Error in Boot Error Variable
             move.w       #$BAD3,d0
             move.w       d0,(r8+OFFSET_M1VAR_BTERR)

             bra          <Signal_Boot_Error

; ###
; ### Check EEPROM Flash
; ###
; ###    Only Flash supports Read Device ID instruction.
; ###    EEPROM does not support this instruction.
; ###    EEPROM returns 0x00 or 0xFF
; ###

Check_EEPROM_Flash
             bmclr        #EEPROMFlash,STATUS_reg.h

             ClearBit     SPISEL_data,SPISEL_reg,DR_offset
             move.w       #READID_0,d0
             bsr                        SPI_TxRx

             move.w       #$0,d0
             bsr                        SPI_TxRx

             move.w       #$0,d0
             bsr                        SPI_TxRx

             move.w       #$0,d0
             bsr                        SPI_TxRx

             move.w       #DUMMY,d0
             bsr                        SPI_TxRx

             SetBit       SPISEL_data,SPISEL_reg,DR_offset

      ; Check if 0x00
             tsteq        d1
             bt                         TryOtherReadID

      ; Check if 0xFF
             bmtsts       #$00ff,d1.l
             bt                         TryOtherReadID
```

```
      ; Otherwise - Flash and Copy SPI_TxRx to M1 for fast transfer
              bmset         #EEPROMFlash,STATUS_reg.h
              bsr                           Copy_SPI_TxRx_M1
              bra                           Done_Check_EEPROM_Flash

TryOtherReadID
              ClearBit      SPISEL_data,SPISEL_reg,DR_offset
              move.w        #READID_1,d0
              bsr                           SPI_TxRx

              move.w        #$0,d0
              bsr                           SPI_TxRx

              move.w        #$0,d0
              bsr                           SPI_TxRx

              move.w        #$0,d0
              bsr                           SPI_TxRx

              move.w        #DUMMY,d0
              bsr                           SPI_TxRx

              SetBit        SPISEL_data,SPISEL_reg,DR_offset

      ; Check if 0x00 - EEPROM
              tsteq         d1
              bt                            Done_Check_EEPROM_Flash

      ; Check if 0xFF - EEPROM
              bmtsts        #$00ff,d1.l
              nop                                       ; reqd by pipeline: see dependency T.1
      ; Otherwise - Flash and Copy SPI_TxRx to M1 for fast transfer
              iff                           bmset         #EEPROMFlash,STATUS_reg.h
              iff                           bsr                           Copy_SPI_TxRx_M1
Done_Check_EEPROM_Flash
              rts


; ###
; ### (A Copy of) SPI Transmit / Receive
; ###
; ###    Will be copied into M1 memory.
; ###

Copy_SPI_TxRx_M1
              push                          r0
              push                          r1
              move.l                        #SPI_TxRx,r0
              adda                          #OFFSET_M1_OVERLAY,r8,r1
              dosetup2      Copy
              doen2                         #SPI_TxRx_SZ
              loopstart2
Copy
              move.b                        (r0)+,d0
              move.b                        d0,(r1)+
              nop
              nop
```

```
              loopend2
              pop                                   r1
              pop                                   r0
              rts


; ###
; ### SPI Transmit / Receive
; ###

SPITxRx
              bmtsts        #EEPROMFlash,STATUS_reg.h
              bt                          FastSPI
              bsr                         SPI_TxRx
              bra                         SPITxRxDone
FastSPI
              push                        r0
              adda                        #OFFSET_M1_OVERLAY,r8,r0
              jsr                         r0
              pop                         r0
SPITxRxDone
              rts


; ###
; ### Send Extra Byte Address: Fast Mode
; ###

SendExtraByteAddress_Fast
              asrr          #16,d0
              push                        r0
              adda                        #OFFSET_M1_OVERLAY,r8,r0
              jsr                         r0
              pop                         r0
              rts


; ###
; ### Send Extra Byte Address: Slow Mode
; ###

SendExtraByteAddress_Slow
              asrr          #16,d0
              bsr                         SPI_TxRx
              rts


; ###
; ### Read Block
; ###
; ###    Processes one Record from the SPI.
; ###
; ###
; ###

ReadBlock

; --- Get Record Size (which includes Checksum Compare Enable in MSB)
```

```
            bsr                     Get_CS_SIZE

; --- Get Next Record Address
            bsr                     Get_NEXT_BLOCK_ADDR_HIGH
            bsr                     Get_NEXT_BLOCK_ADDR_LOW
            bsr                     Check_SEQ_ORDER

; --- Check if Final Record
            bsr                     Check_END_BLOCK

; --- Get Load Address
            bsr                     Get_LOAD_ADDR_HIGH
            bsr                     Get_LOAD_ADDR_LOW

; --- Get all Boot Data Entries
            bsr                     Get_DATA
            bsr                     Save_CALC_CS

; --- Checksum Processing
            bsr                     Get_CS
            bsr                     Get_NotCS
            bsr                     Compare_CS

            rts


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; The functions below show an example of how to burn
; serial memory. Will not be part of ROM code
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      IF BURN_SERIAL

WriteTest
            move.l      #$DEADBEEF,n2
            bmtsts      #EEPROMFlash,STATUS_reg.h
            nop                                 ; reqd by pipeline: see dependency T.1
            ift                     bsr         EraseFlash
            bsr                     Write_Enable
            bsr                     WaitforWriteEnabled
            bsr                     WriteTestData
            bsr                     WaitforNotBusy
            bsr                     ReadTestData
            debug
; ###
; ###
; ###

WriteTestData
            move.l      #$40,d0                                 ; d0 = EEPROM address
            move.l      #DATA_OUT,r0  ; r0 = pts to data to write
            bmtstc      #EEPROMFlash,STATUS_reg.h
            move.w      #DATA_SZ,r5             ; r5 = pts to data size
            ift                     move.l      #EEPROM_BYTES_PER_PAGE,d5
            iff                     move.l      #FLASH_BYTES_PER_PAGE,d5
            move.l      #$DEADBEEF,n2
            bsr                     WriteData
            rts
```

```
; ###
; ###
; ###

ReadTestData
             move.l       #$40,d0                                  ; d0 = EEPROM address
             move.l       #DATA_IN,r6                ; r0 = pts to data to write
             bmtstc       #EEPROMFlash,STATUS_reg.h
             move.w       #DATA_SZ,r5                ; r5 = pts to data size
             bmset        #$0002,STATUS_reg.l; indicate data
             bsr                       ReadField
             rts



      ENDIF
```

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X