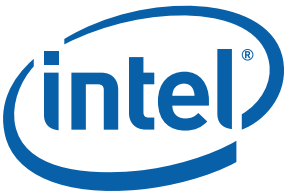


Intel® IXP43X Product Line of Network Processors

Developer's Manual

December 2008

Document Number: 316843; Revision: 003US

Contents

Figures

Tables

Revision History

Date	Revision	Description
December 2008	003	**Section 2.x** Section 2.1.7 - Added DDRII 16-bit SDRAM, 512 Mbit technology support Table 3 - Added GPIO alternate function information **Section 11.x** Section 11.1 and Table 195 - Added DDRII 16-bit SDRAM, 512 Mbit technology support Section 11.2.2.10 - Added procedure to issue EMRS OCD Command during DDR SDRAM Initialization Section 11.6.1 - Added note to DDR SDRAM Initialization Register - SDIR Section 11.6.2 - Added notes for the EDP and ODT bits Added new feature: IEEE-1588 Hardware Assist support Added new feature: Turbo MII Mode support Incorporated specification clarifications, specification changes and document changes from *Intel® IXP43X Product Line of Network Processors Specification Update* (316847-004) Change bars indicate areas of change.
October 2007	002	Section 2.0: • Figure 1, Figure 2, and Figure 5: Removed 266 MHz clock speed • Figure 3, and Figure 4: Modified 266 MHz clock speed to 400 MHz clock speed Section 5.0: Added Note on the transaction type Section 11.0: • Section 11.2.2.15: Added DDR SDRAM Debugging Procedure • Table 208: Added information for 'Receive Enable Delay Register' and 'DDR Drive Strength Control Register' Section 11.6: Added the following registers: • "Receive Enable Delay Register RCVDLY" • "DDR Drive Strength Control Register LEGOVERIDE" Section 12.0: • Removed Table 210 • Removed 266 MHz support
April 2007	001	Initial release

§ §

1.0 Introduction

1.1 About This Document

This document is the authoritative and definitive reference for external architecture of the Intel® IXP43X Product Line of Network Processors based on Intel XScale® Technology.

1.2 Intended Audience

This document is intended for:

- Software engineers who want to program at the register or assembly level
- System architects who want to ensure that a system operates and performs to specification
- Board designers who will use it for critical functional interfaces and bootstrapping option for the IXP43X network processors
- Anyone else seeking detailed knowledge on the functional interworkings of the part

1.3 How to Read This Document

You should be familiar with the ARM* Version 5TE Architecture to understand some aspects of this document. Each chapter in this document focuses on a specific architectural feature of the IXP43X network processors.

1.4 Other Relevant Documents

Document Title	Document #
Intel® IXP43X Product Line of Network Processors Datasheet	316842
Intel® IXP43X Product Line of Network Processors Hardware Design Guidelines	316844
Intel® IXP4XX Product Line of Network Processors Specification Update	306428
Intel® IXP400 Software Programmer's Guide	252539
Intel XScale® Technology *Programmer's Reference Manual*	273436
Enabling Time Synchronization (IEEE-1588) Hardware on Intel® IXP43X Product Line of Network Processors Application Note	313857
Enabling TMII Hardware on Intel® IXP43X Product Line of Network Processors Application Note	319092
PCI Local Bus Specification, Revision 2.2	
Universal Serial Bus Specification, Revision 2.0	
UTOPIA Level 2 Specification, Revision 1.0	
IEEE 802.3 Specification	
IEEE 1149.1 Specification	

Document Title	Document #
UTMI and UTMI+ Specification	
SSP Specification	
JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79	
JEDEC Standard DDR2 SDRAM Specification JESD79-2	

1.5 Terminology and Conventions

This section explains the naming conventions and the terminology used in the document. Common acronyms are defined in Table 1.

Table 1. List of Acronyms (Sheet 1 of 3)

Acronym	Description
AAL	ATM Adaptation Layer
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
AHB	Advanced High-Performance Bus
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASSP	Application Specific Standard Product
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
ATPG	Automatic Test Pattern Generation
BERT	Bit-Error Rate Test
BFA	Basic Frame Alignment
BIU	Bus Interface Unit
Bps	Bytes per second
bps	Bits per second
BIU	Bus Interface Unit
CID	Channel Identifier
CPE	Customer Premise Equipment
CRC	Cyclical Redundancy Check
DAT	Direct Access Test
DES	Data Encryption Standard
DDR	Double Data Rate
DIA	Development Investment Approval
DiffServ	Differentiated Services
DMA	Direct Memory Access
DSL	Digital Subscriber Line
FRAD	Frame Relay Access Device
FRF	Frame Relay Forum

Table 1. List of Acronyms (Sheet 2 of 3)

Acronym	Description
FXO	Foreign Exchange Office
FXS	Foreign Exchange Subscriber
GCI	General Circuit Interface
GE	Gigabit Ethernet
G.SHDSL	ITU G series specification for Symmetric High Bit Rate Digital Subscriber Line
GPIO	General-Purpose Input/Output
HDLC	High-Level Data Link Control
HEC	Header-Error Check
HPI	Host-Port Interface
HSSI	High-Speed Serial Interface
IEEE	Institute of Electrical and Electronic Engineers
IKE	Internet Key Exchange
IP	Internet Protocol
IPsec	Internet Protocol Security
IRQ	Interrupt Request
IXA	Internet Exchange Architecture
IXP	Internet Exchange Processor
JTAG	Joint Test Action Group
LAN	Local Area Network
LUT	Look-Up Table
MAC	Media Access Controller
MCU	Memory Controller Unit
MD5	Message Digest 5
MDIO	Management Data Input/Output
MII	Media-Independent Interface
MLPPP	Multi-Link, Point-to-Point Protocol
MPI	Memory Port Interface
MVIP	Multi-Vendor Integration Protocol
NAPT	Network Address and Protocol Translator
NAT	Network Address Translation
NPE	Network Processor Engine
OC 3	Optical Carrier 3
PBX	Private Branch Exchange
PCI	Peripheral Component Interface
PHY	Physical Layer (Layer 1) Interface
RComp	Resistive Compensation
Rx	Receive
SAR	Segmentation and Reassembly
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol

Table 1. List of Acronyms (Sheet 3 of 3)

Acronym	Description
SNMP	Simple Network Management Protocol
TLU	Test Logic Unit
ToS	Type of Service
Tx	Transmit
UART	Universal Asynchronous Receiver-Transmitter
UTOPIA	Universal Test and Operation PHY Interface for ATM
VCI	Virtual Circuit Identifier
VDSL	Very-High-Speed, Digital Subscriber Line
VoDSL	Voice over Digital Subscriber Line
VoFR	Voice over Frame Relay
VoIP	Voice-Over-Internet Protocol
VPI	Virtual Path Identifier
VPN	Virtual Private Network
WAN	Wide Area Network
xDSL	Any Digital Subscriber Line

1.5.1 Number Representation

All numbers in this document can be assumed to be base 10 unless designated otherwise. In text and pseudo code descriptions, hexadecimal numbers have a prefix of 0x and binary numbers have a prefix of 0b. For example, 107 will be represented as 0x6B in hexadecimal and 0b1101011 in binary.

1.5.2 Signal-Naming Convention

The following naming conventions are used in this document,

- A word refers to 32 bits
- Dword for PCI transactions, PCI specification refers to Dword as 32-bit transactions
- Signal names or register bit names ending with '_N' or '#'are active low, for example RTS_N; Signals or register bits without this are active high

1.5.3 Register Legend

The following register access definitions are used in this document:

Table 2. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write '1' to clear	RW1S	Normal Read Write '1' to set

2.0 Functional Overview

The Intel® IXP43X Product Line of Network Processors is compliant with the Intel® StrongARM* Version 5TE instruction set architecture (ISA). The IXP43X product line of network processors is designed with Intel 0.13-µ semiconductor process technology. This process technology along with compactness of the Intel® StrongARM* RISC ISA, which has got the ability to simultaneously process data with up to two integrated network processing engines (NPEs), and numerous dedicated-function peripheral interfaces, enables the IXP43X network processors to operate over a wide range of low-cost networking applications with industry-leading performance.

As indicated in Figure 1, Figure 2, Figure 3, Figure 4 and Figure 5, the IXP43X network processors combine many features with the Intel XScale® Processor to create a highly integrated processor applicable to LAN/WAN-based networking applications in addition to other embedded networking applications.

This section describes the main features of the IXP43X product line of network processors. Refer to the *Intel® IXP43X Product Line of Network Processors Datasheet* for a detailed list of features that are supported in the IXP43X network processors, categorized by the processor model.

Figure 1. Intel® IXP435 Network Processor Block Diagram

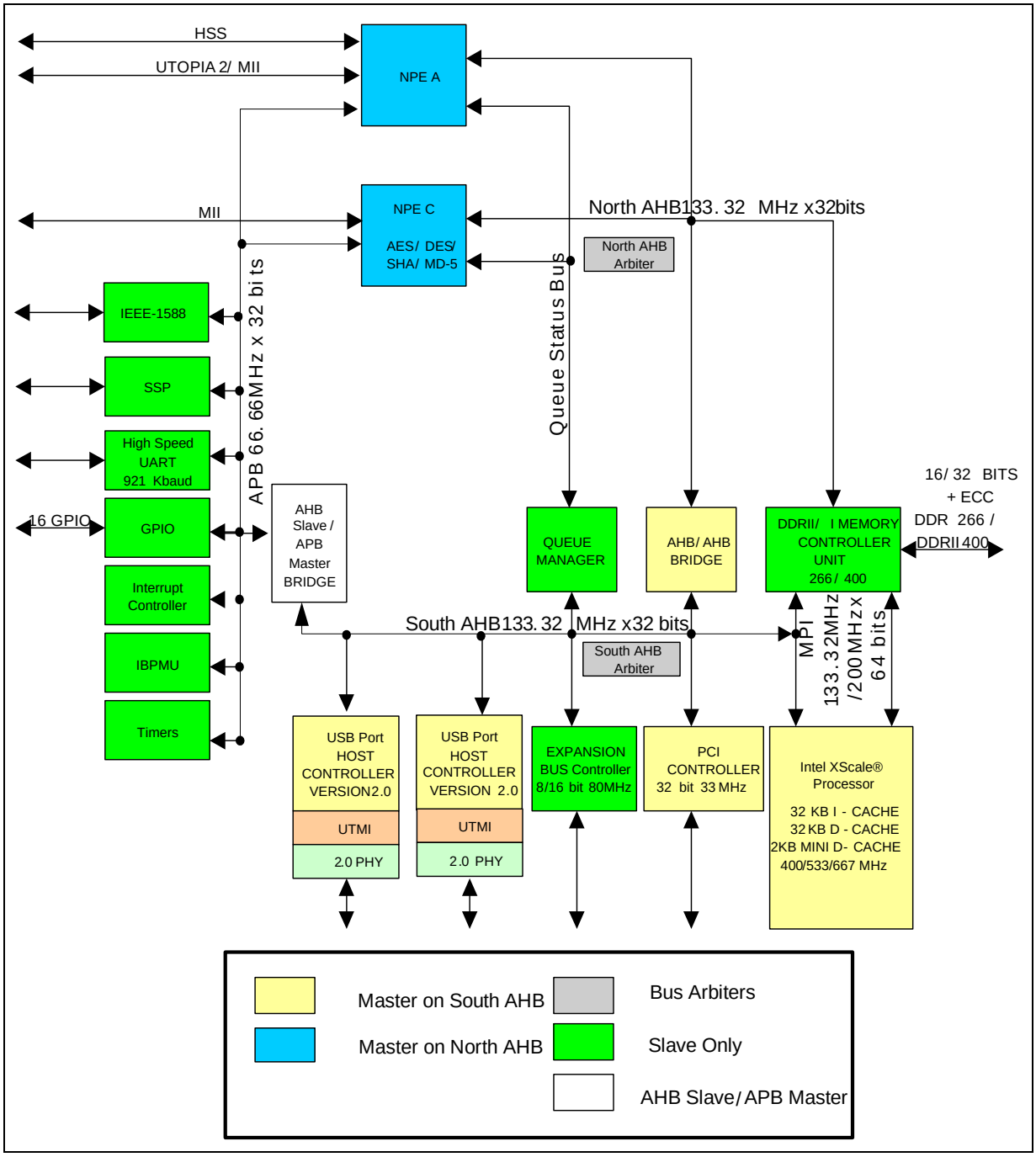


Figure 2. Intel® IXP433 Network Processor Block Diagram

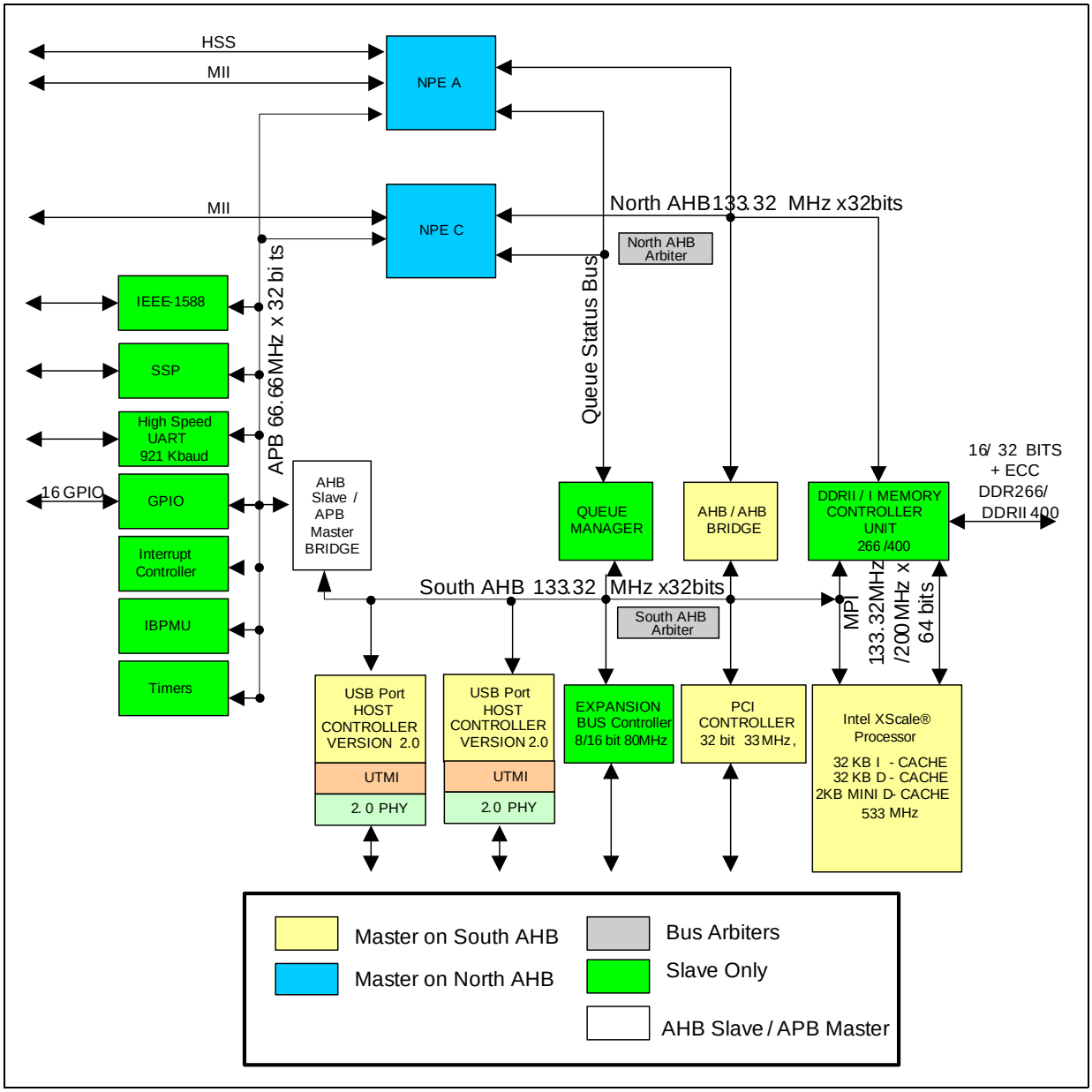


Figure 3. Intel® IXP432 Network Processor Block Diagram

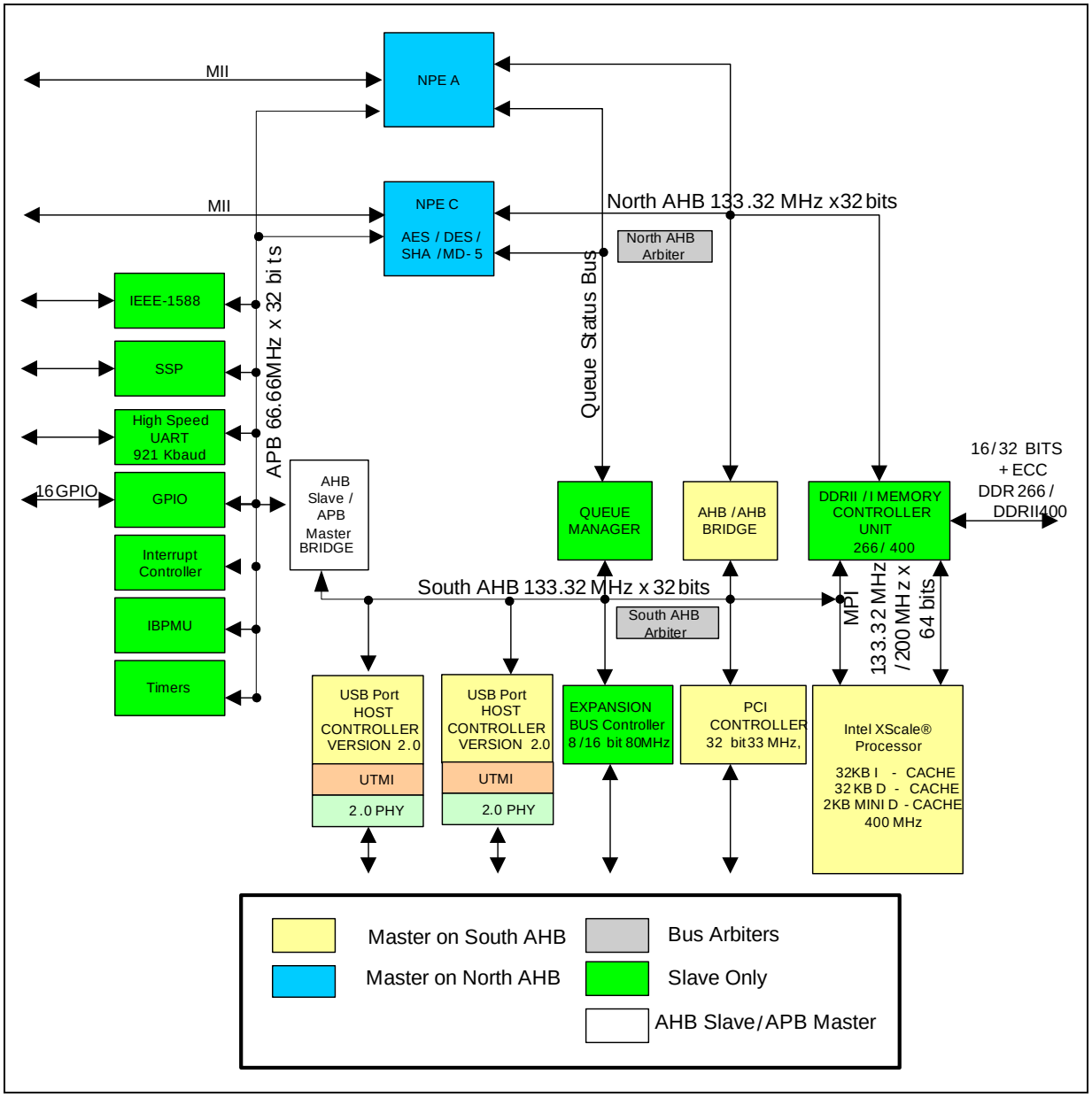


Figure 4. Intel® IXP431 Network Processor Block Diagram

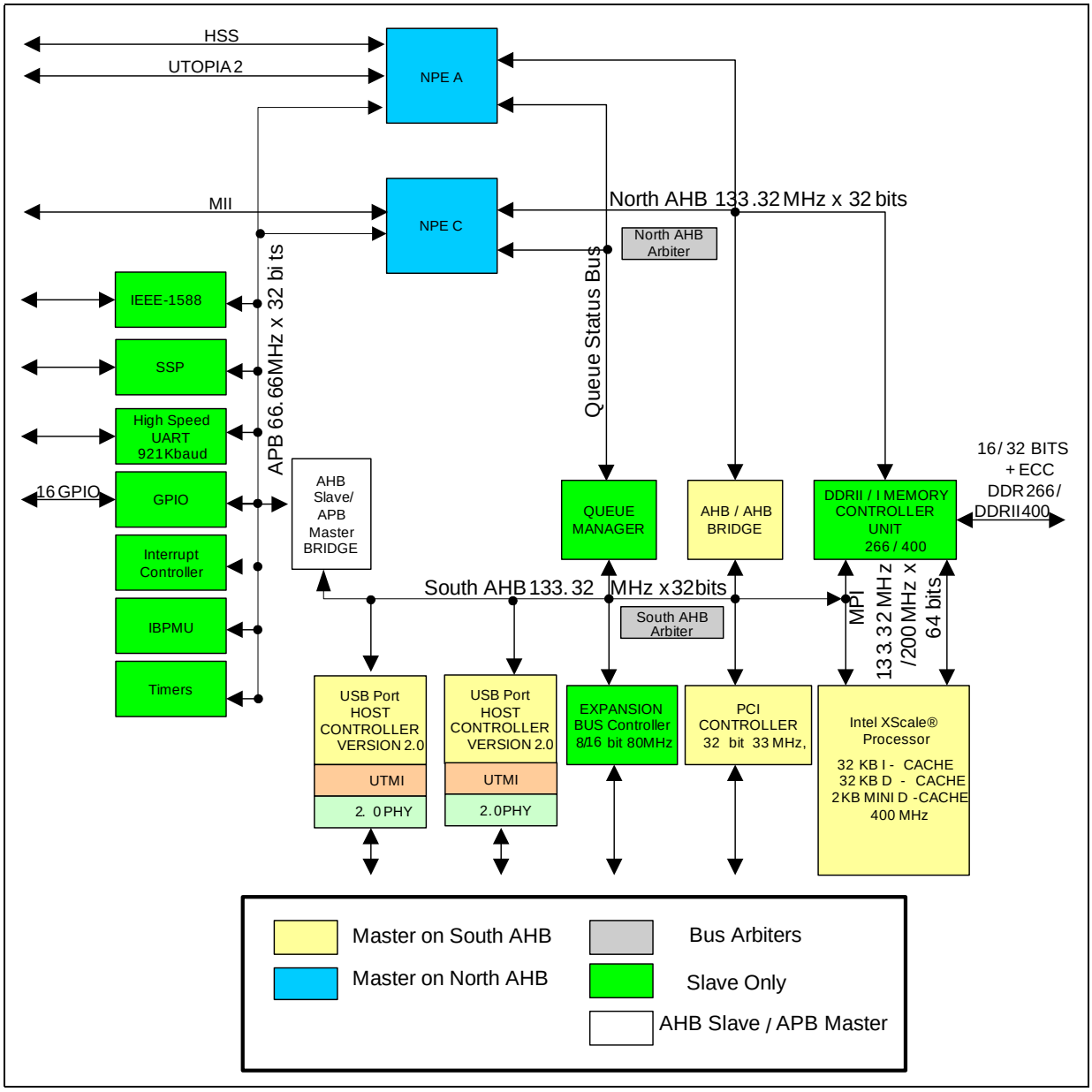
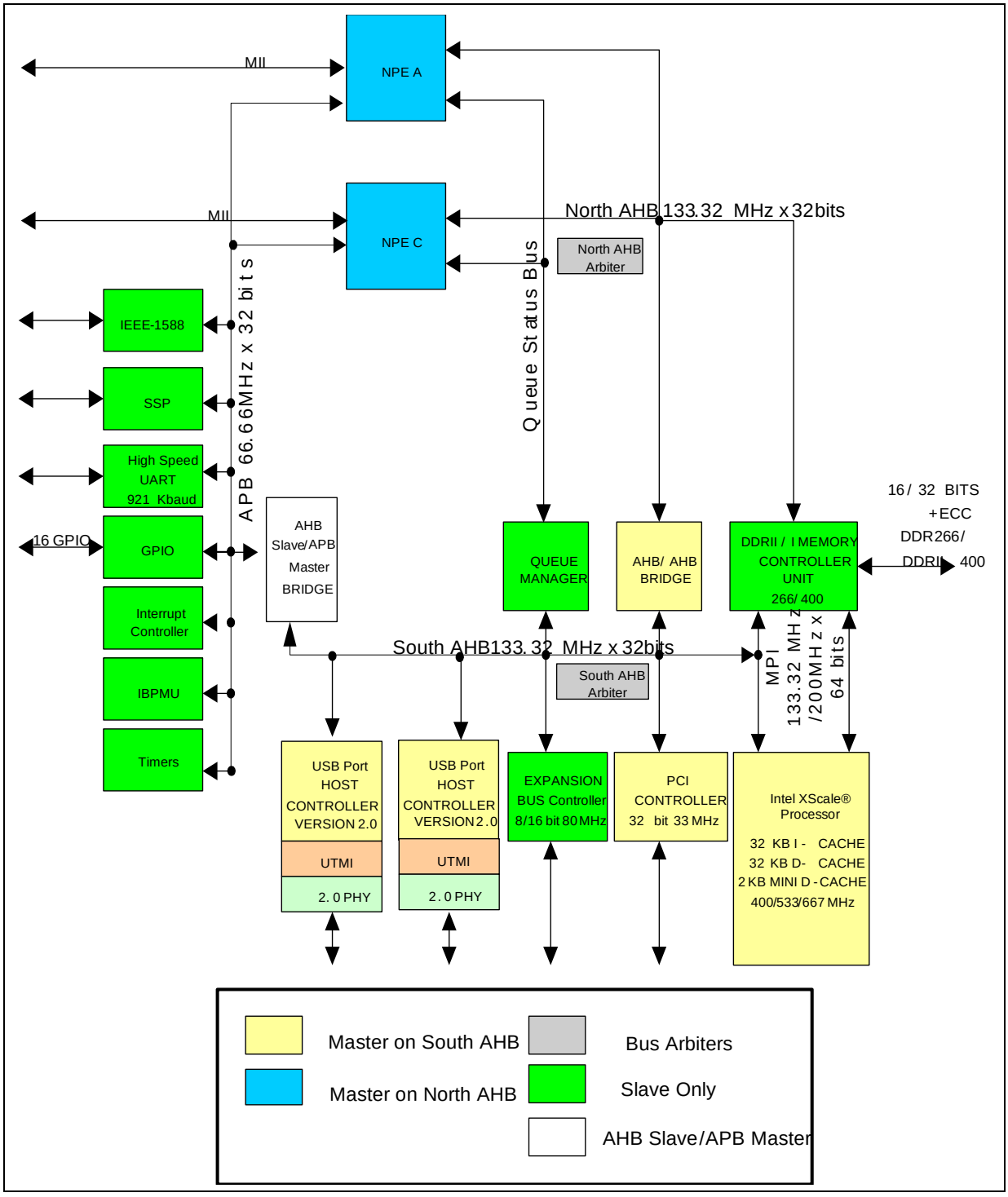


Figure 5. Intel® IXP430 Network Processor Block Diagram



2.1 Key Functional Units

The following sections describe the functional units and their interactions in the system:

Note: Unless otherwise specified, the functional descriptions apply to all the five network processors in the Intel® IXP43X Product Line:

- Intel® IXP435 Network Processor
- Intel® IXP433 Network Processor
- Intel® IXP432 Network Processor
- Intel® IXP431 Network Processor
- Intel® IXP430 Network Processor

Refer to the *Intel® IXP43X Product Line of Network Processors Datasheet* for a detailed list of features that are supported in the IXP43X network processors categorized by the processor model.

2.1.1 Network Processor Engines (NPEs)

The Network Processor Engines (NPEs) are dedicated function processors containing hardware coprocessors, that are integrated into the IXP43X network processors. The NPEs are used to offload processing function required by the Intel XScale processor.

These NPEs are high performance, hardware multi-threaded processors with additional local hardware assist functionality used to offload highly processing intensive functions such as MII (MAC), CRC checking/generation, AAL segmentation and re-assembly, AES, AES-CCM, DES, DES3, SHA-1/256/384/512, MD5, and so on.

All instruction code is stored locally for the NPEs; each of them has a dedicated instruction memory bus and data memory bus.

These NPEs support processing of the dedicated peripherals that can include:

- One UTOPIA Level 2 (Universal Test and Operation PHY Interface for ATM) interface
- One High-Speed Serial (HSS) interface
- Up to two Media-Independent Interface (MII)

There are several possible combination of interfaces for the NPEs contained in the IXP43X network processors. These interface combinations are configured by setting the expansion bus Section 12.5.7, “EXP_UNIT_FUSE_RESET” register during reset. Detailed information on settings can be found in “EXP_UNIT_FUSE_RESET”.

The NPE core is a hardware multi-threaded processor engine separate from, but operating in parallel with the Intel XScale processor. The NPE is used to off-load and accelerate network data packet processing that will otherwise take processing cycle time on the Intel XScale processor. Each NPE core is a 133.32MHz processor core that has self-contained instruction memory and self-contained data memory that operate in parallel. Each NPE core has 4 K x 29bit of instruction memory and 4 K words of data memory.

In addition to having separate instruction/data memory, the NPE core supports hardware multi-threading with support for multiple contexts. The support of hardware multi-threading creates an efficient processor engine with minimal processor stalls due to the ability of the processor to switch contexts in a single clock cycle, based on a prioritized/preemptive basis. The prioritized/pre-emptive nature of the context switching allows time-critical applications to be implemented in a low-latency fashion; this is required when processing multi-media applications.

The NPE core also connects to several hardware-based coprocessors that are used to implement functions that are difficult for a processor to implement. These functions include:

- HSS Serialization/ De-serialization
- DES/3DES/AES
- MD-5
- UTOPIA Level 2 Framing
- CRC checking/generation
- SHA-1/256/384/512
- HDLC bit stuffing/de-stuffing
- Media Access Controller functionality

These coprocessors are implemented in hardware, enabling the coprocessors and the NPE processor core to operate in parallel.

The combined forces of the hardware multi-threading, independent instruction memory, independent data memory, and parallel processing contained on the NPE allows the Intel XScale processor to be utilized for application purposes. The multi-processing capability of the peripheral interface functions allows unparalleled performance to be achieved by the application running on the Intel XScale processor.

Note: All the described NPE functions require Intel supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your Intel local sales representative.

2.1.2 Internal Bus

The internal bus architecture of the IXP43X network processors is designed to allow parallel processing to occur and to isolate bus utilization, based on particular traffic patterns. The bus is segmented into four major buses:

- North - Advanced High Performance Bus (AHB)
- South - AHB
- Memory Port Interface
- Advanced Peripheral Bus (APB)

2.1.2.1 North AHB

The North AHB is a 133.32MHz (4*OSC_IN), 32-bit bus that can be mastered by the NPE A, and NPE C. The targets of the North AHB can be the DDRI/DDRII memory controller or the AHB/AHB bridge. The AHB/AHB bridge allows the NPEs to access peripherals and internal targets on the South AHB.

Data transfers by the NPEs on the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB bridge can be **posted** while writing or **split** while reading.

When a transaction is **posted**, a master on the North AHB requests a write to a peripheral on the South AHB. If the AHB/AHB Bridge has a free FIFO location, the write request is transferred from the master on the North AHB to the AHB/AHB bridge. The AHB/AHB bridge completes the write on the South AHB, when it can obtain access to the peripheral on the South AHB. The North AHB is released to complete another transaction.

When a transaction is **split**, a master on the North AHB requests a read of a peripheral on the South AHB. If the AHB/AHB bridge has a free FIFO location, the read request is transferred from the master on the North AHB to the AHB/AHB bridge. The AHB/AHB bridge completes the read on the South AHB, when it can obtain access to the peripheral on the South AHB.

Once the AHB/AHB bridge has obtained the read information from the peripheral on the South AHB, the AHB/AHB bridge notifies the arbiter on the North AHB, that the AHB/AHB bridge has the data for the master that requested the **split** transfer. The master on the North AHB that requested the split transfer arbitrates for the North AHB and transfers the read data from the AHB/AHB bridge. The North AHB is released to complete another transaction as the North AHB master that requested the **split** transfer waits for the data to arrive.

These **posting** and **splitting** transfers allow control of the North AHB to be given to another master on the North AHB, thus enabling the North AHB to achieve maximum efficiency. Transfers to the AHB/AHB bridge are assumed to be small and infrequent, relative to the traffic passed between the NPEs on the North AHB and the DDRI/DDRII SDRAM.

Arbitration on the North AHB is round-robin. Each transaction cannot be longer than an eight word bursts. This implementation promotes fairness within the system.

2.1.2.2 South AHB

The South AHB is a 133.32MHz (4*OSC_IN), 32-bit bus that can be mastered by the Intel XScale processor, PCI controller, USB Host Controllers, and the AHB/AHB bridge. The targets of the South AHB Bus can be the DDRI/DDRII SDRAM, PCI Controller, Queue Manager, Expansion Bus, or the AHB/APB bridge.

Accessing across the APB/AHB bridge allows interfacing with peripherals attached to the APB. The Expansion bus can be configured to support split transfers.

Arbitration on the South AHB is round-robin. Each transaction cannot be longer than eight word bursts. This implementation promotes fairness within the system.

2.1.2.3 Memory Port Interface

The Memory Port Interface provides a dedicated interface between the Intel XScale processor and the DDRI/DDRII SDRAM. The Memory Port Interface operates at 133.32MHz when DDRI SDRAM is used and 200MHz when DDRII SDRAM is used.

The Memory Port Interface stores memory transactions from the Intel XScale processor that have not been processed by the Memory Controller. The Memory Port Interface supports eight core processor read transactions up to 32 bytes each. The total of core DCU [4 - load requests to unique cache lines], IFU [2 - prefetch], IMM [1 - tablewalk], DMM [1 - tablewalk] equals the maximum number of outstanding transaction the Core Processor Bus Controller can support.

The Memory Port Interface also supports eight Intel XScale processor-posted write transactions up to 16 bytes each.

Arbitration on the Memory Port Interface is not required due to no contention with other masters. Arbitration exists in the DDRI/DDRII memory controller between all the main internal interfaces.

2.1.2.4 APB Bus

The APB Bus is a 66.66MHz (2*OSC_IN), 32-bit bus that can be mastered by the AHB/APB bridge alone. The targets of the APB bus can be:

- Timers
- The internal bus performance monitoring unit (IBPMU)
- GPIOs
- Synchronous Serial Protocol Interface
- UART
- All NPEs
- Interrupt controller
- IEEE 1588 Hardware Assist

The APB interface is also used for NPE code download and configuration.

No arbitration is required due to single master implementation.

2.1.3 MII Interfaces

The IXP43X network processors can be configured to support up to two MII industry-standard interfaces. These interfaces are integrated with IXP43X network processors with separate Media-Access Controllers and Network Processing Engines.

The independent NPEs and MACs allow parallel processing of data traffic on the MII interfaces and off-loading of processing required by the Intel XScale processor. The IXP43X network processors are compliant with the IEEE 802.3 specification.

In addition to the MII interfaces, the IXP43X network processors include a single management data interface that is used to configure and control PHY devices that are connected to the MII interfaces.

TMII (Turbo Media Independent Interface), also called Turbo MII, is used to increase the MII clock from 25 MHz to 50 MHz. The purpose of the Turbo MII is to enhance LAN throughput performance by doubling the MII clock rate. TMII is supported in Intel® IXP400 Software Release 3.01. Please refer to the *Enabling TMII Hardware on Intel® IXP435 Product Line of Networks Processors Application Note* for more information.

Note: All the described NPE functions require Intel supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your Intel local sales representative.

2.1.4 UTOPIA Level 2

The integrated UTOPIA Level 2 interface works with a network processing engine core for several of the IXP43X network processors. The pins of the UTOPIA Level 2 interface are multiplexed with one of the MII interfaces.

The UTOPIA Level 2 interface supports a single or a multiple physical interface configuration with cell level or octet level handshaking. The network processing engine handles segmentation and reassembly of ATM cells, CRC checking/generation, and transfer of data to/from memory. This allows parallel processing of data traffic on the UTOPIA Level 2 interface, off-loading processor overhead required by the Intel XScale processor.

The IXP43X network processors are compliant with the ATM Forum's *UTOPIA Level 2 Specification*, Revision 1.0. The UTOPIA Level 2 interface of the IXP43X network processors is an 8-bit interface.

Note: All the described NPE functions require Intel supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.5 Universal Serial Bus (USB) Version 2.0 Interfaces

The Universal Serial Bus (USB) Host functionality is implemented on the IXP43X network processors. The function being performed is defined by the USB 2.0 specification, maintained by www.usb.org, and the interfaces are EHCI-compliant, as defined by Intel.

List of supported features are:

- Host function
- Low speed interface
- Full speed interface
- High speed interface
- EHCI register interface
- UTMI+ Level 2 Compliant

The following is a partial list of features *not* supported:

- Device function
- OTG function

Note: There is a performance degradation impact on USB2.0 high speed mode while utilizing WinCE* and potentially other operating systems when the Intel XScale processor is configured in little-endian data coherent mode of operation.

2.1.6 PCI Controller

The PCI controller of the IXP43X network processors is compatible with the *PCI Local Bus Specification*, Rev. 2.2. The PCI interface is 33.33MHz 32-bit compatible bus and capable of operating as either a host or an option.

When the PCI Controller is configured as a host, an internal PCI arbiter can be utilized to allow up to four devices to be connected to the IXP43X network processors without an external arbiter.

Features of the PCI Controller include:

- Conforms to PCI Local Bus Specification Revision 2.2
- 32-bit, 0-33MHz PCI bus operation
- Provides initiator (master) and target (slave) PCI interfaces
- Provides AHB-to-PCI and PCI-to-AHB DMA channels
- Includes PCI bus arbiter supporting up to 4 external PCI masters using round-robin arbitration.
- Access to PCI Configuration registers from PCI and AHB busses
- Provides interrupt to processor to indicate transaction errors on the PCI or AHB bus
- Provides interrupt to processor for DMA complete and DMA error
- Provides doorbell interrupt generation capability to PCI and AHB agents
- Byte, half word, word single reads/writes, burst word reads/writes supported on AHB and PCI busses
- Generates memory, I/O, and configuration cycles as PCI master
- Provides AHB masters with full access to the 4Gbyte PCI address space
- Provides PCI-to-AHB address translation to map PCI accesses to AHB address space

Note: The Intel® IXP42X and IXP46X Network Processors support up to 66MHz PCI operation while the Intel® IXP43X Product Line of Network Processors supports up to 33MHz PCI operation.

2.1.7 DDRII/DDRI Memory Controller

The IXP43X network processors integrate a high-performance, multi-ported Memory Controller Unit (MCU) to provide a direct interface between the IXP43X network processors and their local memory subsystem. The MCU supports:

- DDRI 266 or DDRII 400 SDRAM
- 128/256/512-Mbit, 1-Gbit DDRI SDRAM technology support
- 256/512-Mbit DDRII SDRAM technology support
- Only unbuffered DRAM support (No registered DRAM support)
- Dedicated port for the Intel XScale processor to DDRII/DDRI SDRAM (supports critical word first reads)
- Between 32 Mbytes and 1Gbytes of 32-bit DDRI SDRAM
- Between 64 Mbytes and 512MB of 32-bit DDRII SDRAM
- 16 Mbytes of 16-bit DDRI SDRAM (support 128 Mbit technology only)
- 32 / 64 Mbytes of 16-bit DDRII SDRAM (support 256 Mbit / 512 Mbit technology)
- Two AHB ports for access from units other than the Intel XScale processor (no critical word first support)
- All MMR access must go through the South AHB port
- Single-bit error correction, multi-bit detection support (ECC)
- 32, 40-bit wide Memory Interfaces (non-ECC and ECC support), and 16-bit wide Memory Interfaces (non-ECC)

The DDRII/DDRI SDRAM interface provides direct connection to a high bandwidth and reliable memory subsystem. The DDRII/DDRI SDRAM interface comprises a 16-bit/32-bit-wide data path to support up to 1.6 Gbps throughput.

An 8-bit Error Correction Code (ECC) across each 32-bit word improves system reliability. It does not support ECC for 16-bit wide interface option.

Note: ECC is also referred to as CB in many DIMM specifications. The pins used for ECC on the IXP43X network processors are called DDR_CB[7:0] or D_CB[7:0]. Although the memory controller is 32-bits when ECC is enabled, the ECC generated is 8-bits. The ECC circuitry is designed to operate always on a 64 bit word and when operating in a 32 bit mode, the upper 32 bits are driven to zero internally. To summarize, the full 8 bits of ECC is stored and read from a memory array for the ECC logic to work. An 8-bit-wide memory is used while implementing ECC.

The MCU supports two physical banks of DDRII/DDRI SDRAM. The MCU has support for unbuffered DDRI 266 and DDRII 400 in the form of discrete chips only.

The MCU supports a memory subsystem ranging from

- 32 Mbyte to 1 Gbyte for 32-bit memory systems for DDRI SDRAM
- From 64 Mbyte to 512 Mbyte for 32-bit memory systems for DDRII SDRAM
- 16 Mbyte for 16-bit memory systems for DDRI SDRAM (non-ECC),
- 32 / 64 Mbyte for 16-bit memory systems for DDRII SDRAM (non-ECC)

An ECC or non-ECC system can be implemented using x8, or x16 devices. Table 192, Table 193, Table 194 and Table 195 in Section 11.2.2.2, "DDR-I/II SDRAM Bank Sizes and Configurations" illustrate the supported DDRII/DDRI SDRAM configurations.

The 128/256/512-Mbit, 1-Gbit DDR SDRAM technology devices comprise four internal leaves. The MCU controls the leaf selects within 128/256/512-Mbit, 1-Gbit DDR SDRAM technology devices by toggling DDR_BA[0] and DDR_BA[1].

The two DDR SDRAM chip, enables the DDR_CS_N[1:0] support a DDR SDRAM memory subsystem consisting of two physical banks. The base address for the two contiguous banks are programmed in the DDR SDRAM Base Register (SDBR) and is aligned to a 16-Mbyte boundary. The size of each DDRI SDRAM bank is programmed with the DDR SDRAM boundary registers (SBR0 and SBR1).

The memory controller internally interfaces with the North AHB, South AHB, and Memory Port Interface with independent interfaces. This architecture allows DDR SDRAM transfers to be interleaved and pipelined to achieve maximum efficiency.

2.1.8 Expansion Interface

The expansion interface allows easy and glueless connection with the peripheral devices. It also provides input information for device configuration after reset.

Some of the peripheral device types are SRAM, flash, ATM control interfaces, and DSPs used for voice applications. Some voice configurations can be supported by the HSS interface and the Intel XScale processor implementing voice-compression algorithms.

The expansion interface functions support 8 or 16 bit data operation and allows an address range of 512 bytes to 16 Mbytes, using 24 address lines for each of the four independent chip selects.

Access to the expansion bus interface is completed in five phases. Each of the five phases can be lengthened or shortened by setting various configuration registers on a per-chip-select basis. This feature allows the IXP43X network processors to connect to a wide variety of peripheral devices with varying speeds.

The expansion interface supports Intel or Motorola* microprocessor style bus cycles. The bus cycles can be configured to be multiplexed address/data cycles or separate address/data cycles for each of the four chip-selects.

The expansion interface is an asynchronous interface to externally connected chips. To operate, you must supply a clock to the expansion interface of the IXP43X network processors. This clock can be driven from GPIO 15 or an external source. The maximum clock rate that the expansion interface can accept is 80 MHz.

At the deassertion of reset, the address bus is used to capture configuration information from the levels that are applied to the pins at this time. External pull-up/pull-down resistors are used to tie the signals to particular logic levels. Refer to Chapter 12.0, "Expansion Bus Controller" for additional information.

2.1.9 High Speed Serial Interface

The high speed serial interface (HSS) is a six signal interface that supports serial transfer speeds from 512 KHz to 8.192 MHz, for some of the IXP43X network processors.

This interface allows direct connection of up to four T1/E1 framers and CODEC/SLICs to the IXP43X network processors. The high speed serial interface is capable of supporting various protocols, based on the implementation of the code developed for the network processor engine core.

Refer to the *Intel® IXP400 Software Programmer's Guide* for a list of supported protocols.

2.1.10 UART

The UART interface is a 16550-compliant UART with the exception of transmit and receive buffers. The Transmit and receive buffers are 64 bytes deep versus the 16 bytes required by the 16550 UART specification.

The interface can be configured to support speeds from 1,200 Baud to 921 Kbaud. The interface supports configurations of:

- Five, six, seven, or eight data-bit transfers
- One or two stop bits
- Even, odd, or no parity

The request-to-send (RTS_N) and clear-to-send (CTS_N) modem control signals are also available with the interface for hardware flow control.

2.1.11 GPIO

16 GPIO pins are supported by the IXP43X network processors. Table 3 shows the functionality of the GPIO pins. GPIO pins 0 through 15 can be configured to be general-purpose input or general-purpose output. Additionally, GPIO pins 0 through 12 can be configured to be an interrupt input.

GPIO Pin 1 can also be configured as a clock input for an external USB 2.0 Host Bypass clock. When spread spectrum clocking (SSC) is used, an external clock should be used as the source for the USB 2.0 Host clock. See bit 8 (USB Clock) in Table 215, "Configuration Register 0 Description" on page 592 for more details.

GPIO Pin 14 and GPIO 15 can also be configured as a clock output. The output-clock configuration can be set at various speeds, up to 33.33 MHz, with various duty cycles. GPIO Pin 14 is configured as an input, upon reset. GPIO Pin 15 is configured as a clock output, upon reset. GPIO Pin 15 can be used to clock the expansion interface, after reset.

Several other GPIO pins can serve as an alternate function. These functions are outlined in Table 3 below.

Table 3. GPIO Function Table

GPIO Pin Number	GPIO Function	Alternate Function
0	General purpose input/output or interrupt source	Reserved
1†	General purpose input/output or interrupt source	External USB 48 MHz Bypass Clock
2	General purpose input/output or interrupt source	Reserved
3	General purpose input/output or interrupt source	Reserved
4	General purpose input/output or interrupt source	Reserved
5	General purpose input/output or interrupt source	Reserved
6	General purpose input/output or interrupt source	Reserved
7	General purpose input/output or interrupt source	Auxiliary IEEE-1588 Master Snapshot
8	General purpose input/output or interrupt source	Auxiliary IEEE-1588 Slave Snapshot

Table 3. GPIO Function Table

GPIO Pin Number	GPIO Function	Alternate Function
9:12	General purpose input/output or interrupt source	Reserved
13	General purpose input/output	Reserved
14	General purpose input/output or output clock	Output clock 14
15	Output Clock or General purpose input/output	Output clock 15
†	When a spread spectrum clock is used, GPIO Pin 1 should be configured as an input clock for USB Host. See Table 215 for more details.	

2.1.12 Internal Bus Performance Monitoring Unit (IBPMU)

The IXP43X network processors comprise a performance monitoring unit that can be used to capture predefined events within the system outside of the Intel XScale processor. These features aid in measuring and monitoring various system parameters that contribute to the overall performance of the processor.

The Performance Monitoring (PMON) facility provided comprises:

- Eight Programmable Event Counters (PECx) clocked by the AHB clock (133.32 MHz)
- Eight Programmable Event Counters (MPECx) clocked by the MCU clock (133.32 MHz/200 MHz)
- Previous Master/Slave Register
- Event Selection Multiplexor
- Simultaneous event counting

The programmable event counters are 27 bits wide. Each counter can be programmed to observe one event from a defined set of events. An event consists of a set of parameters that define a start and stop condition.

The monitored events are selected by programming the Event Select Registers (ESR).

2.1.13 Interrupt Controller

The IXP43X network processors comprise up to 64 interrupt sources to allow an extension of the FIQ and IRQ interrupt sources of the Intel XScale processor. These sources can originate from some external GPIO pins, internal peripheral interfaces, or internal logic.

The interrupt controller can configure each interrupt source as an FIQ, IRQ, or disabled. The interrupt sources tied to Interrupt 0 to 7 can be prioritized. The remaining interrupts are prioritized in ascending order. For example, Interrupt 8 has a higher priority than 9, 9 has a higher priority than 10, and 30 has a higher priority that 31.

An additional level of priority can be set for interrupts 32 through 63. This priority setting gives any interrupt between 32 through 63 priority, over interrupts 0 through 31.

2.1.14 Timers

The IXP43X network processors comprise four internal timers operating at 66.66 MHz to allow task scheduling and prevent software lock-ups. The device has four 32-bit counters:

- Watch-Dog Timer
- Timestamp Timer
- Two general-purpose Timers

The Timestamp Timer and the two general-purpose timers have the optional ability to use a prescaled clock. A programmable prescaler can be used to divide the input clock by a 16-bit value. The input clock can be either the APB clock (66.66 MHz) or a 20-ns version of the APB clock (50 MHz). All timers use the APB clock by default.

The 16-bit pre-scale value ranges from divide by 2 to 65,536 and results in a new clock enable available for timers that range from 33.33 MHz down to 1,017.26 Hz.

The Timestamp Timer also contains a 32-bit compare register that allows an interrupt to be created at times other than time 0.

2.1.15 IEEE-1588 Hardware Assist

In a distributed control system containing multiple clocks, individual clocks tend to drift apart. Some kind of correction mechanism is necessary to synchronize the individual clocks to maintain global time, which is accurate to some clock resolution. The IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems can be used for this purpose.

The IEEE 1588 standard defines several messages that can be used to exchange timing information. The hardware assist logic required to achieve precision clock synchronization using the IEEE 1588 standard is left to implementation. The IXP43X network processors consist of this IEEE 1588 hardware-assist logic on three of the MII interfaces. Using the hardware assist logic along with software running on the Intel XScale processor, a full source- or sink-capable IEEE-1588 compliant network node can be implemented.

2.1.16 Synchronous Serial Protocol Interface

The IXP43X network processors comprise a dedicated Synchronous Serial Protocol (SSP) interface. The SSP interface is a full-duplex synchronous serial interface. It can connect to a variety of external analog-to-digital (A/D) converters, audio and telecom CODECs, and many other devices that use serial protocols for transferring data.

It supports synchronous serial protocol (SSP) of National Microwire* and Texas Instruments*, and serial peripheral interface (SPI) protocol of Motorola*.

The SSP operates in master mode with the attached peripheral functions as a slave, and supports serial bit rates from 7.2 Kbps to 1.8432 Mbps using the on-chip, 3.6864-MHz clock. Serial data formats can range from 4 to 16 bits in length. Two on-chip register blocks function as independent FIFOs for data, one for each direction. The FIFOs are 16 entries deep x 16 bits wide. Each 32-bit word from the system fills one entry in a FIFO using the lower half 16-bits of a 32-bit word.

2.1.17 AES/DES/SHA/MD-5

The IXP43X network processors implement on chip hardware acceleration for underlying security and authentication algorithms.

The encryption/decryption algorithms supported are AES, single pass AES-CCM, DES, and triple DES. These algorithms are commonly found while implementing IPSec, VPN, WEP, WEP2, WPA, and WPA2.

The authentication algorithms supported are MD-5, SHA-1, SHA-256, SHA-384, and SHA-512. Inclusion of SHA-384 and SHA-512 allows 256-bit key authentication to pair up with 256-bit AES support.

Note: All the described NPE functions require Intel supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your local sales representative.

2.1.18 Queue Manager

The Queue Manager provides a means for maintaining coherency for data handling between various processor cores in the IXP43X network processors (NPE to NPE, NPE to Intel XScale processor, and so on.). It maintains the queues as circular buffers in an embedded 8-Kbyte SRAM. The Queue Manager also implements the status flags and pointers required for each queue.

The Queue Manager manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The Queue Manager interfaces include an Advanced High-performance Bus (AHB) interface to the NPEs and Intel XScale processor or any other AHB bus master, a Flag Bus interface, an event bus to the NPE condition select logic, and two interrupts to the Intel XScale processor.

The AHB interface is used for configuration of the Queue Manager and provides access to queues, queue status, and SRAM. Individual queue status for queues 0-31 is communicated to the NPEs through the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs through the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

2.2 Intel XScale® Processor

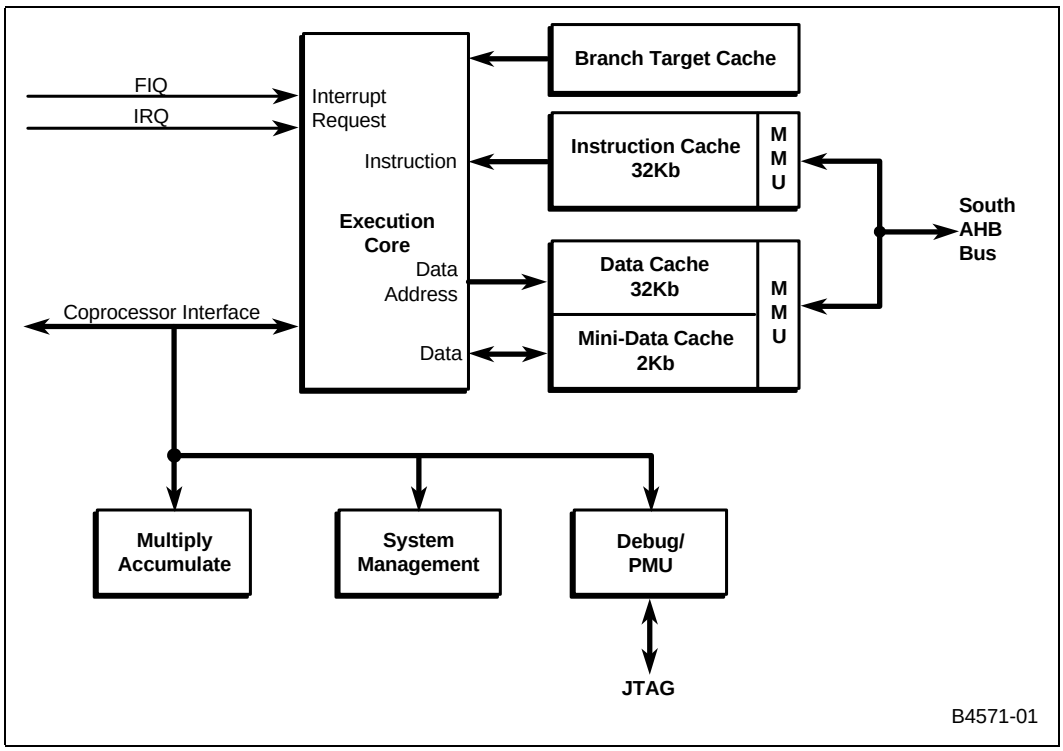
The Intel XScale® technology is compliant with Intel® StrongARM* Version 5TE instruction set architecture (ISA). The Intel XScale processor as shown in Figure 6 is designed with Intel, 0.13-µ production semiconductor process technology. This process technology with the compactness of the Intel® StrongARM* RISC ISA enables the Intel XScale processor to operate over a wide speed and power range, producing industry-leading performance.

The Intel XScale processor features include:

- Seven/eight-stage super-pipeline promotes high-speed, efficient core performance
- 128-entry branch target buffer keeps pipeline filled with statistically correct branch choices
- 32-entry instruction memory-management unit for logical-to-physical address translation, access permissions, and Instruction-Cache (I-cache) attributes
- 32-entry data-memory management unit for logical-to-physical address translation, access permissions, Data-Cache (D-Cache) attributes
- 32-Kbyte instruction cache can hold entire programs, preventing core stalls caused by multi-cycle memory accesses
- 32-Kbyte data cache reduces core stalls caused by multi-cycle memory accesses
- 2-Kbyte mini-data cache for frequently changing data streams avoids "thrashing" of the D-cache
- Four-entry, fill-and-pend buffers to promote core efficiency by allowing "hit-under-miss" operation with data caches
- Eight-entry write buffer allows the core to continue execution as data is written to memory

- Multiple-accumulate coprocessor that can do two simultaneous, 16-bit, SIMD multiplies with 40-bit accumulation for efficient, high-quality media and signal processing
- Performance monitoring unit (PMU) furnishing four 32-bit event counters and one 32-bit cycle counter for analysis of hit rates, and so on.
  This PMU is for the Intel XScale processor only. An additional PMU is supplied for monitoring of internal bus performance.
- JTAG debug unit that uses hardware break points and 256-entry trace history buffer (for flow-change messages) to debug programs

Figure 6. Intel XScale® Processor Block Diagram



2.2.1 Super Pipeline

The super pipeline is composed of integer, multiply-accumulate (MAC), and memory pipes.

The integer pipe has seven stages:

- Branch Target Buffer (BTB)/Fetch 1
- Fetch 2
- Decode
- Register File/Shift
- ALU Execute
- State Execute
- Integer Writeback

The memory pipe has eight stages:

- The first five stages of the Integer pipe (BTB/Fetch 1 through ALU Execute)..... then ends with the following memory stages:
- Data Cache 1
- Data Cache 2
- Data Cache Writeback

The MAC pipe has six to nine stages:

- The first four stages of the Integer pipe (BTB/Fetch 1 through Register File/ Shift)..... then ends with the following MAC stages:
- MAC 1
- MAC 2
- MAC 3
- MAC 4
- Data Cache Writeback

The MAC pipe supports a data-dependent early terminate where stages MAC 2, MAC 3, and/or MAC 4 are bypassed.

Deep pipes promote high instruction execution rates only when a means exists to successfully predict the outcome of branch instructions. The branch target buffer provides such a means.

2.2.2 Branch Target Buffer

Each entry of the 128-entry Branch Target Buffer (BTB) contains the address of a branch instruction, the target address associated with the branch instruction, and previous history of the branch being taken or not taken. The history is recorded as one of the following four states:

- Strongly taken
- Weakly taken
- Weakly not taken
- Strongly not taken

The BTB can be enabled or disabled through Coprocessor 15, Register 1.

When the address of the branch instruction hits in the BTB and its history is strongly or weakly taken, the instruction at the branch target address is fetched. When its history is strongly or weakly not taken, the next sequential instruction is fetched. In either case the history is updated.

Data associated with a branch instruction enters the BTB the first time the branch is taken. This data enters the BTB in a slot with a history of strongly not taken (overwriting previous data when present).

Successfully predicted branches avoid any branch latency penalties in the super pipeline. Unsuccessfully predicted branches result in a four-to-five-cycle, branch-latency penalty in the super pipeline.

2.2.3 Instruction Memory Management Unit

For instruction pre-fetches, the Instruction Memory Management Unit (IMMU) controls logical-to-physical address translation, memory access permissions, memory-domain identifications, and attributes governing operation of the instruction cache.

The IMMU contains a 32-entry, fully associative instruction-translation, look-aside buffer (ITLB) that has a round-robin replacement policy. ITLB entries zero through 30 can be locked.

When an instruction pre-fetch misses in the ITLB, the IMMU invokes an automatic table walk mechanism that fetches an associated descriptor from memory and loads it into the ITLB. The descriptor contains information for logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes governing operation of the I-cache. The IMMU then continues the instruction pre-fetch by using the address translation just entered into the ITLB. When an instruction pre-fetch hits in the ITLB, the IMMU continues the pre-fetch using the address translation already resident in the ITLB.

Access permissions for each up to 16 memory domains can be programmed. When an instruction pre-fetch is attempted to an area of memory in violation of access permissions, the attempt is aborted and a pre-fetch abort is sent to the core for exception processing. The IMMU and DMMU can be enabled or disabled together.

2.2.4 Data Memory Management Unit

For data fetches, the Data Memory Management Unit (DMMU) controls logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes governing operation of the data cache or mini-data cache and write buffer. The DMMU contains a 32-entry, fully associative data-translation, look-aside buffer (DTLB) that has a round-robin replacement policy. DTLB entries 0 through 30 can be locked.

When a data fetch misses in the DTLB, the DMMU invokes an automatic table walk mechanism that fetches an associated descriptor from memory and loads it into the DTLB. The descriptor contains information for logical-to-physical address translation, memory-access permissions, memory-domain identifications, and attributes (governing operation of the D-cache or mini-data cache and write buffer).

The DMMU continues the data fetch by using the address translation just entered into the DTLB. When a data fetch hits in the DTLB, the DMMU continues the fetch using the address translation already resident in the DTLB.

Access permissions for each of up to 16 memory domains can be programmed. When a data fetch is attempted to an area of memory in violation of access permissions, the attempt is aborted and a data abort is sent to the core for exception processing.

The IMMU and DMMU can be enabled or disabled together.

2.2.5 Instruction Cache

The Instruction Cache (I-Cache) can contain high-use, multiple-code segments or entire programs, allowing the Intel XScale processor access to instructions at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory.

The 32-Kbyte I-cache is 32-set/32-way associative, where each set contains 32 ways and each way contains a tag address, a cache line of instructions (eight 32-bit words and one parity bit per word), and a line-valid bit. For each of the 32 sets, 0 through 28 ways can be locked. Unlocked ways are replaceable through a round-robin policy.

The I-cache can be enabled or disabled. Attribute bits within the descriptors contained in the ITLB of the IMMU provide some control over an enabled I-cache.

When a needed line (eight 32-bit words) is not present in the I-cache, the line is fetched critical word first, from memory through a two-level, deep-fetch queue. The fetch queue allows the next instruction to be accessed from the I-cache, but only when its data operands do not depend on the execution results of the instruction being fetched through the queue.

2.2.6 Data Cache

The Data Cache (D-Cache) can contain high-use data such as lookup tables and filter coefficients, allowing the Intel XScale processor access to data at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory.

The 32-Kbyte D-cache is 32-set/32-way associative, where each set contains 32 ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two eight-byte groupings in a line), and one valid bit. For each of the 32 sets, zero through 28 ways can be locked, unlocked, or used as local SRAM. Unlocked ways are replaceable through a round-robin policy.

The D-cache together with the mini-data cache can be enabled or disabled. Attribute bits within the descriptors, contained in the DTLB of the DMMU, provide significant control over an enabled D-cache. These bits specify cache operating modes such as read and write allocate, write-back, write-through, and D-cache versus mini-data cache targeting.

The D-cache and mini-data cache work with the load buffer and pend buffer to provide **hit-under-miss** capability that allows the Intel XScale processor to access other data in the cache after a **miss** is encountered. The D-cache and mini-data cache works in conjunction with the write buffer for data that is to be stored to memory.

2.2.7 Mini-Data Cache

The mini-data cache can contain frequently changing data streams such as MPEG video, allowing the Intel XScale processor access to data streams at core frequencies. This prevents stalls caused by multi-cycle accesses to external memory. The mini-data cache relieves the D-cache of data **thrashing** caused by frequently changing data streams.

The 2-Kbyte, mini-data cache is 32-set/two-way associative, where each set contains two ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two eight-byte groupings in a line), and a valid bit. The mini-data cache uses a round-robin replacement policy, and cannot be locked.

The mini-data cache (together with the D-cache) can be enabled or disabled. Attribute bits contained within a coprocessor register specify operating modes write and/or read allocate, write-back, and write-through.

The mini-data cache (and D-cache) work with the load buffer and pend buffer to provide **hit-under-miss** capability that allows the Intel XScale processor to access other data in the cache after a **miss** is encountered. The mini-data cache (and D-cache) works in conjunction with the write buffer for data that is to be stored to memory.

2.2.8 Fill Buffer and Pend Buffer

The four-entry fill buffer (FB) works with the Intel XScale processor to hold non-cacheable loads until the bus controller can act on them. The FB and the four-entry pend buffer (PB) work with the D-cache and mini-data cache to provide **hit-under-miss** capability, allowing the Intel XScale processor to seek other data in the caches as **miss** data is being fetched from memory.

The FB can contain up to four unique **miss** addresses (logical), allowing four **misses** before the Intel XScale processor is stalled. The PB holds up to four addresses (logical) for additional **misses** to those addresses that are already in the FB. A coprocessor register can specify draining of the fill and pend write buffers.

2.2.9 Write Buffer

The write buffer (WB) holds data for storage to memory until the bus controller can act on it. The WB is eight entries deep, where each entry holds 16 bytes. The WB is constantly enabled and accepts data from the Intel XScale processor, D-cache, or mini-data cache.

Coprocessor 15, Register 1 specifies whether WB coalescing is enabled or disabled. When coalescing is disabled, stores to memory occur in program order regardless of the attribute bits within the descriptors located in the DTLB.

When coalescing is enabled, the attribute bits within the descriptors located in the DTLB are examined to determine when coalescing is enabled for the destination region of memory. When coalescing is enabled in both CP15, R1 and the DTLB, data entering the WB can coalesce with any of the eight entries (16 bytes) and stored to the destination memory region, but possibly out of program order.

Stores to a memory region specified to be non-cacheable and non-bufferable by the attribute bits within the descriptors located in the DTLB causes the Intel XScale processor to stall until the store completes. A coprocessor register can specify draining of the write buffer.

2.2.10 Multiply-Accumulate Coprocessor

For efficient processing of high-quality, media-and-signal-processing algorithms, the Multiply-Accumulate Coprocessor (CP0) provides 40-bit accumulation of 16 x 16, dual-16 x 16 (SIMD), and 32 x 32 signed multiplies. Special MAR and MRA instructions are implemented to move the 40-bit accumulator to two Intel XScale processor-general registers (MAR) and move two Intel XScale processor-general registers to the 40-bit accumulator (MRA). The 40-bit accumulator can be stored or loaded to or from D-cache, mini-data cache, or memory using two STC or LDC instructions.

The 16 x 16 signed multiply-accumulates (MIAxy) multiply either the high/high, low/low, high/low, or low/high 16 bits of a 32-bit Intel XScale processor general register (multiplier) and another 32-bit Intel XScale processor general register (multiplicand) to produce a full, 32-bit product that is sign-extended to 40 bits and added to the 40-bit accumulator.

Dual-signed, 16 x 16 (SIMD) multiply-accumulates (MIAPH) multiply the high/high and low/low 16-bits of a packed 32-bit, Intel XScale processor-general register (multiplier) and another packed 32-bit, Intel XScale processor-general register (multiplicand) to produce two 16-bits products that are both sign-extended to 40 bits and added to the 40-bit accumulator.

The 32 x 32 signed multiply-accumulates (MIA) multiply a 32-bit, Intel XScale processor-general register (multiplier) and another 32-bit,
Intel XScale processor-general register (multiplicand) to produce a 64-bit product where the 40 LSBs are added to the 40-bit accumulator. The 16 x 32 versions of the 32 x 32 multiply-accumulate instructions complete in a single cycle.

2.2.11 Performance Monitoring Unit

The performance monitoring unit (PMU) contains four 32-bit, event counters and one 32-bit, clock counter. The event counters can be programmed to monitor I-cache hit rate, data caches hit rate, ITLB hit rate, DTLB hit rate, pipeline stalls, BTB prediction hit rate, and instruction execution count.

2.2.12 Debug Unit

The debug unit is accessed through the JTAG port. The industry-standard, IEEE 1149.1 JTAG port consists of a test access port (TAP) controller, boundary-scan register, instruction and data registers, and dedicated signals JTG_TMS, JTG_TDI, JTG_TDO, JTG_TCK, and JTG_TRST_N.

The debug unit when used with debugger application code running on a host system outside of the Intel XScale processor allows a program running on the Intel XScale processor to be debugged. It allows the debugger application code or a debug exception to stop program execution and redirect execution to a debug-handling routine.

Debug exceptions are instruction breakpoint, data breakpoint, software breakpoint, external debug breakpoint, exception vector trap, and trace buffer full breakpoint. Once execution has stopped, the debugger application code can examine or modify the core's state, coprocessor state, or memory. The debugger application code can then restart the program execution.

The debug unit has two hardware instruction break point registers, two hardware data breakpoint registers, and a hardware data breakpoint control register. The second data breakpoint register can be alternatively used as a mask register for the first data breakpoint register.

A 256-entry trace buffer provides the ability to capture control flow messages or addresses. A JTAG instruction (LDIC) can be used to download a debug handler through the JTAG port to the mini-instruction cache; the I-cache has a 2-Kbyte mini-instruction cache like the mini-data cache, that is used only to hold a debug handler.

§ §

3.0 Intel XScale® Processor

This chapter provides functional description of the Intel XScale® Processor.

3.1 Memory Management Unit

This section describes the memory management unit implemented in the Intel® IXP43X Product Line of Network Processors.

The Intel XScale processor implements the Memory Management Unit (MMU) Architecture specified in the *ARM* Architecture Reference Manual*. To accelerate virtual-to-physical address translation, Intel XScale processor uses both an instruction Translation Look-Aside Buffer (TLB) and a data TLB to cache the latest translations. Each TLB holds 32 entries and is fully associative.

Not only do the TLBs contain the translated addresses, but also the access rights for memory references.

If an instruction or data TLB miss occurs, a hardware translation-table-walking mechanism is invoked to translate the virtual address to a physical address. Once translated, the physical address is placed in the TLB along with the access rights and attributes of the page or section. These translations can also be locked down in either TLB to guarantee the performance of critical routines.

For more information, refer to "Exceptions" on page 60.

The Intel XScale processor allows system software to associate various attributes with regions of memory:

- Cacheable
- Bufferable
- Line-allocate policy
- Write policy
- I/O
- Mini data cache
- Coalescing

For a description of page attributes, see "Cacheable (C), Bufferable (B), and eXtension (X) Bits" on page 58. For information on where these attributes have been mapped in the MMU descriptors, see "New Page Attributes" on page 159.

Note: The virtual address with which the TLBs are accessed is remapped by the PID register. For a description of the PID register, see "Register 13: Process ID" on page 94.

Intel® StrongARM* MMU Version 5 Architecture introduces the support of tiny pages, that are 1 KByte in size. The reserved field in the first-level descriptor (encoding 0b11) is used as the fine page table base address. The exact bit fields and the format of the first and second-level descriptors is found in "New Page Attributes" on page 159.

The attributes associated with a particular region of memory are configured in the memory management page table and control the behavior of accesses to the instruction cache, data cache, mini-data cache, and the write buffer. These attributes are ignored when the MMU is disabled.

To allow compatibility with older system software, the new Intel XScale processor attributes take advantage of encoding space in the descriptors that is formerly reserved.

3.1.1 Memory Attributes

3.1.1.1 Page (P) Attribute Bit

The selection between address or data coherency is controlled by a software-programmable P-Attribute bit in the Intel XScale processor' Memory Management Unit (MMU) and the BYTE_SWAP_EN bit. The BYTE_SWAP_EN bit is from the Expansion Bus Controller Configuration Register 1, bit 8. This bit resets to 0.

The default endian-conversion method for the IXP43X network processors is address coherency. This is selected for backward compatibility with the Intel® IXP42X and IXP46X Network Processors.

The BYTE_SWAP_EN bit is an enable bit that allows data coherency to be performed, based on the P-Attribute bit.

- When the bit is 0, address coherency is always performed.
- When the bit is 1, the type of coherency performed is dependent on the P-Attribute bit.

The P-Attribute bit is associated with each 1-Mbyte page. The P-Attribute bit is output from the Intel XScale processor with any store or load access associated with that page.

Note: The P-attribute feature allows software to control byte swapping per 1-Mbyte regions. The P-attribute bit selectively enables or disables this byte swap feature. Future Intel XScale processor products with L2 push cache do not support NPE byte-swapping per 1-Mbyte regions. Using the P-attribute bit to byte swap all of the NPE memory region allows compatible software code porting to future Intel XScale processor. Using the P-attribute bit to byte-swap 1-Mbyte regions of the NPE memory does **not** allow compatible software code porting to future Intel XScale® Technology.

3.1.1.2 Cacheable (C), Bufferable (B), and eXtension (X) Bits

3.1.1.2.1 Instruction Cache

When examining these bits in a descriptor, the Instruction Cache only utilizes the C bit. If the C bit is clear, the Instruction Cache considers a code fetch from that memory to be non-cacheable and does not fill a cache entry. If the C bit is set, then fetches from the associated memory region is cached.

3.1.1.2.2 Details on Data Cache and Write Buffer Behavior

If the MMU is disabled, all data accesses is non-cacheable and non-bufferable. This is the same behavior as when the MMU is enabled and a data access uses a descriptor with X, C, and B all set to 0.

The X, C, and B bits determine when the processor should place new data into the data cache. The cache places data into the cache in lines (also called blocks). Thus, the basis for making a decision about placing new data into the cache is a called a **Line-Allocation Policy**.

If the Line-Allocation Policy is read-allocate, all load operations that miss the cache, request a 32-byte cache line from external memory and allocate it into the data cache or mini-data cache. (This statement assumes that the cache is enabled.) Store operations that miss the cache does not cause a line to be allocated.

If read/write-allocate is in effect, load **or** store operations that miss the cache requests a 32-byte cache line from external memory if the cache is enabled.

The other policy determined by the X, C, and B bits is the Write Policy. A write-through policy instructs the data cache to keep external memory coherent by performing stores to both external memory and the cache. A write-back policy only updates external memory when a line in the cache is cleaned or must be replaced with a new line. Generally, write-back provides higher performance because it generates less data traffic to external memory. For more details on cache policies, see "Cacheability" on page 76

3.1.1.2.3 Data Cache and Write Buffer

All of these descriptor bits affect the behavior of the data cache and the Write Buffer.

If the X bit for a descriptor is zero, the C and B bits operate as mandated by the Intel StrongARM architecture, refer to the *ARM* Architecture Reference Manual*. This behavior is explained in Table 4.

If the X bit for a descriptor is one, the C and B bits' meaning is extended, as explained in Table 5.

Table 4. Data Cache and Buffer Behavior When X = 0

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	N	N	-	-	Stall until complete†
0 1	N	Y	-	-	
1 0	Y	Y	Write Through	Read Allocate	
1 1	Y	Y	Write Back	Read Allocate	

† Normally, the processor continues executing after a data access if no dependency on that access is encountered. With this setting, the processor stalls execution until the data access completes. This guarantees to software that the data access has taken effect by the time execution of the data access instruction completes. External data aborts from such accesses is imprecise (but see "Data Aborts" on page 163 for a method to shield code from this imprecision).

Table 5. Data Cache and Buffer Behavior When X = 1

C B	Cacheable	Bufferable	Write Policy	Line Allocation Policy	Notes
0 0	-	-	-	-	Unpredictable -- do not use
0 1	N	Y	-	-	Writes does not coalesce into buffers[1]
1 0	(Mini data cache)	-	-	-	Cache policy is determined by MD field of Auxiliary Control register[2]
1 1	Y	Y	Write Back	Read/Write Allocate	

Notes:
1. Normally, bufferable writes can coalesce with previously buffered data in the same address range.
2. See "Register 1: Control and Auxiliary Control Registers" on page 88 for a description of this register.

3.1.1.2.4 Memory Operation Ordering

A fence memory operation (memop) is one that guarantees all memops issued prior to the fence executes before any memop issued after the fence. Thus software may issue a fence to impose a partial ordering on memory accesses.

Table 6 shows the circumstances where memops act as fences.

Any swap (**SWP** or **SWPB**) to a page that would create a fence on a load or store is a fence.

Table 6. Memory Operations that Impose a Fence

Operation	X	C	B
load	-	0	-
store	1	0	1
load or store	0	0	0

3.1.1.2.5 Exceptions

The MMU may generate prefetch aborts for instruction accesses and data aborts for data memory accesses. The types and priorities of these exceptions are described in "Event Architecture" on page 161.

Data address alignment checking is enabled by setting bit 1 of the Control Register (CP15, register 1). Alignment faults are still reported even if the MMU is disabled. All other MMU exceptions are disabled when the MMU is disabled.

3.1.2 Interaction of the MMU, Instruction Cache, and Data Cache

The MMU, instruction cache, and data/mini-data cache is enabled/disabled independently. The instruction cache is enabled with the MMU enabled or disabled. But, the data cache can only be enabled when the MMU is enabled. Therefore only three of the four combinations of the MMU and data/mini-data cache enables are valid. The invalid combination causes undefined results.

Table 7. Valid MMU & Data / Mini-Data Cache Combinations

MMU	Data / mini-data Cache
Off	Off
On	Off
On	On

3.1.3 MMU Control

3.1.3.1 Invalidate (Flush) Operation

The entire instruction and data TLB is invalidated at the same time with one command or they are invalidated separately. An individual entry in the data or instruction TLB can also be invalidated. See Table 20, “TLB Functions” on page 93 for a listing of commands supported by the Intel XScale processor.

Globally invalidating a TLB does not affect locked TLB entries. But, the invalidate-entry operations can invalidate individual locked entries. In this case, the locked contents remain in the TLB, but never hit’s on an address translation. Effectively, creating a hole is in the TLB. This situation is rectified by unlocking the TLB.

3.1.3.2 Enabling / Disabling

The MMU is enabled by setting bit 0 in coprocessor 15, register 1 (Control Register). When the MMU is disabled, accesses to the instruction cache default to cacheable accesses and all accesses to data memory are made non-cacheable.

A recommended code sequence for enabling the MMU is shown in Example 1 on page 62.

Example 1. Enabling the MMU

```
; This routine provides software with a predictable way of enabling the MMU.

; After the CPWAIT, the MMU is guaranteed to be enabled. Be aware

; that the MMU is enabled sometime after MCR and before the instruction

; that executes after the CPWAIT.

; Programming Note: This code sequence requires a one-to-one virtual to

; physical address mapping on this code since

; the MMU may be enabled part way through. This would allow the instructions

; after MCR to execute properly regardless the state of the MMU.


MRC P15,0,R0,C1,C0,0; Read CP15, register 1

ORR R0, R0, #0x1; Turn on the MMU

MCR P15,0,R0,C1,C0,0; Write to CP15, register 1


; For a description of CPWAIT, see

; "Additions to CP15 Functionality" on page 160

CPWAIT

; The MMU is guaranteed to be enabled at this point; the next instruction or

; data address is translated.
```

3.1.3.3 Locking Entries

Individual entries are locked into the instruction and data TLBs. See Table 21, "Cache Lock-Down Functions" on page 93 for the exact commands. If a lock operation finds the virtual address translation already resident in the TLB, the results are unpredictable. An invalidate by entry command before the lock command ensures proper operation. Software can also accomplish this by invalidating all entries, as shown in Example 2 on page 63.

Locking entries into the instruction TLB or data TLB reduces the available number of entries by the number that is locked down, for hardware to cache other virtual to physical address
translations.

A procedure for locking entries into the instruction TLB is shown in Example 2 on page 63.

If a MMU abort is generated during an instruction or data TLB lock operation, the Fault Status Register is updated to indicate a Lock Abort (see "Data Aborts" on page 163), and the exception is reported as a data abort.

Example 2. Locking Entries into the Instruction TLB

```
R1, R2 and R3 contain the virtual addresses to translate and lock into the instruction TLB.

The value in R0 is ignored in the following instruction. Hardware guarantees that accesses to
CP15 occur in program order


MCR P15,0,R0,C8,C5,0       ; Invalidate the entire instruction TLB


MCR P15,0,R1,C10,C4,0      ; Translate virtual address (R1) and lock into

                           ; instruction TLB

MCR P15,0,R2,C10,C4,0  ; Translate

                           ; virtual address (R2) and lock into instruction TLB

MCR P15,0,R3,C10,C4,0      ; Translate virtual address (R3) and lock into

                           ; instruction TLB


CPWAIT


The MMU is guaranteed to be updated at this point the next instruction does

see the locked instruction TLB entries.
```

Note: If exceptions are allowed to occur in the middle of this routine, the TLB may end up caching a translation that is about to be locked. For example, if R1 is the virtual address of an interrupt service routine and that interrupt occurs immediately after the TLB has been invalidated, the lock operation is ignored when the interrupt service routine returns back to this code sequence. Software should disable interrupts (FIQ or IRQ) in this case.

As a general rule, software should avoid locking in all other exception types.

The proper procedure for locking entries into the data TLB is shown in Example 3 on page 64.

Example 3. Locking Entries into the Data TLB

```
; R1, and R2 contain the virtual addresses to translate and lock into the data TLB


MCR  P15,0,R1,C8,C6,1      ; Invalidate the data TLB entry specified by the

                           ; virtual address in R1

MCR  P15,0,R1,C10,C8,0     ; Translate virtual address (R1) and lock into

                           ; data TLB


; Repeat sequence for virtual address in R2

MCR  P15,0,R2,C8,C6,1      ; Invalidate the data TLB entry specified by the

                           ; virtual address in R2

MCR  P15,0,R2,C10,C8,0     ; Translate virtual address (R2) and lock into

                           ; data TLB


CPWAIT                     ; wait for locks to complete


; The MMU is guaranteed to be updated at this point; the next instruction does

; see the locked data TLB entries.
```

Note: Care must be exercised here when allowing exceptions to occur during this routine whose handlers may have data that lies in a page that is trying to be locked into the TLB.

3.1.3.4 Round-Robin Replacement Algorithm

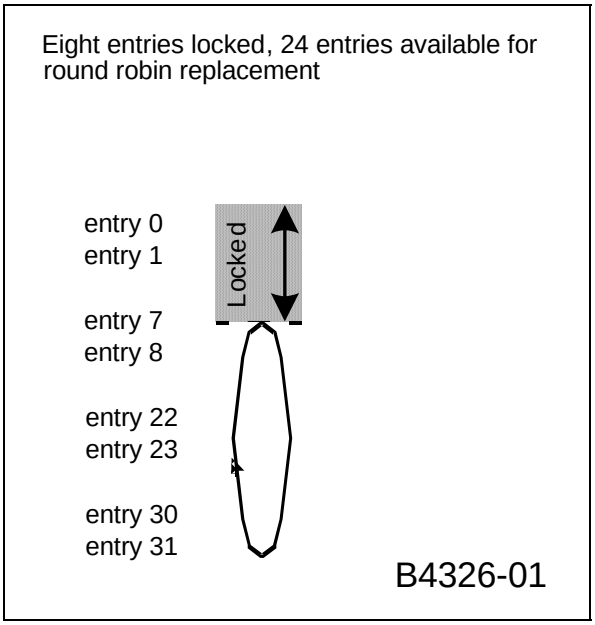
The line replacement algorithm for the TLBs is round-robin; there is a round-robin pointer that keeps track of the next entry to replace. The next entry to replace is the one sequentially after the last entry that was written. For example, if the last virtual to physical address translation is written into entry 5, the next entry to replace is entry 6.

At reset, the round-robin pointer is set to entry 31. Once a translation is written into entry 31, the round-robin pointer gets set to the next available entry, beginning with entry 0 if no entries have been locked down. Subsequent translations move the round-robin pointer to the next sequential entry until entry 31 is reached, where it wraps back to entry 0 upon the next translation.

A lock pointer is used for locking entries into the TLB and is set to entry 0 at reset. A TLB lock operation places the specified translation at the entry designated by the lock pointer, moves the lock pointer to the next sequential entry, and resets the round-robin pointer to entry 31. Locking entries into either TLB effectively reduces the available entries for updating. For example, if the first three entries were locked down, the round-robin pointer would be entry 3 after it rolled over from entry 31.

Only entries 0 through 30 is locked in either TLB; entry 31can never be locked. If the lock pointer is at entry 31, a lock operation updates the TLB entry with the translation and ignore the lock. In this case, the round-robin pointer stays at entry 31.

Figure 7. Example of Locked Entries in TLB



3.2 Instruction Cache

The Intel XScale processor instruction cache enhances performance by reducing the number of instruction fetches from external memory. The cache provides fast execution of cached code. Code can also be locked down when guaranteed or fast access time is required.

Figure 8 shows the cache organization and how the instruction address is used to access the cache.

The instruction cache is available as a 32 K, 32-way set, associative cache**.** Each set is 1,024 bytes in size. Each set contains 32 ways. Each way of a set contains eight 32-bit words and one valid bit, and is referred to as a line. The replacement policy is a round-robin algorithm and the cache also supports the ability to lock code in at a line granularity.

The instruction cache is virtually addressed and virtually tagged.

Note: The virtual address presented to the instruction cache is remapped by the PID register. For a description of the PID register, see "Register 13: Process ID" on page 94.

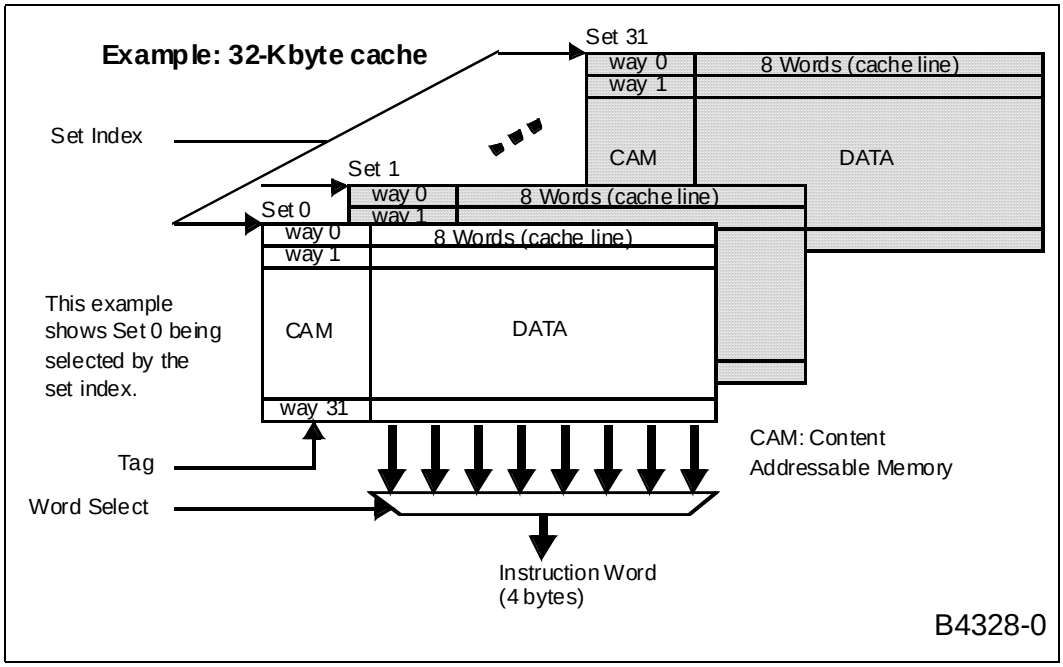
3.2.1 Operation When Instruction Cache is Enabled

When the cache is enabled, it compares every instruction request address against the addresses of instructions that it is currently holding. If the cache contains the requested instruction, the access **hits** the cache, and the cache returns the requested instruction. If the cache does not contain the requested instruction, the access **misses** the cache. The cache requests an eight-word, also known as a line, fetch from external memory that contains the requested instruction using the fetch policy described in "Instruction-Cache 'Miss'" on page 66. As the fetch returns instructions to the cache, the instructions are placed in one of two fetch buffers and the requested instruction is delivered to the instruction decoder.

A fetched line is written into the cache if it is cacheable. Code is designated as cacheable when the Memory Management Unit (MMU) is disabled or when the MMU is enable and the cacheable (C) bit is set to 1 in its corresponding page. See "Memory Management Unit" on page 57 for a discussion on page attributes.

Note: An instruction fetch may **miss** the cache but **hit** one of the fetch buffers. When this happens, the requested instruction is delivered to the instruction decoder in the same manner as a cache **hit**.

Figure 8. Instruction Cache Organization



Disabling the cache prevents any lines from being written into the instruction cache. Although the cache is disabled, it is still accessed and may generate a **hit** if the data is already in the cache.

Disabling the instruction cache **does not** disable instruction buffering that may occur within the instruction fetch buffers. Two 8-word instruction fetch buffers is always enabled in the cache disabled mode. So long as instruction fetches continue to **hit** within either buffer (even in the presence of forward and backward branches), no external fetches for instructions are generated. A miss causes one or the other buffer to be filled from external memory using the fill policy described in "Instruction-Cache 'Miss'" on page 66.

3.2.1.1 Instruction-Cache 'Miss'

An instruction-cache **miss** occurs when the requested instruction is not found in the instruction fetch buffers or instruction cache; a fetch request is then made to external memory. The instruction cache can handle up to two **misses**. Each external fetch request uses a fetch buffer that holds 32-bytes and eight valid bits, one for each word.

A miss causes the following:

- A fetch buffer is allocated
- The instruction cache sends a fetch request to the external bus. This request is for a 32-byte line.

- Instructions words are returned back from the external bus, at a maximum rate of 1 word per core cycle. The instruction cache can have the eight words of data return in any order, and allows the Intel XScale processor to send the requested instruction first, thus reducing fetch latency. (This is referred to as critical word first.)As each word returns, the corresponding valid bit is set for the word in the fetch buffer.
- As soon as the fetch buffer receives the requested instruction, it forwards the instruction to the instruction decoder for execution.
- When all words have returned, the fetched line is written into the instruction cache if it is cacheable and if the instruction cache is enabled. The line chosen for update in the cache is controlled by the round-robin replacement algorithm. This update may evict a valid line at that location. For more information on enabling or disabling instruction cache, refer to “Instruction-Cache Coherence” on page 68
- Once the cache is updated, the eight valid bits of the fetch buffer are invalidated.

3.2.1.2 Instruction-Cache Line-Replacement Algorithm

The line replacement algorithm for the instruction cache is round-robin. Each set in the instruction cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the one after the last line that is written. For example, if the line for the last external instruction fetch is written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way0 if no lines have been locked into that particular set. Locking lines into the instruction cache effectively reduces the available lines for cache updating. For example, if the first three lines of a set were locked down, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. For more details on cache locking, see “Instruction-Cache Coherence” on page 68.

The instruction cache is protected by parity to ensure data integrity. Each instruction cache word has 1 parity bit. (The instruction cache tag is NOT parity protected.) When a parity error is detected on an instruction cache access, a prefetch abort exception occurs if the Intel XScale processor attempts to execute the instruction. Before servicing the exception, hardware places a notification of the error in the Fault Status Register (Coprocessor 15, register 5).

A software exception handler can recover from an instruction cache parity error. The parity error is accomplished by invalidating the instruction cache and the branch target buffer and then returning to the instruction that caused the prefetch abort exception. A simplified code example is shown in Example 4 on page 68. A more complex handler might choose to invalidate the specific line that caused the exception and then invalidate the BTB.

Example 4. Recovering from an Instruction Cache Parity Error

```
; Prefetch abort handler

MCR P15,0,R0,C7,C5,0       ; Invalidate the instruction cache and branch target

                           ; buffer



CPWAIT                     ; wait for effect (see "Additions to CP15 Functionality" on page 160
for a

                        ; description of CPWAIT)



SUBS PC,R14,#4          ; Returns to the instruction that generated the

                           ; parity error



; The Instruction Cache is guaranteed to be invalidated at this point
```

If a parity error occurs on an instruction that is locked in the cache, the software exception handler must unlock the instruction cache, invalidate the cache and then re-lock the code in before it returns to the faulting instruction.

The instruction cache does not detect modification to program memory by loads, stores or actions of other bus masters. Several situations may require program memory modification, such as uploading code from disk.

The application program is responsible for synchronizing code modification and invalidating the cache. In general, software must ensure that modified code space is not accessed until modification and invalidating are completed.

3.2.1.3 Instruction-Cache Coherence

To achieve cache coherence, instruction cache contents is invalidated after code modification in external memory is complete.

If the instruction cache is not enabled, or code is being written to a non-cacheable region, software must still invalidate the instruction cache before using the newly-written code. This precaution ensures that state associated with the new code is not buffered elsewhere in the processor, such as the fetch buffers or the BTB.

Naturally, when writing code as data, care is taken to force it completely out of the processor into external memory before attempting to execute it. If writing into a non-cacheable region, flushing the write buffers is sufficient precaution (see "Register 7: Cache Functions" on page 91 for a description of this operation). If writing to a cacheable region, then the data cache should be submitted to a Clean/Invalidate operation (see "Cacheability" on page 76) to ensure coherency.

After reset, the instruction cache is always disabled, unlocked, and invalidated (flushed).

The instruction cache is enabled by setting bit 12 in coprocessor 15, register 1 (Control Register). This process is illustrated in Example 5, Enabling the Instruction Cache.

Example 5. Enabling the Instruction Cache

```
; Enable the ICache

MRC P15, 0, R0, C1, C0, 0             ; Get the control register

ORR R0, R0, #0x1000                   ; set bit 12 -- the I bit

MCR P15, 0, R0, C1, C0, 0             ; Set the control register


CPWAIT
```

The entire instruction cache along with the fetch buffers are invalidated by writing to coprocessor 15, register 7. (See Table 19, “Cache Functions” on page 92 for the exact command.) The invalidate command does not unlock any lines that were locked in the instruction cache nor does it invalidate those locked lines. To invalidate the entire cache including locked lines, the unlock instruction cache command must be executed before the invalidate command. The unlock command can also be found in Table 21, “Cache Lock-Down Functions” on page 93.

There is an inherent delay from the execution of the instruction cache invalidate command to where the next instruction sees the result of the invalidate. The following routine is used to guarantee proper synchronization.

Example 6. Invalidating the Instruction Cache

```
MCR P15,0,R1,C7,C5,0      ; Invalidate the instruction cache and branch

                          ; target buffer


CPWAIT


; The instruction cache is guaranteed to be invalidated at this point; the next

; instruction sees the result of the invalidate command.
```

The Intel XScale processor also supports invalidating an individual line from the instruction cache. See Table 19, “Cache Functions” on page 92 for the exact command.

Software has the ability to lock performance critical routines into the instruction cache. Up to 28 lines in each set is locked; hardware ignores the lock command if software is trying to lock all the lines in a particular set (that is, ways 28-31can never be locked). When all ways in a particular set are requested to be locked, the instruction cache line is still allocated into the cache but the lock is ignored. The round-robin pointer stays at way 31 for that set.

Cache lines is locked into the instruction cache by initiating a write to coprocessor 15. (See Table 21, “Cache Lock-Down Functions” on page 93 for the exact command.) Register **Rd** contains the virtual address of the line to be locked into the cache.

There are several requirements for locking down code:

- The routine used to lock lines down in the cache is placed in non-cacheable memory, that means the MMU is enabled.

  As a result, no fetches of cacheable code should occur when locking instructions into the cache.

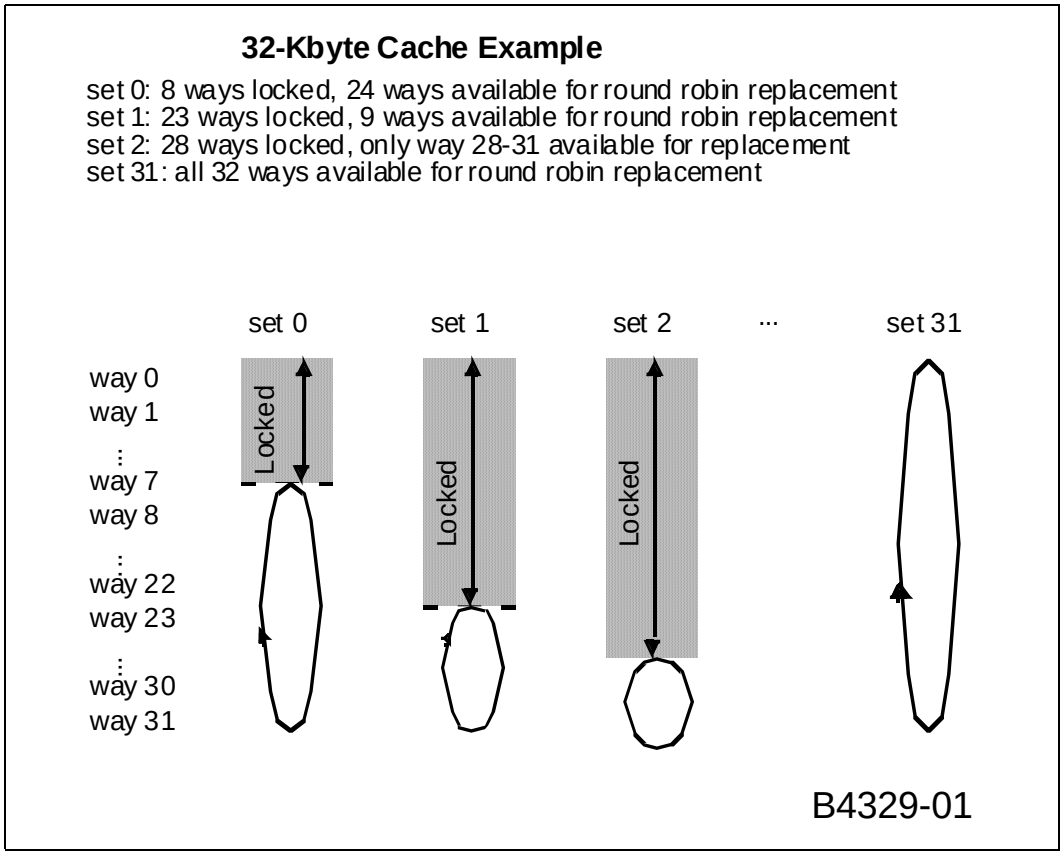
- The code being locked into the cache is cacheable
- The instruction cache is enabled and invalidated prior to locking down lines.

Failure to follow these requirements produces unpredictable results when accessing the instruction cache.

System programmers should ensure that the code to lock instructions into the cache does not reside closer than 128 bytes to a non-cacheable/cacheable page boundary. If the processor fetches ahead into a cacheable page, then the first requirement noted above could be violated.

Lines are locked into a set starting at way 0 and may progress up to way 27, and set a line gets locked into depends on the set index of the virtual address. Figure 9 is an example (32-Kbyte cache) of where lines of code are locked into the cache along with how the round-robin pointer is affected.

Figure 9. Locked Line Effect on Round-Robin Replacement



Software can lock down several various routines located at various memory locations. This may cause some sets to have more locked lines than others as shown in Figure 9.

Example 7 on page 71 shows how a routine, called **lockMe** in this example, might be locked into the instruction cache.

Note: It is possible to receive an exception when locking code (see "Event Architecture" on page 161).

Example 7. Locking Code into the Cache

```
lockMe:                    ; This is the code that is locked into the cache

    mov     r0, #5

    add     r5, r1, r2

    . . .

lockMeEnd:

        . . .


codeLock:                              ; here is the code to lock the lockMe routine

    ldr     r0, =(lockMe AND NOT 31); r0 gets a pointer to the first line we

    should lock

    ldr     r1, =(lockMeEnd AND NOT 31); r1 contains a pointer to the last line we

    should lock


lockLoop:

    mcr     p15, 0, r0, c9, c1, 0      ; lock next line of code into ICache

    cmp     r0, r1                     ; are we done yet?

    add     r0, r0, #32                ; advance pointer to next line

    bne     lockLoop                   ; if not done, do the next line
```

The Intel XScale processor provides a global unlock command for the instruction cache. Writing to coprocessor 15, register 9 unlocks all the locked lines in the instruction cache and leaves them valid. These lines then become available for the round-robin replacement algorithm. (See Table 21, “Cache Lock-Down Functions” on page 93 for the exact command.)

3.3 Branch Target Buffer

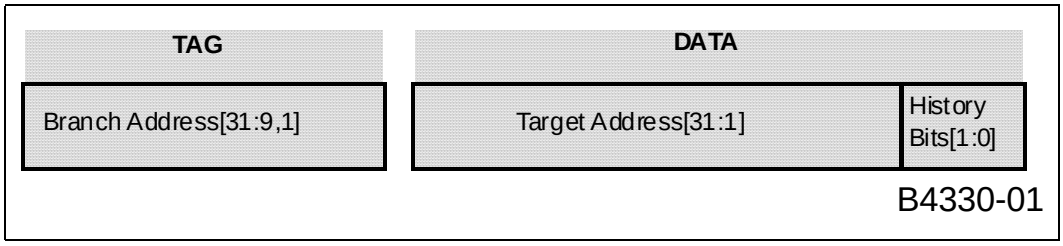
The Intel XScale processor uses dynamic branch prediction to reduce the penalties associated with changing the flow of program execution. The Intel XScale processor features a branch target buffer that provides the instruction cache with the target address of branch type instructions. The branch target buffer is implemented as a 128-entry, direct-mapped cache.

This section is primarily for those optimizing their code for performance. An understanding of the branch target buffer is needed in this case so that code is scheduled to best utilize the performance benefits of the branch target buffer.

3.3.1 Branch Target Buffer (BTB) Operation

The BTB stores the history of branches that have executed along with their targets. Figure 10 shows an entry in the BTB, where the tag is the instruction address of a previously executed branch and the data contains the target address of the previously executed branch along with two bits of history information.

Figure 10. BTB Entry



The BTB takes the current instruction address and checks to see if this address is a branch that is previously seen. The BTB uses bits [8:2] of the current address to read out the tag and then compares this tag to bits [31:9,1] of the current instruction address. If the current instruction address matches the tag in the cache and the history bits indicate that this branch is usually taken in the past, the BTB uses the data (target address) as the next instruction address to send to the instruction cache.

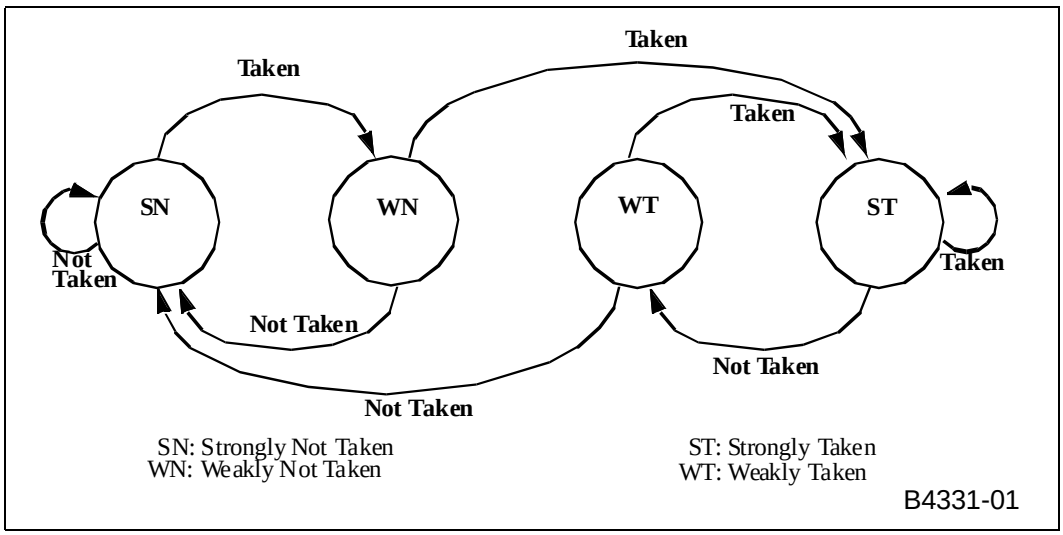
Bit[1] of the instruction address is included in the tag comparison to support Thumb execution. This organization means that two consecutive Thumb branch (B) instructions, with instruction address bits[8:2] the same, contends for the same BTB entry. Thumb also requires 31 bits for the branch target address. The bit[1] is zero in Intel StrongARM mode.

The history bits represent four possible prediction states for a branch entry in the BTB. Figure 11, "Branch History" on page 72 shows these states along with the possible transitions. The initial state for branches stored in the BTB is Weakly-Taken (WT). Every time a branch that exists in the BTB is executed, the history bits are updated to reflect the latest outcome of the branch, taken or not-taken.

"Performance Considerations" on page 166 describes the instructions that are dynamically predicted by the BTB and the performance penalty for incorrectly predicting a branch.

The BTB does not have to be managed explicitly by software; it is disabled by default after reset and is invalidated when the instruction cache is invalidated.

Figure 11. Branch History



3.3.1.1 Reset

After Processor Reset, the BTB is disabled and all entries are invalidated. A new entry is stored into the BTB when the following conditions are met:

- The branch instruction has executed
- The branch is taken
- The branch is not currently in the BTB

The entry is then marked valid and the history bits are set to WT. If another valid branch exists at the same entry in the BTB, it is evicted by the new branch.

Once a branch is stored in the BTB, the history bits are updated upon every execution of the branch as shown in Figure 11.

The BTB is always disabled with Reset. Software can enable the BTB through a bit in a coprocessor register (see "Register 1: Control and Auxiliary Control Registers" on page 88).

Before enabling or disabling the BTB, software must invalidate the BTB (described in the following section). This action ensures correct operation in case stale data is in the BTB. Software should not place any branch instruction between the code that invalidates the BTB and the code that enables/disables it.

There are four ways the contents of the BTB is invalidated.

- Reset
- Software can directly invalidate the BTB via a CP15, register 7 function. Refer to "Register 7: Cache Functions" on page 91.
- The BTB is invalidated when the Process ID Register is written.
- The BTB is invalidated when the instruction cache is invalidated via CP15, register 7 functions.

3.4 Data Cache

The Intel XScale processor data cache enhances performance by reducing the number of data accesses to and from external memory. There are two data cache structures in the Intel XScale processor: a 32-Kbyte data cache and a 2-Kbyte mini-data cache. An eight entry write buffer and a four-entry, fill buffer are also implemented to decouple the Intel XScale processor instruction execution from external memory accesses, and increases overall system performance.

3.4.1 Data Cache Overview

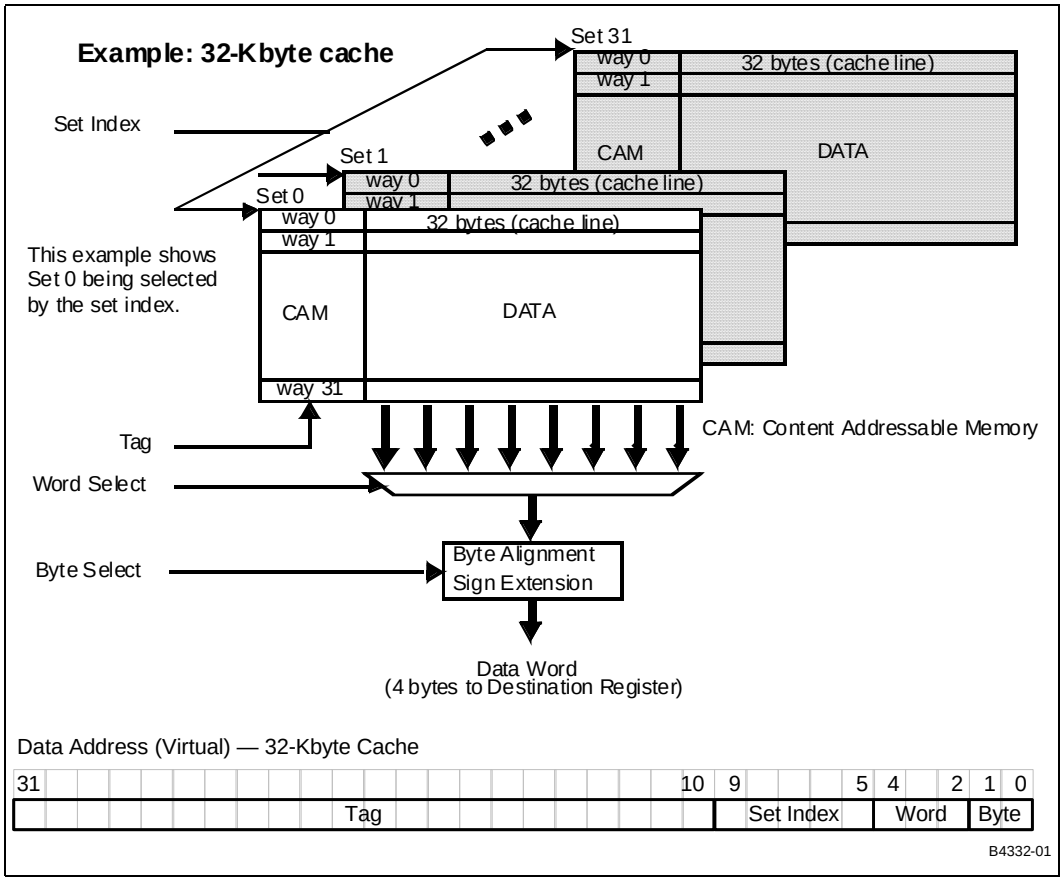
The data cache is a 32-Kbyte, 32-way set, associative cache. The 32-Kbyte cache has 32 sets. Each set contains 32 ways. Each way of a set contains 32 bytes (one cache line) and one valid bit. There also exist two dirty bits for every line, one for the lower 16 bytes and the other one for the upper 16 bytes. When a store hits the cache the dirty bit associated with it is set. The replacement policy is a round-robin algorithm and the cache also supports the ability to reconfigure each line as data RAM.

Figure 12, "Data Cache Organization" on page 74 shows the cache organization and how the data address is used to access the cache.

Cache policies are adjusted for particular regions of memory by altering page attribute bits in the MMU descriptor that controls that memory. See "Memory Attributes" on page 58 for a description of these bits.

The data cache is virtually addressed and virtually tagged. The data cache supports write-back and write-through caching policies. The data cache always allocates a line in the cache when a cacheable read miss occurs and allocates a line into the cache on a cacheable write miss when write allocate is specified by its page attribute. Page attribute bits determine whether a line gets allocated into the data cache or mini-data cache.

Figure 12. Data Cache Organization

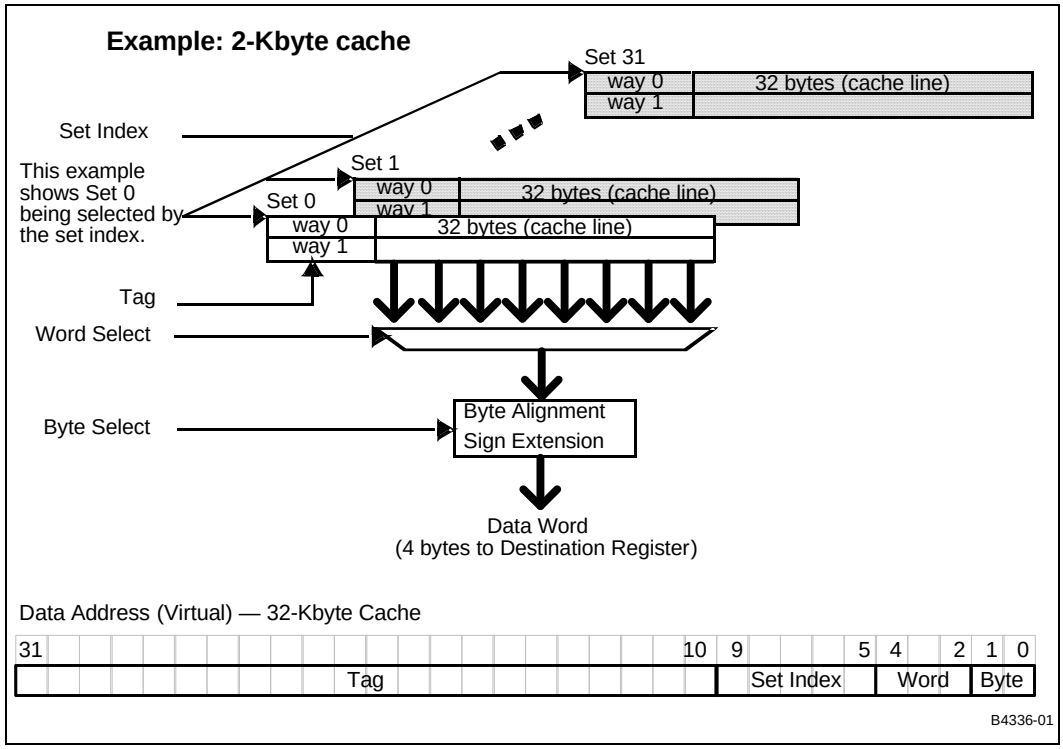


The mini-data cache is 2 Kbytes in size. The 2-Kbyte mini data cache has 32 sets and is two-way set associative. Each way of a set contains 32 bytes (one cache line) and one valid bit. There also exist two dirty bits for every line, one for the lower 16 bytes and the other one for the upper 16 bytes. When a store hits the cache the dirty bit associated with it is set. The replacement policy is a round-robin algorithm.

Figure 13, “Mini-Data Cache Organization” on page 75 shows the cache organization and how the data address is used to access the cache.

The mini-data cache is virtually addressed and virtually tagged and supports the same caching policies as the data cache. Lines can’t be locked into the mini-data cache.

Figure 13. Mini-Data Cache Organization



The Intel XScale processor employs an eight entry write buffer, each entry containing 16 bytes. Stores to external memory are first placed in the write buffer and subsequently taken out when the bus is available.

The write buffer supports the coalescing of multiple store requests to external memory. An incoming store may coalesce with any of the eight entries.

The fill buffer holds the external memory request information for a data cache or mini-data cache fill or non-cacheable read request. Up to four 32-byte read request operations are outstanding in the fill buffer before the Intel XScale processor stalls.

The fill buffer has been augmented with a four entry pend buffer that captures data memory requests to outstanding fill operations. Each entry in the pend buffer contains enough data storage to hold one 32-bit word, specifically for store operations. Cacheable load or store operations that hit an entry in the fill buffer get placed in the pend buffer and are completed when the associated fill completes. Any entry in the pend buffer is pended against any of the entries in the fill buffer; multiple entries in the pend buffer is pended against a single entry in the fill buffer.

Pended operations complete in program order.

The following discussions refer to the data cache and mini-data cache as one cache (data/mini-data) since their behavior is the same when accessed.

When the data/mini-data cache is enabled for an access, the data/mini-data cache compares the address of the request against the addresses of data that it is currently holding. If the line containing the address of the request is resident in the cache, the access **hits** the cache. For a load operation the cache returns the requested data to the destination register and for a store operation the data is stored into the cache. The data associated with the store may also be written to external memory if write-through

caching is specified for that area of memory. If the cache does not contain the requested data, the access 'misses' the cache, and the sequence of events that follows depends on the configuration of the cache, the configuration of the MMU and the page attributes, that are described in "Cacheability" on page 76.

The data/mini-data cache is still accessed even though it is disabled. If a load hits the cache it returns the requested data to the destination register. If a store hits the cache, the data is written into the cache. Any access that misses the cache does not allocate a line in the cache when it is disabled, even if the MMU is enabled and the memory region's cacheability attribute is set.

3.4.2 Cacheability

Data at a specified address is cacheable given the following:

- the MMU is enabled
- the cacheable attribute is set in the descriptor for the accessed address
- and the data/mini-data cache is enabled

The following sequence of events occurs when a cacheable (see "Cacheability" on page 76) load operation misses the cache:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.

   If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it accesses the cache again, to obtain the request data and returns it to the destination register.

   If there is no outstanding fill request for that line, the current load request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor stalls until an entry is available.
2. A line is allocated in the cache to receive the 32-bytes of fill data. The line selected is determined by the round-robin pointer. (See "Cacheability" on page 76.) The line chosen may contain a valid line previously allocated in the cache. In this case both dirty bits are examined and if set, the four words associated with a dirty bit that's asserted is written back to external memory as a four word burst operation.
3. When the data requested by the load is returned from external memory, it is immediately sent to the destination register specified by the load. A system that returns the requested data back first, with respect to the other bytes of the line, obtains the best performance.
4. As data returns from external memory it is written into the cache in the previously allocated line.

A load operation that misses the cache and is NOT cacheable makes a request from external memory for the exact data size of the original load request. For example, **LDRH** requests exactly two bytes from external memory, **LDR** requests 4 bytes from external memory, and so forth. This request is placed in the fill buffer until, the data is returned from external memory, and is then forwarded back to the destination register(s).

A write operation that misses the cache requests a 32-byte cache line from external memory if the access is cacheable and write allocation is specified in the page. In this case the following sequence of events occur:

1. The fill buffer is checked to see if an outstanding fill request already exists for that line.

If so, the current request is placed in the pending buffer and waits until the previously requested fill completes, after which it writes its data into the recently allocated cache line.

If there is no outstanding fill request for that line, the current store request is placed in the fill buffer and a 32-byte external memory read request is made. If the pending buffer or fill buffer is full, the Intel XScale processor stalls until an entry is available.

2. The 32-bytes of data is returned back to the Intel XScale processor in any word order, that is, the eight words in the line is returned in any order.

Note: For performance sake, the order in which the data is returned to the Intel XScale processor does not matter since the store operation has to wait until the entire line is written into the cache before it can complete.

3. When the entire 32-byte line has returned from external memory, a line is allocated in the cache, selected by the round-robin pointer. The line to be written into the cache may replace a valid line previously allocated in the cache. In this case both dirty bits are examined and if any are set, the four words associated with a dirty bit that's asserted is written back to external memory as a 4 word burst operation. This write operation is placed in the write buffer.
4. The line is written into the cache along with the data associated with the store operation.

If the above condition for requesting a 32-byte cache line is not met, a write miss causes a write request to external memory for the exact data size specified by the store operation, assuming the write request doesn't coalesce with another write operation in the write buffer.

The Intel XScale processor supports write-back caching or write-through caching, controlled through the MMU page attributes. When write-through caching is specified, all store operations are written to external memory even if the access hits the cache. This feature keeps the external memory coherent with the cache, that is, no dirty bits are set for this region of memory in the data/mini-data cache. But write through does not guarantee that the data/mini-data cache is coherent with external memory, and is dependent on the system level configuration, specifically if the external memory is shared by another master.

When write-back caching is specified, a store operation that hits the cache does not generate a write to external memory, thus reducing external memory traffic.

The line replacement algorithm for the data cache is round-robin. Each set in the data cache has a round-robin pointer that keeps track of the next line (in that set) to replace. The next line to replace in a set is the next sequential line after the one that was just filled. For example, if the line for the last fill is written into way 5-set 2, the next line to replace for that set would be way 6. None of the other round-robin pointers for the other sets are affected in this case.

After reset, way 31 is pointed to by the round-robin pointer for all the sets. Once a line is written into way 31, the round-robin pointer points to the first available way of a set, beginning with way 0 if no lines have been re-configured as data RAM in that particular set. Re-configuring lines as data RAM effectively reduces the available lines for cache updating. For example, if the first three lines of a set were re-configured, the round-robin pointer would point to the line at way 3 after it rolled over from way 31. Refer to "Reconfiguring the Data Cache as Data RAM" on page 80 for more details on data RAM.

The mini-data cache follows the same round-robin replacement algorithm as the data cache except that there are only two lines the round-robin pointer can point to such that the round-robin pointer always points to the least recently filled line. A least recently used replacement algorithm is not supported because the purpose of the mini-

data cache is to cache data that exhibits low temporal locality, that is, data that is placed into the mini-data cache is typically modified once and then written back out to external memory.

The data cache and mini-data cache are protected by parity to ensure data integrity; there is one parity bit per byte of data. (The tags are NOT parity protected.) When a parity error is detected on a data/mini-data cache access, a data abort exception occurs. Before servicing the exception, hardware sets bit 10 of the Fault Status Register register.

A data/mini-data cache parity error is an imprecise data abort, meaning R14_ABORT (+8) may not point to the instruction that caused the parity error. If the parity error occurred during a load, the targeted register is updated with incorrect data.

A data abort due to a data/mini-data cache parity error may not be recoverable if the data address that caused the abort occurred on a line in the cache that has a write-back caching policy. Prior updates to this line may be lost; in this case the software exception handler should perform a clean and clear operation on the data cache, ignoring subsequent parity errors, and restart the offending process.

The **SWP** and **SWPB** instructions generate an atomic load and store operation allowing a memory semaphore to be loaded and altered without interruption. These accesses may hit or miss the data/mini-data cache depending on configuration of the cache, configuration of the MMU, and the page attributes.

After processor reset, both the data cache and mini-data cache are disabled, all valid bits are set to zero (invalid), and the round-robin bit points to way 31. Any lines in the data cache that were configured as data RAM before reset are changed back to cacheable lines after reset, that is, there are 32 KBytes of data cache and zero bytes of data RAM.

The data cache and mini-data cache are enabled by setting bit 2 in coprocessor 15, register 1 (Control Register). See “Configuration” on page 84, for a description of this register and others.

Equation 8 shows code that enables the data and mini-data caches.

Note: The MMU is enabled to use the data cache.

Example 8. Enabling the Data Cache

```
enableDCache:


       MCR     p15, 0, r0, c7, c10, 4; Drain pending data operations...

                           ; (see Chapter 7.2.8, Register 7: Cache functions)

       MRC     p15, 0, r0, c1, c0, 0; Get current control register

       ORR     r0, r0, #4      ; Enable DCache by setting ‘C’ (bit 2)

       MCR     p15, 0, r0, c1, c0, 0; And update the Control register
```

Individual entries is invalidated and cleaned in the data cache and mini-data cache via coprocessor 15, register 7.

Note: A line locked into the data cache remains locked even after it has been subjected to an invalidate-entry operation. This leaves an unusable line in the cache until a global unlock has occurred. For this reason, do not use these commands on locked lines.

This same register also provides the command to invalidate the entire data cache and mini-data cache. Refer to Table 19, “Cache Functions” on page 92 for a listing of the commands. These global invalidate commands have no effect on lines locked in the data cache. Locked lines is unlocked before they are invalidated. This is accomplished by the Unlock Data Cache command found in Table 21, “Cache Lock-Down Functions” on page 93.

A simple software routine is used to globally clean the data cache. It takes advantage of the line-allocate data cache operation, that allocates a line into the data cache. This allocation evicts any cache dirty data back to external memory. Example 9 on the next page shows how data cache is cleaned.

Example 9. Global Clean Operation

```
; Global Clean/Invalidate THE DATA CACHE
; R1 contains the virtual address of a region of cacheable memory reserved for
; this clean operation
; R0 is the loop count; Iterate 1024 times which is the number of lines in the
; data cache

;; Macro ALLOCATE performs the line-allocation cache operation on the
;; address specified in register Rx.
;;

MACRO ALLOCATE Rx
    MCR P15, 0, Rx, C7, C2, 5
ENDM

MOV  R0, #1024
LOOP1:
ALLOCATE R1                ; Allocate a line at the virtual address
                           ; specified by R1.
ADD R1, R1, #32        ; Increment the address in R1 to the next cache line
SUBS R0, R0, #1        ; Decrement loop count
BNE LOOP1
;
;Clean the Mini-data Cache
; Can't use line-allocate command, so cycle 2KB of unused data through.
; R2 contains the virtual address of a region of cacheable memory reserved for
; cleaning the Mini-data Cache
; R0 is the loop count; Iterate 64 times which is the number of lines in the
; Mini-data Cache.

MOV  R0, #64
LOOP2:
LDR R3,[R2],#32 ; Load and increment to next cache line
SUBS R0, R0, #1        ; Decrement loop count
BNE LOOP2
;

; Invalidate the data cache and mini-data cache
MCR P15, 0, R0, C7, C6, 0
;
```

The line-allocate operation does not require physical memory to exist at the virtual address specified by the instruction, since it does not generate a load/fill request to external memory. Also, the line-allocate operation does not set the 32 bytes of data associated with the line to any known value. Reading this data produces unpredictable results.

The line-allocate command does not operate on the mini Data Cache, so system software must clean this cache by reading 2KByte of contiguous unused data into it. This data is unused and reserved for this purpose so that it is not already in the cache. It must reside in a page that is marked as mini Data Cache cacheable (see “New Page Attributes” on page 159).

The time it takes to execute a global clean operation depends on the number of dirty lines in cache.

3.4.3 Reconfiguring the Data Cache as Data RAM

Software has the ability to lock tags associated with 32-byte lines in the data cache, thus creating the appearance of data RAM. Any subsequent access to this line always hits the cache unless it is invalidated. Once a line is locked into the data cache it is no longer available for cache allocation on a line fill. Up to 28 lines in each set is reconfigured as data RAM, such that the maximum data RAM size is 28 Kbytes for the 32-Kbyte.

Hardware does not support locking lines into the mini-data cache; any attempt to do this produces unpredictable results.

There are two methods for locking tags into the data cache; the method of choice depends on the application. One method is used to lock data that resides in external memory into the data cache and the other method is used to re-configure lines in the data cache as data RAM. Locking data from external memory into the data cache is useful for lookup tables, constants, and any other data that is frequently accessed. Re-configuring a portion of the data cache as data RAM is useful when an application needs scratch memory (bigger than the register file can provide) for frequently used variables. These variables are strewn across memory, making it advantageous for software to pack them into data RAM memory.

Code examples for these two applications are shown in Example 10 on page 81 and Example 11 on page 82. The difference between these two routines is that Example 10 on page 81 actually requests the entire line of data from external memory and Example 11 on page 82 uses the line-allocate operation to lock the tag into the cache. No external memory request is made, that means software can map any unallocated area of memory as data RAM. But, the line-allocate operation does validate the target address with the MMU, so system software must ensure that the memory has a valid descriptor in the page table.

Another item to note in Example 11 on page 82 is that the 32 bytes of data located in a newly allocated line in the cache is initialized by software before it is read. The line allocate operation does not initialize the 32 bytes and therefore reading from that line produces unpredictable results.

In both examples, the code drains the pending loads before and after locking data. This step ensures that outstanding loads do not end up in the wrong place -- unintentionally locked into the cache or mistakenly left out in the proverbial cold.

Note: A drain operation has been placed after the operation that locks the tag into the cache. This drains ensures predictable results if a programmer tries to lock more than 28 lines in a set; the tag gets allocated in this case but not locked into the cache.

Example 10. Locking Data into Data Cache

```
; R1 contains the virtual address of a region of memory to lock,
; configured with C=1 and B=1
; R0 is the number of 32-byte lines to lock into the data cache. In this
; example 16 lines of data are locked into the cache.
; MMU and data cache are enabled prior to this code.

MACRO DRAIN
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores
ENDM

DRAIN

MOV  R2, #0x1
MCR  P15,0,R2,C9,C2,0         ; Put the data cache in lock mode
CPWAIT
MOV  R0, #16
LOOP1:
MCR  P15,0,R1,C7,C10,1        ; Write back the line if it is dirty in the cache
MCR  P15,0,R1, C7,C6,1        ; Flush/Invalidate the line from the cache
LDR  R2, [R1], #32            ; Load and lock 32 bytes of data located at [R1]
                              ; into the data cache. Post-increment the address
                              ; in R1 to the next cache line.
SUBS R0, R0, #1               ; Decrement loop count
BNE LOOP1

           ; Turn off data cache locking
MOV  R2, #0x0
MCR  P15,0,R2,C9,C2,0         ; Take the data cache out of lock mode.
CPWAIT
```

Example 11. Creating Data RAM

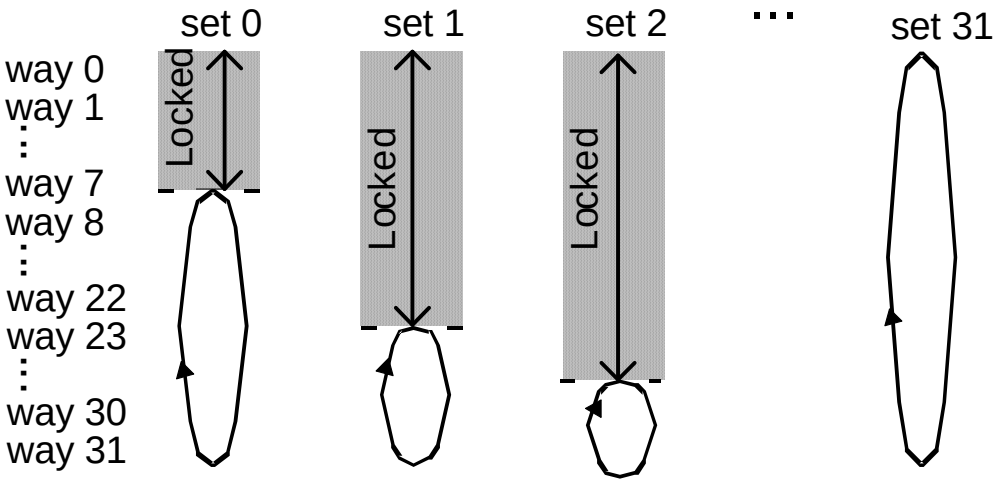
```
; R1 contains the virtual address of a region of memory to configure as data RAM,
; which is aligned on a 32-byte boundary.
; MMU is configured so that the memory region is cacheable.
; R0 is the number of 32-byte lines to designate as data RAM. In this example 16
; lines of the data cache are re-configured as data RAM.
; The inner loop is used to initialize the newly allocated lines
; MMU and data cache are enabled prior to this code.

MACRO ALLOCATE Rx
    MCR P15, 0, Rx, C7, C2, 5
ENDM

MACRO DRAIN
    MCR P15, 0, R0, C7, C10, 4    ; drain pending loads and stores
ENDM

DRAIN
MOV R4, #0x0
MOV R5, #0x0
MOV R2, #0x1
MCR P15,0,R2,C9,C2,0                      ; Put the data cache in lock mode
CPWAIT

MOV R0, #16
LOOP1:
ALLOCATE R1               ; Allocate and lock a tag into the data cache at
                          ; address [R1].
; initialize 32 bytes of newly allocated line
DRAIN
STRD R4, [R1],#8       ;
STRD R4, [R1],#8   ;
STRD R4, [R1],#8       ;
STRD R4, [R1],#8       ;

SUBS R0, R0, #1           ; Decrement loop count
BNE LOOP1
; Turn off data cache locking

DRAIN                             ; Finish all pending operations

MOV R2, #0x0
MCR P15,0,R2,C9,C2,0; Take the data cache out of lock mode.
CPWAIT
```

Tags are locked into the data cache by enabling the data cache lock mode bit located in coprocessor 15, register 9. (See Table 21, “Cache Lock-Down Functions” on page 93 for the exact command.) Once enabled, any new lines allocated into the data cache are locked down.

Note: The **PLD** instruction does not affect the cache contents if it encounters an error when executing. For this reason, system software should ensure the memory address used in the **PLD** is correct. If this cannot be ascertained, replace the **PLD** with a **LDR** instruction that targets a scratch register.

Lines are locked into a set starting at way0 and may progress up to way 27, and set a line gets locked into depends on the set index of the virtual address of the request. Figure 9, “Locked Line Effect on Round-Robin Replacement” on page 70 is an example of where lines of code are locked into the cache along with how the round-robin pointer is affected.

Figure 14. Locked Line Effect on Round-Robin Replacement



Notes:
1. Set 0: 8 ways locked, 24 ways available for round robin replacement
2. Set 1: 23 ways locked, 9 ways available for round robin replacement
3. Set 2: 28 ways locked, only ways 28-31 available for replacement
4. Set 31: all 32 ways available for round robin replacement

Software can lock down data located at various memory locations. This may cause some sets to have more locked lines than others as shown in Figure 9.

Lines are unlocked in the data cache by performing an unlock operation. See "Register 9: Cache Lock Down" on page 93 for more information about locking and unlocking the data cache.

Before locking, the programmer must ensure that no part of the target data range is already resident in the cache. The Intel XScale processor does not prefetch such data, and results in it not being locked into the cache. If there is any doubt as to the location of the targeted memory data, the cache should be cleaned and invalidated to prevent this scenario. If the cache contains a locked region and the programmer wishes to lock again, then the cache is unlocked before being cleaned and invalidated.

See Section 1.5, "Terminology and Conventions" for a definition of coalescing.

The write buffer is always enabled that means, stores to external memory is buffered. The K bit in the Auxiliary Control Register (CP15, register 1) is a global enable/disable for allowing coalescing in the write buffer. When this bit disables coalescing, no coalescing occurs regardless the value of the page attributes. If this bit enables coalescing, the page attributes X, C, and B are examined to see if coalescing is enabled for each region of memory.

All reads and writes to external memory occur in program order when coalescing is disabled in the write buffer. If coalescing is enabled in the write buffer, writes may occur out of program order to external memory. Program correctness is maintained in this case by comparing all store requests with all the valid entries in the fill buffer.

The write buffer and fill buffer support a drain operation, such that before the next instruction executes, all Intel XScale processor data requests to external memory have completed. Refer Table 19, “Cache Functions” on page 92 for the exact command.

Writes to a region marked non-cacheable/non-bufferable (page attributes C, B, and X all 0) causes execution to stall until the write completes.

If software is running in a privileged mode, it can explicitly drain all buffered writes. For details on this operation, see the description of Drain Write Buffer in “Register 7: Cache Functions” on page 91.

3.5 Configuration

This section describes the System Control Coprocessor (CP15) and coprocessor 14 (CP14). CP15 configures the MMU, caches, buffers and other system attributes. Where possible, the definition of CP15 follows the definition of the Intel StrongARM products. CP14 contains the performance monitor registers, clock and power management registers and the debug registers.

CP15 is accessed through **MRC** and **MCR** coprocessor instructions and allowed only in privileged mode. Any access to CP15 in user mode or with **LDC** or **STC** coprocessor instructions causes an undefined instruction exception.

All CP14 registers are accessed through **MRC** and **MCR** coprocessor instructions. **LDC** and **STC** coprocessor instructions can only access the clock and power management registers, and the debug registers. The performance monitoring registers can’t be accessed by **LDC** and **STC** because CRm!= 0x0. Access to all registers is allowed only in privileged mode. Any access to CP14 in user mode causes an undefined instruction exception.

Coprocessors, CP15 and CP14, on the Intel XScale processor do not support access via **CDP**, **MRRC**, or **MCRR** instructions. An attempt to access these coprocessors with these instructions results in an undefined instruction exception.

Many of the MCR commands available in CP15 modify hardware state sometime after execution. A software sequence is available for those wishing to determine when this update occurs and is found in “Additions to CP15 Functionality” on page 160.

Like certain other Intel StrongARM architecture products, the Intel XScale processor includes an extra level of virtual address translation in the form of a PID (Process ID) register and associated logic. For a detailed description of this facility, see “Register 13: Process ID” on page 94. Privileged code must be aware of this facility because, when interacting with CP15, some addresses are modified by the PID and others are not.

An address that has yet to be modified by the PID (PIDified) is known as a virtual address (VA). An address that has been through the PID logic, but not translated into a physical address, is a modified virtual address (MVA). Non-privileged code always deals with VAs, as privileged code that programs CP15 occasionally use MVAs.

The format of **MRC** and **MCR** is shown in Table 8.

cp_num is defined for CP15, CP14 and CP0 on the Intel XScale processor. CP0 supports instructions specific for DSP and is described in “Programming Model” on page 151

Unless otherwise noted, unused bits in coprocessor registers have unpredictable values when read. For compatibility with future implementations, software should not rely on the values in those bits.

Table 8. MRC/MCR Format

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
cond				1	1	1	0	opcode_1			n	CRn				Rd				cp_num				opcode_2			1	CRm			

Bits	Description	Notes
31:28	**cond** - Intel StrongARM condition codes	-
23:21	**opcode_1** - Reserved	Should be programmed to zero for future compatibility
20	**n** - Read or write coprocessor register 0 = MCR 1 = MRC	-
19:16	**CRn** - specifies which coprocessor register	-
15:12	**Rd** - General-Purpose Register, R0..R15	-
11:8	**cp_num** - coprocessor number	Intel XScale processor defines three coprocessors: 0b1111 = CP15 0b1110 = CP14 0x0000 = CP0 ***Note:*** Mappings are implementation defined for all coprocessors below CP14 and above CP0. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:5	**opcode_2** - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.
3:0	**CRm** - Function bits	This field should be programmed to zero for future compatibility unless a value has been specified in the command.

The format of **LDC** and **STC** for CP14 is shown in Table 9. **LDC** and **STC** follow the programming notes in the *ARM* Architecture Reference Manual*.

Note: Access to CP15 with **LDC** and **STC** causes an undefined exception.

LDC and **STC** transfer a single 32-bit word between a coprocessor register and memory. These instructions do not allow the programmer to specify values for **opcode_1**, **opcode_2**, **or Rm**; those fields implicitly contain zero, that means the performance monitoring registers are not accessible.

Table 9. LDC/STC Format when Accessing CP14 (Sheet 1 of 2)

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
cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8_bit_word_offset								

Bits	Description	Notes
31:28	**cond** - Intel StrongARM condition codes	-
24:23,21	**P, U, W** - specifies 1 of 3 addressing modes identified by addressing mode 5 in the *ARM* Architecture Reference Manual*.	-
22	**N** - should be 0 for CP14 coprocessors. Setting this bit to 1 has an undefined effect.	

Table 9. LDC/STC Format when Accessing CP14 (Sheet 2 of 2)

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
cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8_bit_word_offset							

Bits	Description	Notes
20	**L** - Load or Store 0 = STC 1 = LDC	-
19:16	**Rn** - specifies the base register	-
15:12	**CRd** - specifies the coprocessor register	-
11:8	**cp_num** - coprocessor number	Intel XScale processor defines the following: 0b1111 = Undefined Exception 0b1110 = CP14 ***Note:*** Mappings are implementation defined for all coprocessors below CP13. Access to unimplemented coprocessors (as defined by the cpConfig bus) cause exceptions.
7:0	8-bit word offset	-

3.5.1 CP15 Registers

Table 10 lists the CP15 registers implemented in the IXP43X network processors.

Table 10. CP15 Registers

Register (CRn)	Opcode_2	Access	Description
0	0	Read / Write-Ignored	ID
0	1	Read / Write-Ignored	Cache Type
1	0	Read / Write	Control
1	1	Read / Write	Auxiliary Control
2	0	Read / Write	Translation Table Base
3	0	Read / Write	Domain Access Control
4	-	Unpredictable	(Reserved)
5	0	Read / Write	Fault Status
6	0	Read / Write	Fault Address
7	0	Read-unpredictable / Write	Cache Operations
8	0	Read-unpredictable / Write	TLB Operations
9	0	Read / Write	Cache Lock Down
10	0	Read-unpredictable / Write	TLB Lock Down
11 - 12	-	Unpredictable	(Reserved)
13	0	Read / Write	Process ID (PID)
14	0	Read / Write	Breakpoint Registers
15	0	Read / Write	(CRm = 1) CP Access

3.5.1.1 Register 0: ID & Cache Type Registers

Register 0 houses two read-only register that are used for part identification: an ID register and a cache type register.

The ID Register is selected when opcode_2=0. This register returns the code for the IXP43X network processors.

Table 11. ID Register

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
0	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1	Core Gen				Core Revision				Product Number						Product Revision	

reset value: As Shown

Bits	Access	Description
31:24	Read / Write Ignored	Implementation trademark (0x69 = 'i'= Intel Corporation)
23:16	Read / Write Ignored	Architecture version = Intel StrongARM Version 5TE (0x05 is value returned)
15:13	Read / Write Ignored	Core Generation 0b010 = Intel XScale processor This field reflects a specific set of architecture features supported by the core. If new features are added/ deleted/modified this field changes.
12:10	Read / Write Ignored	Core Revision: This field reflects revisions of core generations. Differences may include errata that dictate various operating conditions, software work-around, and so forth. Value returned is 000b
9:4	Read / Write Ignored	Product Number for:IXP43X network processors 000100b
3:0	Read / Write Ignored	Product Revision for:IXP43X network processors 0001b

The Cache Type Register is selected when opcode_2=1 and describes the cache configuration of the Intel XScale processor.

Table 12. Cache Type Register (Sheet 1 of 2)

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
0	0	0	0	1	0	1	1	0	0	0	Dsize			1	0	1	0	1	0	0	0	0	Isize			1	0	1	0	1	0

reset value: As Shown

Bits	Access	Description
31:29	Read-as-Zero / Write Ignored	Reserved
28:25	Read / Write Ignored	Cache class = 0b0101 The caches support locking, write back and round-robin replacement. They do not support address by index.
24	Read / Write Ignored	Harvard Cache
23:21	Read-as-Zero / Write Ignored	Reserved
20:18	Read / Write Ignored	Data Cache Size (Dsize) 0b110 = 32 KB
17:15	Read / Write Ignored	Data cache associativity = 0b101 = 32-way
14	Read-as-Zero / Write Ignored	Reserved
13:12	Read / Write Ignored	Data cache line length = 0b10 = 8 words/line
11:9	Read-as-Zero / Write Ignored	Reserved

Table 12. Cache Type Register (Sheet 2 of 2)

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
0	0	0	0	1	0	1	1	0	0	0	Dsize			1	0	1	0	1	0	0	0	0	Isize			1	0	1	0	1	0

reset value: As Shown

Bits	Access	Description
8:6	Read / Write Ignored	Instruction cache size (Isize) 0b110 = 32 KB
5:3	Read / Write Ignored	Instruction cache associativity = 0b101 = 32-way
2	Read-as-Zero / Write Ignored	Reserved
1:0	Read / Write Ignored	Instruction cache line length = 0b10 = 8 words/line

3.5.1.2 Register 1: Control and Auxiliary Control Registers

Register 1 is made up of two registers, one that is compliant with Intel StrongARM Version 5TE and referred by opcode_2 = 0x0, and the other which is specific to the Intel XScale processor is referred by opcode_2 = 0x1. The latter is known as the Auxiliary Control Register.

The Exception Vector Relocation bit (bit 13 of the Intel StrongARM control register) allows the vectors to be mapped into high memory rather than their default location at address 0. This bit is readable and writable by software. If the MMU is enabled, the exception vectors is accessed via the usual translation method involving the PID register (see "Register 13: Process ID" on page 94) and the TLBs. To avoid automatic application of the PID to exception vector accesses, software may relocate the exceptions to high memory.

Table 13. Intel® StrongARM* Control Register (Sheet 1 of 2)

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
																		V	I	Z	0	R	S	B	1	1	1	1	C	A	M

reset value: writeable bits set to 0

Bits	Access	Description
31:14	Read-Unpredictable / Write-as-Zero	Reserved
13	Read / Write	Exception Vector Relocation (V). 0 = Base address of exception vectors is 0x0000,0000 1 = Base address of exception vectors is 0xFFFF,0000
12	Read / Write	Instruction Cache Enable/Disable (I) 0 = Disabled 1 = Enabled
11	Read / Write	Branch Target Buffer Enable (Z) 0 = Disabled 1 = Enabled
10	Read-as-Zero / Write-as-Zero	Reserved
9	Read / Write	**ROM Protection (R)** This selects the access checks performed by the memory management unit. See the *ARM* Architecture Reference Manual* for more information.
8	Read / Write	**System Protection (S)** This selects the access checks performed by the memory management unit. See the *ARM* Architecture Reference Manual* for more information.
7	Read / Write	Big- / Little-Endian (B) 0 = Little-endian operation 1 = Big-endian operation
6:3	Read-as-One / Write-as-One	= 0b1111

Table 13. Intel® StrongARM* Control Register (Sheet 2 of 2)

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
																		V	I	Z	0	R	S	B	1	1	1	1	C	A	M

reset value: writeable bits set to 0

Bits	Access	Description
2	Read / Write	Data cache enable/disable (C) 0 = Disabled 1 = Enabled
1	Read / Write	Alignment fault enable/disable (A) 0 = Disabled 1 = Enabled
0	Read / Write	Memory management unit enable/disable (M) 0 = Disabled 1 = Enabled

The mini-data cache attribute bits, in the Auxiliary Control Register, are used to control the allocation policy for the mini-data cache and whether it uses write-back caching or write-through caching.

The configuration of the mini-data cache should be setup before any data access is made that is cached in the mini-data cache. Once data is cached, software must ensure that the mini-data cache has been cleaned and invalidated before the mini-data cache attributes are changed.

Table 14. Auxiliary Control Register

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
																						PTEX				MD		C	B	P	K

reset value: writeable bits set to 0

Bits	Access	Description
31:6	Read-Unpredictable / Write-as-Zero	Reserved
5:4	Read / Write	Mini Data Cache Attributes (MD) All configurations of the Mini-data cache are cacheable, stores are buffered in the write buffer and stores is coalesced in the write buffer as long as coalescing is globally enable (bit 0 of this register). 0b00 = Write back, Read allocate 0b01 = Write back, Read/Write allocate 0b10 = Write through, Read allocate 0b11 = Unpredictable
3:2	Read-Unpredictable/ Write-as-Zero	(Reserved)
1	Read/Write	Page Table Memory Attribute (P) This is a request to the core memory bus for a hardware page table walk. An ASSP may use this bit to direct the external bus controller to perform some special operation on the memory access.
0	Read / Write	Write Buffer Coalescing Disable (K) This bit globally disables the coalescing of all stores in the write buffer no matter what the value of the Cacheable and Bufferable bits are in the page table descriptors. 0 = Enabled 1 = Disabled

3.5.1.3 Register 2: Translation Table Base Register

Table 15. Translation Table Base Register

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
Translation Table Base																															

reset value: unpredictable

Bits	Access	Description
31:14	Read / Write	**Translation Table Base** - Physical address of the base of the first-level table
13:0	Read-unpredictable / Write-as-Zero	Reserved

3.5.1.4 Register 3: Domain Access Control Register

Table 16. Domain Access Control Register

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
D15				D14				D13				D12				D11				D10				D9				D8			
D7				D6				D5				D4				D3				D2				D1				D0			

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**Access permissions for all 16 domains** - The meaning of each field is found in the *ARM* Architecture Reference Manual*.

3.5.1.5 Register 4: Reserved

Register 4 is reserved. Reading and writing this register yields unpredictable results.

3.5.1.6 Register 5: Fault Status Register

The Fault Status Register (FSR) indicates the fault that has occurred, and could be a prefetch abort or a data abort. Bit 10 extends the encoding of the status field for prefetch aborts and data aborts. The definition of the extended status field is found in "Event Architecture" on page 161. Bit 9 indicates that a debug event occurred and the exact source of the event is found in the debug control and status register (CP14, register 10). When bit 9 is set, the domain and extended status field are undefined.

Upon entry into the prefetch abort or data abort handler, hardware updates this register with the source of the exception. Software is not required to clear these fields.

Table 17. Fault Status Register

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
																					X	D	0	Domain				Status			

reset value: unpredictable

Bits	Access	Description
31:11	Read-unpredictable / Write-as-Zero	Reserved
10	Read / Write	Status Field Extension (X) This bit is used to extend the encoding of the Status field, when there is a prefetch abort and when there is a data abort. The definition of this field is found in "Event Architecture" on page 161
9	Read / Write	Debug Event (D) This flag indicates a debug event has occurred and that the cause of the debug event is found in the MOE field of the debug control register (CP14, register 10)
8	Read-as-zero / Write-as-Zero	= 0
7:4	Read / Write	**Domain** - Specifies which of the 16 domains is being accessed when a data abort occurred
3:0	Read / Write	**Status** - Type of data access being attempted

3.5.1.7 Register 6: Fault Address Register

Table 18. Fault Address Register

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
Fault Virtual Address																															

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**Fault Virtual Address** - Contains the MVA of the data access that caused the memory abort

3.5.1.8 Register 7: Cache Functions

All the functions defined in existing Intel StrongARM products appear here. The Intel XScale processor adds other functions also. This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

The Drain Write Buffer function not only drains the write buffer but also drains the fill buffer.

The Intel XScale processor does not check permissions on addresses supplied for cache or TLB functions. Due to the fact only privileged software may execute these functions, full accessibility is assumed. Cache functions does not generate any of the following:

- Translation faults
- Domain faults
- Permission faults

The invalidate instruction cache line command does not invalidate the BTB. If software invalidates a line from the instruction cache and modifies the same location in external memory, it must invalidate the BTB also. Not invalidating the BTB in this case may cause unpredictable results.

Disabling/enabling a cache has no effect on contents of the cache: valid data stays valid, locked items remain locked. All operations defined in Table 19 work regardless of whether the cache is enabled or disabled.

Since the Clean DCache Line function reads from the data cache, it is capable of generating a parity fault. The other operations does not generate parity faults.

Table 19. Cache Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D cache & BTB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c7, c7, 0
Invalidate I cache & BTB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 0
Invalidate I cache line	0b001	0b0101	MVA	MCR p15, 0, Rd, c7, c5, 1
Invalidate D cache	0b000	0b0110	Ignored	MCR p15, 0, Rd, c7, c6, 0
Invalidate D cache line	0b001	0b0110	MVA	MCR p15, 0, Rd, c7, c6, 1
Clean D cache line	0b001	0b1010	MVA	MCR p15, 0, Rd, c7, c10, 1
Drain Write (& Fill) Buffer	0b100	0b1010	Ignored	MCR p15, 0, Rd, c7, c10, 4
Invalidate Branch Target Buffer	0b110	0b0101	Ignored	MCR p15, 0, Rd, c7, c5, 6
Allocate Line in the Data Cache	0b101	0b0010	MVA	MCR p15, 0, Rd, c7, c2, 5

The line-allocate command allocates a tag into the data cache specified by bits [31:5] of Rd. If a valid dirty line (with another MVA) already exists at this location it is evicted. The 32 bytes of data associated with the newly allocated line are not initialized and therefore generates unpredictable results if read.

Line allocate command is used for cleaning the entire data cache on a context switch and also when reconfiguring portions of the data cache as data RAM. In both cases, Rd is a virtual address that maps to some non-existent physical memory. When creating data RAM, software must initialize the data RAM before read accesses can occur. Specific uses of these commands is found in Chapter 3.0, “Data Cache”.

Other items to note about the line-allocate command are:

- It forces all pending memory operations to complete.
- Bits [31:5] of Rd is used to specific the virtual address of the line to be allocated into the data cache.
- If the targeted cache line is already resident, this command has no effect.
- The command cannot be used to allocate a line in the mini Data Cache.
- The newly allocated line is not marked as **dirty** so it never gets evicted. But, if a valid store is made to that line it is marked as **dirty** and gets written back to external memory if another line is allocated to the same cache location. This eviction produces unpredictable results.

  To avoid this situation, the line-allocate operation should only be used if one of the following is guaranteed:

  — The virtual address associated with this command is not one that is generated during normal program execution. This is the case when line-allocate is used to clean/invalidate the entire cache.

  — The line-allocate operation is used only on a cache region destined to be locked. When the region is unlocked, it is invalidated before making another data access.

3.5.1.9 Register 8: TLB Operations

Disabling/enabling the MMU has no effect on the contents of either TLB: valid entries stay valid, locked items remain locked. All operations defined in Table 20 work regardless of whether the TLB is enabled or disabled.

This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 20. TLB Functions

Function	opcode_2	CRm	Data	Instruction
Invalidate I&D TLB	0b000	0b0111	Ignored	MCR p15, 0, Rd, c8, c7, 0
Invalidate I TLB	0b000	0b0101	Ignored	MCR p15, 0, Rd, c8, c5, 0
Invalidate I TLB entry	0b001	0b0101	MVA	MCR p15, 0, Rd, c8, c5, 1
Invalidate D TLB	0b000	0b0110	Ignored	MCR p15, 0, Rd, c8, c6, 0
Invalidate D TLB entry	0b001	0b0110	MVA	MCR p15, 0, Rd, c8, c6, 1

3.5.1.10 Register 9: Cache Lock Down

Register 9 is used for locking down entries into the instruction cache and data cache. (The protocol for locking down entries is found in Chapter 3.0, “Data Cache”.)

Table 21 shows the command for locking down entries in the instruction and data cache. The entry to lock in the instruction cache is specified by the virtual address in Rd. The data cache locking mechanism follows another procedure than the instruction cache. The data cache is placed in lock down mode such that all subsequent fills to the data cache result in that line being locked in, as controlled by Table 22.

Lock/unlock operations on a disabled cache have an undefined effect.

Read and write access is allowed to the data cache lock register bit[0]. All other accesses to register 9 should be write-only; reads, as with an MRC, have an undefined effect.

Table 21. Cache Lock-Down Functions

Function	opcode_2	CRm	Data	Instruction
Fetch and Lock I cache line	0b000	0b0001	MVA	MCR p15, 0, Rd, c9, c1, 0
Unlock Instruction cache	0b001	0b0001	Ignored	MCR p15, 0, Rd, c9, c1, 1
Read data cache lock register	0b000	0b0010	Read lock mode value	MRC p15, 0, Rd, c9, c2, 0
Write data cache lock register	0b000	0b0010	Set/Clear lock mode	MCR p15, 0, Rd, c9, c2, 0
Unlock Data Cache	0b001	0b0010	Ignored	MCR p15, 0, Rd, c9, c2, 1

Table 22. Data Cache Lock Register

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
																															L

reset value: writeable bits set to 0

Bits	Access	Description
31:1	Read-unpredictable / Write-as-Zero	Reserved
0	Read / Write	**Data Cache Lock Mode (L)** 0 = No locking occurs 1 = Any fill into the data cache while this bit is set gets locked in

3.5.1.11 Register 10: TLB Lock Down

Register 10 is used for locking down entries into the instruction TLB, and data TLB. (The protocol for locking down entries is found in Chapter 3.0, "Memory Management Unit".) Lock/unlock operations on a TLB when the MMU is disabled have an undefined effect.

This register should be accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 23 shows the command for locking down entries in the instruction TLB, and data TLB. The entry to lock is specified by the virtual address in Rd.

Table 23. TLB Lockdown Functions

Function	opcode_2	CRm	Data	Instruction
Translate and Lock I TLB entry	0b000	0b0100	MVA	MCR p15, 0, Rd, c10, c4, 0
Translate and Lock D TLB entry	0b000	0b1000	MVA	MCR p15, 0, Rd, c10, c8, 0
Unlock I TLB	0b001	0b0100	Ignored	MCR p15, 0, Rd, c10, c4, 1
Unlock D TLB	0b001	0b1000	Ignored	MCR p15, 0, Rd, c10, c8, 1

3.5.1.12 Register 11-12: Reserved

These registers are reserved. Reading and writing them yields unpredictable results.

3.5.1.13 Register 13: Process ID

The Intel XScale processor supports the remapping of virtual addresses through a Process ID (PID) register. This remapping occurs before the instruction cache, instruction TLB, data cache and data TLB are accessed. The PID register controls when virtual addresses are remapped and to what value.

The PID register is a 7-bit value that is ORed with bits 31:25 of the virtual address when they are zero. This action effectively remaps the address to one of 128 **slots** in the 4 Gbytes of address space. If bits 31:25 are not zero, no remapping occurs. This feature is useful for operating system management of processes that may map to the same virtual address space. In those cases, the virtually mapped caches on the Intel XScale processor would not require invalidating on a process switch.

Table 24. Accessing Process ID

Function	opcode_2	CRm	Instruction
Read Process ID Register	0b000	0b0000	MRC p15, 0, Rd, c13, c0, 0
Write Process ID Register	0b000	0b0000	MCR p15, 0, Rd, c13, c0, 0

Table 25. Process ID Register

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
Process ID																															

reset value: 0x0000,0000

Bits	Access	Description
31:25	Read / Write	**Process ID** - This field is used for remapping the virtual address when bits 31-25 of the virtual address are zero.
24:0	Read-as-Zero / Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility

3.5.1.14 The PID Register Affect On Addresses

All addresses generated and used by User Mode code are eligible for being **PIDified** as described in the previous section. Privileged code, but, is aware of certain special cases where address generation does not follow the usual flow.

The PID register is not used to remap the virtual address when accessing the Branch Target Buffer (BTB). Any writes to the PID register invalidates the BTB, and that prevents any virtual addresses from being double mapped between two processes.

A breakpoint address (see "Register 14: Breakpoint Registers" on page 95) is expressed as an MVA when written to the breakpoint register. This requirement means the value of the PID is combined appropriately with the address before it is written to the breakpoint register. All virtual addresses in translation descriptors (see "Memory Management Unit" on page 57) are MVAs.

3.5.1.15 Register 14: Breakpoint Registers

The Intel XScale processor contains two instruction breakpoint address registers (IBCR0 and IBCR1), one data breakpoint address register (DBR0), one configurable data mask/address register (DBR1), and one data breakpoint control register (DBCON). The Intel XScale processor also supports a 256-entry, trace buffer that records program execution information. The registers to control the trace buffer are located in CP14.

Refer to "Software Debug" on page 99 for more information on these features of the Intel XScale processor.

Table 26. Accessing the Debug Registers

Function	opcode_2	CRm	Instruction
Access Instruction Breakpoint Control Register 0 (IBCR0)	0b000	0b1000	MRC p15, 0, Rd, c14, c8, 0; read MCR p15, 0, Rd, c14, c8, 0 ; write
Access Instruction Breakpoint Control Register 1(IBCR1)	0b000	0b1001	MRC p15, 0, Rd, c14, c9, 0; read MCR p15, 0, Rd, c14, c9, 0 ; write
Access Data Breakpoint Address Register (DBR0)	0b000	0b0000	MRC p15, 0, Rd, c14, c0, 0 ; read MCR p15, 0, Rd, c14, c0, 0 ; write
Access Data Mask/Address Register (DBR1)	0b000	0b0011	MRC p15, 0, Rd, c14, c3, 0 ; read MCR p15, 0, Rd, c14, c3, 0 ; write
Access Data Breakpoint Control Register (DBCON)	0b000	0b0100	MRC p15, 0, Rd, c14, c4, 0 ; read MCR p15, 0, Rd, c14, c4, 0 ; write

3.5.1.16 Register 15: Coprocessor Access Register

This register is selected when opcode_2 = 0 and CRm = 1.

This register controls access rights to all the coprocessors in the system except for CP15 and CP14. Both CP15 and CP14 can only be accessed in privilege mode. This register is accessed with an MCR or MRC with the CRm field set to 1.

This register controls access to CP0, atypical use for this register is for an operating system to control resource sharing among applications. Initially, all applications are denied access to shared resources by clearing the appropriate coprocessor bit in the Coprocessor Access Register. An application may request the use of a shared resource (for example, the accumulator in CP0) by issuing an access to the resource, and results in an undefined exception. The operating system may grant access to this coprocessor by setting the appropriate bit in the Coprocessor Access Register and return to the application where the access is retried.

Sharing resources among various applications requires a state saving mechanism. Two possibilities are:

- The operating system, during a context switch, could save the state of the coprocessor if the last executing process had access rights to the coprocessor.
- The operating system, during a request for access, saves off the old coprocessor state and saves it with last process to have access to it.

Under both scenarios, the OS must restore state when a request for access is made. This means the OS has to maintain a list of what processes are modifying CP0 and their associated state.

Example 12. Disallowing access to CP0

```
;; The following code clears bit 0 of the CPAR.

;; This causes the processor to fault if software

;; attempts to access CP0.


      LDR R0, =0x3FFE                    ; bit 0 is clear

      MCR P15, 0, R0, C15, C1, 0         ; move to CPAR

      CPWAIT                             ; wait for effect
```

Table 27. Coprocessor Access Register

31:16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	CP13	CP12	CP11	CP10	CP9	CP8	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0

reset value: 0x0000,0000

Bits	Access	Description
31:16	Read-unpredictable / Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility
15:14	Read-as-Zero/Write-as-Zero	**Reserved** - Should be programmed to zero for future compatibility
13:1	Read / Write	**Coprocessor Access Rights**- Each bit in this field corresponds to the access rights for each coprocessor.
0	Read / Write	**Coprocessor Access Rights**- This bit corresponds to the access rights for CP0. 0 = Access denied. Any attempt to access the corresponding coprocessor generates an undefined exception. 1 = Access allowed. Includes read and write accesses.

3.5.2 CP14 Registers

Table 28 lists the CP14 registers implemented in the Intel XScale processor.

Table 28. CP14 Registers

Description	Access	Register# (CRn)	Register# (CRm)
Performance Monitoring	Read / Write	0,1,4,5,8	1
		0-3	2
Clock and Power Management	Read / Write	6-7	0
Software Debug	Read / Write	8-15	0

All other registers are reserved in CP14. Reading and writing them yields unpredictable results.

3.5.2.1 Performance Monitoring Registers

The performance monitoring unit contains a control register (PMNC), a clock counter (CCNT), interrupt enable register (INTEN), overflow flag register (FLAG), event selection register (EVTSEL) and four event counters (PMN0 through PMN3). The format of these registers is found in "Performance Monitoring" on page 141, along with a description on how to use the performance monitoring facility.

Opcode_2 should be zero on all accesses.

These registers can't be accessed by **LDC** and **STC** coprocessor instructions.

Table 29. Accessing the Performance Monitoring Registers

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

3.5.2.2 Clock and Power Management Registers

These registers contain functions for managing the core clock and power. These registers are not implemented for the IXP43X network processors and are reserved for future use.

Table 30. PWRMODE Register

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
																															M

reset value: writeable bits set to 0

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero	Reserved
1:0	Read / Write	**Mode (M)** 0 = ACTIVE Never change from 00b

The Intel XScale processor clock frequency cannot be changed by the software on the IXP43X network processors.

Table 31. Clock and Power Management

Function	Data	Instruction
Read CCLKCFG	ignored	MRC p14, 0, Rd, c6, c0, 0
Write CCLKCFG	CCLKCFG value	MCR p14, 0, Rd, c6, c0, 0

Table 32. CCLKCFG Register

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
																												CCLKCFG			

reset value: unpredictable

Bits	Access	Description
31:0	Read-unpredictable / Write-as-Zero always	Reserved (write as zero)

3.5.2.3 Software Debug Registers

Software debug is supported by address breakpoint registers (Coprocessor 15, register 14), serial communication over the JTAG interface and a trace buffer. Registers 8 and 9 are used for the serial interface and registers 10 through 13 support a 256 entry trace buffer. Register 14 and 15 are the debug link register and debug SPSR (saved program status register). These registers are explained in more detail in “Software Debug” on page 99.

Opcode_2 and CRm should be zero.

Table 33. Accessing the Debug Registers (Sheet 1 of 2)

Function	CRn (Register #)	Instruction
Access Transmit Debug Register (TX)	0b1000	MRC p14, 0, Rd, c8, c0, 0 MCR p14, 0, Rd, c8, c0, 0
Access Receive Debug Register (RX)	0b1001	MCR p14, 0, Rd, c9, c0, 0 MRC p14, 0, Rd, c9, c0, 0
Access Debug Control and Status Register (DBGCSR)	0b1010	MCR p14, 0, Rd, c10, c0, 0 MRC p14, 0, Rd, c10, c0, 0
Access Trace Buffer Register (TBREG)	0b1011	MCR p14, 0, Rd, c11, c0, 0 MRC p14, 0, Rd, c11, c0, 0

Table 33. Accessing the Debug Registers (Sheet 2 of 2)

Function	CRn (Register #)	Instruction
Access Checkpoint 0 Register (CHKPT0)	0b1100	MCR p14, 0, Rd, c12, c0, 0 MRC p14, 0, Rd, c12, c0, 0
Access Checkpoint 1 Register (CHKPT1)	0b1101	MCR p14, 0, Rd, c13, c0, 0 MRC p14, 0, Rd, c13, c0, 0
Access Transmit and Receive Debug Control Register	0b1110	MCR p14, 0, Rd, c14, c0, 0 MRC p14, 0, Rd, c14, c0, 0

3.6 Software Debug

This section describes the software debug and related features implemented in the IXP43X network processors, namely:

- Debug modes, registers and exceptions
- A serial debug communication link via the JTAG interface
- A trace buffer
- A mechanism to load the instruction cache through JTAG
- Debug Handler SW requirements and suggestions

3.6.1 Definitions

Debug handler: Debug handler is an event handler that runs on the IXP43X network processors, when a debug event occurs

Debugger: The debugger is software that runs on a host system outside of the IXP43X network processors

3.6.2 Debug Registers

CP15 Registers
CRn = 14; CRm = 8: instruction breakpoint register 0 (IBCR0)
CRn = 14; CRm = 9: instruction breakpoint register 1 (IBCR1)
CRn = 14; CRm = 0: data breakpoint register 0 (DBR0)
CRn = 14; CRm = 3: data breakpoint register 1 (DBR1)
CRn = 14; CRm = 4: data breakpoint control register (DBCON)

CP15 registers are accessible using MRC and MCR. CRn and CRm specify the register to access. The opcode_1 and opcode_2 fields are not used and should be set to 0.

CP14 Registers
CRn = 8; CRm = 0: TX Register (TX)
CRn = 9; CRm = 0: RX Register (RX)
CRn = 10; CRm = 0: Debug Control and Status Register (DCSR)
CRn = 11; CRm = 0: Trace Buffer Register (TBREG)
CRn = 12; CRm = 0: Checkpoint Register 0 (CHKPT0)
CRn = 13; CRm = 0: Checkpoint Register 1 (CHKPT1)
CRn = 14; CRm = 0: TXRX Control Register (TXRXCTRL)

CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers causes an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.

Software access to all debug registers is done in privileged mode. User mode access generates an undefined instruction exception. Specifying registers that do not exist has unpredictable results.

The TX and RX registers, certain bits in the TXRXCTRL register, and certain bits in the DCSR is accessed by a debugger through the JTAG interface.

3.6.3 Debug Modes

The debug unit for the IXP43X network processors, when used with a debugger application, allows software running on the target of the IXP43X network processors to be debugged. The debug unit allows the debugger to stop program execution and re-direct execution to a debug handling routine. Once program execution has stopped, the debugger can examine or modify processor state, coprocessor state, or memory. The debugger can then restart execution of the application.

One of the following two debug modes is entered on the IXP43X network processors:

- Halt mode
- Monitor mode

3.6.3.1 Halt Mode

When the debug unit is configured for halt mode, the reset vector is overloaded to serve as the debug vector. A new processor mode, DEBUG mode (CPSR[4:0] = 0x15), is added to allow debug exceptions to be handled similarly to other types of Intel StrongARM exceptions.

When a debug exception occurs, the processor switches to debug mode and redirects execution to a debug handler, via the reset vector. After the debug handler begins execution, the debugger can communicate with the debug handler to examine or alter processor state or memory through the JTAG interface.

The debug handler is downloaded and locked directly into the instruction cache through JTAG so external memory is not required to contain debug handler code.

3.6.3.2 Monitor Mode

In monitor mode, debug exceptions are handled like Intel StrongARM prefetch aborts or Intel StrongARM data aborts, depending on the cause of the exception.

When a debug exception occurs, the processor switches to abort mode and branches to a debug handler using the pre-fetch abort vector or data abort vector. The debugger then communicates with the debug handler to access processor state or memory contents.

3.6.4 Debug Control and Status Register (DCSR)

The DCSR register is the main control register for the debug unit. Table 34 shows the format of the register. The DCSR register is accessed in privileged modes by software running on the core or by a debugger through the JTAG interface. Refer to “SELDCSR JTAG Register” on page 113 for details about accessing DCSR through JTAG.

Table 34. Debug Control and Status Register (DCSR)

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
GE	H							TF	TI		TD	TA	TS	TU	TR											SA	MOE			M	E

Bits	Access	Description	Reset Value	TRST Value
31	SW Read / Write JTAG Read-Only	Global Enable (GE) 0: disables all debug functionality 1: enables all debug functionality	0	unchanged
30	SW Read Only JTAG Read / Write	Halt Mode (H) 0: Monitor Mode 1: Halt Mode	unchanged	0
29:24	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
23	SW Read Only JTAG Read / Write	Trap FIQ (TF)	unchanged	0
22	SW Read Only JTAG Read / Write	Trap IRQ (TI)	unchanged	0
21	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
20	SW Read Only JTAG Read / Write	Trap Data Abort (TD)	unchanged	0
19	SW Read Only JTAG Read / Write	Trap Prefetch Abort (TA)	unchanged	0
18	SW Read Only JTAG Read / Write	Trap Software Interrupt (TS)	unchanged	0
17	SW Read Only JTAG Read / Write	Trap Undefined Instruction (TU)	unchanged	0
16	SW Read Only JTAG Read / Write	Trap Reset (TR)	unchanged	0
15:6	Read-undefined / Write-As-Zero	Reserved	undefined	undefined
5	SW Read / Write JTAG Read-Only	Sticky Abort (SA)	0	unchanged
4:2	SW Read / Write JTAG Read-Only	Method Of Entry (MOE) 000: Processor Reset 001: Instruction Breakpoint Hit 010: Data Breakpoint Hit 011: BKPT Instruction Executed 100: External Debug Event Asserted 101: Vector Trap Occurred 110: Trace Buffer Full Break 111: Reserved	0b000	unchanged
1	SW Read / Write JTAG Read-Only	Trace Buffer Mode (M) 0: Wrap around mode 1: fill-once mode	0	unchanged
0	SW Read / Write JTAG Read-Only	Trace Buffer Enable (E) 0: Disabled 1: Enabled	0	unchanged

3.6.4.1 Global Enable Bit (GE)

The Global Enable bit disables and enables all debug functionality (except the reset vector trap). Following a processor reset, this bit is clear so all debug functionality is disabled. When debug functionality is disabled, the BKPT instruction becomes a loop and external debug breaks, hardware breakpoints, and non-reset vector traps are ignored.

3.6.4.2 Halt Mode Bit (H)

The Halt Mode bit configures the debug unit for halt mode or monitor mode.

3.6.4.3 Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR)

The Vector Trap bits allow instruction breakpoints to be set on exception vectors without using up any of the breakpoint registers. When a bit is set, it acts as if an instruction breakpoint is set up on the corresponding exception vector. A debug exception is generated before the instruction in the exception vector executes.

Software running on the IXP43X network processors must set the Global Enable bit and the debugger must set the Halt Mode bit and the appropriate vector trap bit through JTAG to set up a non-reset vector trap.

To set up a reset vector trap, the debugger sets the Halt Mode bit and reset vector trap bit through JTAG. The Global Enable bit does not effect the reset vector trap. A reset vector trap is set up before or during a processor reset. When processor reset is de-asserted, a debug exception occurs before the instruction in the reset vector executes.

3.6.4.4 Sticky Abort Bit (SA)

The Sticky Abort bit is only valid in Halt mode. It indicates a data abort occurred within the special debug state (see “Halt Mode” on page 103). Since special debug state disables all exceptions, a data abort exception does not occur. The processor sets the Sticky Abort bit to indicate a data abort is detected. The debugger can use this bit to determine if a data abort is detected during the special debug state. The sticky abort bit is cleared by the debug handler before exiting the debug handler.

3.6.4.5 Method of Entry Bits (MOE)

The Method of Entry bits specify the cause of the most recent debug exception. When multiple exceptions occur in parallel, the processor places the highest priority exception (based on the priorities in Table 35) in the MOE field.

3.6.4.6 Trace Buffer Mode Bit (M)

The Trace Buffer Mode bit selects one of the two trace buffer modes:

- Wrap-around mode — Trace buffer fills up and wraps around until a debug exception occurs.
- Fill-once mode — The trace buffer automatically generates a debug exception (trace buffer full break) when it becomes full.

3.6.4.7 Trace Buffer Enable Bit (E)

The Trace Buffer Enable bit enables and disables the trace buffer. Both DCSR.e and DCSR.ge is set to enable the trace buffer. The processor automatically clears this bit to disable the trace buffer when a debug exception occurs. For more details on the trace buffer refer to “Trace Buffer” on page 119.

3.6.5 Debug Exceptions

A debug exception causes the processor to re-direct execution to a debug event handling routine. The debug architecture of the IXP43X network processors defines the following debug exceptions:

- Instruction breakpoint
- Data breakpoint
- Software breakpoint
- External debug break
- Exception vector trap
- Trace-buffer full break

When a debug exception occurs, the processor's actions depend on whether the debug unit is configured for Halt mode or Monitor mode.

Table 35 shows the priority of debug exceptions relative to other processor exceptions.

Table 35. Event Priority

Event	Priority
Reset	1
Vector Trap	2
Data abort (precise)	3
Data bkpt	4
Data abort (imprecise)	5
External debug break, trace-buffer full	6
FIQ	7
IRQ	8
Instruction breakpoint	9
Pre-fetch abort	10
Undef, SWI, software Bkpt	11

3.6.5.1 Halt Mode

The debugger turns on Halt mode through the JTAG interface by scanning in a value that sets the bit in DCSR. The debugger turns off Halt mode through JTAG, by scanning in a new DCSR value or by a TRST. Processor reset does not effect the value of the Halt mode bit.

When halt mode is active, the processor uses the reset vector as the debug vector. The debug handler and exception vectors is downloaded directly into the instruction cache, to intercept the default vectors and reset handler, or they are resident in external memory. Downloading into the instruction cache allows a system with memory problems, or no external memory, to be debugged. Refer top "Downloading Code in ICache" on page 125 for details about downloading code into the instruction cache.

During Halt mode, the software running on the IXP43X network processors cannot access DCSR or any of hardware breakpoint registers unless the processor is in a special debug state (SDS) as described below:

When a debug exception occurs during Halt mode, the processor takes the following actions:

- Disables the trace buffer
- Sets DCSR.moe encoding
- Processor enters a special debug state (SDS)
- For data breakpoints, trace buffer full break, and external debug break:
  R14_dbg = PC of the next instruction to execute + 4
  for instruction breakpoints and software breakpoints and vector traps:
  R14_dbg = PC of the aborted instruction + 4
- SPSR_dbg = CPSR
- CPSR[4:0] = 0b10101 (DEBUG mode)
- CPSR[5] = 0
- CPSR[6] = 1
- CPSR[7] = 1
- PC = 0x0

Note: When the vector table is relocated (CP15 Control Register[13] = 1), the debug vector is relocated to 0xffff0000.

Following a debug exception, the processor switches to debug mode and enters SDS, and allows the following special functionality:

- All events are disabled. SWI or undefined instructions have unpredictable results. The processor ignores pre-fetch aborts, FIQ and IRQ (SDS disables FIQ and IRQ regardless of the enable values in the CPSR). The processor reports data aborts detected during SDS by setting the Sticky Abort bit in the DCSR, but does not generate an exception (processor also sets up FSR and FAR as it normally would for a data abort).
- Normally, during halt mode, software cannot write the hardware breakpoint registers or the DCSR. But, during the SDS, software has write access to the breakpoint registers (see "HW Breakpoint Resources" on page 105) and the DCSR (see Table 34, "Debug Control and Status Register (DCSR)" on page 101).
- The IMMU is disabled. In halt mode, since the debug handler would typically be downloaded directly into the IC, it would not be appropriate to do TLB accesses or translation walks, since there may not be any external memory or if there is, the translation table or TLB may not contain a valid mapping for the debug handler code. To avoid these problems, the processor internally disables the IMMU during SDS.
- The PID is disabled for instruction fetches. This prevents fetches of the debug handler code from being remapped to another address than where the code is downloaded.

The SDS remains in effect regardless of the processor mode. This allows the debug handler to switch to other modes, maintaining SDS functionality. Entering user mode may cause unpredictable behavior. The processor exits SDS following a CPSR restore operation.

When exiting, the debug handler should use:

```
subs pc, lr, #4
```

This restores CPSR, turns off all of SDS functionality, and branches to the target instruction.

3.6.5.2 Monitor Mode

In monitor mode, the processor handles debug exceptions like normal Intel StrongARM exceptions. If debug functionality is enabled (DCSR[31] = 1) and the processor is in Monitor mode, debug exceptions causes a data abort or a pre-fetch abort.

The following debug exceptions cause data aborts:

- Data breakpoint
- External debug break
- Trace-buffer full break

The following debug exceptions cause pre-fetch aborts:

- Instruction breakpoint
- BKPT instruction

The processor ignores vector traps during monitor mode.

When an exception occurs in monitor mode, the processor takes the following actions:

- Disables the trace buffer
- Sets DCSR.moe encoding
- Sets FSR[9]
- R14_abt = PC of the next instruction to execute + 4 (for Data Aborts)
  R14_abt = PC of the faulting instruction + 4 (for Prefetch Aborts)
- SPSR_abt = CPSR
- CPSR[4:0] = 0b10111 (ABORT mode)
- CPSR[5] = 0
- CPSR[6] = unchanged
- CPSR[7] = 1
- PC = 0xc (for Prefetch Aborts),
  PC = 0x10 (for Data Aborts)

During abort mode, external debug breaks and trace buffer full breaks are internally ended. When the processor exits abort mode, through a CPSR restore or a write directly to the CPSR, the pended debug breaks immediately generates a debug exception. Any pending debug breaks are cleared out when any type of debug exception occurs.

When exiting, the debug handler should do a CPSR restore operation that branches to the next instruction to be executed in the program under debug.

3.6.6 HW Breakpoint Resources

The debug architecture of the IXP43X network processors defines two instruction and two data breakpoint registers, denoted IBCR0, IBCR1, DBR0, and DBR1.

The instruction and data address breakpoint registers are 32-bit registers. The instruction breakpoint causes a break before execution of the target instruction. The data breakpoint causes a break after the memory access has been issued.

In this section Modified Virtual Address (MVA) refers to the virtual address ORed with the PID. Refer to "Register 13: Process ID" on page 94 for more details on the PID. The processor does not OR the PID with the specified breakpoint address prior to doing address comparison. This is done by the programmer and written to the breakpoint register as the MVA. This applies to data and instruction breakpoints.

3.6.6.1 Instruction Breakpoints

The Debug architecture defines two instruction breakpoint registers (IBCR0 and IBCR1). The format of these registers is shown in Table 36., Instruction Breakpoint Address and Control Register (IBCRx). In Intel StrongARM mode, the upper 30 bits contain a word aligned MVA to break on. In Thumb mode, the upper 31 bits contain a half-word aligned MVA to break on. In both modes, bit 0 enables and disables that instruction breakpoint register. Enabling instruction breakpoints debug is globally disabled (DCSR.GE=0) may result in unpredictable behavior.

Table 36. Instruction Breakpoint Address and Control Register (IBCRx)

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
IBCRx																															E

reset value: unpredictable address, disabled

Bits	Access	Description
31:1	Read / Write	Instruction Breakpoint MVA in Intel® StrongARM* mode, IBCRx[1] is ignored
0	Read / Write	**IBCRx Enable (E)** - 0 = Breakpoint disabled 1 = Breakpoint enabled

An instruction breakpoint generates a debug exception before the instruction at the address specified in the ICBR executes. When an instruction breakpoint occurs, the processor sets the DBCR.moe bits to 0b001.

Software must disable the breakpoint before exiting the handler. This allows the breakpointed instruction to execute after the exception is handled.

Single step execution is accomplished using the instruction breakpoint registers and must be completely handled in software (on the host or by the debug handler).

3.6.6.2 Data Breakpoints

The debug architecture of the IXP43X network processors defines two data breakpoint registers (DBR0, DBR1). The format of the registers is shown in Table 37.

Table 37. Data Breakpoint Register (DBRx)

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
DBRx																															

reset value: unpredictable

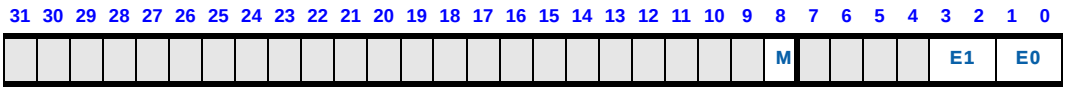
Bits	Access	Description
31:0	Read / Write	**DBR0**: Data Breakpoint MVA **DBR1**: Data Address Mask OR Data Breakpoint MVA

DBR0 is a dedicated data address breakpoint register. DBR1 is programmed for one of two operations:

- Data address mask
- Second data address breakpoint

The DBCON register controls the functionality of DBR1, and the enables for both DBRs. DBCON also controls what type of memory access to break on.

Table 38. Data Breakpoint Controls Register (DBCON)



reset value: 0x00000000

Bits	Access	Description
31:9	Read-as-Zero / Write-ignored	Reserved
8	Read / Write	**DBR1 Mode (M)** - 0: DBR1 = Data Address Breakpoint 1: DBR1 = Data Address Mask
7:4	Read-as-Zero / Write-ignored	Reserved
3:2	Read / Write	**DBR1 Enable (E1)** - When DBR1 = Data Address Breakpoint 0b00: DBR1 disabled 0b01: DBR1 enabled, Store only 0b10: DBR1 enabled, Any data access, load or store 0b11: DBR1 enabled, Load only When DBR1 = Data Address Mask this field has no effect
1:0	Read / Write	**DBR0 Enable (E0)** - 0b00: DBR0 disabled 0b01: DBR0 enabled, Store only 0b10: DBR0 enabled, Any data access, load or store 0b11: DBR0 enabled, Load only

When DBR1 is programmed as a data address mask, it is used in conjunction with the address in DBR0. The bits set in DBR1 are ignored by the processor when comparing the address of a memory access with the address in DBR0. Using DBR1 as a data address mask allows a range of addresses to generate a data breakpoint. When DBR1 is selected as a data address mask, it is unaffected by the E1 field of DBCON. The mask is used only when DBR0 is enabled.

When DBR1 is programmed as a second data address breakpoint, it functions independently of DBR0. In this case, the DBCON.E1 controls DBR1.

A data breakpoint is triggered if the memory access matches the access type and the address of any byte within the memory access matches the address in DBRx. For example, LDR triggers a breakpoint if DBCON.E0 is 0b10 or 0b11, and the address of any of the 4 bytes accessed by the load matches the address in DBR0.

The processor does not trigger data breakpoints for the PLD instruction or any CP15, register 7, 8, 9, or 10 functions. Any other type of memory access can trigger a data breakpoint. For data breakpoint purposes the SWP and SWPB instructions are treated as stores - they do not cause a data breakpoint if the breakpoint is set up to break on loads only and an address match occurs.

On unaligned memory accesses, breakpoint address comparison is done on a word-aligned address (aligned down to word boundary).

When a memory access triggers a data breakpoint, the breakpoint is reported after the access is issued. The memory access is not aborted by the processor. The actual timing of when the access completes with respect to the start of the debug handler depends on the memory configuration.

On a data breakpoint, the processor generates a debug exception and re-directs execution to the debug handler before the next instruction executes. The processor reports the data breakpoint by setting the DCSR.MOE to 0b010. The link register of a data breakpoint is always PC (of the next instruction to execute) + 4, regardless of whether the processor is configured for monitor mode or halt mode.

3.6.7 Software Breakpoints

Mnemonics: BKPT (See *ARM* Architecture Reference Manual*, ARMv5T)

Operation: If DCSR[31] = 0, BKPT is a nop;
If DCSR[31] =1, BKPT causes a debug exception

The processor handles the software breakpoint as described in "Debug Exceptions" on page 103.

3.6.8 Transmit/Receive Control Register

Communications between the debug handler and debugger are controlled through handshaking bits that ensures the debugger and debug handler make synchronized accesses to TX and RX. The debugger side of the handshaking is accessed through the DBGTX ("DBGTX JTAG Register" on page 115) and DBGRX ("DBGRX JTAG Register" on page 116) JTAG Data Registers, depending on the direction of the data transfer.The debug handler uses separate handshaking bits in TXRXCTRL register for accessing TX and RX.

The TXRXCTRL register also contains two other bits that support high-speed download. One bit indicates an overflow condition that occurs when the debugger attempts to write the RX register before the debug handler has read the previous data written to RX. The other bit is used by the debug handler as a branch flag during high-speed download.

All of the bits in the TXRXCTRL register are placed such that they are read directly into the CC flags in the CPSR with an MRC (with Rd = PC). The subsequent instruction can then conditionally execute based on the updated CC value.

Table 39. TX RX Control Register (TXRXCTRL)

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
RR	OV	D	TR																												

reset value: 0x00000000

Bits	Access	Description
31	SW Read-only / Write-ignored JTAG Write-only	RR RX Register Ready
30	SW Read / Write	OV RX overflow sticky flag
29	SW Read-only/ Write-ignored JTAG Write-only	D High-speed download flag
28	SW Read-only/ Write-ignored JTAG Write-only	TR TX Register Ready
27:0	Read-as-Zero / Write-ignored	Reserved

3.6.8.1 RX Register Ready Bit (RR)

The debugger and debug handler use the RR bit to synchronize accesses to RX. Normally, the debugger and debug handler use a handshaking scheme that requires both sides to poll the RR bit. To support higher download performance for large amounts of data, a high-speed download handshaking scheme is used where only the debug handler polls the RR bit before accessing the RX register, when the debugger continuously downloads data.

Table 40 shows the normal handshaking used to access the RX register.

Table 40. Normal RX Handshaking

Debugger Actions
• Debugger wants to send data to debug handler. • Before writing new data to the RX register, the debugger polls RR through JTAG until the bit is cleared. • After the debugger reads a '0' from the RR bit, it scans data into JTAG to write to the RX register and sets the valid bit. The write to the RX register automatically sets the RR bit.
Debug Handler Actions
• Debug handler is expecting data from the debugger. • The debug handler polls the RR bit until it is set, indicating data in the RX register is valid. • Once the RR bit is set, the debug handler reads the new data from the RX register. The read operation automatically clears the RR bit.

When data is being downloaded by the debugger, part of the normal handshaking is bypassed to allow the download rate to be increased. Table 41 shows the handshaking used when the debugger is doing a high-speed download.

Note: Before the high-speed download can start, both the debugger and debug handler is synchronized, such that the debug handler is executing a routine that supports the high-speed download.

Although it is similar to the normal handshaking, the debugger polling of RR is bypassed with the assumption that the debug handler can read the previous data from RX before the debugger can scan in the new data.

Table 41. High-Speed Download Handshaking States

Debugger Actions
• Debugger wants to transfer code into the IXP43X network processors system memory. • Prior to starting download, the debugger must polls RR bit until it is clear. Once the RR bit is clear, indicating the debug handler is ready, the debugger starts the download. • The debugger scans data into JTAG to write to the RX register with the download bit and the valid bit set. Following the write to RX, the RR bit and D bit are automatically set in TXRXCTRL. • Without polling of RR to see whether the debug handler has read the data just scanned in, the debugger continues scanning in new data into JTAG for RX, with the download bit and the valid bit set. • An overflow condition occurs if the debug handler does not read the previous data before the debugger completes scanning in the new data, (See "Overflow Flag (OV)" on page 110 for more details on the overflow condition). • After completing the download, the debugger clears the D bit allowing the debug handler to exit the download loop.
Debug Handler Actions
• Debug is handler in a routine waiting to write data out to memory. The routine loops based on the D bit in TXRXCTRL. • The debug handler polls the RR bit until it is set. It then reads the Rx register, and writes it out to memory. The handler loops, repeating these operations until the debugger clears the D bit.

3.6.8.2 Overflow Flag (OV)

The Overflow flag is a sticky flag that is set when the debugger writes to the RX register as the RR bit is set.

The flag is used during high-speed download to indicate that some data is lost. The assumption during high-speed download is that the time it takes for the debugger to shift in the next data word is greater than the time necessary for the debug handler to process the previous data word. So, before the debugger shifts in the next data word, the handler is polling for that data.

But, if the handler incurs stalls that are long enough such that the handler is still processing the previous data when the debugger completes shifting in the next data word, an overflow condition occurs and the OV bit is set.

Once set, the overflow flag remains set, until cleared by a write to TXRXCTRL with an MCR. After the debugger completes the download, it can examine the OV bit to determine if an overflow occurred. The debug handler software is responsible for saving the address of the last valid store before the overflow occurred.

3.6.8.3 Download Flag (D)

The value of the download flag is set by the debugger through JTAG. This flag is used during high-speed download to replace a loop counter.

The download flag becomes especially useful when an overflow occurs. If a loop counter is used, and an overflow occurs, the debug handler cannot determine how many data words overflowed. Therefore the debug handler counter may get out of synchronization with the debugger — the debugger may finish downloading the data, but the debug handler counter may indicate there is more data to be downloaded - this may result in unpredictable behavior of the debug handler.

Using the download flag, the debug handler loops until the debugger clears the flag. Therefore, when doing a high-speed download, for each data word downloaded, the debugger should set the D bit.

3.6.8.4 TX Register Ready Bit (TR)

The debugger and debug handler use the TR bit to synchronize accesses to the TX register. The debugger and debug handler must poll the TR bit before accessing the TX register. Table 42 shows the handshaking used to access the TX register.

Table 42. TX Handshaking

Debugger Actions
• Debugger is expecting data from the debug handler. • Before reading data from the TX register, the debugger polls the TR bit through JTAG until the bit is set. NOTE: When polling TR, the debugger must scan out the TR bit and the TX register data. • Reading a '1' from the TR bit, indicates that the TX data scanned out is valid • The action of scanning out data when the TR bit is set, automatically clears TR.
Debug Handler Actions
• Debug handler wants to send data to the debugger (in response to a previous request). • The debug handler polls the TR bit to determine when the TX register is empty (any previous data has been read out by the debugger). The handler polls the TR bit until it is clear. • Once the TR bit is clear, the debug handler writes new data to the TX register. The write operation automatically sets the TR bit.

3.6.8.5 Conditional Execution Using TXRXCTRL

All of the bits in TXRXCTRL are placed such that they are read directly into the CC flags using an MCR instruction. To simplify the debug handler, the TXRXCTRL register should be read using the following instruction:

```
mrc p14, 0, r15, C14, C0, 0
```

This instruction directly updates the condition codes in the CPSR. The debug handler can then conditionally execute based on each C bit. Table 43 shows the mnemonic extension to conditionally execute based on whether the TXRXCTRL bit is set or clear.

Table 43. TXRXCTRL Mnemonic Extensions

TXRXCTRL bit	Mnemonic Extension to Execute If Bit Set	Mnemonic Extension to Execute If Bit Clear
31 (to N flag)	MI	PL
30 (to Z flag)	EQ	NE
29 (to C flag)	CS	CC
28 (to V flag)	VS	VC

The following example is a code sequence where the debug handler polls the TXRXCTRL handshaking bit to determine when the debugger has completed its write to RX and the data is ready for the debug handler to read.

```
loop:   mcr     p14, 0, r15, c14, c0, 0# read the handshaking bit in TXRXCTRL

        mcrmi   p14, 0, r0, c9, c0, 0  # if RX is valid, read it

        bpl     loop                   # if RX is not valid, loop
```

3.6.9 Transmit Register

The TX register is the debug handler transmit buffer. The debug handler sends data to the debugger through this register.

Table 44. TX Register

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
TX																															

reset value: unpredictable

Bits	Access	Description
31:0	SW Read / Write JTAG Read-only	Debug handler writes data to send to debugger

Since the TX register is accessed by the debug handler (using MCR/MRC) and the debugger (through JTAG), handshaking is required to prevent the debug handler from writing new data before the debugger reads the previous data.

The TX register handshaking is described in Table 42, “TX Handshaking” on page 111.

3.6.10 Receive Register

The RX register is the receive buffer used by the debug handler to get data sent by the debugger through the JTAG interface.

Table 45. RX Register

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
RX																															

reset value: unpredictable

Bits	Access	Description
31:0	SW Read-only JTAG Write-only	Software reads to receives data/commands from debugger

Since the RX register is accessed by the debug handler (using MRC) and the debugger (through JTAG), handshaking is required to prevent the debugger from writing new data to the register before the debug handler reads the previous data out. The handshaking is described in “RX Register Ready Bit (RR)” on page 109.

3.6.11 Debug JTAG Access

There are four JTAG instructions used by the debugger during software debug: LDIC, SELDCSR, DBGTX and DBGRX. LDIC is described in “Downloading Code in ICache” on page 125. The other three JTAG instructions are described in this section.

SELDCSR, DBGTX and DBGRX use a common 36-bit shift register (DBG_SR). New data is shifted in and captured data out through the DBG_SR. In the UPDATE_DR state, the new data shifted into the appropriate data register.

3.6.11.1 SELDCSR JTAG Command

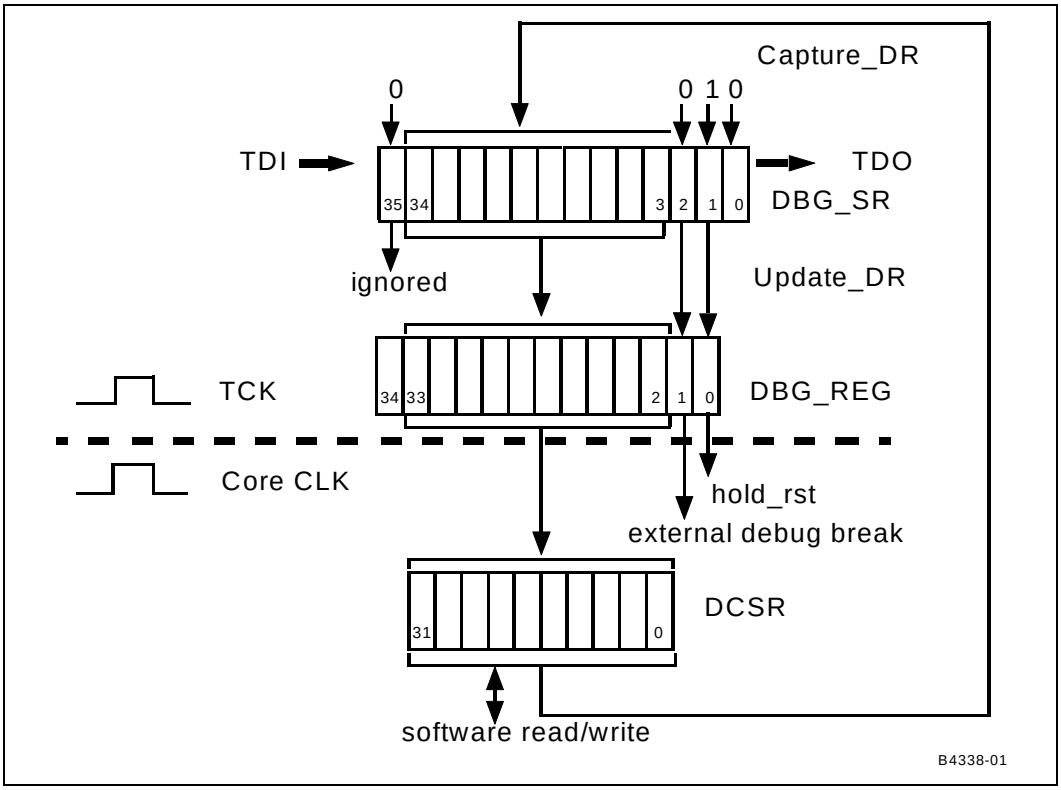
The ‘SELDCSR’ JTAG instruction selects the DCSR JTAG data register. The JTAG op code is ‘01001’. When the SELDCSR JTAG instruction is in the JTAG instruction register, the debugger can directly access the Debug Control and Status Register (DCSR). The debugger can only modify certain bits through JTAG, but can read the entire register.

The SELDCSR instruction also allows the debugger to generate an external debug break.

3.6.11.2 SELDCSR JTAG Register

Placing the **SELDCSR** JTAG instruction in the JTAG IR, selects the DCSR JTAG Data register (Figure 15), allowing the debugger to access the DCSR, generate an external debug break, set the hold_rst signal, and is used when loading code into the instruction cache during reset.

Figure 15. SELDCSR Hardware



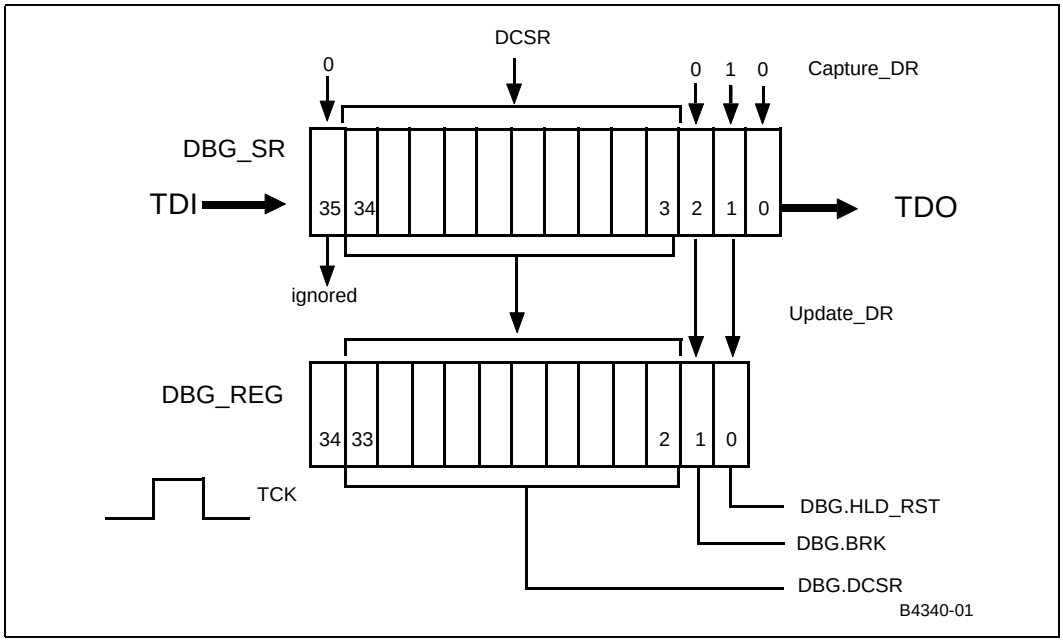
A Capture_DR loads the current DCSR value into DBG_SR[34:3]. The other bits in DBG_SR are loaded as shown in Figure 15.

A new DCSR value is scanned into DBG_SR, and the previous value out, during the Shift_DR state. When scanning in a new DCSR value into the DBG_SR, care is taken to also set up DBG_SR[2:1] to prevent undesirable behavior.

Update_DR parallel loads the new DCSR value into DBG_REG[33:2]. This value is then loaded into the actual DCSR register. All bits defined as JTAG writable in Table 34, “Debug Control and Status Register (DCSR)” on page 101 are updated.

An external host and the debug handler running on the IXP43X network processors must synchronize access the DCSR. If one side writes the DCSR at the same side the other side reads the DCSR, the results are unpredictable.

Figure 16. SELDCSR Data Register



3.6.11.2.1 DBG.HLD_RST

The debugger uses DBG.HLD_RST when loading code into the instruction cache during a processor reset. Details about loading code into the instruction cache are in "Downloading Code in ICache" on page 125.

The debugger must set DBG.HLD_RST before or during assertion of the reset pin. Once DBG.HLD_RST is set, the reset pin is de-asserted, and the processor internally remains in reset. The debugger can then load debug handler code into the instruction cache before the processor begins executing any code.

Once the code download is complete, the debugger must clear DBG.HLD_RST. This takes the processor out of reset, and execution begins at the reset vector.

A debugger sets DBG.HLD_RST in one of two ways:

- By taking the JTAG state machine into the Capture_DR state, and that automatically loads DBG_SR[1] with '1', then the Exit2 state, followed by the Update_Dr state. This sets the DBG.HLD_RST, clear DBG.BRK, and leave the DCSR unchanged (the DCSR bits captured in DBG_SR[34:3] are written back to the DCSR on the Update_DR).
- Alternatively, a '1' is scanned into DBG_SR[1], with the appropriate value scanned in for the DCSR and DBG.BRK.

DBG.HLD_RST can only be cleared by scanning in a '0' to DBG_SR[1] and scanning in the appropriate values for the DCSR and DBG.BRK.

3.6.11.2.2 DBG.BRK

DBG.BRK allows the debugger to generate an external debug break and asynchronously re-direct execution to a debug handling routine.

A debugger sets an external debug break by scanning data into the DBG_SR with DBG_SR[2] set and the desired value to set the DCSR JTAG writable bits in DBG_SR[34:3].

Once an external debug break is set, it remains set internally until a debug exception occurs. In Monitor mode, external debug breaks detected during abort mode are pended until the processor exits abort mode. In Halt mode, breaks detected during SDS are pended until the processor exits SDS. When an external debug break is detected outside of these two cases, the processor ceases executing instructions as quickly as possible. This minimizes breakpoint skid, by reducing the number of instructions that can execute after the external debug break is requested. But, the processor continues to process instructions that may have already begun execution. Debug mode is not to be entered until all processor activity has ceased in an orderly fashion.

3.6.11.2.3 DBG.DCSR

The DCSR is updated with the value loaded into DBG.DCSR following an Update_DR. Only bits specified as writable by JTAG in Table 34 are updated.

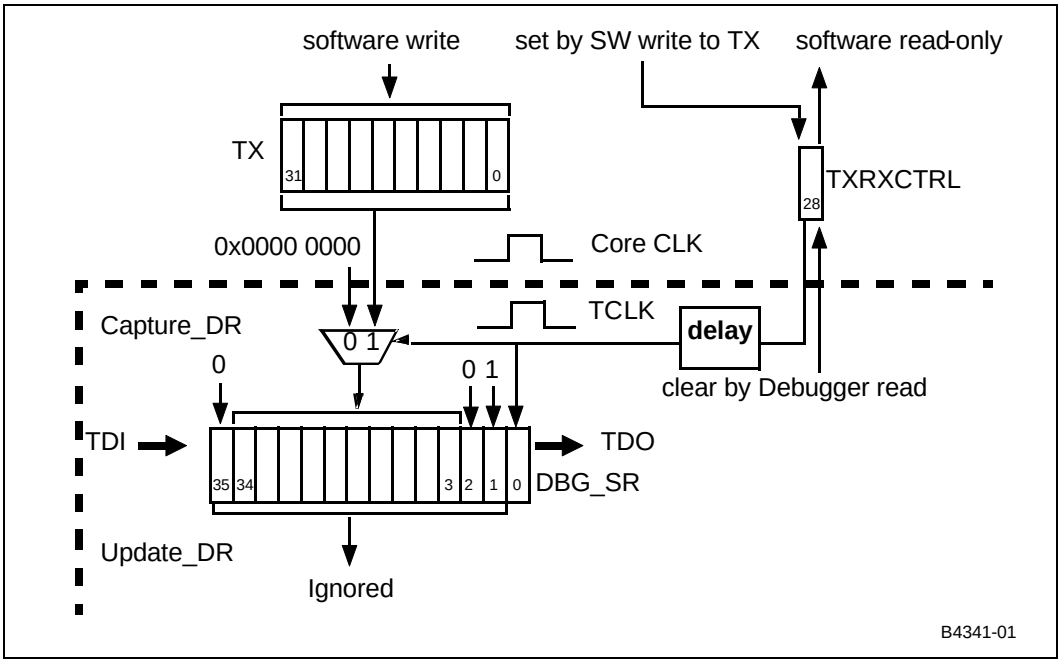
3.6.11.3 DBGTX JTAG Command

The 'DBGTX' JTAG instruction selects the DBGTX JTAG data register. The JTAG op code for this instruction is '0b10000'. Once the DBGTX data register is selected, the debugger can receive data from the debug handler.

3.6.11.4 DBGTX JTAG Register

The DBGTX JTAG instruction selects the Debug JTAG Data register (Figure 17). The debugger uses the DBGTX data register to poll for breaks (internal and external) to debug mode and once in debug mode, to read data from the debug handler.

Figure 17. DBGTX Hardware



A Capture_DR loads the TX register value into DBG_SR[34:3] and TXRXCTRL[28] into DBG_SR[0]. The other bits in DBG_SR are loaded as shown in Figure 21.

The captured TX value is scanned out during the Shift_DR state.

Data scanned in is ignored on an Update_DR.

A '1' captured in DBG_SR[0] indicates the captured TX data is valid. After doing a Capture_DR, the debugger must place the JTAG state machine in the Shift_DR state to guarantee that a debugger read clears TXRXCTRL[28].

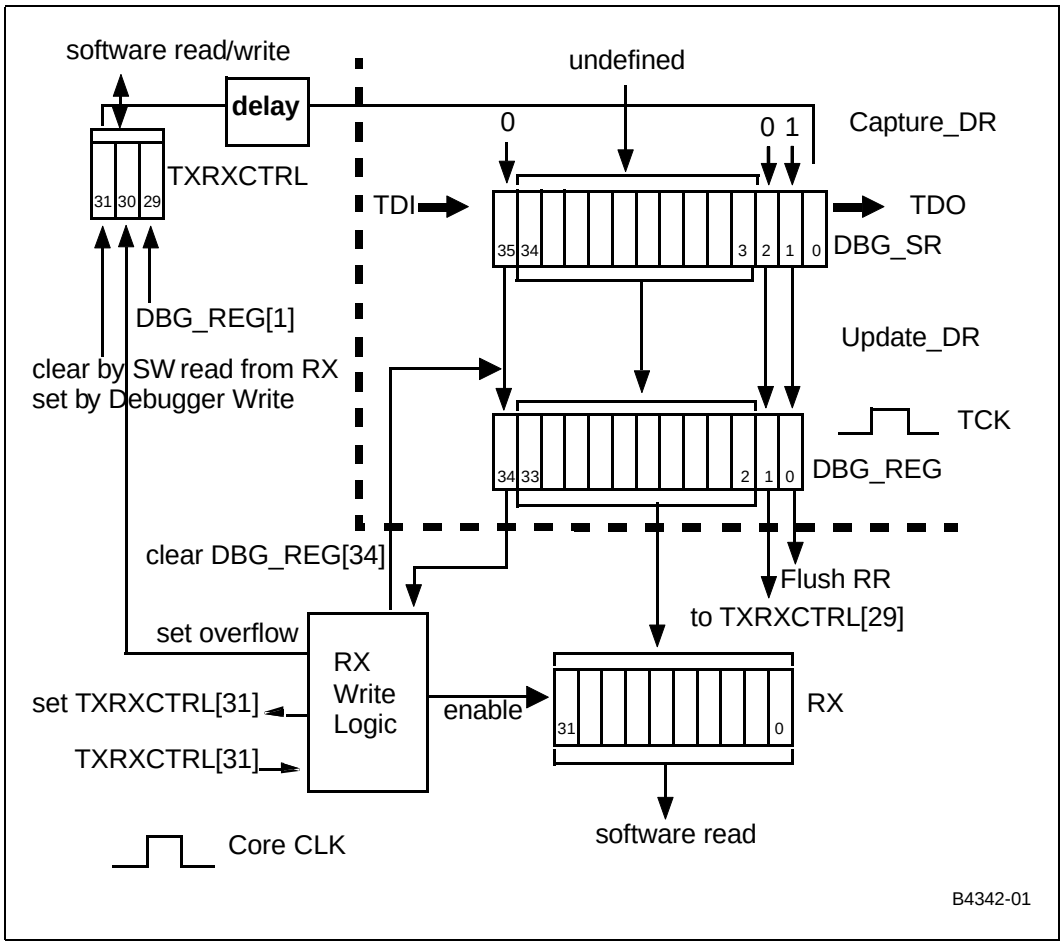
3.6.11.5 DBGRX JTAG Command

The 'DBGRX' JTAG instruction selects the DBGRX JTAG data register. The JTAG op code for this instruction is '0b00010'. Once the DBGRX data register is selected, the debugger can send data to the debug handler through the RX register.

3.6.11.6 DBGRX JTAG Register

The DBGRX JTAG instruction selects the DBGRX JTAG Data register. The debugger uses the DBGRX data register to send data or commands to the debug handler.

Figure 18. DBGRX Hardware



A Capture_DR loads TXRXCTRL[31] into DBG_SR[0]. The other bits in DBG_SR are loaded as shown in Figure 18.

The captured data is scanned out during the Shift_DR state.

Care is taken during scanning in data. When polling TXRXCTRL[31], incorrectly setting DBG_SR[35] or DBG_SR[1] may cause unpredictable behavior following an Update_DR.

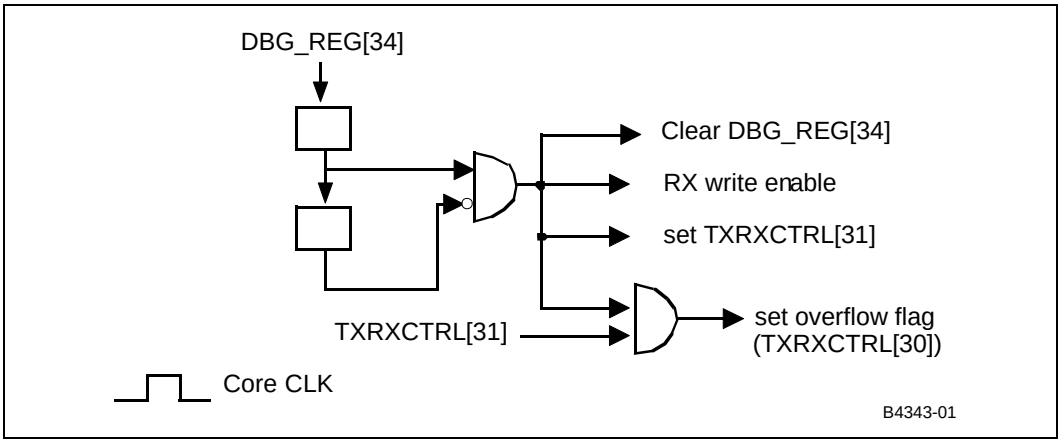
Update_DR parallel loads DBG_SR[35:1] into DBG_REG[34:0]. Whether the new data gets written to the RX register or an overflow condition is detected depends on the inputs to the RX write logic.

3.6.11.6.1 RX Write Logic

The RX write logic (Figure 20) serves 4 functions:

- Enable the debugger write to RX - the logic ensures only new, valid data from the debugger is written to RX. In particular, when the debugger polls TXRXCTRL[31] to see whether the debug handler has read the previous data from RX. The JTAG state machine must go through Update_DR, and should not modify RX.
- Clear DBG_REG[34] - mainly to support high-speed download. During high-speed download, the debugger continuously scan in a data to send to the debug handler and sets DBG_REG[34] to signal the data is valid. Since DBG_REG[34] is never cleared by the debugger in this case, the '0' to '1' transition used to enable the debugger write to RX would not occur.
- Set TXRXCTRL[31] - When the debugger writes new data to RX, the logic automatically sets TXRXCTRL[31], signalling to the debug handler that the data is valid.
- Set the overflow flag (TXRXCTRL[30] - During high-speed download, the debugger does not poll to see if the handler has read the previous data. If the debug handler stalls long enough, the debugger may overwrite the previous data before the handler can read it. The logic sets the overflow flag when the previous data has not been read yet, and the debugger has just written new data to RX.

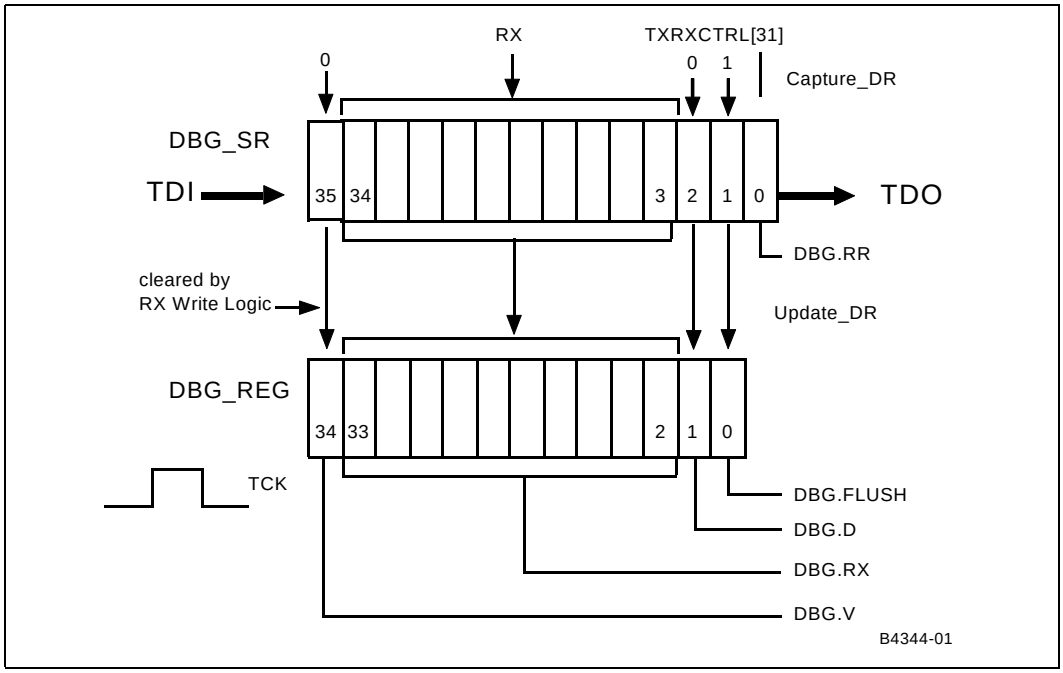
Figure 19. RX Write Logic



3.6.11.6.2 DBGRX Data Register

The bits in the DBGRX data register (Figure 20) are used by the debugger to send data to the processor. The data register also contains a bit to flush previously written data and a high-speed download flag.

Figure 20. DBGRX Data Register



3.6.11.6.3 DBG.RR

The debugger uses DBG.RR as part of the synchronization that occurs between the debugger and debug handler for accessing RX. This bit contains the value of TXRXCTRL[31] after a Capture_DR. The debug handler automatically sets TXRXCTRL[31] by doing a write to RX.

The debugger polls DBG.RR to determine when the handler has read the previous data from RX.

The debugger sets TXRXCTRL[31] by setting the DBG.V bit.

3.6.11.6.4 DBG.V

The debugger sets this bit to indicate the data scanned into DBG_SR[34:3] is valid data to write to RX. DBG.V is an input to the RX Write Logic and is also cleared by the RX Write Logic.

When this bit is set, the data scanned into the DBG_SR is written to RX following an Update_DR. If DBG.V is not set and the debugger does an Update_DR, RX is unchanged.

This bit does not affect the actions of DBG.FLUSH or DBG.D.

3.6.11.6.5 DBG.RX

DBG.RX is written into the RX register based on the output of the RX Write Logic. Any data that must be sent from the debugger to the processor is loaded into DBG.RX with DBG.V set to 1. DBG.RX is loaded from DBG_SR[34:3] when the JTAG enters the Update_DR state.

DBG.RX is written to RX following an Update_DR when the RX Write Logic enables the RX register.

3.6.11.6.6 DBG.D

DBG.D is provided for use during high speed download. This bit is written directly to TXRXCTRL[29]. The debugger sets DBG.D when downloading a block of code or data to the system memory of the IXP43X network processors. The debug handler then uses TXRXCTRL[29] as a branch flag to determine the end of the loop.

Using DBG.D as a branch flags eliminates the need for a loop counter in the debug handler code. This avoids the problem were the debugger's loop counter is out of synchronization with the debug handler's counter because of overflow conditions that may have occurred.

3.6.11.6.7 DBG.FLUSH

DBG.FLUSH allows the debugger to flush any previous data written to RX. Setting DBG.FLUSH clears TXRXCTRL[31].

3.6.11.7 Debug JTAG Data Register Reset Values

Upon asserting TRST, the DEBUG data register is reset. Assertion of the reset pin does not affect the DEBUG data register. Table 46 shows the reset and TRST values for the data register.

Note: These values apply for DBG_REG for SELDCSR, DBGTX and DBGRX.

Table 46. DEBUG Data Register Reset Values

Bit	TRST	RESET
DBG_REG[0]	0	unchanged
DBG_REG[1]	0	unchanged
DBG_REG[33:2]	unpredictable	unpredictable
DBG_REG[34]	0	unchanged

3.6.12 Trace Buffer

The 256-entry trace buffer provides the ability to capture control flow information to be used for debugging an application. Two modes are supported:

- The buffer fills up completely and generates a debug exception. Then SW empties the buffer.
- The buffer fills up and wraps around until it is disabled. Then SW empties the buffer.

3.6.12.1 Trace Buffer CP Registers

CP14 defines three registers (see Table 47) for use with the trace buffer. These CP14 registers are accessible using MRC, MCR, LDC and STC (CDP to any CP14 registers causes an undefined instruction trap). The CRn field specifies the number of the register to access. The CRm, opcode_1, and opcode_2 fields are not used and should be set to 0.

Table 47. CP 14 Trace Buffer Register Summary

CP14 Register Number	Register Name
11	Trace Buffer Register (TBREG)
12	Checkpoint 0 Register (CHKPT0)
13	Checkpoint 1 Register (CHKPT1)

Any access to the trace buffer registers in User mode causes an undefined instruction exception. Specifying registers that do not exist has unpredictable results.

3.6.12.1.1 Checkpoint Registers

When the debugger reconstructs a trace history, it is required to start at the oldest trace buffer entry and construct a trace going forward. In fill-once mode and wrap-around mode when the buffer does not wrap around, the trace is reconstructed by starting from the point in the code where the trace buffer is first enabled.

The difficulty occurs in wrap-around mode when the trace buffer wraps around at least once. In this case the debugger gets a snapshot of the last N control flow changes in the program, where N <= size of buffer. The debugger does not know the starting address of the oldest entry read from the trace buffer. The checkpoint registers provide reference addresses to help reduce this problem.

Table 48. Checkpoint Register (CHKPTx)

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
CHKPTx																															

reset value: Unpredictable

Bits	Access	Description
31:0	Read/Write	**CHKPTx**: target address for corresponding entry in trace buffer

The two checkpoint registers (CHKPT0, CHKPT1) on the IXP43X network processors provides the debugger with two reference addresses to use for re-constructing the trace history.

When the trace buffer is enabled, reading and writing to either checkpoint register has unpredictable results. When the trace buffer is disabled, writing to a checkpoint register sets the register to the value written. Reading the checkpoint registers returns the value of the register.

In normal usage, the checkpoint registers are used to hold target addresses of specific entries in the trace buffer. Only direct and indirect entries get check-pointed. Exception and roll-over messages are never check-pointed. When an entry is check-pointed, the processor sets bit 6 of the message byte to indicate this (refer to Table 50., Message Byte Formats)

When the trace buffer contains only one check-pointed entry, the corresponding checkpoint register is CHKPT0. When the trace buffer wraps around, two entries typically is check-pointed, usually about half a buffers length apart. In this case, the first (oldest) check-pointed entry read from the trace buffer corresponds to CHKPT1, the second check-pointed entry corresponds to CHKPT0.

Although the checkpoint registers are provided for wrap-around mode, they are still valid in fill-once mode.

3.6.12.1.2 Trace Buffer Register (TBREG)

The trace buffer is read through TBREG, using MRC and MCR. Software should only read the trace buffer when it is disabled. Reading the trace buffer as it is enabled, may cause unpredictable behavior of the trace buffer. Writes to the trace buffer have unpredictable results.

Reading the trace buffer returns the oldest byte in the trace buffer in the least significant byte of TBREG. The byte is a message byte or one byte of the 32 bit address associated with an indirect branch message.Table 49 shows the format of the trace buffer register.

Table 49. TBREG Format

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
																								Data							

reset value: unpredictable

Bits	Access	Description
31:8	Read-as-Zero/Write-ignored	Reserved
7:0	Read / Write-unpredictable	Message Byte or Address Byte

3.6.13 Trace Buffer Entries

Trace buffer entries consist of one or five bytes. Most entries are one byte messages indicating the type of control flow change. The target address of the control flow change represented by the message byte is encoded in the message byte (like for exceptions) or is determined by looking at the instruction word (like for direct branches). Indirect branches require five bytes per entry. One byte is the message byte identifying it as an indirect branch. The other four bytes make up the target address of the indirect branch. The following sections describe the trace buffer entries in detail.

3.6.13.1 Message Byte

There are two message formats, (exception and non-exception) as shown in Figure 21.

Figure 21. Message Byte Formats

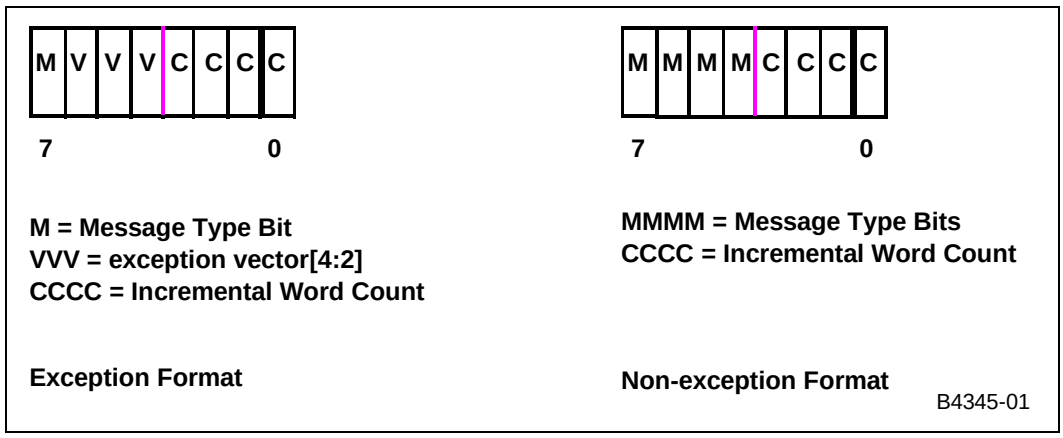


Table 50 shows all of the possible trace messages.

Table 50. Message Byte Formats

Message Name	Message Byte Type	Message Byte Format	# Address Bytes
Exception	exception	0b0VVV CCCC	0
Direct Branch[1]	non-exception	0b1000 CCCC	0
Check-Pointed Direct Branch[1]	non-exception	0b1100 CCCC	0
Indirect Branch[2]	non-exception	0b1001 CCCC	4
Check-Pointed Indirect Branch[2]	non-exception	0b1101 CCCC	4
Roll-over	non-exception	0b1111 1111	0

Notes:
1. Direct branches include Intel StrongARM and THUMB bl, b.
2. Indirect branches include Intel StrongARM ldm, ldr, and dproc to PC; Intel StrongARM and THUMB bx, blx(1) and blx(2); and THUMB pop.

3.6.13.1.1 Exception Message Byte

When any kind of exception occurs, an exception message is placed in the trace buffer. In an exception message byte, the message type bit (M) is always 0.

The vector exception (VVV) field is used to specify bits[4:2] of the vector address (offset from the base of default or relocated vector table). The vector allows the host SW to identify the exception that occurred.

The incremental word count (CCCC) is the instruction count since the last control flow change (not including the current instruction for undef, SWI, and pre-fetch abort). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition of the instruction not matching the CC flags.

A count value of 0 indicates that 0 instructions executed since the last control flow change and the current exception. For example, if a branch is immediate followed by a SWI, a direct branch exception message (for the branch) is followed by an exception message (for the SWI) in the trace buffer. The count value in the exception message is 0, meaning that 0 instructions executed after the last control flow change (the branch) and before the current control flow change (the SWI). Instead of the SWI, if an IRQ is handled immediately after the branch before any other instructions executed, the count is 0, since no instructions executed after the branch and before the interrupt is handled.

A count of 0b1111 indicates that 15 instructions executed between the last branch and the exception. In this case, an exception is caused by the 16th instruction (if it is an undefined instruction exception, pre-fetch abort, or SWI) or handled before the 16th instruction executed (for FIQ, IRQ, or data abort).

3.6.13.1.2 Non-Exception Message Byte

Non-exception message bytes are used for direct branches, indirect branches, and rollovers.

In a non-exception message byte, the four-bit message type field (MMMM) specifies the type of message (refer to Table 50).

The incremental word count (CCCC) is the instruction count since the last control flow change (excluding the current branch). The instruction count includes instructions that were executed and conditional instructions that were not executed due to the condition

of the instruction not matching the CC flags. In the case of back-to-back branches the word count would be 0 indicating that no instructions executed after the last branch and before the current one.

A rollover message is used to keep track of long traces of code that do not have control flow changes. The rollover message means that 16 instructions have executed since the last message byte is written to the trace buffer.

If the incremental counter reaches its maximum value of 15, a rollover message is written to the trace buffer following the next instruction (and is the 16th instruction to execute). This is shown in Example 13. The count in the rollover message is 0b1111, indicating that 15 instructions have executed after the last branch and before the current non-branch instruction that caused the rollover message.

Example 13. Rollover Messages Examples

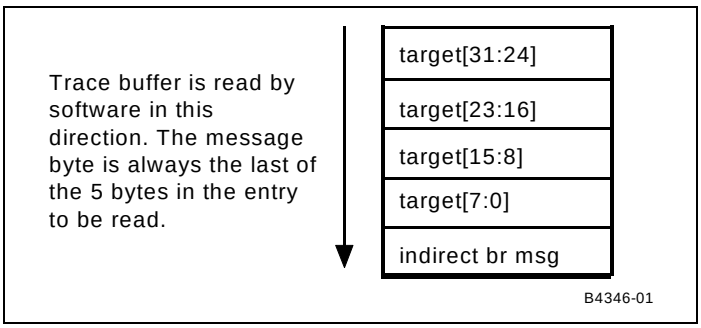
```
count = 5
BL label1          <-- branch message placed in trace buffer after branch executes
count = 0              count = 0b0101
MOV
count = 1
MOV
count = 2
MOV
...

count = 14
MOV
count = 15
MOV                <-- rollover message placed in trace buffer after 16th instruction executes
count = 0              count = 0b1111
```

If the 16th instruction is a branch (direct or indirect), the appropriate branch message is placed in the trace buffer instead of the roll-over message. The incremental counter is still set to 0b1111, meaning 15 instructions executed between the last branch and the current branch.

3.6.13.1.3 Address Bytes

Only indirect branch entries contain address bytes in addition to the message byte. Indirect branch entries always have four address bytes indicating the target of that indirect branch. When reading the trace buffer the MSB of the target address is read out first; the LSB is the fourth byte read out; and the indirect branch message byte is the fifth byte read out. The byte organization of the indirect branch message is shown in Figure 22.

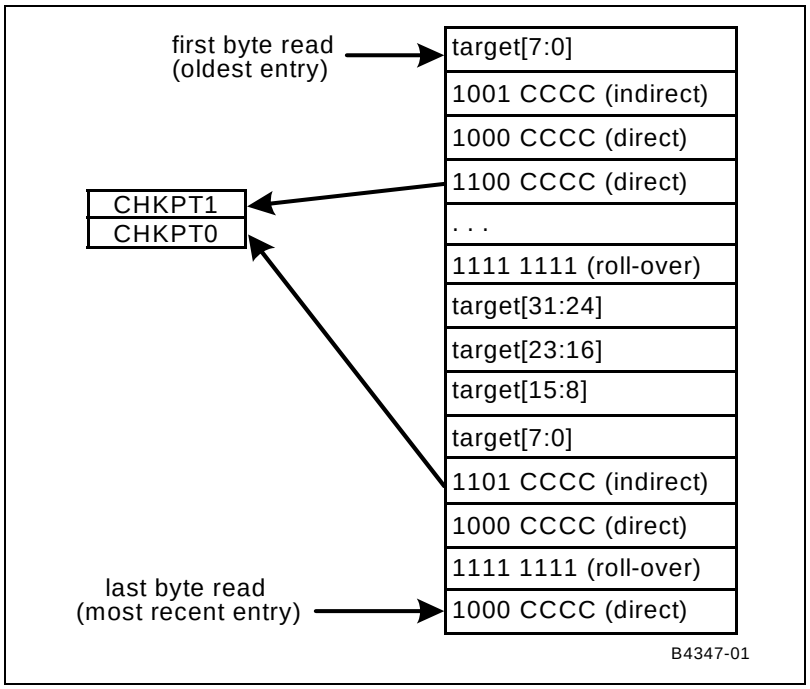
Figure 22. Indirect Branch Entry Address Byte Organization



3.6.13.2 Trace Buffer Usage

The trace buffer for the IXP43X network processors is 256 bytes in length. The first byte read from the buffer represents the oldest trace history information in the buffer. The last (256th) byte read represents the most recent entry in the buffer. The last byte read from the buffer is always a message byte. This provides the debugger with a starting point for parsing the entries out of the buffer. Because the debugger needs the last byte as a starting point when parsing the buffer, the entire trace buffer is read (256 bytes on the IXP43X network processors) before the buffer is parsed. Figure 23 is a high level view of the trace buffer.

Figure 23. High-Level View of Trace Buffer



The trace buffer is initialized prior to its initial usage, then again prior to each subsequent usage. Initialization is done be reading the entire trace buffer. The process of reading the trace buffer also clears it out (all entries are set to 0b0000 0000), so when the trace buffer has been used to capture a trace, the process of reading the captured trace data also re-initializes the trace buffer for its next usage.

The trace buffer is used to capture a trace up to a processor reset. A processor reset disables the trace buffer, but the contents are unaffected. The trace buffer captures a trace up to the processor reset.

The trace buffer does not capture reset events or debug exceptions.

Since the trace buffer is cleared out before it is used, all entries are initially 0b0000 0000. In fill-once mode, these 0's is used to identify the first valid entry in the trace buffer. In wrap around mode, in addition to identifying the first valid entry, these 0 entries is used to determine whether a wrap around occurred.

As the trace buffer is read, the oldest entries are read first. Reading a series of 5 (or more) consecutive **0b0000 0000** entries in the oldest entries indicates that the trace buffer has not wrapped around and the first valid entry is the first non-zero entry read out.

Reading 4 or less consecutive **0b0000 0000** entries requires a bit more intelligence in the host SW. The host SW must determine whether these 0s are part of the address of an indirect branch message, or whether they are part of the **0b0000 0000** that the trace buffer is initialized with. If the first non-zero message byte is an indirect branch message, then these 0s are part of the address since the address is always read before the indirect branch message (see "Address Bytes" on page 123). If the first non-zero entry is any other type of message byte, then these 0s indicate that the trace buffer has not wrapped around and that first non-zero entry is the start of the trace.

If the oldest entry from the trace buffer is non-zero, then the trace buffer has wrapped around or just filled up.

Once the trace buffer has been read and parsed, the host SW should re-create the trace history from oldest trace buffer entry to latest. Trying to re-create the trace going backwards from the latest trace buffer entry may not work in most cases, because once a branch message is encountered, it may not be possible to determine the source of the branch.

In fill-once mode, the return from the debug handler to the application should generate an indirect branch message. The address placed in the trace buffer is that of the target application instruction. Using this as a starting point, re-creating a trace going forward in time should be straightforward.

In wrap around mode, the host SW should use the checkpoint registers and address bytes from indirect branch entries to re-create the trace going forward. The drawback is that some of the oldest entries in the trace buffer are untraceable, depending on where the earliest checkpoint (or indirect branch entry) is located. The best case is when the oldest entry in the trace buffer is check pointed, so the entire trace buffer is used to re-create the trace. The worst case is when the first checkpoint is in the middle of the trace buffer and no indirect branch messages exist before this checkpoint. In this case, the host SW would have to start at its known address (the first checkpoint), that is half way through the buffer and work forward from there.

3.6.14 Downloading Code in ICache

A 2-K mini instruction cache that is physically separate from the 32-K main instruction cache is used as an on-chip instruction RAM on the IXP43X network processors. An external host can download code directly into either instruction cache through JTAG. In addition to downloading code, several cache functions are supported.

Note: A cache line fill from external memory is never written into the mini-instruction cache. The only way to load a line into the mini-instruction cache is through JTAG.

The IXP43X network processors support loading the instruction cache during reset and during program execution. Loading the instruction cache during normal program execution requires a strict handshaking protocol between the software running on the IXP43X network processors and the external host.

In the remainder of this section the term 'instruction cache' applies to main or mini instruction cache.

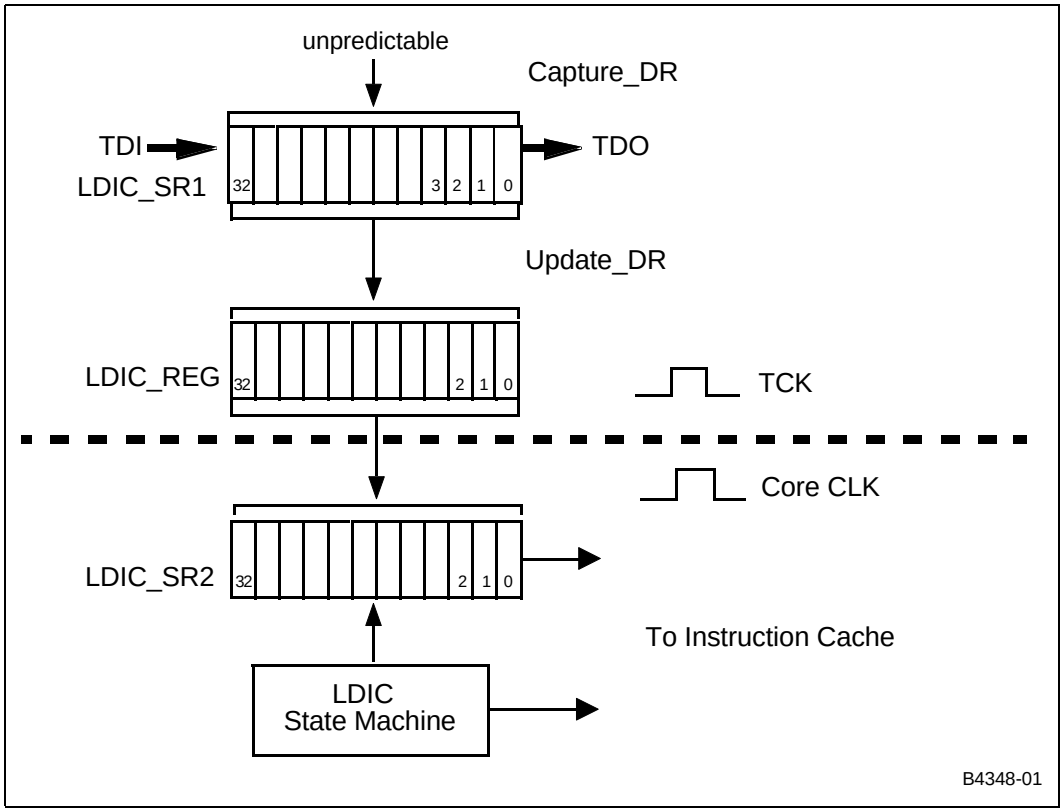
3.6.14.1 LDIC JTAG Command

The LDIC JTAG instruction selects the JTAG data register for loading code into the instruction cache. The JTAG op code for this instruction is '00111'. The LDIC instruction is in the JTAG instruction register to load code directly into the instruction cache through JTAG.

3.6.14.2 LDIC JTAG Data Register

The LDIC JTAG Data Register is selected when the LDIC JTAG instruction is in the JTAG IR. An external host can load and invalidate lines in the instruction cache through this data register.

Figure 24. LDIC JTAG Data Register Hardware



The data loaded into LDIC_SR1 during a Capture_DR is unpredictable.

All LDIC functions and data consists of 33-bit packets that are scanned into LDIC_SR1 during the Shift_DR state.

Update_DR parallel loads LDIC_SR1 into LDIC_REG and is then synchronized with the IXP43X network processors clock and loaded into the LDIC_SR2. Once data is loaded into LDIC_SR2, the LDIC State Machine turns on and serially shifts the contents if LDIC_SR2 to the instruction cache.

Note: There is a delay from the time of the Update_DR to the time the entire contents of LDIC_SR2 have been shifted to the instruction cache. Removing the LDIC JTAG instruction from the JTAG IR before the entire contents of LDIC_SR2 are sent to the instruction cache, results in unpredictable behavior. Therefore, following the Update_DR for the last LDIC packet, the LDIC instruction must remain in the JTAG IR for a minimum of 15 TCKs. This ensures the last packet is correctly sent to the instruction cache.

3.6.14.3 LDIC Cache Functions

The IXP43X network processors support four cache functions that are executed through JTAG. Two functions allow an external host to download code into the main instruction cache or the mini instruction cache through JTAG. Two additional functions are supported to allow lines to be invalidated in the instruction cache. The following table shows the cache functions supported through JTAG.

Table 51. LDIC Cache Functions

Function	Encoding	Arguments	
		Address	# Data Words
Invalidate IC Line	0b000	VA of line to invalidate	0
Invalidate Mini IC	0b001	-	0
Load Main IC	0b010	VA of line to load	8
Load Mini IC	0b011	VA of line to load	8
RESERVED	0b100-0b111	-	-

Invalidate IC line invalidates the line in the instruction cache containing specified virtual address. If the line is not in the cache, the operation has no effect. It does not take any data arguments.

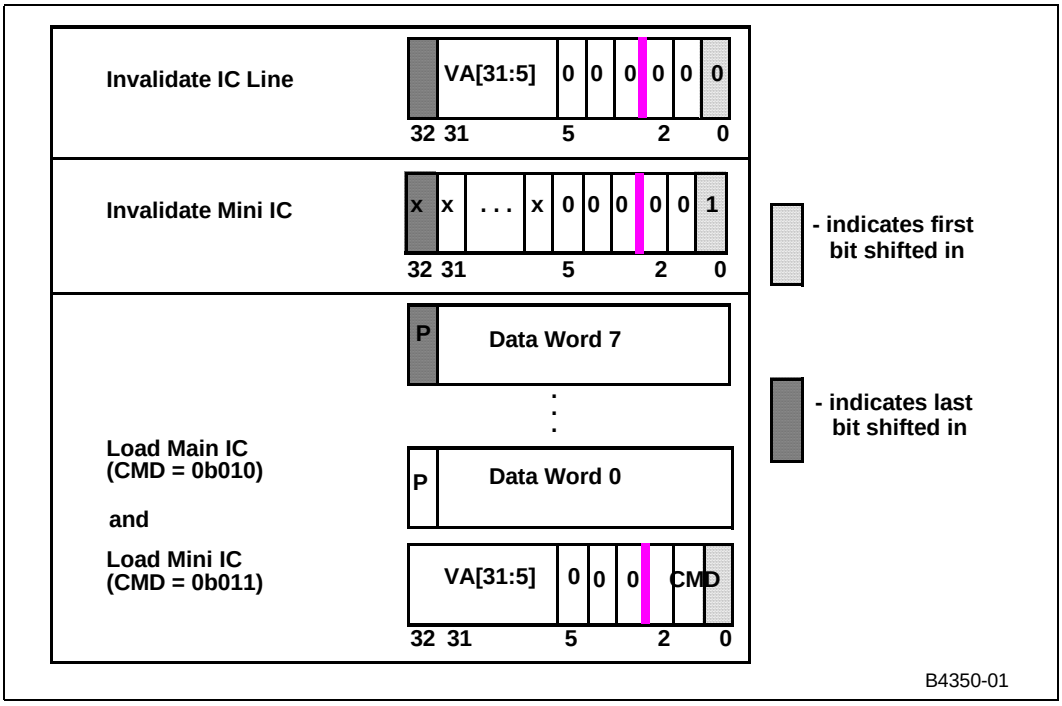
Invalidate Mini IC invalidates the entire mini instruction cache. It does not effect the main instruction cache. It does not require a virtual address or any data arguments.

Note: The LDIC Invalidate Mini IC function does not invalidate the BTB (like the CP15 Invalidate IC function) so software must do this manually where appropriate.

Load Main IC and Load Mini IC write one line of data (eight Intel StrongARM instructions) into the specified instruction cache at the specified virtual address.

Each cache function is downloaded through JTAG in 33 bit packets. Figure 25 shows the packet formats for each of the JTAG cache functions. Invalidate IC Line and Invalidate Mini IC each require 1 packet. Load Main IC and Load Mini IC each require 9 packets.

Figure 25. Format of LDIC Cache Functions



All packets are 33 bits in length. Bits [2:0] of the first packet specify the function to execute. For functions that require an address, bits[32:6] of the first packet specify an eight-word aligned address (Packet1[32:6] = VA[31:5]). For Load Main IC and Load Mini IC, eight additional data packets are used to specify eight Intel StrongARM instructions to be loaded into the target instruction cache. Bits[31:0] of the data packets contain the data to download. Bit[32] of each data packet is the value of the parity for the data in that packet.

As shown in Figure 25, the first bit shifted in TDI is bit 0 of the first packet. After each 33-bit packet, the host must take the JTAG state machine into the Update_DR state. After the host does an Update_DR and returns the JTAG state machine back to the Shift_DR state, the host can immediately begin shifting in the next 33-bit packet.

3.6.14.4 Loading IC During Reset

Code is downloaded into the instruction cache through JTAG during a processor reset. This feature is used during software debug to download the debug handler prior to starting an application program. The downloaded handler can then intercept the reset vector and do any necessary setup before the application code executes

In general, any code downloaded into the instruction cache through JTAG, is downloaded to addresses that are not already valid in the instruction cache. Failure to meet this requirement results in unpredictable behavior by the processor. During a processor reset, the instruction cache is typically invalidated, with the exception of the following modes:

- LDIC mode — Active when LDIC JTAG instruction is loaded in the JTAG IR; prevents the mini instruction cache and the main instruction cache from being invalidated during reset.

- HALT mode — Active when the Halt Mode bit is set in the DCSR; prevents only the mini instruction cache from being invalidated; main instruction cache is invalidated by reset.

During a cold reset (where both a processor reset and a JTAG reset occurs) it is guaranteed that the instruction cache is invalidated since the JTAG reset takes the processor out of any of the modes listed above.

During a warm reset, if a JTAG reset does not occur, the instruction cache is not invalidated by reset when any of the above modes are active. This situation requires special attention if code needs be downloaded during the warm reset.

Note: When Halt Mode is active, reset can invalidate the main instruction cache. Thus debug handler code downloaded during reset can only be loaded into the mini instruction cache. But, code is dynamically downloaded into the main instruction cache. (refer to "Dynamically Loading IC After Reset" on page 132).

The following sections describe the steps necessary to ensure code is correctly downloaded into the instruction cache.

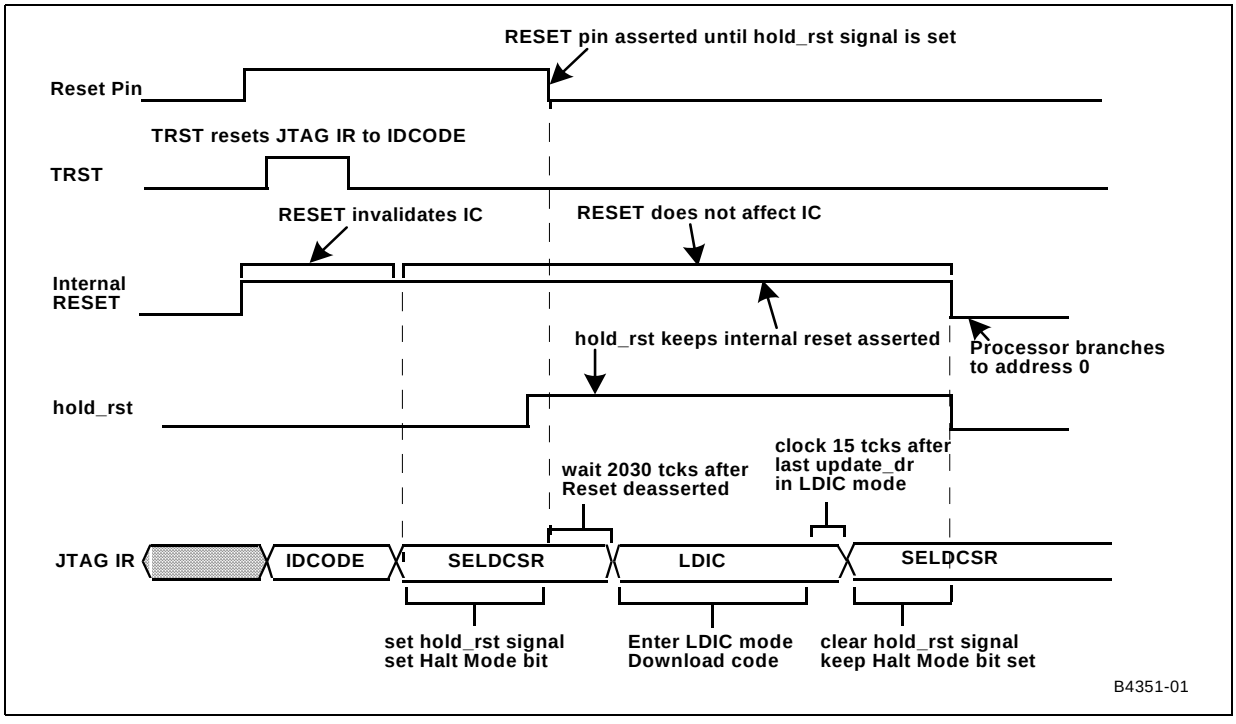
3.6.14.4.1 Loading IC During Cold Reset for Debug

The Figure 26 shows the actions necessary to download code into the instruction cache during a cold reset for debug.

Note: In the Figure 26 hold_rst is a signal that gets set and cleared through JTAG When the JTAG IR contains the SELDCSR instruction, the hold_rst signal is set to the value scanned into DBG_SR[1].

Note: In the Figure 26 TRST is an internal signal, it is the inverted value of the JTG_TRST_N pin.

Figure 26. Code Download During a Cold Reset For Debug



An external host should take the following steps to load code into the instruction cache following a cold reset:

1. Assert the Reset and JTG_TRST_N pins: This resets the JTAG IR to IDCODE and invalidates the instruction cache (main and mini).
2. Load the SELDCSR JTAG instruction into JTAG IR and scan in a value to set the Halt Mode bit in DCSR and to set the hold_rst signal. For details of the SELDCSR, refer to "SELDCSR JTAG Register" on page 113.
3. After hold_rst is set, de-assert the Reset pin. Internally the processor remains held in reset.
4. After Reset is de-asserted, wait 2030 TCKs.
5. Load the LDIC JTAG instruction into JTAG IR.
6. Download code into instruction cache in 33-bit packets as described in "LDIC Cache Functions" on page 127.
7. After code download is complete, clock a minimum of 15 TCKs following the last update_dr in LDIC mode.
8. Place the SELDCSR JTAG instruction into the JTAG IR and scan in a value to clear the hold_rst signal. The Halt Mode bit must remain set to prevent the instruction cache from being invalidated.
9. When hold_rst is cleared, internal reset is de-asserted, and the processor executes the reset vector at address 0.

An additional issue for debug is setting up the reset vector trap. This is done before the internal reset signal is de-asserted. As described in "Vector Trap Bits (TF,TI,TD,TA,TS,TU,TR)" on page 102, the Halt Mode and the Trap Reset bits in the DCSR is set prior to de-asserting reset to trap the reset vector. There are two possibilities for setting up the reset vector trap:

- The reset vector trap is set up before the instruction cache is loaded by scanning in a DCSR value that sets the Trap Reset bit in addition to the Halt Mode bit and the hold_rst signal; OR
- The reset vector trap is set up after the instruction cache is loaded. In this case, the DCSR should be set up to do a reset vector trap, with the Halt Mode bit and the hold_rst signal remaining set.

In either case, when the debugger clears the hold_rst bit to de-assert internal reset, the debugger must set the Halt Mode and Trap Reset bits in the DCSR.

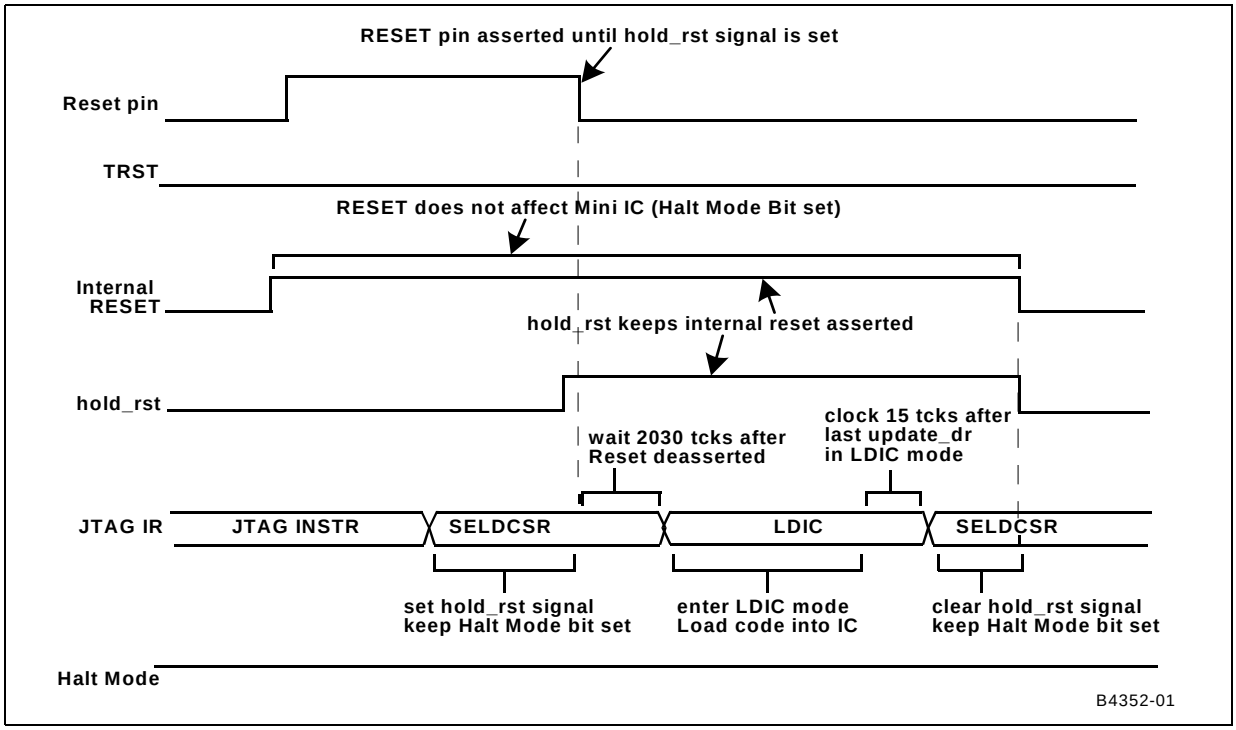
3.6.14.4.2 Loading IC During a Warm Reset for Debug

Loading the instruction cache during a warm reset is a slightly different situation than during a cold reset. For a warm reset, the main issue is whether the instruction cache gets invalidated by the processor reset or not. There are several possible scenarios:

- As reset is asserted, TRST is also asserted.
  In this case the instruction cache is invalidated, so the actions taken to download code are identical to those described in "Loading IC During Cold Reset for Debug" on page 129
- When reset is asserted, TRST is not asserted, but the processor is not in Halt Mode.
  In this case, the instruction cache is also invalidated, so the actions are the same as described in "Loading IC During Cold Reset for Debug" on page 129, after the LDIC instruction is loaded into the JTAG IR.
- When reset is asserted, TRST is not asserted, and the processor is in Halt Mode.
  In this last scenario, the mini instruction cache does not get invalidated by reset, since the processor is in Halt Mode. This scenario is described in more detail in this section.

In the last scenario described above is shown in Figure 28.

Figure 27. Code Download During a Warm Reset For Debug



As shown in Figure 27, reset does not invalidate the instruction cache because of the processor is in Halt Mode. Since the instruction cache is not invalidated, it may contain valid lines. The host must avoid downloading code to virtual addresses that are already valid in the instruction cache (mini IC or main IC), otherwise the processor may behave unpredictably.

There are several possible solutions that ensure code is not downloaded to a VA that already exists in the instruction cache.

Since the mini instruction cache is not invalidated, any code previously downloaded into the mini IC is valid in the mini IC, so it is not necessary to download the same code again.

If it is necessary to download code into the instruction cache:

1. Assert TRST.
   This clears the Halt Mode bit allowing the instruction cache to be invalidated.
2. Clear the Halt Mode bit through JTAG.
   This allows the instruction cache to be invalidated by reset.
3. Place the LDIC JTAG instruction in the JTAG IR, then proceed with the normal code download, using the Invalidate IC Line function before loading each line.
   This requires 10 packets to be downloaded per cache line instead of the 9 packets described in “LDIC Cache Functions” on page 127

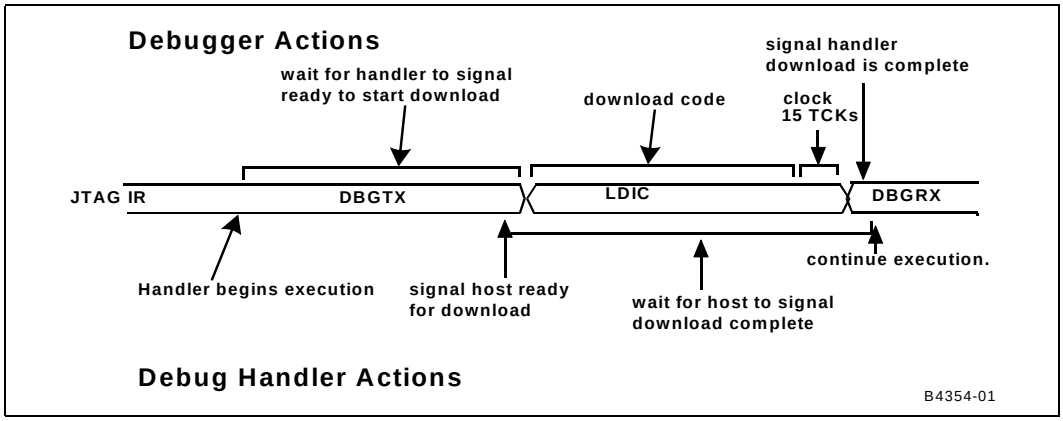
3.6.14.5 Dynamically Loading IC After Reset

An external host can load code into the instruction cache **on the fly** or **dynamically**. This occurs when the host downloads code and the processor is not being reset. This requires strict synchronization between the code running on the IXP43X network processors and the external host. The guidelines for downloading code during program execution must be followed to ensure proper operation of the processor. The description in this section focuses on using a debug handler running on the IXP43X network processors to synchronize with the external host, but the details apply for any application that is running as code is dynamically downloaded.

To dynamically download code during software debug, there is a minimal debug handler stub, responsible for doing the handshaking with the host, resident in the instruction cache. This debug handler stub should be downloaded into the instruction cache during processor reset using the method described in "Loading IC During Reset" on page 128. "Dynamic Code Download Synchronization" on page 133 describes the details for implementing the handshaking in the debug handler.

Figure 28 shows a high level view of the actions taken by the host and debug handler during dynamic code download.

Figure 28. Downloading Code in IC During Program Execution



The following steps describe the details for downloading code:

- Since the debug handler is responsible for synchronization during the code download, the handler is executing before the host can begin the download. The debug handler execution starts when the application running on the IXP43X network processors generate a debug exception or when the host generates an external debug break.
- As the DBGTX JTAG instruction is in the JTAG IR (see "DBGTX JTAG Command" on page 115), the host polls DBG_SR[0], waiting for the debug handler to set it.
- When the debug handler gets to the point where it is OK to begin the code download, it writes to TX, and that automatically sets DBG_SR[0]. This signals the host it is OK to begin the download. The debug handler then begins polling TXRXCTRL[31] waiting for the host to clear it through the DBGRX JTAG register (to indicate the download is complete).
- The host writes LDIC to the JTAG IR, and downloads the code. For each line downloaded, the host must invalidate the target line before downloading code to that line. Failure to invalidate a line prior to writing it may cause unpredictable operation by the processor.
- When the host completes its download, the host must wait a minimum of 15 TCKs, then switch the JTAG IR to DBGRX, and complete the handshaking (by scanning in

a value that sets DBG_SR[35]). This clears TXRXCTL[31] and allows the debug handler code to exit the polling loop. The data scanned into DBG_SR[34:3] is implementation specific.

- After the handler exits the polling loop, it branches to the downloaded code.

Note: This debug handler stub must reside in the instruction cache and execute out of the cache during synchronization. The processor should not be doing any code fetches to external memory when code is being downloaded.

3.6.14.5.1 Dynamic Code Download Synchronization

The following pieces of code are necessary in the debug handler to implement the synchronization used during dynamic code download. The pieces are ordered in the handler as shown below.

Table 52. Debug-Handler Code to Implement Synchronization During Dynamic Code Download

```
# Before the download can start, all outstanding instruction fetches must complete.
# The MCR invalidate IC by line function serves as a barrier instruction in
# the core. All outstanding instruction fetches are guaranteed to complete before
# the next instruction executes.

# NOTE1: the actual address specified to invalidate is implementation defined, but
# must not have any harmful effects.

# NOTE2: The placement of the invalidate code is implementation defined, the only
# requirement is that it is placed such that by the time the debugger starts
# loading the instruction cache, all outstanding instruction fetches have completed

    mov r5, address

    mcr p15, 0, r5, c7, c5, 1

# The host waits for the debug handler to signal that it is ready for the
# code download. This is done using the TX register access handshaking
# protocol. The host polls the TR bit through JTAG until it is set, then begins
# the code download. The following MCR does a write to TX, automatically
# setting the TR bit.

# NOTE: The value written to TX is implementation defined.

    mcr p14, 0, r6, c8, c0, 0

# The debug handler waits until the download is complete before continuing. The
# debugger uses the RX handshaking to signal the debug handler when the download
# is complete. The debug handler polls the RR bit until it is set. A debugger write
# to RX automatically sets the RR bit, allowing the handler to proceed.

# NOTE: The value written to RX by the debugger is implementation defined - it is a bogus value
signalling the handler to continue or it is a target address for the handler to branch to.

loop:

    mrc     p14, 0, r15, c14, c0, 0       @ handler waits for signal from debugger

    bpl     loop

    mrc     p14, 0, r0, c8, c0, 0         @ debugger writes target address to RX

    bx      r0
```

In a very simple debug handler stub, the above parts may form the complete handler downloaded during reset (with some handler entry and exit code). When a debug exception occurs, routines are downloaded as necessary. This basically allows the entire handler to be dynamic.

Another possibility is for a more complete debug handler is downloaded during reset. The debug handler may support some operations, such as read memory, write memory, and so on. But, other operations, such as reading or writing a group of CP register, is downloaded dynamically. This method could be used to dynamically download infrequently used debug handler functions, as the more common operations remain static in the mini-instruction cache.

The Intel Debug Handler is a complete debug handler that implements the more commonly used functions, and allows less frequently used functions to be dynamically downloaded.

3.6.14.6 Mini-Instruction Cache Overview

The mini instruction cache is a smaller version of the main instruction cache. (For more details on the main instruction cache, see “Instruction Cache” on page 65.) It is a 2-KByte, two-way set associative cache. There are 32 sets, each containing two ways with each way containing eight words. The cache uses the round-robin replacement policy.

The mini instruction cache is virtually addressed and addresses are remapped by the PID. But, since the debug handler executes in special debug state, address translation and PID remapping are turned off. For application code, accesses to the mini instruction cache use the normal address translation and PID mechanisms.

Normal application code is never cached in the mini instruction cache on an instruction fetch. The only way to get code into the mini instruction cache is through the JTAG LDIC function. Code downloaded into the mini instruction cache is essentially locked; it cannot be overwritten by application code running on the IXP43X network processors. It is not locked against code downloaded through the JTAG LDIC functions.

Application code can invalidate a line in the mini instruction cache using a CP15 Invalidate IC line function to an address that hits in the mini instruction cache. But, a CP15 global invalidate IC function does not affect the mini instruction cache.

The mini instruction cache is globally invalidated through JTAG by the LDIC Invalidate IC function or by a processor reset when the processor is not in HALT or LDIC mode. A single line in the mini instruction cache is invalidated through JTAG by the LDIC Invalidate IC-line function.

3.6.15 Halt Mode Software Protocol

This section describes the overall debug process in Halt Mode. It describes how to start and end a debug session and details for implementing a debug handler. Intel provides a standard Debug Handler that implements some of the techniques in this section. The Intel Debug Handler itself is a a document describing additional handler implementation techniques and requirements.

3.6.15.1 Starting a Debug Session

Prior to starting a debug session in Halt Mode, the debugger must download code into the instruction cache during reset, via JTAG. (“Downloading Code in ICache” on page 125). This downloaded code should consist of:

- A debug handler
- An override default vector table

- An override relocated vector table (if necessary)

When the processor is still in reset, the debugger should set up the DCSR to trap the reset vector. This causes a debug exception to occur immediately when the processor comes out of reset. Execution is redirected to the debug handler allowing the debugger to perform any necessary initialization. The reset vector trap is the only debug exception that can occur with debug globally disabled (DCSR[31]=0). Therefore, the debugger must also enable debug prior to existing the handler to ensure all subsequent debug exceptions correctly break to the debug handler.

3.6.15.1.1 Setting up Override Vector Tables

The override default vector table intercepts the reset vector and branches to the debug handler when a debug exception occurs. If the vector table is relocated, the debug vector is relocated to address 0xffff0000. Thus, an override relocated vector table is required to intercept vector 0xffff0000 and branch to the debug handler.

Both override vector tables also intercept the other debug exceptions, so they are set up to branch to a debugger specific handler or go to the application's handlers.

It is possible that the application modifies its vector table in memory, so the debugger may not be able to set up the override vector table to branch to the application's handlers. The Debug Handler is used to work around this problem by reading memory and branching to the appropriate address. Vector traps are used to get to the debug handler, or the override vector tables can redirect execution to a debug handler routine that examines memory and branches to the application's handler.

3.6.15.1.2 Placing the Handler in Memory

The debug handler is not required to be placed at a specific pre-defined address. But, there are some limitations on where the handler is placed due to the override vector tables and the two-way set associative mini instruction cache.

In the override vector table, the reset vector must branch to the debug handler using:

- A direct branch, that limits the start of the handler code to within 32 Mbytes of the reset vector, or
- An indirect branch with a data processing instruction. The data processing instruction creates an address using immediate operands and then branches to the target. An LDR to the PC does not work because the debugger cannot set up data in memory before starting the debug handler

The two-way set associative limitation is due to the fact that when the override default and relocated vector tables are downloaded, they take up both ways of Set 0 (w/ addresses 0x0 and 0xffff0000). Therefore, debug handler code cannot be downloaded to an address that maps into Set 0, otherwise it overwrites one of the vector tables (avoid addresses w/ lower 12 bits=0).

The instruction cache two-way set limitation is not a problem when the reset vector uses a direct branch, since the branch offset is adjusted accordingly. But, it makes using indirect branches more complicated. Now, the reset vector actually needs multiple data processing instructions to create the target address and branch to it.

One possibility is to set up vector traps on the non-reset exception vectors. These vector locations can then be used to extend the reset vector.

Another solution is to have the reset vector do a direct branch to some intermediate code. This intermediate code can then uses several instructions to create the debug handler start address and branch to it. This would require another line in the mini

instruction cache, since the intermediate code must also be downloaded. This method also requires that the layout of the debug handler be well thought out to avoid the intermediate code overwriting a line of debug handler code, or vice versa.

For the indirect branch cases, a temporary scratch register is necessary to hold intermediate values when computing the final target address. DBG_r13 is used for this purpose (see "Debug Handler Restrictions" on page 136 for restrictions on DBG_r13 usage).

3.6.15.2 Implementing a Debug Handler

The debugger uses the debug handler to examine or modify processor state by sending commands and reading data through JTAG. The API between the debugger and debug handler is specific to a debugger implementation. Intel provides a standard debug handler and API that is used by third-party vendors. Issues and details for writing a debug handler are discussed in this section and in the Intel Debug Handler.

3.6.15.2.1 Debug Handler Entry

When the debugger requests an external debug break or is waiting for an internal break, it should poll the TR bit through JTAG to determine when the processor has entered Debug Mode. The debug handler entry code must do a write to TX to signal the debugger that the processor has entered Debug Mode. The write to TX sets the TR bit, signalling the host that a debug exception has occurred and the processor has entered Debug Mode. The value of the data written to TX is implementation defined (debug break message, contents of register to save on host, and so on.).

3.6.15.2.2 Debug Handler Restrictions

The Debug Handler executes in debug mode and is similar to other privileged processor modes, but, there are some differences. Following are restrictions on Debug Handler code and differences between debug mode and other privileged modes.

- The processor is in special debug state following a debug exception, and thus has special functionality as described in "Halt Mode" on page 103.
- Although address translation and PID remapping are disabled for instruction accesses (as defined in special debug state), data accesses use the normal address translation and PID remapping mechanisms.
- Debug mode does not have a dedicated stack pointer, DBG_r13. Although DBG_r13 exists, it is not a general purpose register. Its contents are unpredictable and should not be relied upon across any instructions or exceptions. But, DBG_r13 is used, by data processing (non RRX) and MCR/MRC instructions, as a temporary scratch register.
- The following instructions should not be executed in debug mode, they may result in unpredictable behavior:
  - LDM
  - LDR w/ Rd=PC
  - LDR w/ RRX addressing mode
  - SWP
  - LDC
  - STC
- The handler executes in debug mode and is switched to other modes to access banked registers. The handler must not enter User Mode; any User Mode registers that must be accessed are accessed in System Mode. Entering User Mode may cause unpredictable behavior.

3.6.15.2.3 Dynamic Debug Handler

On the IXP43X network processors, the debug handler and override vector tables reside in the 2-KByte, mini instruction cache, separate from the main instruction cache. A **static** Debug Handler is downloaded during reset. This is the base handler code, necessary to do common operations such as handler entry/exit, parse commands from the debugger, read/write Intel StrongARM registers, read/write memory, and so on.

Some functions may require large amounts of code or may not be used very often. As long as there is space in the mini-instruction cache, these functions are downloaded as part of the static Debug Handler. But, if space is limited, the debug handler also has a dynamic capability that allows a function to be downloaded when it is needed. There are three methods for implementing a dynamic debug handler (using the mini instruction cache, main instruction cache, or external memory). Each method has their limitations and advantages. "Dynamically Loading IC After Reset" on page 132 describes how do dynamically load the mini or main instruction cache.

- Using the Mini IC

  The static debug handler can support a command that can have functionality dynamically mapped to it. This dynamic command does not have any specific functionality associated with it until the debugger downloads a function into the mini instruction cache. When the debugger sends the dynamic command to the handler, new functionality is downloaded, or the previously downloaded functionality is used.

  There are also variations where the debug handler supports multiple dynamic commands, each mapped to another dynamic function; or a single dynamic command that can branch to one of several downloaded dynamic functions based on a parameter passed by the debugger.

  Debug Handlers that allow code to be dynamically downloaded into the mini instruction cache must be carefully written to avoid inadvertently overwriting a critical piece of debug handler code. Dynamic code is downloaded to the way pointed to by the round-robin pointer. Thus, it is possible for critical debug handler code to be overwritten, if the pointer does not select the expected way.

  To avoid this problem, the debug handler should be written to avoid placing critical code in either way of a set that is intended for dynamic code download. This allows code to be downloaded into either way, and the only code that is overwritten is the previously downloaded dynamic function. This method requires that space within the mini instruction cache be allocated for dynamic download, limiting the space available for the static Debug Handler. Also, the space available may not be suitable for a larger dynamic function.

  Once downloaded, a dynamic function essentially becomes part of the Debug Handler. Since it is in the mini instruction cache, it does not get overwritten by application code. It remains in the cache until it is replaced by another dynamic function or the lines where it is downloaded are invalidated.

- Using the Main IC.

  The steps for downloading dynamic functions into the main instruction cache is similar to downloading into the mini instruction cache. But, using the main instruction cache has its advantages.

  Using the main instruction cache eliminates the problem of inadvertently overwriting static Debug Handler code by writing to the wrong way of a set, since the main and mini instruction caches are separate. The debug handler code need not be specially mapped out to avoid this problem. Also, space for dynamic functions need not be allocated in the mini instruction cache and dynamic functions are not limited to the size allocated.

  The dynamic function can actually be downloaded anywhere in the address space. The debugger specifies the location of the dynamic function by writing the address to RX when it signals to the handler to continue. The debug handler then does a branch-and-link to that address.

If the dynamic function is already downloaded in the main instruction cache, the debugger immediately downloads the address, signalling the handler to continue.

The static Debug Handler must support only one dynamic function command. Multiple dynamic functions are downloaded to various addresses and the debugger uses the function's address to specify the dynamic function that has to execute.

Since the dynamic function is being downloaded into the main instruction cache, the downloaded code may overwrite valid application code, and conversely, application code may overwrite the dynamic function. The dynamic function is only guaranteed to be in the cache from the time it is downloaded to the time the debug handler returns to the application (or the debugger overwrites it).

- External memory

  Dynamic functions can also we downloaded to external memory (or they may already exist there). The debugger can download to external memory using the write-memory commands. Then the debugger executes the dynamic command using the address of the function to identify the function to execute. This method has the many of the same advantages as downloading into the main instruction cache.

  Depending on the memory system, this method could be much slower than downloading directly into the instruction cache. Another problem is the application may write to the memory where the function is downloaded. If it is guaranteed that the application does not modify the downloaded dynamic function, the debug handler can save the time it takes to re-download the code. Otherwise, the ensure the application does not corrupt the dynamic functions, the debugger should re-download any dynamic functions it uses.

For all three methods, the downloaded code executes in the context of the debug handler. The processor is in special debug state, so all of the special functionality applies.

The downloaded functions may also require some common routines from the static debug handler, such as the polling routines for reading RX or writing TX. To simplify the dynamic functions, the debug handler should define a set of registers to contain the addresses of the most commonly used routines. The dynamic functions can then access these routines using indirect branches (BLX). This helps reduce the amount of code in the dynamic function since common routines need not be replicated within each dynamic function.

3.6.15.2.4 High-Speed Download

Special debug hardware has been added to support a high-speed download mode to increase the performance of downloads to system memory (vs. writing a block of memory using the standard handshaking).

The basic assumption is that the debug handler can read any data sent by the debugger and write it to memory, before the debugger can send the next data. Thus, in the time it takes for the debugger to scan in the next data word and do an Update_DR, the handler is already in its polling loop, waiting for it. Using this assumption, the debugger does not have to poll RR to see whether the handler has read the previous data - it assumes the previous data has been consumed and immediately starts scanning in the next data word.

The pitfall is when the write to memory stalls long enough that the assumption fails. In this case the download with normal handshaking is used (or high-speed download can still be used, but a few extra TCKs in the Pause_DR state is necessary to allow a little more time for the store to complete).

The hardware support for high-speed download includes the Download bit (DCSR[29]) and the Overflow Flag (DCSR[30]).

The download bit acts as a branch flag, signalling to the handler to continue with the download. This removes the need for a counter in the debug handler.

The overflow flag indicates that the debugger attempted to download the next word before the debugger read the previous word.

More details on the Download bit, Overflow flag and high-speed download, in general, is found in "Transmit/Receive Control Register" on page 108.

Following is example code showing how the Download bit and Overflow flag are used in the debug handler:

Table 53. Debug Handler Code: Download Bit and Overflow Flag

```
hs_write_word_loop:

hs_write_overflow:

    bl      read_RX                @ read data word from host


    @@ read TXRXCTRL into the CCs

    mrc     p14, 0, r15, c14, c0, 0

    bcc     hs_write_done      @ if D bit clear, download complete, exit loop.

    beq     hs_write_overflow  @ if overflow detected, loop until host clears D bit


    str     r0, [r6], #4       @ store only if there is no overflow.


    b       hs_write_word_loop     @ get next data word


hs_write_done:

    @@ after the loop, if the overflow flag was set, return error message to host

    moveq   r0, #OVERFLOW_RESPONSE

    beq     send_response

    b       write_common_exit
```

3.6.15.3 Ending a Debug Session

Prior to ending a debug session, the debugger should take the following actions:

1. Clear the DCSR (disable debug, exit Halt Mode, clear all vector traps, disable the trace buffer)
2. Turn off all breakpoints.
3. Invalidate the mini instruction cache.
4. Invalidate the main instruction cache.
5. Invalidate the BTB.

These actions ensure that the application program executes correctly after the debugger has been disconnected.

3.6.16 Software Debug Notes and Errata

- Trace buffer message count value on data aborts:
  LDR to non-PC that aborts gets counted in the exception message. But an LDR to the PC that aborts does not get counted on exception message.
- SW Note on data abort generation in special debug state.
  - Avoid code that could generate exact data aborts.
  - If this cannot be done, then handler must be written such that a memory access is followed by 1 nops. In this case, certain memory operations must be avoided - LDM, STM, STRD, LDC, SWP.
- Data abort on special debug state:
  When write-back is on for a memory access that causes a data abort, the base register is updated with the write-back value. This is inconsistent with normal (non-SDS) behavior where the base remains unchanged if write-back is on and a data abort occurs.
- Trace Buffer wraps around and loses data in Halt Mode when configured for fill-once mode:
  It is possible to overflow (and lose) data from the trace buffer in fill-once mode, in Halt Mode. When the trace buffer fills up, it has space for 1 indirect branch message (5 bytes) and 1 exception message (1 Byte).
  If the trace buffer fills up with an indirect branch message and generates a trace buffer full break at the same time as a data abort occurs, the data abort has higher priority, so the processor first goes to the data abort handler. This data abort is placed into the trace buffer without losing any data.
  But, if another imprecise data abort is detected at the start of the data abort handler, it has higher priority than the trace buffer full break, so the processor goes back to the data abort handler. This 2nd data abort also gets written into the trace buffer. This causes the trace buffer to wrap-around and one trace buffer entry is lost (oldest entry is lost). Additional trace buffer entries is lost if imprecise data aborts continue to be detected before the processor can handle the trace buffer full break (that turns off the trace buffer).
  This trace buffer overflow problem is avoided by enabling vector traps on data aborts.
- TXRXCTRL.RR prevents TX register from being updated (even if TXRXCTRL.TR is clear). This is fixed on B-step.
  The problem is that there is incorrect and unnecessary interaction between the RX ready (RR) flag and writing the TX register. The debug handler looks at the TX ready bit before writing to the TX register. If this bit is clear, then the handler should be able to write to the TX register. In the current implementation even if the TR bit is clear, if the RR bit is set, TX is unchanged when the handler writes to it. It is OK to prevent a write to TX when the TR bit is set (since the host has not read the previous data in the TX, and we don't want a write to TX to overwrite previous data).
- The TXRXCTRL.OV bit (overflow flag) does not get set during high-speed download when the handler reads the RX register at the same time the debugger writes to it.
  If the debugger writes to RX at the same time the handler reads from RX, the handler read returns the newly written data and the previous data is lost. In this specific case, the overflow flag does not get set, so the debugger is unaware that the download is not successful.

3.7 Performance Monitoring

This section describes the performance monitoring facility of the IXP43X network processors. The events that are monitored can provide performance information for compiler writers, system application developers and software programmers.

3.7.1 Overview

The hardware for the IXP43X network processors provides four 32-bit performance counters that allow four unique events to be monitored simultaneously. In addition, the IXP43X network processors implement a 32-bit clock counter that is used in conjunction with the performance counters; its main purpose is to count the number of core clock cycles and is useful in measuring total execution time.

The IXP43X network processors can monitor occurrence events or duration events. When counting occurrence events, a counter is incremented each time a specified event takes place and when measuring duration, a counter counts the number of processor clocks that occur as a specified condition is true. If any of the five counters overflow, an interrupt request occurs if it is enabled. Each counter has its own interrupt request enable. The counters continue to monitor events even after an overflow occurs, until disabled by software.

Each of these counters are programmed to monitor any one of various events.

To further augment performance monitoring, the clock counter of the IXP43X network processors is used to measure the executing time of an application. This information combined with a duration event can feedback a percentage of time the event occurred with respect to overall execution time.

All of the performance monitoring registers are accessible through Coprocessor 14 (CP14). Refer to Table 28 for more details on accessing these registers with **MRC** and **MCR** coprocessor instructions. Access is allowed in privileged mode only.

Note: These registers can't be access with **LDC** or **STC** coprocessor instructions.

Table 54. Performance Monitoring Registers

Description	CRn Register #	CRm Register #	Instruction
(PMNC) Performance Monitor Control Register	0b0000	0b0001	Read: MRC p14, 0, Rd, c0, c1, 0 Write: MCR p14, 0, Rd, c0, c1, 0
(CCNT) Clock Counter Register	0b0001	0b0001	Read: MRC p14, 0, Rd, c1, c1, 0 Write: MCR p14, 0, Rd, c1, c1, 0
(INTEN) Interrupt Enable Register	0b0100	0b0001	Read: MRC p14, 0, Rd, c4, c1, 0 Write: MCR p14, 0, Rd, c4, c1, 0
(FLAG) Overflow Flag Register	0b0101	0b0001	Read: MRC p14, 0, Rd, c5, c1, 0 Write: MCR p14, 0, Rd, c5, c1, 0
(EVTSEL) Event Selection Register	0b1000	0b0001	Read: MRC p14, 0, Rd, c8, c1, 0 Write: MCR p14, 0, Rd, c8, c1, 0
(PMN0) Performance Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performance Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performance Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performance Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

3.7.2 Register Description

3.7.2.1 Clock Counter (CCNT)

The format of CCNT is shown in Table 55. The clock counter is reset to '0' by setting bit 2 in the Performance Monitor Control Register (PMNC) or is set to a predetermined value by directly writing to it. It counts core clock cycles. When CCNT reaches its maximum value 0xFFFF,FFFF, the next clock cycle causes it to roll over to zero and set the overflow flag (bit 0) in FLAG. An interrupt request occurs if it is enabled via bit 0 in INTEN.

Table 55. Clock Count Register (CCNT)

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
Clock Counter																															

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**32-bit clock counter** - Reset to '0' by PMNC register. When the clock counter reaches its maximum value 0xFFFF,FFFF, the next cycle causes it to roll over to zero and generate an interrupt request if enabled.

3.7.2.2 Performance Count Registers

There are four 32-bit event counters; their format is shown in Table 56. The event counters are reset to '0' by setting bit 1 in the PMNC register or is set to a predetermined value by directly writing to them. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it should count causes it to roll over to zero and set its corresponding overflow flag (bit 1,2,3 or 4) in FLAG. An interrupt request is generated if its corresponding interrupt enable (bit 1,2,3 or 4) is set in INTEN.

Table 56. Performance Monitor Count Register (PMN0 - PMN3)

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
Event Counter																															

reset value: unpredictable

Bits	Access	Description
31:0	Read / Write	**32-bit event counter** - Reset to '0' by PMNC register. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it should count causes it to roll over to zero and generate an interrupt request if enabled.

3.7.2.3 Performance Monitor Control Register

The performance monitor control register (PMNC) is a coprocessor register that:

- Contains the PMU ID
- Extends CCNT counting by six more bits (cycles between counter rollover = 2^{38})
- Resets all counters to zero
- And enables the entire mechanism

Table 57 shows the format of the PMNC register.

Table 57. Performance Monitor Control Register

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
ID																												D	C	P	E

reset value: E **and** ID **are 0, others unpredictable**

Bits	Access	Description
31:24	Read / Write Ignored	Performance Monitor Identification (ID) - IXP43X network processors = 0x14
23:4	Read-unpredictable / Write-as-0	Reserved
3	Read / Write	**Clock Counter Divider (D)** - 0 = CCNT counts every processor clock cycle 1 = CCNT counts every 64th processor clock cycle
2	Read-unpredictable / Write	**Clock Counter Reset (C)** - 0 = no action 1 = reset the clock counter to '0x0'
1	Read-unpredictable / Write	**Performance Counter Reset (P)** - 0 = no action 1 = reset all performance counters to '0x0'
0	Read / Write	**Enable (E)** - 0 = all counters are disabled 1 = all counters are enabled

3.7.2.4 Interrupt Enable Register

Each counter can generate an interrupt request when it overflows. INTEN enables interrupt requesting for each counter.

Table 58. Interrupt Enable Register

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
																											P3	P2	P1	P0	C

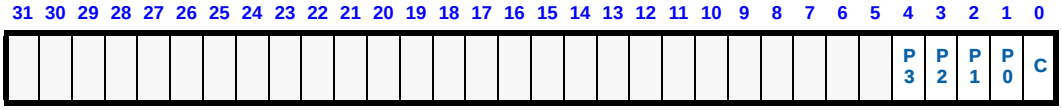
reset value: [4:0] = 0b00000**, others unpredictable**

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	**PMN3 Interrupt Enable (P3)** - 0 = disable interrupt 1 = enable interrupt
3	Read / Write	**PMN2 Interrupt Enable (P2)** - 0 = disable interrupt 1 = enable interrupt
2	Read / Write	**PMN1 Interrupt Enable (P1)** - 0 = disable interrupt 1 = enable interrupt
1	Read / Write	**PMN0 Interrupt Enable (P0)** - 0 = disable interrupt 1 = enable interrupt
0	Read / Write	**CCNT Interrupt Enable (C)** - 0 = disable interrupt 1 = enable interrupt

3.7.2.5 Overflow Flag Status Register

FLAG identifies the counter that has overflowed and also indicates an interrupt has been requested if the overflowing counter's corresponding interrupt enable bit (contained within INTEN) is asserted. An overflow is cleared by writing a '1' to the overflow bit.

Table 59. Overflow Flag Status Register



reset value: [4:0] = 0b00000, **others unpredictable**

Bits	Access	Description
31:5	Read-unpredictable / Write-as-0	Reserved
4	Read / Write	**PMN3 Overflow Flag (P3)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
3	Read / Write	**PMN2 Overflow Flag (P2)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
2	Read / Write	**PMN1 Overflow Flag (P1)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
1	Read / Write	**PMN0 Overflow Flag (P0)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
0	Read / Write	**CCNT Overflow Flag (C)** - Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit

3.7.2.6 Event Select Register

EVTSEL is used to select events for PMN0, PMN1, PMN2 and PMN3. Refer to Table 61, "Performance Monitoring Events" on page 146 for a list of possible events.

Table 60. Event Select Register

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
evtCount3	evtCount2	evtCount1	evtCount0

reset value: unpredictable

Bits	Access	Description
31:24	Read / Write	**Event Count 3 (evtCount3)** - Identifies the source of events that PMN3 counts. See Table 61 for a description of the values this field may contain.
23:16	Read / Write	**Event Count 2 (evtCount2)** - Identifies the source of events that PMN2 counts. See Table 61 for a description of the values this field may contain.
15:8	Read / Write	**Event Count 1 (evtCount1)** - Identifies the source of events that PMN1 counts. See Table 61 for a description of the values this field may contain.
7:0	Read / Write	**Event Count 0 (evtCount0)** - Identifies the source of events that PMN0 counts. See Table 61 for a description of the values this field may contain.

3.7.3 Managing the Performance Monitor

The following are a few notes about controlling the performance monitoring mechanism:

- An interrupt request is generated when a counter's overflow flag is set and its associated interrupt enable bit is set in INTEN. The interrupt request remains asserted until software clears the overflow flag by writing a one to the flag that is set.

Note: The product specific interrupt unit and the CPSR must have enabled the interrupt for software to receive it.

- The interrupt request can also be deasserted by clearing the corresponding interrupt enable bit. Disabling the facility (PMNC.E) doesn't deassert the interrupt request.
- The counters continue to record events even after they overflow.
- To change an event for a performance counter, first disable the facility (PMNC.E) and then modify EVTSEL. Not doing so causes unpredictable results.
- To increase the monitoring duration, software can extend the count duration beyond 32 bits by counting the number of overflow interrupts each 32-bit counter generates. This is done in the interrupt service routine (ISR) where an increment to some memory location every time the interrupt occurs, enables longer durations of performance monitoring. This does intrude upon program execution but is negligible, since the ISR execution time is in the order of tens of cycles compared to the number of cycles it took to generate an overflow interrupt (2^{32}).
- Power is saved by selecting event 0xFF for any unused event counter. This only applies when other event counters are in use. When the performance monitor is not used at all (PMNC.E = 0x0), hardware ensures minimal power consumption.

3.7.4 Performance Monitoring Events

Table 61 lists events that are monitored. Each of the performance monitor count registers (PMN0, PMN1, PMN2, and PMN3) can count any listed event. Software selects the event that is counted by each PMNx register by programming the evtCountx fields of EVTSEL.

Table 61. Performance Monitoring Events

Event Number (evtCountx)	Event Definition
0x0	Instruction cache miss requires fetch from external memory.
0x1	Instruction cache cannot deliver an instruction. This could indicate an ICache miss or an ITLB miss. This event occurs every cycle where the condition is present.
0x2	Stall due to a data dependency. This event occurs every cycle where the condition is present.
0x3	Instruction TLB miss.
0x4	Data TLB miss.
0x5	Branch instruction executed, branch may or may not have changed program flow. (Counts only B and BL instructions, in both Intel StrongARM and Thumb mode)
0x6	Branch incorrectly predicted. (Counts only B and BL instructions, in both Intel StrongARM and Thumb mode.)
0x7	Instruction executed.
0x8	Stall because the data cache buffers are full. This event occurs every cycle where the condition is present.
0x9	Stall because the data cache buffers are full. This event occurs once for each contiguous sequence of this type of stall.
0xA	Data cache access, not including Cache Operations (defined in "Register 7: Cache Functions" on page 91)
0xB	Data cache miss, not including Cache Operations (defined in "Register 7: Cache Functions" on page 91)
0xC	Data cache write-back. This event occurs once for each 1/2 line (four words) that are written back from the cache.
0xD	Software changed the PC. All 'b', 'bl', 'blx', 'mov[s] pc, Rm', 'ldm Rn, {Rx, pc}', 'ldr pc, [Rm]', pop {pc} is counted. An 'mcr p<cp>, 0,pc, ...', is not. The count also does not increment when an event occurs and the PC changes to the event address, for example, IRQ, FIQ, SWI, and so on.
0x10 through 0x17	Reserved.
all others	Reserved, unpredictable results

Some of the typical combinations of counted events are listed in this section and summarized in Table 62. In this section, such an event is referred as a combination mode.

Table 62. Common Uses of the PMU

Mode	EVTSEL.evtCount0	EVTSEL.evtCount1
Instruction Cache Efficiency	0x7 (instruction count)	0x0 (ICache miss)
Data Cache Efficiency	0xA (Dcache access)	0xB (DCache miss)
Instruction Fetch Latency	0x1 (ICache cannot deliver)	0x0 (ICache miss)
Data/Bus Request Buffer Full	0x8 (DBuffer stall duration)	0x9 (DBuffer stall)
Stall/Writeback Statistics	0x2 (data stall)	0xC (DCache writeback)
Instruction TLB Efficiency	0x7 (instruction count)	0x3 (ITLB miss)
Data TLB Efficiency	0xA (Dcache access)	0x4 (DTLB miss)

Note: PMN0 and PMN1 were used for illustration purposes only. Given there are four event counters, more elaborate combination of events could be constructed. For example,

one performance run could select 0xA, 0xB, 0xC, 0x9 events from the data cache performance statistics that could be gathered (like hit rates, number of write-backs per data cache miss, and number of times the data cache buffers fill up per request).

3.7.4.1 Instruction Cache Efficiency Mode

PMN0 totals the number of instructions that were executed, and does not include instructions fetched from the instruction cache that were never executed. This can happen if a branch instruction changes the program flow; the instruction cache may retrieve the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time.

Statistics derived from these two events:

- Instruction cache miss-rate. This is derived by dividing PMN1 by PMN0.
- The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI). CPI is derived by dividing CCNT by PMN0, where CCNT is used to measure total execution time.

3.7.4.2 Data Cache Efficiency Mode

PMN0 totals the number of data cache accesses, and includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note: **STM** and **LDM** each counts as several accesses to the data cache depending on the number of registers specified in the register list. **LDRD** registers two accesses.

PMN1 counts the number of data cache and mini-data cache misses. Cache operations do not contribute to this count. See "Register 7: Cache Functions" on page 91 for a description of these operations.

The statistic derived from these two events is:

Data cache miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.4.3 Instruction Fetch Latency Mode

PMN0 accumulates the number of cycles when the instruction-cache is not able to deliver an instruction to the IXP43X network processors due to an instruction-cache miss or instruction-TLB miss. This event means that the processor core is stalled.

PMN1 counts the number of instruction fetch requests to external memory. Each of these requests loads 32 bytes at a time. This is the same event as measured in instruction cache efficiency mode.

Statistics derived from these two events:

- The average number of cycles the processor stalled waiting for an instruction fetch from external memory to return. This is calculated by dividing PMN0 by PMN1. If the average is high, then the IXP43X network processors is starved of the bus external to the IXP43X network processors.
- The percentage of total execution cycles the processor stalled waiting on an instruction fetch from external memory to return. This is calculated by dividing PMN0 by CCNT, and is used to measure total execution time.

3.7.4.4 Data/Bus Request Buffer Full Mode

The Data Cache has buffers available to service cache misses or uncacheable accesses. For every memory request that the Data Cache receives from the processor core a buffer is speculatively allocated in case an external memory request is required or temporary storage is needed for an unaligned access.

If no buffers are available, the Data Cache stalls the processor core. How often the Data Cache stalls depends on the performance of the bus external to the IXP43X network processors and what the memory access latency is for Data Cache miss requests to external memory. If the memory access latency for the IXP43X network processors is high, possibly due to starvation, these Data Cache buffers will become full. This performance monitoring mode is provided to see if the IXP43X network processors is starved of the bus external to the IXP43X network processors, and affects the performance of the application running on the IXP43X network processors.

PMN0 accumulates the number of clock cycles the processor is being stalled due to this condition and PMN1 monitors the number of times this condition occurs.

Statistics derived from these two events:

- **The average number of cycles the processor stalled on a data-cache access that may overflow the data-cache buffers.** This is calculated by dividing PMN0 by PMN1. This statistic lets you know if the duration event cycles are due to many requests or are attributed to just a few requests. If the average is high, the IXP43X network processors is starved of the bus external to the IXP43X network processors.
- **The percentage of total execution cycles the processor stalled because a Data Cache request buffer is not available.** This is calculated by dividing PMN0 by CCNT, and is used to measure total execution time.

3.7.4.5 Stall/Write-Back Statistics

When an instruction requires the result of a previous instruction and that result is not yet available, the IXP43X network processors stall to preserve the correct data dependencies. PMN0 counts the number of stall cycles due to data-dependencies. Not all data-dependencies cause a stall; only the following dependencies cause such a stall penalty:

- Load-use penalty: attempting to use the result of a load before the load completes. To avoid the penalty, software should delay using the result of a load until it is available. This penalty shows the latency effect of data-cache access.
- Multiply/Accumulate-use penalty: attempting to use the result of a multiply or multiply-accumulate operation before the operation completes. Again, to avoid the penalty, software should delay using the result until it is available.
- ALU use penalty: there are a few isolated cases where back-to-back ALU operations may result in one cycle delay in the execution. These cases are defined in Table 3.9, “Performance Considerations” on page 166.

PMN1 counts the number of write-back operations emitted by the data cache. These write-backs occur when the data cache evicts a dirty line of data to make room for a newly requested line or as the result of clean operation (CP15, register 7).

Statistics derived from these two events:

- **The percentage of total execution cycles the processor stalled because of a data dependency.** This is calculated by dividing PMN0 by CCNT, and is used to measure total execution time. Often a compiler can reschedule code to avoid these penalties when given the right optimization switches.

- Total number of data write-back requests to external memory is derived solely with PMN1.

3.7.4.6 Instruction TLB Efficiency Mode

PMN0 totals the number of instructions that were executed, and does not include instructions that were translated by the instruction TLB and never executed. This can happen if a branch instruction changes the program flow; the instruction TLB may translate the next sequential instructions after the branch, before it receives the target address of the branch.

PMN1 counts the number of instruction TLB table-walks, that occurs when there is a TLB miss. If the instruction TLB is disabled PMN1 does not increment.

Statistics derived from these two events:

- Instruction TLB miss-rate. This is derived by dividing PMN1 by PMN0.
- **The average number of cycles it took to execute an instruction or commonly referred to as cycles-per-instruction (CPI).** The CPI is derived by dividing CCNT by PMN0, where CCNT is used to measure total execution time.

3.7.4.7 Data TLB Efficiency Mode

PMN0 totals the number of data cache accesses, and includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note: **STM** and **LDM** each counts as several accesses to the data TLB depending on the number of registers specified in the register list. **LDRD** registers two accesses.

PMN1 counts the number of data TLB table-walks, that occurs when there is a TLB miss. If the data TLB is disabled PMN1 does not increment.

The statistic derived from these two events is Data TLB miss-rate. This is derived by dividing PMN1 by PMN0.

3.7.5 Multiple Performance Monitoring Run Statistics

There are times when more than four events must be monitored for performance tuning. In this case, multiple performance monitoring runs is done, capturing various events from each run. For example, the first run could monitor the events associated with instruction cache performance and the second run could monitor the events associated with data cache performance. By combining the results, statistics like total number of memory requests could be derived.

3.7.6 Examples

In this example, the events selected with the Instruction Cache Efficiency mode are monitored and CCNT is used to measure total execution time. Sampling time ends when PMN0 overflows and generates an IRQ interrupt.

Example 14. Configuring the Performance Monitor

```
; Configure the performance monitor with the following values:
;   EVTSEL.evtCount0 = 7, EVTSEL.evtCount1 = 0 instruction cache efficiency
;   INTEN.inten = 0x7 set all counters to trigger an interrupt on overflow
;   PMNC.C = 1   reset CCNT register
;   PMNC.P = 1   reset PMN0 and PMN1 registers
;   PMNC.E = 1   enable counting
MOV  R0,#0x700
MCR  P14,0,R0,C8,c1,0  ; setup EVTSEL
MOV  R0,#0x7
MCR  P14,0,R0,C4,c1,0  ; setup INTEN
MCR  P14,0,R0,C0,c1,0  ; setup PMNC, reset counters & enable
; Counting begins
```

Counter overflow is dealt with in the IRQ interrupt service routine as shown below:

Example 15. Interrupt Handling

```
IRQ_INTERRUPT_SERVICE_ROUTINE:
; Assume that performance counting interrupts are the only IRQ in the system
MRC  P14,0,R1,C0,c1,0  ; read the PMNC register
BIC  R2,R1,#1            ; clear the enable bit, preserve other bits in PMNC
MCR  P14,0,R2,C0,c1,0  ; disable counting
MRC  P14,0,R3,C1,c1,0  ; read CCNT register
MRC  P14,0,R4,C0,c2,0  ; read PMN0 register
MRC  P14,0,R5,C1,c2,0  ; read PMN1 register

<process the results>
SUBS PC,R14,#4         ; return from interrupt
```

As an example, assume the following values in CCNT, PMN0, PMN1 and PMNC:

Example 16. Computing the Results

```
; Assume CCNT overflowed


CCNT = 0x0000,0020 ;Overflowed and continued counting

Number of instructions executed = PMN0 = 0x6AAA,AAAA

Number of instruction cache miss requests = PMN1 = 0x0555,5555

Instruction Cache miss-rate = 100 * PMN1/PMN0 = 5%

CPI = (CCNT + 2^32)/Number of instructions executed = 2.4 cycles/instruction
```

In the contrived example above, the instruction cache had a miss-rate of 5% and CPI is 2.4.

3.8 Programming Model

This section describes the programming model of the IXP43X network processors, namely the implementation options and extensions to the Intel StrongARM Version 5TE architecture.

3.8.1 Intel® StrongARM* Architecture Compatibility

The IXP43X network processors implement the integer instruction set architecture as specified in the Intel StrongARM V5TE. T refers to the thumb instruction set and E refers to the DSP-Enhanced instruction set.

The Intel StrongARM V5TE introduces a few more architecture features over Intel StrongARM V4, specifically the addition of tiny pages (1 Kbyte), a new instruction that counts the leading zeroes (**CLZ**) in a data value, enhanced Intel StrongARM-Thumb transfer instructions and a modification of the system control coprocessor, CP15.

3.8.2 Intel® StrongARM* Architecture Implementation Options

3.8.2.1 Big-Endian versus Little-Endian

The IXP43X network processors can operate in big or little-endian mode. The B-bit of the Control Register, coprocessor 15, register 1, bit 7 (Section 3.5.1.2, “Register 1: Control and Auxiliary Control Registers”) contained within the IXP43X network processors select the endianess mode of the Intel XScale processor.

Note: This bit takes effect even if the MMU is disabled.

If you choose little-endian then you have further options that control whether address and/or data coherency modes. Refer P-Attribute bit in the MMU (Section 3.1.1.1, “Page (P) Attribute Bit”) and Expansion Bus configuration Register Bit 8 BYTE_SWAP_ENABLE.

Note: The NPEs on the IXP43X network processors are **Big-Endian only**; so if you change the endianess of the Intel XScale processor to little-endian for your operating system, then this has an impact on how the NPEs and the Intel XScale processor exchange data. The IXP400 software release handles this.

3.8.2.2 26-Bit Architecture

The Intel XScale processor does not support 26-bit architecture.

3.8.2.3 Thumb

The Intel XScale processor supports the thumb instruction set.

3.8.2.4 Intel® StrongARM* DSP-Enhanced Instruction Set

The Intel XScale processor implements DSP-enhanced instruction set of Intel StrongARM and is a set of instructions that boost the performance of signal processing applications. There are new multiply instructions that operate on 16-bit data values and new saturation instructions. Saturated instructions are used to ensure accuracy during DSP operations to ensure the signed extension is maintained during an overflow arithmetic operation. Further information on saturated integer arithmetic is found in the *ARM* Architecture Reference Manual*.

Some of the new instructions are:

- SMLAxy — 32<=16x16+32
- SMLAWy — 32<=32x16+32
- SMLALxy — 64<=16x16+64
- SMULxy — 32<=16x16
- SMULWy — 32<=32x16
- QADD — Adds two registers and saturates the result if an overflow occurred
- QDADD — Doubles and saturates one of the input registers then add and saturate
- QSUB — Subtracts two registers and saturates the result if an overflow occurred
- QDSUB — Doubles and saturates one of the input registers then subtract and saturate

The Intel XScale processor also implements Load Two words (LDRD), Store Two Words (STRD) and cache preload (PLD) instructions with the following implementation notes:

- PLD is interpreted as a read operation by the MMU and is ignored by the data breakpoint unit, that is, PLD never generates data breakpoint events.
- PLD to a non-cacheable page performs no action. Also, if the targeted cache line is already resident, this instruction has no affect.
- Both LDRD and STRD instructions generates an alignment exception when the address bits [2:0] = 0b100.

The transfers of two Intel StrongARM register values to a coprocessor (MCRR) and the transfer of values from a coprocessor to two registers (MRRC) is supported on the IXP43X network processors only when directed to coprocessor 0 and is used to access the internal accumulator. See “Internal Accumulator Access Format” on page 156 for more information. Access to coprocessors 15 and 14 generate an undefined instruction exception.

3.8.2.5 Base Register Update

If a data abort is signalled on a memory instruction that specifies write-back, the contents of the base register is not updated. This holds for all load and store instructions. This behavior matches that of the first generation Intel StrongARM processor and is referred to in the Intel StrongARM V5TE architecture as the Base Restored Abort Model.

3.8.3 Extensions to Intel® StrongARM* Architecture

The Intel XScale processor adds a few extensions to the Intel StrongARM Version 5TE architecture to meet the needs of various markets and design requirements. The following is a list of the extensions that are discussed in the next sections.

- A DSP coprocessor (CP0) has been added that contains a 40-bit accumulator and eight new instructions.
- New page attributes are added to the page table descriptors. The C and B page attribute encoding is extended by one more bit to allow for more encodings: write allocate and mini-data cache.
- Additional functionality has been added to coprocessor 15. Coprocessor 14 is also created.
- Enhancements were made to the event architecture, and includes instruction cache and data cache parity error exceptions, breakpoint events, and imprecise external data aborts.

3.8.3.1 DSP Coprocessor 0 (CP0)

The Intel XScale processor adds a DSP coprocessor to the architecture for the purpose of increasing the performance and the precision of audio processing algorithms. This coprocessor contains a 40-bit accumulator and eight new instructions.

The 40-bit accumulator is referenced by several new instructions that were added to the architecture; **MIA**, **MIAPH** and **MIAxy** are multiply/accumulate instructions that reference the 40-bit accumulator instead of a register specified accumulator. **MAR** and **MRA** provide the ability to read and write the 40-bit accumulator.

Access to CP0 is always allowed in all processor modes when bit 0 of the Coprocessor Access Register is set. Any access to CP0 when this bit is clear, causes an undefined exception. (See “Register 15: Coprocessor Access Register” on page 95 for more details).

Note: Only privileged software can set this bit in the Coprocessor Access Register located in CP15.

The 40-bit accumulator must be saved on a context switch if multiple processes are using it.

Two new instruction formats were added for coprocessor 0: Multiply with Internal Accumulate Format and Internal Accumulate Access Format. The formats and instructions are described next.

3.8.3.1.1 Multiply with Internal Accumulate Format

A new multiply format has been created to define operations on 40-bit accumulators. Table 8, “MRC/MCR Format” on page 85 shows the layout of the new format. The op code for this format lies within the coprocessor register transfer instruction type. These instructions have their own syntax.

Table 63. Multiply with Internal Accumulate Format

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
cond				1	1	1	0	0	0	1	0	opcode_3				Rs				0	0	0	0	acc			1	Rm			

Bits	Description	Notes
31:28	**cond** - Intel StrongARM condition codes	-
19:16	**opcode_3** - specifies the type of multiply with internal accumulate	Intel XScale processor defines the following: 0b0000 = **MIA** 0b1000 = **MIAPH** 0b1100 = **MIABB** 0b1101 = **MIABT** 0b1110 = **MIATB** 0b1111 = **MIATT** The effect of all other encodings are unpredictable.
15:12	**Rs** - Multiplier	
7:5	**acc** - select 1 of 8 accumulators	Intel XScale processor only implements acc0; access to any other acc has unpredictable effect.
3:0	**Rm** - Multiplicand	-

Two new fields were created for this format, acc and opcode_3. The acc field specifies one of eight internal accumulators to operate on and opcode_3 defines the operation for this format. The Intel XScale processor defines a single 40-bit accumulator referred to as acc0; future implementations may define multiple internal accumulators. The Intel XScale processor uses opcode_3 to define six instructions, **MIA**, **MIAPH**, **MIABB, MIABT, MIATB and MIATT.**

Table 64. MIA{<cond>} acc0, Rm, Rs

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
cond				1	1	1	0	0	0	1	0	0	0	0	0	Rs				0	0	0	0	0	0	0	1	Rm			

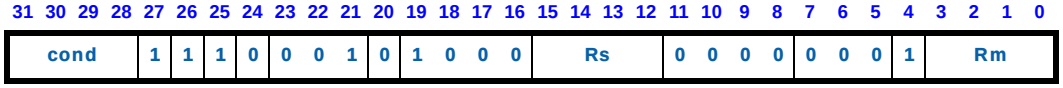
```
Operation:  if ConditionPassed(<cond>) then

                acc0 = (Rm[31:0] * Rs[31:0])[39:0] + acc0[39:0]

Exceptions: none

Qualifiers  Condition Code

            No condition code flags are updated

Notes:      Early termination is supported. Instruction timings is found

            in "Multiply Instruction Timings" on page 170.

            Specifying R15 for register Rs or Rm has unpredictable results.

            acc0 is defined to be 0b000 on Intel XScale processor.
```

The **MIA** instruction operates similarly to **MLA** except that the 40-bit accumulator is used. **MIA** multiplies the signed value in register Rs (multiplier) by the signed value in register Rm (multiplicand) and then adds the result to the 40-bit accumulator (acc0).

MIA does not support unsigned multiplication; all values in Rs and Rm is interpreted as signed data values. **MIA** is useful for operating on signed 16-bit data that is loaded into a general purpose register by **LDRSH**.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 65. MIAPH{<cond>} acc0, Rm, Rs




```
Operation: if ConditionPassed(<cond>) then

               acc0 = sign_extend(Rm[31:16] * Rs[31:16]) +

                      sign_extend(Rm[15:0] * Rs[15:0]) +

                      acc0[39:0]

Exceptions:no ne

Qualifiers Condition Code

           S bit is always cleared; no condition code flags are updated

Notes:     Instruction timings is found
           in "Multiply Instruction Timings" on page 170.

           Specifying R15 for register Rs or Rm has unpredictable results.

           acc0 is defined to be 0b000 on Intel XScale processor
```

The **MIAPH** instruction performs two16-bit signed multiplies on packed half word data and accumulates these to a single 40-bit accumulator. The first signed multiplication is performed on the lower 16 bits of the value in register Rs with the lower 16 bits of the value in register Rm. The second signed multiplication is performed on the upper 16 bits of the value in register Rs with the upper 16 bits of the value in register Rm. Both signed 32-bit products are sign extended and then added to the value in the 40-bit accumulator (acc0).

The instruction is only executed if the condition specified in the instruction matches the condition code status.

Table 66. MIAxy{<cond>} acc0, Rm, Rs

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
cond				1	1	1	0	0	0	1	0	1	1	x	y	Rs				0	0	0	0	0	0	0	1	Rm			

```
Operation: if ConditionPassed(<cond>) then

               if (bit[17] == 0)

                   <operand1> = Rm[15:0]

               else

                   <operand1> = Rm[31:16]


               if (bit[16] == 0)

                   <operand2> = Rs[15:0]

               else

                   <operand2> = Rs[31:16]


               acc0[39:0] = sign_extend(<operand1> * <operand2>) + acc0[39:0]


Exceptions: none

Qualifiers  Condition Code

            S bit is always cleared; no condition code flags are updated

Notes:      Instruction timings is found
            in "Multiply Instruction Timings" on page 170.

            Specifying R15 for register Rs or Rm has unpredictable results.

            acc0 is defined to be 0b000 on Intel XScale processor.
```

The **MIAxy** instruction performs one16-bit signed multiply and accumulates these to a single 40-bit accumulator. **x** refers to the upper half or lower half of register Rm (multiplicand) and **y** refers to the upper or lower half of Rs (multiplier). A value of 0x1 selects bits [31:16] of the register and is specified in the mnemonic as T (for top). A value of 0x0 selects bits [15:0] of the register and is specified in the mnemonic as B (for bottom).

MIAxy does not support unsigned multiplication; all values in Rs and Rm is interpreted as signed data values.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

3.8.3.1.2 Internal Accumulator Access Format

The Intel XScale processor defines a new instruction format for accessing internal accumulators in CP0. Table 67, "Internal Accumulator Access Format" on page 157 shows that the op code falls into the coprocessor register transfer space.

The RdHi and RdLo fields allow up to 64 bits of data transfer between Intel StrongARM registers and an internal accumulator. The acc field specifies 1 of 8 internal accumulators to transfer data to/from. The Intel XScale processor implements a single 40-bit accumulator referred to as acc0; future implementations can specify multiple internal accumulators of varying sizes, up to 64 bits.

Access to the internal accumulator is allowed in all processor modes (user and privileged) as long bit 0 of the Coprocessor Access Register is set. (See "Register 15: Coprocessor Access Register" on page 95 for more details).

The IXP43X network processors implement two instructions, **MAR** and **MRA** that move two Intel StrongARM registers to acc0 and move acc0 to two Intel StrongARM registers respectively.

Table 67. Internal Accumulator Access Format

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
cond				1	1	0	0	0	1	0	L	RdHi				RdLo				0	0	0	0	0	0	0	0	0	acc		

Bits	Description	Notes
31:28	**cond** - Intel StrongARM condition codes	-
20	**L** - move to/from internal accumulator 0= move to internal accumulator (MAR) 1= move from internal accumulator (MRA)	-
19:16	**RdHi** - specifies the high order eight (39:32) bits of the internal accumulator.	On a read of the acc, this 8-bit high order field is sign extended. On a write to the acc, the lower 8 bits of this register is written to acc[39:32]
15:12	**RdLo** - specifies the low order 32 bits of the internal accumulator	-
7:4	Should be zero	
3	Should be zero	-
2:0	**acc** - specifies 1 of 8 internal accumulators	Intel XScale processor only implements acc0; access to any other acc is unpredictable

Note: **MAR** has the same encoding as **MCRR** (to coprocessor 0) and **MRA** has the same encoding as **MRRC** (to coprocessor 0). These instructions move 64-bits of data to/from Intel StrongARM registers from/to coprocessor registers. **MCRR** and **MRRC** are defined in Intel StrongARM's DSP instruction set.

Disassemblers not aware of **MAR** and **MRA** produces the following syntax:

```
MCRR{<cond>} p0, 0x0, RdLo, RdHi, c0

MRRC{<cond>} p0, 0x0, RdLo, RdHi, c0
```

Table 68. MAR{<cond>} acc0, RdLo, RdHi

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
cond				1	1	0	0	0	1	0	0	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0

```
Operation:  if ConditionPassed(<cond>) then
                acc0[39:32] = RdHi[7:0]
                acc0[31:0] = RdLo[31:0]

Exceptions: none

Qualifiers  Condition Code
            No condition code flags are updated

Notes:      Instruction timings is found in

            "Multiply Instruction Timings" on page 170

            Specifying R15 as RdHi or RdLo has unpredictable results.
```

The MAR instruction moves the value in register RdLo to bits[31:0] of the 40-bit accumulator (acc0) and moves bits[7:0] of the value in register RdHi into bits[39:32] of acc0.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.

Table 69. MRA{<cond>} RdLo, RdHi, acc0

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
cond				1	1	0	0	0	1	0	1	RdHi				RdLo				0	0	0	0	0	0	0	0	0	0	0	0

```
Operation:  if ConditionPassed(<cond>) then
                RdHi[31:0] = sign_extend(acc0[39:32])
                RdLo[31:0] = acc0[31:0]

Exceptions: none

Qualifiers  Condition Code
            No condition code flags are updated

Notes:      Instruction timings is found in
            "Multiply Instruction Timings" on page 170

            Specifying the same register for RdHi and RdLo has unpredictable
            results.

            Specifying R15 as RdHi or RdLo has unpredictable results.
```

The MRA instruction moves the 40-bit accumulator value (acc0) into two registers. Bits[31:0] of the value in acc0 are moved into the register RdLo. Bits[39:32] of the value in acc0 are sign extended to 32 bits and moved into the register RdHi.

The instruction is only executed if the condition specified in the instruction matches the condition code status.

This instruction executes in any processor mode.

3.8.3.2 New Page Attributes

The Intel XScale processor extends the page attributes defined by the C and B bits in the page descriptors with an additional X bit. This bit allows four more attributes to be encoded when X=1. These new encodings include allocating data for the mini-data cache and write-allocate caching. A full description of the encodings is found in "Memory Attributes" on page 58.

The Intel XScale processor retains Intel StrongARM definitions of the C and B encoding when X = 0, and is different than the Intel StrongARM products. The memory attribute for the mini-data cache has been moved and replaced with the write-through caching attribute.

When write-allocate is enabled, a store operation that misses the data cache (cacheable data only) generates a line fill. If disabled, a line fill only occurs when a load operation misses the data cache (cacheable data only).

Write-through caching causes all store operations to be written to memory, whether they are cacheable or not cacheable. This feature is useful for maintaining data cache coherency.

Bit 1 in the Control Register (coprocessor 15, register 1, opcode=1) is used reserved for the P bit memory attribute for memory accesses made during page table walks. The P bit is not implemented on the IXP43X network processors.

These attributes are programmed in the translation table descriptors, that are highlighted in:

- Table 70, "First-Level Descriptors" on page 159
- Table 71, "Second-Level Descriptors for Coarse Page Table" on page 160
- Table 72, "Second-Level Descriptors for Fine Page Table" on page 160

Two second-level descriptor formats have been defined for the IXP43X network processors, one is used for the coarse page table and the other is used for the fine page table.

Table 70. First-Level Descriptors

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
SBZ																														0	0
Coarse page table base address																						P	Domain				SBZ			0	1
Section base address												SBZ				TEX			AP			P	Domain				0	C	B	1	0
Fine page table base address																				SBZ		P	Domain				SBZ			1	1

Table 71. Second-Level Descriptors for Coarse Page Table

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
SBZ																														0	0
Large page base address																TEX				AP3		AP2		AP1		AP0		C		B	0	1
Small page base address																				AP3		AP2		AP1		AP0		C		B	1	0
Extended small page base address																				SBZ		TEX				AP		C		B	1	1

Table 72. Second-Level Descriptors for Fine Page Table

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
SBZ																														0	0
Large page base address																TEX				AP3		AP2		AP1		AP0		C		B	0	1
Small page base address																				AP3		AP2		AP1		AP0		C		B	1	0
Tiny Page Base Address																						TEX				AP		C		B	1	1

The TEX (Type Extension) field is present in several of the descriptor types. In the Intel XScale processor, only the LSB of this field is defined; this is called the X bit. The remaining bits are reserved for future use and should be programmed as zero (SBZ) on the IXP43X network processors.

A Small Page descriptor does not have a TEX field. For these descriptors, TEX is implicitly zero; that is, they operate as if the X bit had a ‘0’ value.

The X bit, when set, modifies the meaning of the C and B bits. Description of page attributes and their encoding is found in “Memory Management Unit” on page 57.

3.8.3.3 Additions to CP15 Functionality

To accommodate the functionality in the Intel XScale processor, registers in CP15 and CP14 have been added or augmented. See “Configuration” on page 84 for details.

At times it is necessary to be able to guarantee exactly when a CP15 update takes effect. For example, when enabling memory address translation (turning on the MMU), it is vital to know when the MMU is actually guaranteed to be in operation. To address this need, a processor-specific code sequence is defined for the Intel XScale processor. The sequence — called CPWAIT — is shown in Example 12 on page 96.

Example 17. CPWAIT: Canonical Method to Wait for CP15 Update

```
;; The following macro should be used when software needs to be

;; assured that a CP15 update has taken effect.

;; It may only be used while in a privileged mode, because it

;; accesses CP15.


MACRO CPWAIT


        MRC P15, 0, R0, C2, C0, 0             ; arbitrary read of CP15

        MOV R0, R0                            ; wait for it

        SUB PC, PC, #4                        ; branch to next instruction


        ; At this point, any previous CP15 writes are

        ; guaranteed to have taken effect.

ENDM
```

When setting multiple CP15 registers, system software may opt to delay the assurance of their update. This is accomplished by executing CPWAIT only after the sequence of MCR instructions.

The CPWAIT sequence guarantees that CP15 side-effects are complete by the time the CPWAIT is complete. It is possible, that the CP15 side-effect takes place before CPWAIT completes or is issued. Programmers should take care that this does not affect the correctness of their code.

3.8.3.4 Event Architecture

3.8.3.4.1 Exception Summary

Table 73 shows all the exceptions that the Intel XScale processor may generate, and the attributes of each. Subsequent sections give details on each exception.

Table 73. Exception Summary (Sheet 1 of 2)

Exception Description	Exception Type[1]	Precise	Updates FAR
Reset	Reset	N	N
FIQ	FIQ	N	N
IRQ	IRQ	N	N
External Instruction	Prefetch	Y	N
Instruction MMU	Prefetch	Y	N
Instruction Cache Parity	Prefetch	Y	N

Notes:
1. Exception types are those described in the Intel StrongARM manual, section 2.5.
2. Refer to "Software Debug" on page 99 for more details

Table 73. Exception Summary (Sheet 2 of 2)

Exception Description	Exception Type[1]	Precise	Updates FAR
Lock Abort	Data	Y	N
MMU Data	Data	Y	Y
External Data	Data	N	N
Data Cache Parity	Data	N	N
Software Interrupt	Software Interrupt	Y	N
Undefined Instruction	Undefined Instruction	Y	N
Debug Events[2]	varies	varies	N

Notes:
1. Exception types are those described in the Intel StrongARM manual, section 2.5.
2. Refer to "Software Debug" on page 99 for more details

3.8.3.4.2 Event Priority

The Intel XScale processor follows the exception priority specified in the *ARM* Architecture Reference Manual*. The processor has additional exceptions that might be generated during debugging. For information on these debug exceptions, see "Software Debug" on page 99.

Table 74. Event Priority

Exception	Priority
Reset	1 (Highest)
Data Abort (Precise & Imprecise)	2
FIQ	3
IRQ	4
Prefetch Abort	5
Undefined Instruction, SWI	6 (Lowest)

3.8.3.4.3 Prefetch Aborts

The Intel XScale processor detects three types of prefetch aborts: Instruction MMU abort, external abort on an instruction access, and an instruction cache parity error. These aborts are described in Table 75.

When a prefetch abort occurs, hardware reports the highest priority one in the extended Status field of the Fault Status Register. The value placed in R14_ABORT (the link register in abort mode) is the address of the aborted instruction + 4.

Table 75. Processors': Encoding of Fault Status for Prefetch Aborts

Priority	Sources	FS[10,3:0]†	Domain	FAR
Highest	Instruction MMU Exception Several exceptions can generate this encoding: - translation faults - domain faults, and - permission faults It is up to software to figure out which one occurred.	0b10000	invalid	invalid
	External Instruction Error Exception This exception occurs when the external memory system reports an error on an instruction cache fetch.	0b10110	invalid	invalid
Lowest	Instruction Cache Parity Error Exception	0b11000	invalid	invalid

† All other encodings not listed in the table are reserved.

3.8.3.4.4 Data Aborts

Two types of data aborts exist in the Intel XScale processor: precise and imprecise. A precise data abort is defined as one where R14_ABORT always contains the PC (+8) of the instruction that caused the exception. An imprecise abort is one where R14_ABORT contains the PC (+4) of the next instruction to execute and not the address of the instruction that caused the abort. In other words, instruction execution has advanced beyond the instruction that caused the data abort.

On the Intel XScale processor precise data aborts are recoverable and imprecise data aborts are not recoverable.

Precise Data Aborts

- A lock abort is a precise data abort; the extended Status field of the Fault Status Register is set to 0xb10100. This abort occurs when a lock operation directed to the MMU (instruction or data) or instruction cache causes an exception, due to a translation fault, access permission fault or external bus fault.

  The Fault Address Register is undefined and R14_ABORT is the address of the aborted instruction + 8.

- A data MMU abort is precise. These are due to an alignment fault, translation fault, domain fault, permission fault or external data abort on an MMU translation. The status field is set to a predetermined Intel StrongARM definition and is shown in Table 76.

  The Fault Address Register is set to the effective data address of the instruction and R14_ABORT is the address of the aborted instruction + 8.

Table 76. Intel XScale® Processor Encoding of Fault Status for Data Aborts (Sheet 1 of 2)

Priority	Sources		FS[10,3:0]†	Domain	FAR
Highest	Alignment		0b000x1	invalid	valid
	External Abort on Translation	First level Second level	0b01100 0b01110	invalid valid	valid valid
	Translation	Section Page	0b00101 0b00111	invalid valid	valid valid
	Domain	Section Page	0b01001 0b01011	valid valid	valid valid

† All other encodings not listed in the table are reserved.

Table 76. Intel XScale® Processor Encoding of Fault Status for Data Aborts (Sheet 2 of 2)

Priority	Sources		FS[10,3:0]†	Domain	FAR
	Permission	Section Page	0b01101 0b01111	valid valid	valid valid
	Lock Abort This data abort occurs on an MMU lock operation (data or instruction TLB) or on an Instruction Cache lock operation.		0b10100	invalid	invalid
	Imprecise External Data Abort		0b10110	invalid	invalid
Lowest	Data Cache Parity Error Exception		0b11000	invalid	invalid

† All other encodings not listed in the table are reserved.

Imprecise Data Aborts

- A data cache parity error is imprecise; the extended Status field of the Fault Status Register is set to 0xb11000.
- All external data aborts except for those generated on a data MMU translation are imprecise.

The Fault Address Register for all imprecise data aborts is undefined and R14_ABORT is the address of the next instruction to execute + 4, and is the same for both Intel StrongARM and Thumb mode.

Although the Intel XScale processor guarantees the Base Restored Abort Model for precise aborts, it cannot do so in the case of imprecise aborts. A Data Abort handler may encounter an updated base register if it is invoked because of an imprecise abort.

Imprecise data aborts may create scenarios that are difficult for an abort handler to recover. Both external data aborts and data cache parity errors may result in corrupted data in the targeted registers. Because these faults are imprecise, it is possible that the corrupted data has been used before the Data Abort fault handler is invoked. Because of this, software should treat imprecise data aborts as unrecoverable.

Note: Even memory accesses marked as **stall until complete** (see "Details on Data Cache and Write Buffer Behavior" on page 58) can result in imprecise data aborts. For these types of accesses, the fault is somewhat less imprecise than the general case: it is guaranteed to be raised within three instructions of the instruction that caused it. In other words, if a **stall until complete** LD or ST instruction triggers an imprecise fault, then that fault is seen by the program within three instructions.

With this knowledge, it is possible to write code that accesses **stall until complete** memory with impunity. Place several NOP instructions after such an access. If an imprecise fault occurs, it does so during the NOPs; the data abort handler sees identical register and memory state as it would with a precise exception, and so should be able to recover. An example of this is shown in Example 18 on page 165.

Example 18. Shielding Code from Potential Imprecise Aborts

```
;; Example of code that maintains architectural state through the

;; window where an imprecise fault might occur.


       LD      R0, [R1]                      ; R1 points to stall-until-complete

                                             ;   region of memory

       NOP

       NOP

       NOP

       ; Code beyond this point is guaranteed not to see any aborts

       ; from the LD.
```

If a system design precludes events that could cause external aborts, then such precautions are not necessary.

Multiple Data Aborts

Multiple data aborts are detected by hardware but only the highest priority one is reported. If the reported data abort is precise, software can correct the cause of the abort and re-execute the aborted instruction. If the lower priority abort still exists, it is reported. Software can handle each abort separately until the instruction successfully executes.

If the reported data abort is imprecise, software must check the SPSR to see if the previous context is executing in abort mode. If this is the case, the link back to the current process has been lost and the data abort is unrecoverable.

3.8.3.4.5 Events from Preload Instructions

A **PLD** instruction never causes the Data MMU to fault for any of the following reasons:

- Domain fault
- Permission fault
- Translation fault

If execution of the **PLD** would cause one of the above faults, then the **PLD** causes no effect.

This feature allows software to issue **PLD**s speculatively. For example, Example 19 on page 166 places a **PLD** instruction early in the loop. This **PLD** is used to fetch data for the next loop iteration. In this example, the list is terminated with a node that has a null pointer. When execution reaches the end of the list, the **PLD** on address 0x0 does not cause a fault. Rather, it is ignored and the loop terminates normally.

Example 19. Speculatively issuing PLD

```
;; R0 points to a node in a linked list. A node has the following layout:

;; Offset   Contents

;;----------------------------------

;;      0   data

;;      4   pointer to next node

;; This code computes the sum of all nodes in a list. The sum is placed into R9.

;;

        MOV R9, #0      ; Clear accumulator

sumList:

        LDR R1, [R0, #4]   ; R1 gets pointer to next node

        LDR R3, [R0]       ; R3 gets data from current node

        PLD [R1]           ; Speculatively start load of next node

        ADD R9, R9, R3     ; Add into accumulator

        MOVS R0, R1        ; Advance to next node. At end of list?

        BNE sumList        ; If not then loop
```

3.8.3.4.6 Debug Events

Debug events are covered in "Debug Exceptions" on page 103.

3.9 Performance Considerations

This section describes relevant performance considerations that compiler writers, application programmers, and system designers must be aware of, to efficiently use the IXP43X network processors. Performance numbers discussed here include interrupt latency, branch prediction, and instruction latencies.

3.9.1 Interrupt Latency

Minimum Interrupt Latency is defined as the minimum number of cycles from the assertion of any interrupt signal (IRQ or FIQ) to the execution of the instruction at the vector for that interrupt. This number assumes best case conditions exist when the interrupt is asserted, for example, the system isn't waiting on the completion of some other operation.

A sometimes more useful number to work with is the Maximum Interrupt Latency. This is typically a complex calculation that depends on what else is going on in the system at the time the interrupt is asserted. Some examples that can adversely affect interrupt latency are:

- The instruction currently executing could be a 16-register LDM
- The processor could fault just when the interrupt arrives
- The processor could be waiting for data from a load, doing a page table walk, and so forth.

- High core-to-system (bus) clock ratios

Maximum Interrupt Latency is reduced by:

- Ensuring that the interrupt vector and interrupt service routine are resident in the instruction cache. This is accomplished by locking them down into the cache.
- Removing or reducing the occurrences of hardware page table walks. This also is accomplished by locking down the application's page table entries into the TLBs, along with the page table entry for the interrupt service routine.

3.9.2 Branch Prediction

The IXP43X network processors implement dynamic branch prediction for the Intel StrongARM instructions **B** and **BL** and for the thumb instruction **B**. Any instruction that specifies the PC as the destination is predicted as not taken. For example, an **LDR** or a **MOV** that loads or moves directly to the PC is predicted not taken and incur a branch latency penalty.

These instructions, Intel StrongARM **B,** Intel StrongARM **BL** and thumb **B,** enter into the branch target buffer when they are **taken** for the first time. (A **taken** branch refers to when they are evaluated to be true.) Once in the branch target buffer, the IXP43X network processors dynamically predict the outcome of these instructions based on previous outcomes. Table 77 shows the branch latency penalty when these instructions are correctly predicted and when they are not. A penalty of zero for correct prediction means that the IXP43X network processors can execute the next instruction in the program flow in the cycle following the branch.

Table 77. Branch Latency Penalty

Core Clock Cycles		Description
Intel® StrongARM*	Thumb	
+0	+ 0	**Predicted Correctly**. The instruction is in the branch target cache and is correctly predicted.
+4	+ 5	**Mispredicted**. There are three occurrences of branch misprediction, all of which incur a 4-cycle branch delay penalty. 1. The instruction is in the branch target buffer and is predicted not-taken, but is actually taken. 2. The instruction is not in the branch target buffer and is a taken branch. 3. The instruction is in the branch target buffer and is predicted taken, but is actually not-taken

3.9.3 Addressing Modes

All load and store addressing modes implemented in the IXP43X network processors do not add to the instruction latencies numbers.

3.9.4 Instruction Latencies

The latencies for all the instructions are shown in the following sections with respect to their functional groups: branch, data processing, multiply, status register access, load/store, semaphore, and coprocessor.

The following section explains how to read these tables.

3.9.4.1 Performance Terms

- Issue Clock (cycle 0)

The first cycle when an instruction is decoded **and** allowed to proceed to further stages in the execution pipeline (that is, when the instruction is actually issued).

- Cycle Distance from A to B

  The cycle distance from cycle **A** to cycle **B** is **(B-A)** -- that is, the number of cycles from the start of cycle **A** to the start of cycle **B**. Example: the cycle distance from cycle 3 to cycle 4 is one cycle.

- Issue Latency

  The cycle distance **from** the first issue clock of the current instruction **to** the issue clock of the next instruction. The actual number of cycles is influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.

- Result Latency

  The cycle distance **from** the first issue clock of the current instruction **to** the issue clock of the first instruction that can use the result without incurring a resource dependency stall. The actual number of cycles is influenced by cache-misses, resource-dependency stalls, and resource availability conflicts.

- Minimum Issue Latency (without Branch Misprediction)

  The minimum cycle distance **from** the issue clock of the current instruction **to** the first possible issue clock of the next instruction assuming best case conditions (that is, that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; the current instruction does not incur resource dependency stalls during execution that cannot be detected at issue time; and if the instruction uses dynamic branch prediction, correct prediction is assumed).

- Minimum Result Latency

  The required minimum cycle distance **from** the issue clock of the current instruction **to** the issue clock of the first instruction that can use the result without incurring a resource dependency stall assuming best case conditions (that is, that the issuing of the next instruction is not stalled due to a resource dependency stall; the next instruction is immediately available from the cache or memory interface; and the current instruction does not incur resource dependency stalls during execution that cannot be detected at issue time).

- Minimum Issue Latency (with Branch Misprediction)

  The minimum cycle distance **from** the issue clock of the current branching instruction **to** the first possible issue clock of the next instruction. This definition is identical to Minimum Issue Latency except that the branching instruction has been incorrectly predicted. It is calculated by adding Minimum Issue Latency (without Branch Misprediction) to the minimum branch latency penalty number from Table 77, and is four cycles.

- Minimum Resource Latency

  The minimum cycle distance from the issue clock of the current multiply instruction to the issue clock of the next multiply instruction assuming the second multiply does not incur a data dependency and is immediately available from the instruction cache or memory interface.

  For the following code fragment, here is an example of computing latencies:

Example 20. Computing Latencies

```
UMLAL   r6,r8,r0,r1

ADD     r9,r10,r11

SUB     r2,r8,r9

MOV     r0,r1
```

Table 78 shows how to calculate Issue Latency and Result Latency for each instruction. Looking at the issue column, the **UMLAL** instruction starts to issue on cycle 0 and the next instruction, **ADD,** issues on cycle 2, so the Issue Latency for **UMLAL** is two. From the code fragment, there is a result dependency between the **UMLAL** instruction and the **SUB** instruction. In Table 78, **UMLAL** starts to issue at cycle 0 and the **SUB** issues at cycle 5. thus the Result Latency is five.

Table 78. Latency Example

Cycle	Issue	Executing
0	umlal (1st cycle)	--
1	umlal (2nd cycle)	umlal
2	add	umlal
3	sub (stalled)	umlal & add
4	sub (stalled)	umlal
5	sub	umlal
6	mov	sub
7	--	mov

3.9.4.2 Branch Instruction Timings

Table 79. Branch Instruction Timings (Those Predicted by the BTB)

Mnemonic	Minimum Issue Latency When Correctly Predicted by the BTB	Minimum Issue Latency with Branch Misprediction
B	1	5
BL	1	5

Table 80. Branch Instruction Timings (Those not Predicted by the BTB)

Mnemonic	Minimum Issue Latency When the Branch is not Taken	Minimum Issue Latency When the Branch is Taken
BLX(1)	N/A	5
BLX(2)	1	5
BX	1	5
Data Processing Instruction with PC as the destination	Same as Table 81	4 + numbers in Table 81
LDR PC,< >	2	8
LDM with PC in register list	3 + numreg†	10 + max (0, numreg-3)

† numreg is the number of registers in the register list including the PC.

3.9.4.3 Data Processing Instruction Timings

Table 81. Data Processing Instruction Timings

Mnemonic	<shifter operand> is NOT a Shift/Rotate by Register		<shifter operand> is a Shift/Rotate by Register OR <shifter operand> is RRX	
	Minimum Issue Latency	Minimum Result Latency†	Minimum Issue Latency	Minimum Result Latency†
ADC	1	1	2	2
ADD	1	1	2	2
AND	1	1	2	2
BIC	1	1	2	2
CMN	1	1	2	2
CMP	1	1	2	2
EOR	1	1	2	2
MOV	1	1	2	2
MVN	1	1	2	2
ORR	1	1	2	2
RSB	1	1	2	2
RSC	1	1	2	2
SBC	1	1	2	2
SUB	1	1	2	2
TEQ	1	1	2	2
TST	1	1	2	2

† If the next instruction should use the result of the data processing for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.4 Multiply Instruction Timings

Table 82. Multiply Instruction Timings (Sheet 1 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency†	Minimum Resource Latency (Throughput)
MLA	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4
MUL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	2	1
		1	2	2	2
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	3	2
		1	3	3	3
	all others	0	1	4	3
		1	4	4	4

† If the next instruction must use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

Table 82. Multiply Instruction Timings (Sheet 2 of 2)

Mnemonic	Rs Value (Early Termination)	S-Bit Value	Minimum Issue Latency	Minimum Result Latency†	Minimum Resource Latency (Throughput)
SMLAL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMLALxy	N/A	N/A	2	RdLo = 2; RdHi = 3	2
SMLAWy	N/A	N/A	1	3	2
SMLAxy	N/A	N/A	1	2	1
SMULL	Rs[31:15] = 0x00000 or Rs[31:15] = 0x1FFFF	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00 or Rs[31:27] = 0x1F	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5
SMULWy	N/A	N/A	1	3	2
SMULxy	N/A	N/A	1	2	1
UMLAL	Rs[31:15] = 0x00000	0	2	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	2	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	2	RdLo = 4; RdHi = 5	4
		1	5	5	5
UMULL	Rs[31:15] = 0x00000	0	1	RdLo = 2; RdHi = 3	2
		1	3	3	3
	Rs[31:27] = 0x00	0	1	RdLo = 3; RdHi = 4	3
		1	4	4	4
	all others	0	1	RdLo = 4; RdHi = 5	4
		1	5	5	5

† If the next instruction must use the result of the multiply for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

Table 83. Multiply Implicit Accumulate Instruction Timings

Mnemonic	Rs Value (Early Termination)	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MIA	Rs[31:15] = 0x0000 or Rs[31:15] = 0xFFFF	1	1	1
	Rs[31:27] = 0x0 or Rs[31:27] = 0xF	1	2	2
	all others	1	3	3
MIAxy	N/A	1	1	1
MIAPH	N/A	1	2	2

Table 84. Implicit Accumulator Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency	Minimum Resource Latency (Throughput)
MAR	2	2	2
MRA	1	(RdLo = 2; RdHi = 3)†	2

† If the next instruction must use the result of the MRA for a shift by immediate or as Rn in a QDADD or QDSUB, one extra cycle of result latency is added to the number listed.

3.9.4.5 Saturated Arithmetic Instructions

Table 85. Saturated Data Processing Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
QADD	1	2
QSUB	1	2
QDADD	1	2
QDSUB	1	2

3.9.4.6 Status Register Access Instructions

Table 86. Status Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRS	1	2
MSR	2 (6 if updating mode bits)	1

3.9.4.7 Load/Store Instructions

Table 87. Load and Store Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDR	1	3 for load data; 1 for writeback of base
LDRB	1	3 for load data; 1 for writeback of base
LDRBT	1	3 for load data; 1 for writeback of base
LDRD	1 (+1 if Rd is R12)	3 for Rd; 4 for Rd+1; 1 (+1 if Rd is R12) for write-back of base
LDRH	1	3 for load data; 1 for writeback of base
LDRSB	1	3 for load data; 1 for writeback of base

Table 87. Load and Store Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDRSH	1	3 for load data; 1 for writeback of base
LDRT	1	3 for load data; 1 for writeback of base
PLD	1	N/A
STR	1	1 for writeback of base
STRB	1	1 for writeback of base
STRBT	1	1 for writeback of base
STRD	2	2 for write-back of base
STRH	1	1 for writeback of base
STRT	1	1 for writeback of base

Table 88. Load and Store Multiple Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
LDM[1]	2 + numreg[2]	5-18 for load data (4 + numreg for last register in list; 3 + numreg for 2nd to last register in list; 2 + numreg for all other registers in list); 2+ numreg for write-back of base
STM	2 + numreg	2 + numreg for write-back of base

Notes:
1. See Table 80 on page 169 for LDM timings when R15 is in the register list.
2. numreg is the number of registers in the register list.

3.9.4.8 Semaphore Instructions

Table 89. Semaphore Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
SWP	5	5
SWPB	5	5

3.9.4.9 Coprocessor Instructions

Table 90. CP15 Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC†	4	4
MCR	2	N/A

† MRC to R15 is unpredictable.

Table 91. CP14 Register Access Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
MRC	8	8
MRC to R15	9	9
MCR	8	N/A
LDC	11	N/A
STC	8	N/A

3.9.4.10 Miscellaneous Instruction Timing

Table 92. Exception-Generating Instruction Timings

Mnemonic	Minimum latency to first instruction of exception handler
SWI	6
BKPT	6
UNDEFINED	6

Table 93. Count Leading Zeros Instruction Timings

Mnemonic	Minimum Issue Latency	Minimum Result Latency
CLZ	1	1

3.9.4.11 Thumb Instructions

In general, the timing of Thumb instructions are the same as their equivalent Intel StrongARM instructions, except for the cases listed below.

- If the equivalent Intel StrongARM instruction maps to one in Table 79 on page 169, the **Minimum Issue Latency with Branch Misprediction** goes from 5 to 6 cycles. This is due to the branch latency penalty. (See Table 77 on page 167.)
- If the equivalent Intel StrongARM instruction maps to one in Table 80 on page 169, the **Minimum Issue Latency when the Branch is Taken** increases by 1 cycle. This is due to the branch latency penalty. (See Table 77 on page 167.)
- A Thumb BL instruction when H = 0 has the same timing as an Intel StrongARM data processing instruction.

The mapping of Thumb instructions to Intel StrongARM instructions is found in the *ARM* Architecture Reference Manual*.

3.10 Optimization Guide

3.10.1 Introduction

This document contains optimization techniques for achieving the highest performance from the IXP43X network processors architecture. It is written for developers who are optimizing compilers or performance analysis tools for the devices based on these processors. It can also be used by application developers to obtain the best performance from their assembly language code. The optimizations presented in this section are based on the IXP43X network processors and applied to all products that are based on it.

The IXP43X network processors architecture includes a super-pipelined RISC architecture with an enhanced memory pipeline. The instruction set for IXP43X network processors is based on the Intel StrongARM V5TE architecture, but the IXP43X network processors include new instructions. Code generated for the SA110, SA1100 and SA1110 executes on the IXP43X network processors, but to obtain the maximum performance of your application code, it should be optimized for the IXP43X network processors using the techniques presented in this document.

3.10.1.1 About This Section

This guide assumes that you are familiar with the Intel StrongARM instruction set and the C language. It consists of the following sections:

- "Introduction" on page 174 — Outlines the contents of this guide
- "Processor Pipeline" on page 175 — Provides an overview of pipeline behavior for the IXP43X network processors
- "Basic Optimizations" on page 180 — Outlines basic optimizations that are applied to the IXP43X network processors
- "Cache and Prefetch Optimizations" on page 186 — Contains optimizations for efficient use of caches. Also included are optimizations that take advantage of the prefetch instruction of the IXP43X network processors
- "Instruction Scheduling" on page 197 — Shows how to optimally schedule code for the pipeline of the IXP43X network processors
- "Optimizing C Libraries" on page 205 — Contains information relating to optimizations for C library routines.
- "Optimizations for Size" on page 205 — Contains optimizations that reduce the size of the generated code. Thumb optimizations are also included.

3.10.2 Processor Pipeline

One of the biggest differences between the IXP43X network processors and Intel StrongARM processors is the pipeline. Many of the differences are summarized in Figure 29. This section provides a brief description of the structure and behavior of the IXP43X network processors pipeline.

3.10.2.1 General Pipeline Characteristics

As the pipelines for the IXP43X network processors are scalar and single issue, instructions can occupy all the three pipelines at once. Out of order completion is possible. The following sections discuss general pipeline characteristics.

3.10.2.1.1 Number of Pipeline Stages

The IXP43X network processors have a longer pipeline (seven stages versus five stages) and operate at a much higher frequency than the Intel® *IXP42X Network Processors*. This allows for greater overall performance. But, the longer pipeline has several negative consequences:

- Larger branch misprediction penalty (four cycles in the IXP43X network processors instead of one in Intel StrongARM Architecture). This is mitigated by dynamic branch prediction.
- Larger load use delay (LUD) - LUDs arise from load-use dependencies. A load-use dependency gives rise to a LUD if the result of the load instruction cannot be made available by the pipeline in due time for the subsequent instruction. An optimizing compiler should find independent instructions to fill the slot following the load.
- Certain instructions incur a few extra cycles of delay on the IXP43X network processors as compared to Intel StrongARM processors (**LDM**, **STM**).
- Decode and register file lookups are spread out over two cycles in the IXP43X network processors, instead of one cycle in the Intel® *IXP42X Network Processors*.

3.10.2.1.2 Processor Pipeline Organization

The single-issue super-pipeline of the IXP43X network processors comprises of a main execution pipeline, MAC (Multiply-Accumulate) pipeline, and a memory access pipeline. These are shown in Figure 29, with the main execution pipeline shaded.

Figure 29. RISC Super-Pipeline

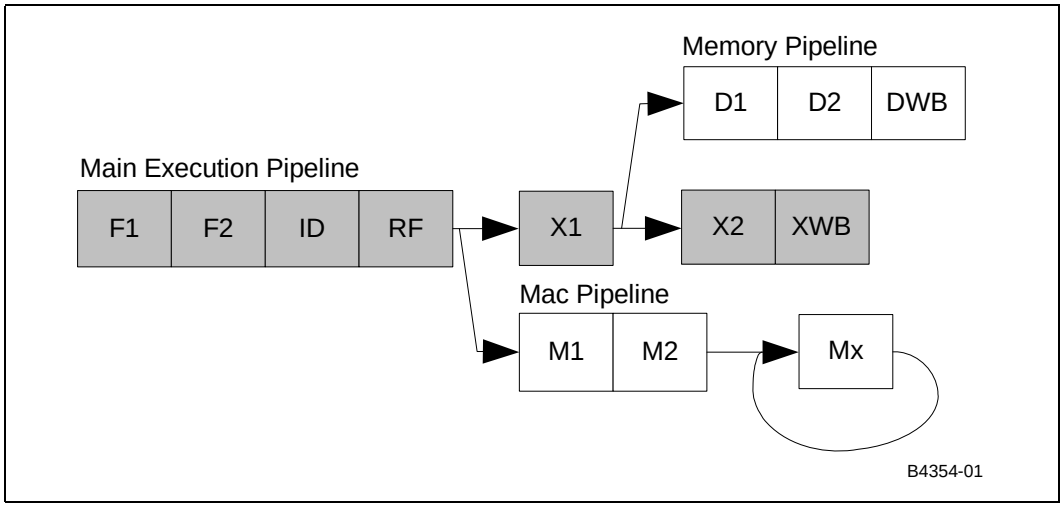


Table 94 gives a brief description of each pipe-stage.

Table 94. Pipelines and Pipe Stages

Pipe / Pipe State	Description	Covered In
Main Execution Pipeline	Handles data processing instructions	"Main Execution Pipeline" on page 178
IF1/IF2	Instruction Fetch	"
ID	Instruction Decode	"
RF	Register File / Operand Shifter	"
X1	ALU Execute	"
X2	State Execute	"
XWB	Write-back	"
Memory Pipeline	Handles load/store instructions	"Memory Pipeline" on page 179
D1/D2	Data Cache Access	"
DWB	Data cache writeback	"
MAC Pipeline	Handles all multiply instructions	"Multiply/Multiply Accumulate (MAC) Pipeline" on page 179
M1-M5	Multiplier stages	"
MWB (not shown)	MAC write-back - may occur during M2-M5	"

3.10.2.1.3 Out-of-Order Completion

Sequential consistency of instruction execution relates to two aspects: first, to the order in which the instructions are completed; and second, to the order in which memory is accessed due to load and store instructions. The IXP43X network processors preserve a weak processor consistency because instructions may complete out of order, provided that no data dependencies exist.

As instructions are issued in-order, the main execution pipeline, memory, and MAC pipelines are not lock-stepped, and, therefore, have various execution times. This means that instructions may finish out of program order. Short 'younger' instructions

may finish earlier than long ‘older’ ones. (The term ‘to finish’ is used here to indicate that the operation has been completed and the result has been written back to the register file.)

3.10.2.1.4 Register Scoreboarding

In certain situations, the pipeline should be stalled because of register dependencies between instructions. A register dependency occurs when a previous MAC or load instruction is about to modify a register value that has not been returned to the register file and the current instruction needs access to the same register. Only the destination of MAC operations and memory loads are scoreboarded. The destinations of ALU instructions are not scoreboarded.

If no register dependencies exist, the pipeline is not stalled. For example, if a load operation has missed the data cache, subsequent instructions that do not depend on the load may complete independently.

3.10.2.1.5 Use of Bypassing

The pipeline of the IXP43X network processors make extensive use of bypassing to minimize data hazards. Bypassing allows results forwarding from multiple sources, thus eliminating the need to stall the pipeline.

3.10.2.2 Instruction Flow Through the Pipeline

The pipeline of the IXP43X network processors issues a single instruction per clock cycle. Instruction execution begins at the F1 pipe stage and completes at the WB pipe stage.

Although a single instruction is issued per clock cycle, all three pipelines (MAC, memory, and main execution) are processing instructions simultaneously. If there are no data hazards, then each instruction completes independently of the others.

Each pipe stage takes a single clock cycle or machine cycle to perform its subtask with the exception of the MAC unit.

3.10.2.2.1 Intel® StrongARM* V5TE Instruction Execution

Figure 29 on page 176 uses arrows to show the possible flow of instructions in the pipeline. Instruction execution flows from the F1 pipe stage to the RF pipe stage. The RF pipe stage may issue a single instruction to the X1 pipe stage or the MAC unit (multiply instructions go to the MAC, as all others continue to X1). This means that M1 or X1 is idle.

All load/store instructions are routed to the memory pipeline after the effective addresses have been calculated in X1.

The Intel StrongARM V5TE bx (branch and exchange) instruction, that is used to branch between Intel StrongARM and thumb code, causes the entire pipeline to be flushed (The bx instruction is not dynamically predicted by the BTB). If the processor is in Thumb mode, then the ID pipe stage dynamically expands each Thumb instruction into a normal Intel StrongARM V5TE RISC instruction and execution resumes as usual.

3.10.2.2.2 Pipeline Stalls

The progress of an instruction can stall anywhere in the pipeline. Several pipe stages may stall for various reasons. It is important to understand when and how hazards occur in the pipeline of the IXP43X network processors. Performance degradation is significant if care is not taken to minimize pipeline stalls.

3.10.2.3 Main Execution Pipeline

3.10.2.3.1 F1 / F2 (Instruction Fetch) Pipe Stages

The job of the instruction fetch stages F1 and F2 is to present the next instruction to be executed to the ID stage. Several important functional units reside within the F1 and F2 stages, including:

- Branch Target Buffer (BTB)
- Instruction Fetch Unit (IFU)

An understanding of the BTB (See "Branch Target Buffer" on page 71) and IFU are important for performance considerations. A summary of operation is provided here so that the reader may understand its role in the F1 pipe stage.

- Branch Target Buffer (BTB)
  The BTB predicts the outcome of branch type instructions. Once a branch type instruction reaches the X1 pipe stage, its target address is known. If this address is different from the address that the BTB predicted, the pipeline is flushed, execution starts at the new target address, and the branch's history is updated in the BTB.
- Instruction Fetch Unit (IFU)
  The IFU is responsible for delivering instructions to the instruction decode (ID) pipe stage. One instruction word is delivered each cycle (if possible) to the ID. The instruction could come from one of two sources: instruction cache or fetch buffers.

3.10.2.3.2 ID (Instruction Decode) Pipe Stage

The ID pipe stage accepts an instruction word from the IFU and sends register decode information to the RF pipe stage. The ID is able to accept a new instruction word from the IFU on every clock cycle where there is no stall. The ID pipe stage is responsible for:

- General instruction decoding (extracting the op code, operand addresses, destination addresses and the offset).
- Detecting undefined instructions and generating an exception.
- Dynamic expansion of complex instructions into sequence of simple instructions. Complex instructions are defined as ones that take more than one clock cycle to issue, such as **LDM**, **STM**, and **SWP**.

3.10.2.3.3 RF (Register File / Shifter) Pipe Stage

The main function of the RF pipe stage is to read and write to the Register File Unit, or RFU. It provides source data to:

- EX for ALU operations
- MAC for multiply operations
- Data Cache for memory writes
- Coprocessor interface

The ID unit decodes the instruction and specifies the registers that are accessed in the RFU. Based upon this information, the RFU determines if it must stall the pipeline due to a register dependency. A register dependency occurs when a previous instruction is about to modify a register value that has not been returned to the RFU and the current instruction must access that same register. If no dependencies exist, the RFU selects the appropriate data from the register file and pass it to the next pipe stage. When a register dependency does exist, the RFU keeps track of the register that is unavailable and when the result is returned, the RFU stops stalling the pipe.

The Intel StrongARM architecture specifies that one of the operands for data processing instructions as the shifter operand, where a 32-bit shift is performed before it is used as an input to the ALU. This shifter is located in the second half of the RF pipe stage.

3.10.2.3.4 X1 (Execute) Pipe Stages

The X1 pipe stage performs the following functions:

- ALU calculation - the ALU performs arithmetic and logic operations, as required for data processing instructions and load/store index calculations.
- Determine conditional instruction execution - The instruction's condition is compared to the CPSR prior to execution of each instruction. Any instruction with a false condition is cancelled, and does not cause any architectural state changes, including modifications of registers, memory, and PSR.
- Branch target determination - If a branch is incorrectly predicted by the BTB, the X1 pipe stage flushes all of the instructions in the previous pipe stages and sends the branch target address to the BTB, that restarts the pipeline

3.10.2.3.5 X2 (Execute 2) Pipe Stages

The X2 pipe stage contains the Program Status Registers (PSRs). This pipe stage selects what is going to be written to the RFU in the WB cycle: PSRs (MRS instruction), ALU output, or other items.

3.10.2.3.6 WB (Write-Back)

When an instruction has reached the write-back stage, it is considered complete. Changes are written to the RFU.

3.10.2.4 Memory Pipeline

The memory pipeline consists of two stages, D1 and D2. The data cache unit, or DCU, consists of the data-cache array, mini-data cache, fill buffers, and write buffers. The memory pipeline handles load / store instructions.

3.10.2.4.1 D1 and D2 Pipe Stage

Operation begins in D1 after the X1 pipe stage has calculated the effective address for load/stores. The data cache and mini-data cache returns the destination data in the D2 pipe stage. Before data is returned in the D2 pipe stage, sign extension and byte alignment occurs for byte and half-word loads.

3.10.2.5 Multiply/Multiply Accumulate (MAC) Pipeline

The Multiply-Accumulate (MAC) unit executes the multiply and multiply-accumulate instructions supported by the IXP43X network processors. The MAC implements the 40-bit accumulator register (acc0) on the IXP43X network processors and handles the instructions that transfers its value to and from general purpose Intel StrongARM registers.

The following are important characteristics about the MAC:

- The MAC is not truly pipelined, as the processing of a single instruction may require use of the same data path resources for several cycles before a new instruction is accepted. The type of instruction and source arguments determines the number of cycles required.
- No more than two instructions can occupy the MAC pipeline concurrently.
- When the MAC is processing an instruction, another instruction may not enter M1 unless the original instruction completes in the next cycle.

- The MAC unit can operate on 16-bit packed signed data. This reduces register pressure and memory traffic size. Two 16-bit data items are loaded into a register with one LDR.
- The MAC can achieve throughput of one multiply per cycle when performing a 16-by-32-bit multiply.

3.10.2.5.1 Behavioral Description

The execution of the MAC unit starts at the beginning of the M1 pipe stage, where it receives two 32-bit source operands. Results are completed N cycles later (where N is dependent on the operand size) and returned to the register file. For more information on MAC instruction latencies, refer to "Instruction Latencies" on page 167.

An instruction that occupies the M1 or M2 pipe stages also occupies the X1 and X2 pipe stage, respectively. Each cycle, a MAC operation progresses for M1 to M5. A MAC operation may complete between M2-M5. If a MAC operation enters M3-M5, it is considered committed because it modifies architectural state regardless of subsequent events.

3.10.3 Basic Optimizations

This section outlines optimizations specific to the Intel StrongARM architecture. These optimizations have been modified to suit IXP43X network processors where needed.

3.10.3.1 Conditional Instructions

The architecture of the IXP43X network processors provides the ability to execute instructions conditionally. This feature combined with the ability of the IXP43X network processors' instructions, to modify the condition codes makes possible a wide array of optimizations.

3.10.3.1.1 Optimizing Condition Checks

The instructions in the IXP43X network processors can selectively modify the state of the condition codes. When generating code for if-else and loop conditions it is often beneficial to make use of this feature to set condition codes, thereby eliminating the need for a subsequent compare instruction.

Consider the C code segment:

```
if (a + b)
```

Code generated for the if condition without using an add instruction to set condition codes is:

```
;Assume r0 contains the value a, and r1 contains the value b
    add   r0,r0,r1
    cmp   r0, #0
```

But, code is optimized as follows making use of add instruction to set condition codes:

```
;Assume r0 contains the value a, and r1 contains the value b
    adds  r0,r0,r1
```

The instructions that increment or decrement the loop counter can also be used to modify the condition codes. This eliminates the need for a subsequent compare instruction. A conditional branch instruction can then be used to exit or continue with the next loop iteration.

Consider the following C code segment:

```
for (i = 10; i != 0; i--)
{
    do something;
}
```

The optimized code generated for the above code segment would look like:

```
L6:
.
.
    subs r3, r3, #1
    bne  .L6
```

It is also beneficial to rewrite loops whenever possible so as to make the loop exit conditions check against the value 0. For example, the code generated for the code segment below needs a compare instruction to check for the loop exit condition.

```
for (i = 0; i < 10; i++)
{
    do something;
}
```

If the loop were rewritten as follows, the code generated avoids using the compare instruction to check for the loop exit condition.

```
for (i = 9; i >= 0; i--)
{
    do something;
}
```

3.10.3.1.2 Optimizing Branches

Branches decrease application performance by indirectly causing pipeline stalls. Branch prediction improves the performance by lessening the delay inherent in fetching a new instruction stream. The number of branches that can accurately be predicted is limited by the size of the branch target buffer. Since the total number of branches executed in a program is relatively large compared to the size of the branch target buffer; it is often beneficial to minimize the number of branches in a program. Consider the following C code segment.

```
int foo(int a)

{

    if (a > 10)

        return 0;

    else

        return 1;

}
```

The code generated for the if-else portion of this code segment using branches is:

```
    cmp   r0, #10
    ble   L1
    mov   r0, #0
    b     L2
L1:
    mov   r0, #1
L2:
```

The code generated above takes three cycles to execute the else part and four cycles for the if-part assuming best case conditions and no branch misprediction penalties. In the case of the IXP43X network processors, a branch misprediction incurs a penalty of four cycles. If the branch is incorrectly predicted 50 percent of the time, and if we assume that both the if-part and the else-part are equally likely to be taken, on an average the code above takes 5.5 cycles to execute.

$$\left(\frac{50}{100} \times 4 + \frac{3+4}{2}\right) = 5.5 \qquad cycles$$

If you use the IXP43X network processors to execute instructions conditionally, the code generated for the above if-else statement is:

```
    cmp   r0, #10
    movgt r0, #0
    movle r0, #1
```

The above code segment would not incur any branch misprediction penalties and would take three cycles to execute assuming best case conditions. As is seen, using conditional instructions speeds up execution significantly. But, the use of conditional instructions should be carefully considered to ensure that it does improve performance. To decide when to use conditional instructions over branches consider the following hypothetical code segment:

```
if (cond)
    if_stmt
else
    else_stmt
```

Assume that we have the following data:

- $N1_B$..... Number of cycles to execute the if_stmt assuming the use of branch instructions
- $N2_B$.....Number of cycles to execute the else_stmt assuming the use of branch instructions
- P1.....Percentage of times the if_stmt is likely to be executed
- P2..... Percentage of times we are likely to incur a branch misprediction penalty
- $N1_C$... Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be true

- $N2_C$ ...Number of cycles to execute the if-else portion using conditional instructions assuming the if-condition to be false

Once we have the above data, use conditional instructions when:

$$\left(N1_C \times \frac{P1}{100}\right) + \left(N2_C \times \frac{100-P1}{100}\right) \leq \left(N1_B \times \frac{P1}{100}\right) + \left(N2_B \times \frac{100-P1}{100}\right) + \left(\frac{P2}{100} \times 4\right)$$

The following example illustrates a situation where we are better off using branches over conditional instructions. Consider the code sample shown below:

```
    cmp   r0, #0
    bne   L1
    add   r0, r0, #1
    add   r1, r1, #1
    add   r2, r2, #1
    add   r3, r3, #1
    add   r4, r4, #1
    b     L2
L1:
    sub   r0, r0, #1
    sub   r1, r1, #1
    sub   r2, r2, #1
    sub   r3, r3, #1
    sub   r4, r4, #1
L2:
```

In the above code sample, the cmp instruction takes 1 cycle to execute, the if-part takes 7 cycles to execute and the else-part takes 6 cycles to execute. If we were to change the code above so as to eliminate the branch instructions by making use of conditional instructions, the if-else part would always take 10 cycles to complete.

If we make the assumptions that both paths are equally likely to be taken and that branches are mis-predicted 50% of the time, the costs of using conditional execution Vs using branches is computed as follows:

Cost of using conditional instructions:

$$1 + \left(\frac{50}{100} \times 10\right) + \left(\frac{50}{100} \times 10\right) = 11 \quad cycles$$

Cost of using branches:

$$1 + \left(\frac{50}{100} \times 7\right) + \left(\frac{50}{100} \times 6\right) + \left(\frac{50}{100} \times 4\right) = 9.5 \quad cycles$$

As is seen, we get better performance by using branch instructions in the above scenario.

3.10.3.1.3 Optimizing Complex Expressions

Conditional instructions should also be used to improve the code generated for complex expressions such as the C shortcut evaluation feature. Consider the following C code segment:

```
int foo(int a, int b)
{
    if (a != 0 && b != 0)
        return 0;
    else
        return 1;
}
```

The optimized code for the if condition is:

```
    cmp   r0, #0
    cmpne r1, #0
```

Similarly, the code generated for the following C segment

```
int foo(int a, int b)
{
    if (a != 0 || b != 0)
        return 0;
    else
        return 1;
}
```

is:

```
    cmp   r0, #0
    cmpeq r1, #0
```

The use of conditional instructions in the above fashion improves performance by minimizing the number of branches, thereby minimizing the penalties caused by branch incorrect predictions. This approach also reduces the utilization of branch prediction resources.

3.10.3.2 Bit Field Manipulation

The shift and logical operations of the IXP43X network processors provide a useful way of manipulating bit fields. Bit field operations is optimized as follows:

```
;Set the bit number specified by r1 in register r0
    mov   r2, #1
    orr   r0, r0, r2, asl r1
;Clear the bit number specified by r1 in register r0
    mov   r2, #1
    bic   r0, r0, r2, asl r1
;Extract the bit-value of the bit number specified by r1 of the
;value in r0 storing the value in r0
    mov   r1, r0, asr r1
    and   r0, r1, #1
;Extract the higher order 8 bits of the value in r0 storing
;the result in r1
    mov   r1, r0, lsr #24
```

3.10.3.3 Optimizing the Use of Immediate Values

The **MOV** or **MVN** instruction of the IXP43X network processors should be used when loading an immediate (constant) value into a register. Refer to the *ARM* Architecture Reference Manual* for the set of immediate values that are used in a **MOV** or **MVN** instruction. It is also possible to generate a whole set of constant values using a combination of **MOV**, **MVN**, **ORR**, **BIC**, and **ADD** instructions. The **LDR** instruction has

the potential of incurring a cache miss in addition to polluting the data and instruction caches. The code samples below illustrate cases when a combination of the above instructions are used to set a register to a constant value:

```
;Set the value of r0 to 127
    mov   r0, #127
;Set the value of r0 to 0xfffffefb.
    mvn   r0, #260
;Set the value of r0 to 257
    mov   r0, #1
    orr   r0, r0, #256
;Set the value of r0 to 0x51f
    mov   r0, #0x1f
    orr   r0, r0, #0x500
;Set the value of r0 to 0xf100ffff
    mvn   r0, #0xff, 16
    bic   r0, r0, #0xe, 8
; Set the value of r0 to 0x12341234
    mov   r0, #0x8d, 30
    orr   r0, r0, #0x1, 20
    add   r0, r0, r0, LSL #16 ; shifter delay of 1 cycle
```

Note: It is possible to load any 32-bit value into a register using a sequence of four instructions.

3.10.3.4 Optimizing Integer Multiply and Divide

Multiplication by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Multiplication of R0 by 2^n
    mov   r0, r0, LSL #n
;Multiplication of R0 by 2^n+1
    add   r0, r0, r0, LSL #n
```

Multiplication by an integer constant that is expressed as $(2^n + 1) \cdot (2^m)$ can similarly be optimized as:

```
;Multiplication of r0 by an integer constant that is
;expressed as (2^n+1)*(2^m)
    add   r0, r0, r0, LSL #n
    mov   r0, r0, LSL #m
```

Note: The above optimization should only be used in cases where the multiply operation cannot be advanced far enough to prevent pipeline stalls.

Dividing an unsigned integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing an unsigned value by an integer constant
;that is represented as 2^n
    mov   r0, r0, LSR #n
```

Dividing a signed integer by an integer constant should be optimized to make use of the shift operation whenever possible.

```
;Dividing r0 containing a signed value by an integer constant
;that is represented as 2^n
    mov   r1, r0, ASR #31
    add   r0, r0, r1, LSR #(32 - n)
    mov   r0, r0, ASR #n
```

The add instruction would stall for 1 cycle. The stall is prevented by filling in another instruction before add.

3.10.3.5 Effective Use of Addressing Modes

The IXP43X network processors provide a variety of addressing modes that make indexing an array of objects highly efficient. For a detailed description of these addressing modes refer to the *ARM* Architecture Reference Manual*. The following code samples illustrate how various kinds of array operations are optimized to make use of these addressing modes:

```
;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the next word


    str   r1,[r0], #4


;Increment the contents of r0 to make it point to the next word
;and set the contents of the word pointed to the value contained
;in r1


    str   r1, [r0, #4]!


;Set the contents of the word pointed to by r0 to the value
;contained in r1 and make r0 point to the previous word


    str   r1,[r0], #-4


;Decrement the contents of r0 to make it point to the previous
;word and set the contents of the word pointed to the value
;contained in r1


    str   r1,[r0, #-4]!
```

3.10.4 Cache and Prefetch Optimizations

This section considers how to use the various cache memories in all their modes and then examines when and how to use prefetch to improve execution efficiencies.

3.10.4.1 Instruction Cache

IXP43X network processors has separate instruction and data caches. Only fetched instructions are held in the instruction cache even though both data and instructions may reside within the same memory space with each other. Functionally, the instruction cache is enabled or disabled. There is no performance benefit in not using the instruction cache. The exception is that code, that locks code into the instruction cache, must itself execute from non-cached memory.

3.10.4.1.1 Cache Miss Cost

The performance of the IXP43X network processors is highly dependent on reducing the cache miss rate.

Note: This cycle penalty becomes significant when the core is running much faster than external memory. Executing non-cached instructions severely curtails the processor's

performance in this case and it is very important to do everything possible to minimize cache misses.

For the IXP43X network processors, care must be taken to optimize code to have a maximum cache hit when accesses have been requested to the Expansion Bus Interface or the PCI Bus Controller. These design recommendations are due to the latency that is associated with accessing the PCI Bus Controller and Expansion Bus Controller. Retries is issued to the Intel XScale processor until the requested transaction is completed.

3.10.4.1.2 Round Robin Replacement Cache Policy

Both the data and the instruction caches use a round robin replacement policy to evict a cache line. The simple consequence of this is that at sometime every line is evicted, assuming a non-trivial program. The less obvious consequence is that predicting when and over which cache lines evictions take place is very difficult to predict. This information is gained by experimentation using performance profiling.

3.10.4.1.3 Code Placement to Reduce Cache Misses

Code placement can greatly affect cache misses. One way to view the cache is to think of it as 32 sets of 32 bytes, that span an address range of 1,024 bytes. When running, the code maps into 32 blocks modular 1,024 of cache space. Any sets, that are overused, thrashes the cache. The ideal situation is for the software tools to distribute the code on a temporal evenness over this space.

This is very difficult if not impossible for a compiler to do. Most of the input needed to best estimate how to distribute the code does come from profiling followed by compiler based two pass optimizations.

3.10.4.1.4 Locking Code into the Instruction Cache

One very important instruction cache feature is the ability to lock code into the instruction cache. Once locked into the instruction cache, the code is always available for fast execution. Another reason for locking critical code into cache is that with the round robin replacement policy, eventually the code is evicted, even if it is a very frequently executed function. Key code components to consider for locking are:

- Interrupt handlers
- Real time clock handlers
- OS critical code
- Time critical application code

The disadvantage to locking code into the cache is that it reduces the cache size for the rest of the program. How much code to lock is very application dependent and requires experimentation to optimize.

Code placed into the instruction cache should be aligned on a 1,024-byte boundary and placed sequentially together as tightly as possible so as not to waste precious memory space. Making the code sequential also insures even distribution across all cache ways. Though it is possible to choose randomly located functions for cache locking, this approach runs the risk of landing multiple cache ways in one set and few or none in another set. This distribution unevenness can lead to excessive thrashing of the Data and Mini Caches.

3.10.4.2 Data and Mini Cache

The IXP43X network processors allow the user to define memory regions whose cache policies are set by the user (see "Cacheability" on page 76). Supported policies and configurations are:

- Non Cacheable with no coalescing of memory writes.
- Non Cacheable with coalescing of memory writes.
- Mini-Data cache with write coalescing, read allocate, and write-back caching.
- Mini-Data cache with write coalescing, read allocate, and write-through caching.
- Mini-Data cache with write coalescing, read-write allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-back caching.
- Data cache with write coalescing, read allocate, and write-through caching.
- Data cache with write coalescing, read-write allocate, and write-back caching.

To support allocating variables to these various memory regions, the tool chain (compiler, assembler, linker and debugger), must implement named sections.

The performance of your application code depends on what cache policy you are using for data objects. A description of when to use a particular policy is described below.

The IXP43X network processors allow dynamic modification of the cache policies at run time, but, the operation requires considerable processing time and therefore should not be used by applications.

If the application is running under an OS, then the OS may restrict you from using certain cache policies.

3.10.4.2.1 Non-Cacheable Regions

It is recommended that non-cache memory (X=0, C=0, and B=0) be used only if necessary as is often necessary for I/O devices. Accessing non-cacheable memory is likely to cause the processor to stall frequently due to the long latency of memory reads.

3.10.4.2.2 Write-Through and Write-Back Cached Memory Regions

Write through memory regions generate more data traffic on the bus. Therefore is not recommended that the write-through policy be used. The write back policy must be used whenever possible.

But, in a multiprocessor environment it is necessary to use a write through policy if data is shared across multiple processors. In such a situation all shared memory regions should use write through policy. Memory regions that are private to a particular processor should use the write back policy.

3.10.4.2.3 Read Allocate and Read-Write Allocate Memory Regions

Most of the regular data and the stack for your application should be allocated to a read-write allocate region. It is expected that you are writing and reading from them often.

Data that is write only (or data that is written to and subsequently not used for a long time) should be placed in a read allocate region. Under the read-allocate policy if a cache write miss occurs a new cache line is not allocated, and hence does not evict critical data from the data cache.

3.10.4.2.4 Creating On-Chip RAM

Part of the Data cache is converted into fast on chip RAM. Access to objects in the on-chip RAM does not incur cache miss penalties, thereby reducing the number of processor stalls. Application performance is improved by converting a part of the cache into on chip RAM and allocating frequently allocated variables to it. Due to the round-robin replacement policy of the IXP43X network processors, all data is eventually evicted. Therefore to prevent critical or frequently used data from being evicted it should be allocated to on-chip RAM.

The following variables are good candidates for allocating to the on-chip RAM:

- Frequently used global data used for storing context for context switching.
- Global variables that are accessed in time critical functions such as interrupt service routines.

The on-chip RAM is created by locking a memory region into the Data cache (see "Reconfiguring the Data Cache as Data RAM" on page 80 for more details).

When creating the on-chip RAM, care must be taken to ensure that all sets in the on-chip RAM area of the Data cache have approximately the same number of ways locked, otherwise some sets has more ways locked than the others. This uneven allocation increases the level of thrashing in some sets and leave other sets under utilized.

For example, consider three arrays arr1, arr2 and arr3 of size 64 bytes each that are being allocated to the on-chip RAM and assume that the address of arr1 is 0, address of arr2 is 1024, and the address of arr3 is 2048. All three arrays is within the same sets, that is, set0 and set1, as a result three ways in both sets set0 and set1, is locked, leaving 29 ways for use by other variables.

This is overcome by allocating on-chip RAM data in sequential order. In the above example allocating arr2 to address 64 and arr3 to address 128, allows the three arrays to use only 1 way in sets 0 through 5.

3.10.4.2.5 Mini-Data Cache

The mini-data cache is best used for data structures, that have short temporal lives, and/or cover vast amounts of data space. Addressing these types of data spaces from the Data cache would corrupt much if not all of the Data cache by evicting valuable data. Eviction of valuable data reduces performance. Placing this data instead in Mini-data cache memory region would prevent data cache corruption when providing the benefits of cached accesses.

A prime example of using the mini-data cache would be for caching the procedure call stack. The stack is allocated to the mini-data cache so that its use does not trash the main dcache. This would keep local variables from global data.

Following are examples of data that could be assigned to mini-dcache:

- The stack space of a frequently occurring interrupt, the stack is used only during the duration of the interrupt, and is usually very small.
- Video buffers, these are usual large and can occupy the whole cache.

Over use of the mini-data cache thrashes the cache. This is easy to do because the mini-data cache only has two ways per set. For example, a loop and uses a simple statement such as:

```
for (i=0; I< IMAX; i++)
{
    A[i] = B[i] + C[i];
}
```

Where A, B, and C reside in a mini-data cache memory region and each is array is aligned on a 1-K boundary quickly thrashes the cache.

3.10.4.2.6 Data Alignment

Cache lines begin on 32-byte address boundaries. To maximize cache line use and minimize cache pollution, data structures should be aligned on 32-byte boundaries and sized to multiple cache line sizes. Aligning data structures on cache address boundaries simplifies later addition of prefetch instructions to optimize performance.

Not aligning data on cache lines has the disadvantage of moving the prefetch address correspondingly to the misalignment. Consider the following example:

```
struct {
    long ia;
    long ib;
    long ic;
    long id;
} tdata[IMAX];

for (i=0, i<IMAX; i++)
{
    PREFETCH(tdata[i+1]);
    tdata[i].ia = tdata[i].ib + tdata[i].ic _tdata[i].id];
    ....
    tdata[i].id = 0;
}
```

In this case if tdata[] is not aligned to a cache line, then the prefetch using the address of tdata[i+1].ia may not include element id. If the array is aligned on a cache line + 12 bytes, then the prefetch would halve to be placed on &tdata[i+1].id.

If the structure is not sized to a multiple of the cache line size, then the prefetch address must be advanced appropriately and requires extra prefetch instructions. Consider the following example:

```
struct {
    long ia;
    long ib;
    long ic;
    long id;
    long ie;
} tdata[IMAX];

ADDRESS preadd = tdata

for (i=0, i<IMAX; i++)
{
    PREFETCH(predata+=16);
    tdata[I].ia = tdata[I].ib + tdata[I].ic _tdata[I].id] +
    tdata[I].ie;
    ....
    tdata[I].ie = 0;
}
```

In this case, the prefetch address is advanced by size of half a cache line and every other prefetch instruction is ignored. Further, an additional register is required to track the next prefetch address.

Generally, not aligning and sizing data adds extra computational overhead.

Additional prefetch considerations are discussed in greater detail in following sections.

3.10.4.2.7 Literal Pools

The IXP43X network processors do not have a single instruction that can move all literals (a constant or address) to a register. One technique to load registers with literals in the IXP43X network processors is by loading the literal from a memory location that has been initialized with the constant or address. These blocks of constants are referred to as literal pools. See “Basic Optimizations” on page 180 for more information on how to do this.

It is better to place all the literals together in a pool of memory known a literal pool. These data blocks are located in the text or code address space so that they are loaded using PC relative addressing. But, references to the literal pool area load the data into the data cache instead of the instruction cache. Therefore it is possible that the literal is present in both the data and instruction caches, resulting in waste of space.

For maximum efficiency, the compiler should align all literal pools on cache boundaries and size each pool to a multiple of 32 bytes (the size of a cache line). One additional optimization would be group highly used literal pool references into the same cache line. The advantage is that once one of the literals has been loaded, the other seven is available immediately from the data cache.

3.10.4.3 Cache Considerations

3.10.4.3.1 Cache Conflicts, Pollution, and Pressure

Cache pollution occurs when unused data is loaded in the cache and cache pressure occurs when data that is not temporal to the current process is loaded into the cache.

3.10.4.3.2 Memory Page Thrashing

Memory page thrashing occurs because of the nature of DDRI SDRAM. DDRI SDRAMs are typically divided into four banks. Each bank can have one selected page where a page address size for current memory components is often defined as 4 k. Memory lookup time or latency time for a selected page address is currently two to three bus clocks. Thrashing occurs when subsequent memory accesses within the same memory bank access various pages. The memory page change adds three to four bus clock cycles to memory latency. This added delay extends the prefetch distance correspondingly making it more difficult to hide memory access latencies. This type of thrashing is resolved by placing the conflicting data structures into various memory banks or by paralleling the data structures such that the data resides within the same memory page. It is also extremely important to insure that instruction and data sections are in various memory banks, or they continually trashes the memory page selection.

3.10.4.4 Prefetch Considerations

The IXP43X network processors have a true prefetch load instruction (PLD). The purpose of this instruction is to preload data into the data and mini-data caches. Data prefetching allows hiding of memory transfer latency as the processor continues to execute instructions. The prefetch is important to compiler and assembly code because judicious use of the prefetch instruction can enormously improve throughput performance of the IXP43X network processors. Data prefetch is applied not only to

loops but also to any data references within a block of code. Prefetch also applies to data writing when the memory type is enabled as write-allocate.

The prefetch load instruction of the IXP43X network processors is a true prefetch instruction because the load destination is the data or mini-data cache and not a register. Compilers for processors that have data caches, but do not support prefetch, sometimes use a load instruction to preload the data cache. This technique has the disadvantages of using a register to load data and requiring additional registers for subsequent preloads and thus increasing register pressure. By contrast, the prefetch is used to reduce register pressure instead of increasing it.

The prefetch load is a hint instruction and does not guarantee that the data is loaded. Whenever the load would cause a fault or a table walk, then the processor ignores the prefetch instruction, the fault or table walk, and continue processing the next instruction. This is particularly advantageous in the case where a linked list or recursive data structure is terminated by a NULL pointer. Prefetching the NULL pointer does not fault program flow.

3.10.4.4.1 Prefetch Loop Limitations

It is not always advantages to add prefetch to a loop. Loop characteristics that limit the use value of prefetch are discussed below.

3.10.4.4.2 Compute versus Data Bus Bound

At the extreme, a loop, that is data bus bound, does not benefit from prefetch because all the system resources to transfer data are quickly allocated and there are no instructions that can profitably be executed. On the other end of the scale, compute bound loops allow complete hiding of all data transfer latencies.

3.10.4.4.3 Low Number of Iterations

Loops with very low iteration counts has the advantages of prefetch completely mitigated. A loop with a small fixed number of iterations is faster if the loop is completely unrolled rather than trying to schedule prefetch instructions.

3.10.4.4.4 Bandwidth Limitations

Overuse of prefetches can usurp resources and degrade performance. This happens because once the bus traffic requests exceed the system resource capacity, the processor stalls. The data transfer resources for the IXP43X network processors are:

- Four fill buffers
- Four pending buffers
- Eight half-cache line write buffer

DDRI SDRAM resources are typically:

- Four memory banks
- One page buffer per bank referencing a 4K address range
- Four transfer request buffers

Consider how these resources work together. A fill buffer is allocated for each cache read miss. A fill buffer is also allocated each cache write miss if the memory space is write allocate along with a pending buffer. A subsequent read to the same cache line does not require a new fill buffer, but does require a pending buffer and a subsequent write also requires a new pending buffer. A fill buffer is also allocated for each read to a non-cached memory and a write buffer is needed for each memory write to non-cached

memory that is non-coalescing. Consequently, a **STM** instruction listing eight registers and referencing non-cached memory uses eight write buffers assuming they don't coalesce and two write buffers if they do coalesce. A cache eviction requires a write buffer for each dirty bit set in the cache line. The prefetch instruction requires a fill buffer for each cache line and 0, 1, or 2 write buffers for an eviction.

When adding prefetch instructions, caution must be asserted to insure that the combination of prefetch and instruction bus requests do not exceed the system resource capacity described above or performance is degraded instead of improved. The important points are to spread prefetch operations over calculations so as to allow bus traffic to free flow and to minimize the number of necessary prefetches.

3.10.4.4.5 Cache Memory Considerations

Stride, the way data structures are walked through, can affect the temporal quality of the data and reduce or increase cache conflicts. The data cache and mini-data caches for the IXP43X network processors have 32 sets of 32 bytes. This means that each cache line in a set is on a modular 1-K-address boundary. The caution is to choose data structure sizes and stride requirements that do not overwhelm a given set causing conflicts and increased register pressure. Register pressure is increased because additional registers are required to track prefetch addresses. The effects are affected by rearranging data structure components to use more parallel access to search and compare elements. Similarly rearranging sections of data structures so that sections often written fit in the same half cache line, 16 bytes for the IXP43X network processors, can reduce cache eviction write-backs. On a global scale, techniques such as array merging can enhance the spatial locality of the data.

As an example of array merging, consider the following code:

```
int a_array[NMAX];
int b_array[NMAX];
int ix;

for (i=0; i<NMAX]; i++)
{
    ix = b[i];
    if (a[i] != 0)
        ix = a[i];
    do_other calculations;
}
```

In the above code, data is read from both arrays a and b, but a and b are not spatially close. Array merging can place a and b specially close.

```
struct {
    int a;
    int b;
} c_arrays;

int ix;

for (i=0; i<NMAX]; i++)
{
    ix = c[i].b;
    if (c[i].a != 0)
        ix = c[i].a;
    do_other_calculations;
}
```

As an example of rearranging often written to sections in a structure, consider the code sample:

```
struct employee {
    struct employee *prev;
    struct employee *next;
    float Year2DatePay;
    float Year2DateTax;
    int ssno;
    int empid;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};
```

In the data structure shown above, the fields Year2DatePay, Year2DateTax, Year2Date401KDed, and Year2DateOtherDed are likely to change with each pay check. The remaining fields change very rarely. If the fields are laid out as shown above, assuming that the structure is aligned on a 32-byte boundary, modifications to the Year2Date fields is likely to use two write buffers when the data is written out to memory. But, we can restrict the number of write buffers that are commonly used to 1 by rearranging the fields in the above data structure as shown below:

```
struct employee {
    struct employee *prev;
    struct employee *next;
    int ssno;
    int empid;
    float Year2DatePay;
    float Year2DateTax;
    float Year2Date401KDed;
    float Year2DateOtherDed;
};
```

3.10.4.4.6 Cache Blocking

Cache blocking techniques, such as strip-mining, are used to improve temporal locality of the data. Given a large data set that is reused across multiple passes of a loop, data blocking divides the data into smaller chunks that are loaded into the cache during the first loop and then be available for processing on subsequence loops thus minimizing cache misses and reducing bus traffic.

As an example of cache blocking consider the following code:

```
for(i=0; i<10000; i++)
    for(j=0; j<10000; j++)
        for(k=0; k<10000; k++)
            C[j][k] += A[i][k] * B[j][i];
```

The variable A[i][k] is completely reused. Accessing C[j][k] in the j and k loops can displace A[i][j] from the cache. Using blocking the code becomes:

```
for(i=0; i<10000; i++)
    for(j1=0; j<100; j++)
        for(k1=0; k<100; k++)
            for(j2=0; j<100; j++)
                for(k2=0; k<100; k++)
                {
                    j = j1 * 100 + j2;
                    k = k1 * 100 + k2;
                    C[j][k] += A[i][k] * B[j][i];
                }
```

3.10.4.4.7 Prefetch Unrolling

When iterating through a loop, data transfer latency are hidden by prefetching ahead one or more iterations. The solution incurs an unwanted side affect that the final interactions of a loop loads useless data into the cache, polluting the cache, increasing bus traffic and possibly evicting valuable temporal data. This problem is resolved by prefetch unrolling. For example consider:

```
for(i=0; i<NMAX; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
```

Interactions i-1 and i, prefetches superfluous data. The problem is avoid by unrolling the end of the loop.

```
for(i=0; i<NMAX-2; i++)
{
    prefetch(data[i+2]);
    sum += data[i];
}
sum += data[NMAX-2];
sum += data[NMAX-1];
```

Unfortunately, prefetch loop unrolling does not work on loops with indeterminate iterations.

3.10.4.4.8 Pointer Prefetch

Not all looping constructs contain induction variables. But, prefetching techniques can still be applied. Consider the following linked list traversal example:

```
while(p) {
    do_something(p->data);
    p = p->next;
}
```

The pointer variable p becomes a pseudo induction variable and the data pointed to by p->next is pre-fetched to reduce data transfer latency for the next iteration of the loop. Linked lists should be converted to arrays as much as possible.

```
while(p) {
    prefetch(p->next);
    do_something(p->data);
    p = p->next;
}
```

Recursive data structure traversal is another construct where prefetching is applied. This is similar to linked list traversal. Consider the following pre-order traversal of a binary tree:

```
preorder(treeNode *t) {
    if(t) {
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

The pointer variable **t** becomes the pseudo induction variable in a recursive loop. The data structures pointed to by the values t->left and t->right are pre-fetched for the next iteration of the loop.

```
preorder(treeNode *t) {
    if(t) {
        prefetch(t->right);
        prefetch(t->left);
        process(t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

Note: The order reversal of the prefetches in relationship to the usage. If there is a cache conflict and data is evicted from the cache then only the data from the first prefetch is lost.

3.10.4.4.9 Loop Interchange

As mentioned earlier, the sequence where data is accessed affects cache thrashing. Usually, it is best to access data in a contiguous spatially address range. But, arrays of data may have been laid out such that indexed elements are not physically next to each other. Consider the following C code that places array elements in row major order.

```
for(j=0; j<NMAX; j++)
    for(i=0; i<NMAX; i++)
    {
        prefetch(A[i+1][j]);
        sum += A[i][j];
    }
```

In the above example, A[i][j] and A[i+1][j] are not sequentially next to each other. This situation causes an increase in bus traffic when prefetching loop data. In some cases where the loop mathematics are unaffected, the problem is resolved by induction variable interchange. The above examples becomes:

```
for(i=0; i<NMAX; i++)
    for(j=0; j<NMAX; j++)
    {
        prefetch(A[i][j+1]);
        sum += A[i][j];
    }
```

3.10.4.4.10 Loop Fusion

Loop fusion is a process of combining multiple loops, that reuse the same data, in to one loop. The advantage of this is that the reused data is immediately accessible from the data cache. Consider the following example:

```
for(i=0; i<NMAX; i++)
{
    prefetch(A[i+1], c[i+1], c[i+1]);
    A[i] = b[i] + c[i];
}
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], c[i+1], A[i+1]);
    D[i] = A[i] + c[i];
}
```

The second loop reuses the data elements A[i] and c[i]. Fusing the loops together produces:

```
for(i=0; i<NMAX; i++)
{
    prefetch(D[i+1], A[i+1], c[i+1], b[i+1]);
    ai = b[i] + c[i];
    A[i] = ai;
    D[i] = ai + c[i];
}
```

3.10.4.4.11 Prefetch to Reduce Register Pressure

Pre-fetch is used to reduce register pressure. When data is needed for an operation, the load is scheduled far enough in advance to hide the load latency. But, the load ties up the receiving register until the data is used. For example:

```
    ldr   r2, [r0]
; Process code { not yet cached latency > 60 core clocks }
    add   r1, r1, r2
```

In the above case, r2 is unavailable for processing until the add statement. Prefetching the data load frees the register for use. The example code becomes:

```
    pld   [r0] ;prefetch the data keeping r2 available for use
; Process code
    ldr   r2, [r0]
; Process code { ldr result latency is 3 core clocks }
    add   r1, r1, r2
```

With the added prefetch, register r2 is used for other operations until almost just before it is needed.

3.10.5 Instruction Scheduling

This section discusses instruction scheduling optimizations. Instruction scheduling refers to the rearrangement of a sequence of instructions for the purpose of minimizing pipeline stalls. Reducing the number of pipeline stalls improves application performance. When making this rearrangement, care should be taken to ensure that the rearranged sequence of instructions has the same effect as the original sequence of instructions.

3.10.5.1 Scheduling Loads

On the IXP43X network processors, an **LDR** instruction has a result latency of three cycles assuming the data being loaded is in the data cache. If the instruction after the **LDR** must use the result of the load, then it would stall for 2 cycles. If possible, the instructions surrounding the **LDR** instruction should be rearranged.

to avoid this stall. Consider the following example:

```
    add   r1, r2, r3

    ldr   r0, [r5]

    add   r6, r0, r1

    sub   r8, r2, r3

    mul   r9, r2, r3
```

In the code shown above, the **ADD** instruction following the **LDR** would stall for two cycles because it uses the result of the load. The code is rearranged as follows to prevent the stalls:

```
ldr   r0, [r5]

add   r1, r2, r3

sub   r8, r2, r3

add   r6, r0, r1

mul   r9, r2, r3
```

Note: This rearrangement may not be always possible. Consider the following example:

```
cmp   r1, #0

addne r4, r5, #4

subeq r4, r5, #4

ldr   r0, [r4]

cmp   r0, #10
```

In the example above, the **LDR** instruction cannot be moved before the **ADDNE** or the **SUBEQ** instructions because the **LDR** instruction depends on the result of these instructions. Rewrite the above code to make it run faster at the expense of increasing code size:

```
cmp   r1, #0

ldrne r0, [r5, #4]

ldreq r0, [r5, #-4]

addne r4, r5, #4

subeq r4, r5, #4

cmp   r0, #10
```

The optimized code takes six cycles to execute compared to the seven cycles taken by the unoptimized version.

The result latency for an **LDR** instruction is significantly higher if the data being loaded is not in the data cache. To minimize the number of pipeline stalls in such a situation the **LDR** instruction should be moved as far away as possible from the instruction that uses result of the load.

Note: This may at times cause certain register values to be spilled to memory due to the increase in register pressure. In such cases, use a preload instruction or a preload hint to ensure that the data access in the **LDR** instruction hits the cache when it executes. A preload hint should be used in cases where we cannot be sure whether the load instruction would be executed. A preload instruction should be used in cases where we are sure that the load instruction would be executed. Consider the following code sample:

```
; all other registers are in use
    sub   r1, r6, r7
    mul   r3,r6, r2
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    add   r0,r4, r5
    ldr   r6, [r0]
    add   r8, r6, r8
    add   r8, r8, #4
    orr   r8,r8, #0xf
; The value in register r6 is not used after this
```

In the code sample above, the **ADD** and the **LDR** instruction are moved before the **MOV** instruction.

Note: This would prevent pipeline stalls if the load hits the data cache. But, if the load is likely to miss the data cache, move the **LDR** instruction so that it executes as early as possible - before the **SUB** instruction. Moving the **LDR** instruction before the **SUB** instruction would change the program semantics. It is possible to move the **ADD** and the **LDR** instructions before the **SUB** instruction if we allow the contents of the register r6 to be spilled and restored from the stack as shown below:

```
; all other registers are in use
    str   r6,[sp, #-4]!
    add   r0,r4,r5
    ldr   r6, [r0]
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    add   r8, r6, r8
    ldr   r6, [sp], #4
    add   r8, r8, #4
    orr   r8,r8, #0xf
    sub   r1, r6, r7
    mul   r3,r6, r2
; The value in register r6 is not used after this
```

As is seen above, the contents of the register r6 have been spilled to the stack and subsequently loaded back to the register r6 to retain the program semantics. Another way to optimize the code above is with the use of the preload instruction as shown below:

```
; all other registers are in use
    add   r0,r4, r5
    pld   [r0]
    sub   r1, r6, r7
    mul   r3,r6, r2
    mov   r2, r2, LSL #2
    orr   r9, r9, #0xf
    ldr   r6, [r0]
    add   r8, r6, r8
    add   r8, r8, #4
    orr   r8,r8, #0xf
; The value in register r6 is not used after this
```

The IXP43X network processors have four fill-buffers that are used to fetch data from external memory when a data-cache miss occurs. The IXP43X network processors stall when all fill buffers are in use. This happens when more than 4 loads are outstanding and are being fetched from memory. As a result, the code written should ensure that no more than four loads are outstanding at the same time. For example, the number of loads issued sequentially should not exceed four.

Note: A preload instruction may cause a fill buffer to be used. As a result, the number of preload instructions outstanding should also be considered to arrive at the number of loads that are outstanding.

Similarly, the number of write buffers also limits the number of successive writes that are issued before the processor stalls. No more than eight stores are issued.

Note: If the data caches are using the write-allocate with write-back policy, then a load operation may cause stores to the external memory if the read operation evicts a cache line that is dirty (modified). The number of sequential stores is limited by this fact.

3.10.5.1.1 Scheduling Load and Store Double (LDRD/STRD)

The IXP43X network processors introduce two new double word instructions: **LDRD** and **STRD**. **LDRD** loads 64 bits of data from an effective address into two consecutive registers, conversely, **STRD** stores 64 bits from two consecutive registers to an effective address. There are two important restrictions on how these instructions are used:

- The effective address must be aligned on an 8-byte boundary
- The specified register must be even (r0, r2, and so on.).

If this situation occurs, using **LDRD**/**STRD** instead of **LDM**/**STM** to do the same thing is more efficient because **LDRD**/**STRD** issues in only one/two clock cycle(s), as opposed to **LDM**/**STM** that issues in four clock cycles. Avoid **LDRD**s targeting R12; this incurs an extra cycle of issue latency.

The **LDRD** instruction has a result latency of 3 or 4 cycles depending on the destination register being accessed (assuming the data being loaded is in the data cache).

```
    add   r6, r7, r8
    sub   r5, r6, r9
; The following ldrd instruction would load values
; into registers r0 and r1
    ldrd r0, [r3]
    orr   r8, r1, #0xf
    mul   r7, r0, r7
```

In the code example above, the **ORR** instruction would stall for three cycles because of the four cycle result latency for the second destination register of an **LDRD** instruction. The code shown above is rearranged to remove the pipeline stalls:

```
; The following ldrd instruction would load values
; into registers r0 and r1
    ldrd  r0, [r3]
    add   r6, r7, r8
    sub   r5, r6, r9
    mul   r7, r0, r7
    orr   r8, r1, #0xf
```

Any memory operation following a **LDRD** instruction (**LDR**, **LDRD**, **STR** and so on) would stall for 1 cycle.

```
; The str instruction below would stall for 1 cycle
    ldrd  r0, [r3]
    str   r4, [r5]
```

3.10.5.1.2 Scheduling Load and Store Multiple (LDM/STM)

LDM and **STM** instructions have an issue latency of 2-20 cycles depending on the number of registers being loaded or stored. The issue latency is typically two cycles plus an additional cycle for each of the registers being loaded or stored assuming a

data cache hit. The instruction following an LDM would stall whether or not this instruction depends on the results of the load. A **LDRD** or **STRD** instruction does not suffer from this drawback (except when followed by a memory operation) and should be used where possible. Consider the task of adding two 64-bit integer values. Assume that the addresses of these values are aligned on an 8-byte boundary. This is achieved using the **LDM** instructions as shown below:

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
    ldm   r0, {r2, r3}
    ldm   r1, {r4, r5}
    adds  r0, r2, r4
    adc   r1,r3, r5
```

If the code were written as shown above, assuming all the accesses hit the cache, the code would take 11 cycles to complete. Rewriting the code as shown below using **LDRD** instruction would take only seven cycles to complete. The performance would increase further if we can fill in other instructions after **LDRD** to reduce the stalls due to the result latencies of the **LDRD** instructions.

```
; r0 contains the address of the value being copied
; r1 contains the address of the destination location
    ldrd  r2, [r0]
    ldrd  r4, [r1]
    adds  r0, r2, r4
    adc   r1,r3, r5
```

Similarly, the code sequence shown below takes five cycles to complete.

```
    stm   r0, {r2, r3}
    add   r1, r1, #1
```

The alternative version that is shown below would only take three cycles to complete.

```
    strd  r2, [r0]
    add   r1, r1, #1
```

3.10.5.2 Scheduling Data Processing Instructions

Most data processing instructions for the IXP43X network processors has a result latency of one cycle. This means that the current instruction is able to use the result from the previous data processing instruction. The result latency is two cycles if the current instruction must use the result of the previous data processing instruction for a shift by immediate. As a result, the following code segment would incur a one-cycle stall for the MOV instruction:

```
    sub   r6, r7, r8
    add   r1, r2, r3
    mov   r4, r1, LSL #2
```

The code above is rearranged as follows to remove the one-cycle stall:

```
    add   r1, r2, r3
    sub   r6, r7, r8
    mov   r4, r1, LSL #2
```

All data processing instructions incur a two cycle issue penalty and a two-cycle result penalty when the shifter operand is a shift/rotate by a register or shifter operand is RRX. Since the next instruction would always incur a 2 cycle issue penalty, there is no way to avoid such a stall except by re-writing the assembler instruction. Consider the following segment of code:

```
mov   r3, #10
mul   r4, r2, r3
add   r5, r6, r2, LSL r3
sub   r7, r8, r2
```

The subtract instruction would incur a one-cycle stall due to the issue latency of the add instruction as the shifter operand is shift by a register. The issue latency is avoided by changing the code as follows:

```
mov   r3, #10
mul   r4, r2, r3
add   r5, r6, r2, LSL #10
sub   r7, r8, r2
```

3.10.5.3 Scheduling Multiply Instructions

Multiply instructions can cause pipeline stalls due to resource conflicts or result latencies. The following code segment would incur a stall of zero to three cycles depending on the values in registers r1, r2, r4 and r5 due to resource conflicts.

```
mul   r0, r1, r2
mul   r3, r4, r5
```

The following code segment would incur a stall of one to three cycles, depending on the values in registers r1 and r2 due to result latency.

```
mul   r0, r1, r2
mov   r4, r0
```

Note: A multiply instruction that sets the condition codes blocks the whole pipeline. A four-cycle multiply operation that sets the condition codes behaves the same as a 4 cycle issue operation. Consider the following code segment:

```
muls  r0, r1, r2
add   r3, r3, #1
sub   r4, r4, #1
sub   r5, r5, #1
```

The add operation above would stall for three cycles if the multiply takes four cycles to complete. It is better to replace the code segment above with the following sequence:

```
mul   r0, r1, r2
add   r3, r3, #1
sub   r4, r4, #1
sub   r5, r5, #1
cmp   r0, #0
```

Refer to "Instruction Latencies" on page 167 to get the instruction latencies for various multiply instructions. The multiply instructions should be scheduled taking into consideration these instruction latencies.

3.10.5.4 Scheduling SWP and SWPB Instructions

The **SWP** and **SWPB** instructions have a five-cycle issue latency. As a result of this latency, the instruction following the **SWP**/**SWPB** instruction would stall for 4 cycles. **SWP** and **SWPB** instructions should, therefore, be used only where absolutely needed.

For example, the following code is used to swap the contents of two memory locations:

```
; Swap the contents of memory locations pointed to by r0 and r1
    ldr   r2, [r0]
    swp   r2, [r1]
    str   r2, [r1]
```

The code above takes nine cycles to complete. The rewritten code below, takes six cycles to execute:

```
; Swap the contents of memory locations pointed to by r0 and r1
    ldr   r2, [r0]
    ldr   r3, [r1]
    str   r2, [r1]
    str   r3, [r0]
```

3.10.5.5 Scheduling the MRA and MAR Instructions (MRRC/MCRR)

The **MRA** (**MRRC**) instruction has an issue latency of one cycle, a result latency of two or three cycles depending on the destination register value being accessed and a resource latency of two cycles.

Consider the code sample:

```
    mra   r6, r7, acc0
    mra   r8, r9, acc0
    add   r1, r1, #1
```

The code shown above would incur a one-cycle stall due to the two-cycle resource latency of an **MRA** instruction. The code is rearranged as shown below to prevent this stall.

```
    mra   r6, r7, acc0
    add   r1, r1, #1
    mra   r8, r9, acc0
```

Similarly, the code shown below would incur a two-cycle penalty due to the three-cycle result latency for the second destination register.

```
    mra   r6, r7, acc0
    mov   r1, r7
    mov   r0, r6
    add   r2, r2, #1
```

The stalls incurred by the code shown above is prevented by rearranging the code:

```
    mra   r6, r7, acc0
    add   r2, r2, #1
    mov   r0, r6
    mov   r1, r7
```

The **MAR** (**MCRR**) instruction has an issue latency, a result latency, and a resource latency of two cycles. Due to the two-cycle issue latency, the pipeline would always stall for one cycle following a **MAR** instruction. The use of the **MAR** instruction should, therefore, be used only where absolutely necessary.

3.10.5.6 Scheduling the MIA and MIAPH Instructions

The **MIA** instruction has an issue latency of one cycle. The result and resource latency can vary from one to three cycles depending on the values in the source register.

Consider the following code sample:

```
mia   acc0, r2, r3
mia   acc0, r4, r5
```

The second **MIA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle resource latency.

Similarly, consider the following code sample:

```
mia   acc0, r2, r3
mra   r4, r5, acc0
```

The **MRA** instruction above can stall from zero to two cycles depending on the values in the registers r2 and r3 due to the one-to-three-cycle result latency.

The **MIAPH** instruction has an issue latency of one cycle, result latency of two cycles and a resource latency of two cycles.

Consider the code sample shown below:

```
add   r1, r2, r3
miaph acc0, r3, r4
miaph acc0, r5, r6
mra   r6, r7, acc0
sub   r8, r3, r4
```

The second **MIAPH** instruction would stall for one-cycle due to a two-cycle resource latency. The **MRA** instruction would stall for one-cycle due to a two-cycle result latency. These stalls is avoided by rearranging the code as follows:

```
miaph acc0, r3, r4
add   r1, r2, r3
miaph acc0, r5, r6
sub   r8, r3, r4
mra   r6, r7, acc0
```

3.10.5.7 Scheduling MRS and MSR Instructions

The **MRS** instruction has an issue latency of one cycle and a result latency of two cycles. The **MSR** instruction has an issue latency of 2 cycles (6 if updating the mode bits) and a result latency of one cycle.

Consider the code sample:

```
mrs   r0, cpsr
orr   r0, r0, #1
add   r1, r2, r3
```

The **ORR** instruction above would incur a one cycle stall due to the two-cycle result latency of the **MRS** instruction. In the code example above, the **ADD** instruction is moved before the **ORR** instruction to prevent this stall.

3.10.5.8 Scheduling CP15 Coprocessor Instructions

The **MRC** instruction has an issue latency of one cycle and a result latency of three cycles. The **MCR** instruction has an issue latency of one cycle.

Consider the code sample:

```
add   r1, r2, r3
mrc   p15, 0, r7, C1, C0, 0
mov   r0, r7
add   r1, r1, #1
```

The **MOV** instruction above would incur a two-cycle latency due to the three-cycle result latency of the mrc instruction. The code shown above is rearranged as follows to avoid these stalls:

```
mrc   p15, 0, r7, C1, C0, 0
add   r1, r2, r3
add   r1, r1, #1
mov   r0, r7
```

3.10.6 Optimizing C Libraries

Many of the standard C library routines benefit greatly by being optimized for the architecture of the IXP43X network processors. The following string and memory manipulation routines should be tuned to obtain the best performance from the architecture (instruction selection, cache usage and data prefetch):

strcat, strchr, strcmp, strcoll, strcpy, strcspn, strlen, strncat, strncmp, strpbrk, strrchr, strspn, strstr, strtok, strxfrm, memchr, memcmp, memcpy, memmove, memset

3.10.7 Optimizations for Size

For applications such as cell phone software it is necessary to optimize the code for improved performance and minimizing code size. Optimizing for smaller code size, in general, lowers the performance of your application. This section contains techniques for optimizing for code size using the instruction set for the IXP43X network processors.

3.10.7.1 Space/Performance Trade Off

Many optimizations mentioned in the previous sections improve the performance of Intel StrongARM code. But, using these instructions results in increased code size. Use the following optimizations to reduce the space requirements of the application code.

3.10.7.1.1 Multiple Word Load and Store

The **LDM**/**STM** instructions are one word long and let you load or store multiple registers at once. Use the **LDM**/**STM** instructions instead of a sequence of loads/stores to consecutive addresses in memory whenever possible.

3.10.7.1.2 Use of Conditional Instructions

Using conditional instructions to expand if-then-else statements as described in “Conditional Instructions” on page 180 results in increasing the size of the generated code. Therefore, do not use conditional instructions if application code space requirements are an issue.

3.10.7.1.3 Use of PLD Instructions

The preload instruction **PLD** is only a hint, it does not change the architectural state of the processor. Using them does not change the behavior of your code, therefore, you should avoid using these instructions when optimizing for space.

§ §

4.0 Network Processor Engines (NPE)

The Network Processor Engines (NPEs) are dedicated function processors containing hardware coprocessors that are integrated into the Intel® IXP43X Product Line of Network Processors. The NPEs are used to offload processing functions required by the Intel XScale® Processor.

The NPEs are high performance, hardware multi-threaded processors with additional local hardware assist functionality used to offload processor intensive functions such as MII (MAC), CRC checking/generation, AAL, DES, AES, SHA, MD5, and so on.

Note: Certain network processor functions are not available depending on which variant of the Intel® IXP43X Product Line is used. Table 95 shows the network processor functions that are enabled in different network processors in the Intel® IXP43X Product Line.

Table 95. Network Processor Functions

Device	NPE A				NPE C	
	UTOPIA	HSS	HDLC	ETHA	ETHC	AES/DES/SHA/MD5
Intel® IXP435 Network Processor	YES	YES	YES	YES	YES	YES
Intel® IXP433 Network Processor	NO	YES	YES	YES	YES	NO
Intel® IXP432 Network Processor	NO	NO	NO	YES	YES	YES
Intel® IXP431 Network Processor	YES	YES	YES	NO	YES	NO
Intel® IXP430 Network Processor	NO	NO	NO	YES	YES	NO

Instruction code for the NPEs is stored locally through a dedicated instruction memory bus and data memory bus. The NPEs support processing of dedicated peripherals interfaces in the IXP43X network processors. The peripherals supported by the use of the NPEs are up to two MII interfaces - UTOPIA Level 2 interface and a high-speed serial interface.

The NPE core is a hardware multi-threaded processor engine that is used to accelerate functions that are difficult to achieve high performance in a standard RISC processor. Each NPE core is a 133-MHz processor core that has self-contained instruction memory and data memory that operate in parallel.

In addition to having separate instruction/data memory, the NPE core supports hardware multi-threading with support for multiple contexts. The support of hardware multi-threading allows an efficient processor engine with minimal processor stalls due to the ability of the processor core to switch context to a new context in a single clock cycle based upon a prioritized/pre-emptive basis.

The prioritized/pre-emptive nature of the context switching allows time critical applications to be implemented in a low-latency fashion, which is required when processing multi-media applications. The NPE core also connects several hardware-based coprocessors. The coprocessors are used to implement several functions that are difficult for a processor to implement. The type of functions implemented by the coprocessors are serialization/de-serialization, CRC checking/ generation, DES/3DES, AES, SHA, MD-5, and HDLC bit-stuffing/de-stuffing. These coprocessors are implemented in hardware thus allowing the coprocessors and the Network Processor Engine core to operate in parallel.

The combined forces of hardware multi-threading, independent instruction memory, independent data memory, and parallel processing allows the Intel XScale processor to be utilized for application purposes. The multi-processing capability of the peripheral interface functions allows unparalleled performance to be achieved by the application running on the Intel XScale processor.

Note: All the described NPE functions require Intel supplied software executing on the NPEs. For further information, see the *Intel® IXP400 Software Programmer's Guide*. For information on the availability of the NPE software and its enabling functions, contact your Intel local sales representative.

4.1 HDLC Coprocessor

The primary function of the High-level Data Link Control (HDLC) Coprocessor is to encapsulate data into HDLC frames. Extracting data from an HDLC frame is also supported. HDLC processing is bit-oriented in nature. It is inefficient to do this in software and therefore, hardware support is required. HDLC processing is performed on data in the NPE cores data memory. The HDLC coprocessor is instantiated once and runs off the 133 MHz clock.

4.1.1 HDLC Coprocessor Features List

The hardware support for HDLC processing will only be for the bit-level processing. Specifically, only the following features are supported and performed in the order listed:

On transmit (encapsulating data in HDLC frames):

- FCS (CRC) generation – both 16- and 32-bit CRC polynomials are supported.
- Bit stuffing – any sequence of five consecutive ones is followed by a 0. ITU-T Q.921 refers to this as "transparency".
- Flag generation – the flag sequence is defined as 0x7E (01111110). Two flags encapsulate an HDLC frame. The start/end flags of successive frames may be shared. Transmit can be configured to start a frame with 0-2 consecutive flags (0 implying start/end flag sharing). However, in FCS 32-bit mode, the number of start-of-frame flags is limited to 0 or 1 flags for frame lengths less than or equal to 4 bytes. Bit sharing between consecutive flags is not possible.
- Idle generation – two idle modes are supported - continuous flags and continuous ones.
- Abort sequence generation – the abort sequence is defined as 0x7F (1111111). This must be generated by the NPE core; the coprocessor does not assist this.
- Pre- and post-processing bit flipping on byte boundary capability.

On receive (removing framing to recover data):

- Error sequence recognition – The coprocessor will recognize an abort sequence and set the RxStat.ABT status bit. This will also set the RxStat.EOF status bit.

- Discard of inter-frame data – flags, idle indicators, and any data not within a frame, is discarded without alerting the NPE core. This includes any data at startup before the first flag is received. Two idle modes are supported: continuous flags and continuous 1s.
- Flag recognition – the flag sequence is used to indicate that a valid frame is in progress. Flag recognition is performed in conformance with the transmit rules above. Bit sharing between successive flags is supported on receive; two consecutive flags may share their start/end bits.
- Bit destuffing – any 0 that follows five consecutive ones is removed.
- FCS checking – the FCS value is checked using the same polynomial as for transmit and the RxStat.FCS status bit is set if the check fails. This bit is only valid at end-of-frame if no other error condition (abort or residual character error) is present.
- Pre and post-processing bit flipping on byte boundary capability.

No provision has been made for a 56K mode for HDLC data.

A frame contains data in an integer number of bytes (before bit stuffing). Depending on their size, frames are handled as described in Table 96.

Table 96. Received Frame Length Handling

FCS Configuration	Frame Length	Hardware Action	Firmware Notification
CRC-16	1-3 bytes	Invalid frame. Discarded.	No
	>= 4 bytes	Valid frame. Passed to NPE core.	Yes
CRC-32	1-3 bytes	Invalid frame. Discarded.	No
	4-5 bytes	Invalid frame. Passed to NPE core. Alignment error asserted.	Yes
	>= 6 bytes	Valid frame. Passed to NPE core.	Yes

§ §

5.0 Internal Bus

The internal bus architecture of the Intel® IXP43X Product Line of Network Processors is designed to allow parallel processing and isolate bus utilization based upon particular traffic patterns. The bus is segmented into four major buses:

- MPI
- North AHB
- South AHB
- APB

The Memory Port Interface provides a dedicated interface between the Intel XScale® Processor and the DDR I/II SDRAM. This interface is a 133/200-MHz, 64-bit bus that is mastered by the Intel XScale processor only. The only target of this interface is the DDR I/II SDRAM controller.

By providing a dedicated path that supports multiple outstanding queues, split, and posted transactions, this interface can achieve better memory performance over previous generation products. Isolation of the interface also allows more DDR I/II SDRAM performance to be allocated to the Intel XScale processor.

The North AHB is a 133.32 MHz (4*OSC_IN), 32-bit bus that can be mastered by the Network Processor Engine (NPE A or NPE C). The targets of the North AHB can either be the DDR I/II SDRAM controller or AHB/AHB Bridge. The AHB/AHB Bridge allows access by the NPEs to the peripherals and internal targets on the South AHB.

Data transfers by the NPEs from the North AHB to the South AHB are targeted predominately to the queue manager. Transfers to the AHB/AHB Bridge can be **posted** while writing or **split** while reading thus allowing control of the North AHB to be given to another master on the North AHB and allowing the bus to achieve maximum efficiency.

The AHB/AHB Bridge supports only SINGLE, INCR4 and INCR8 non wrapping burst access for 32 bit words, and SINGLE access for Byte and Half Word. The AHB/AHB Bridge returns an error response to NPEs for all other burst mode transfer.

Note: Users will have no control on the transaction type (SINGLE, INCR or INCR8). The transaction type is optimized according to the transfer sizes.

Transfers to the AHB/AHB Bridge are considered to be small and infrequent relative to the traffic passed between the NPEs on the North AHB and the DDR I/II SDRAM.

The South AHB is a 133.32MHz (4*OSC_IN), 32-bit bus that can be mastered by the Intel XScale processor, PCI Controller, USB2 Host controllers and the AHB/AHB Bridge. The targets of the South AHB can be DDR I/II SDRAM, PCI Controller, Queue Manager, Expansion Bus Controllers, USB2 Host controllers or the AHB/APB Bridge. Accessing across the AHB/APB Bridge allows interfacing with peripherals attached to the APB Bus.

The APB is a 66.66MHz (2*OSC_IN), 32-bit bus that can be mastered by the AHB/APB Bridge only. The targets of the APB can be High-Speed UART Interface, Console UART Interface, all NPEs, Internal Bus Performance Monitoring Unit (PMU), Interrupt Controller, GPIO, SSP and Timers. The APB interface to the NPEs is used for NPE code download, part configuration, and status collection.

The maximum length that any AHB master can hold the AHB is for eight 32-bit words. This feature allows fairness among all masters on the AHBs.

5.1 Internal Bus Arbiters

The IXP43X network processors contain two internal bus arbiters for the following two AHB transactions:

- North
- South

The arbiters ensure that at any particular time only one AHB master has access to a given AHB. The arbiters perform this function by observing all the AHB master requests to the given AHB segment and deciding which AHB master will be the next owner of the AHB.

The arbiters have a standard interface to all bus masters and split-capable slaves in the system. Any AHB master can request an AHB at any cycle. The arbiters sample the AHB requests. If a particular AHB master is requesting the AHB and is next in the round-robin list, the arbiter will grant the master the AHB.

The arbiters also have the capability to handle split transfers. A split transfer is when:

- An AHB master requests a read from a split capable AHB target
- The split-capable AHB target issues a split transfer indication to the arbiter
- The arbiter allows other transactions to take place on the AHB while the AHB master that issued the request that resulted in the split transfer waits on the read data to be returned from the split capable AHB target
- The split capable AHB target completes the read transaction and notifies the arbiter
- The arbiter grants the AHB master that requested the split transfer the bus in the normal round-robin progression
- The read data is transferred from the split capable AHB target to the AHB master that issued the request that resulted in the split transfer

All split capable AHB targets split a single AHB master read request at any given instance. If the split capable AHB target receives another read request while servicing a split transaction, the split capable AHB target will issue a retry. The only split capable targets on the South AHB is the Expansion Bus. The only split capable target on the North AHB is the AHB/AHB Bridge.

The arbiters send event information to the Internal Bus Performance Monitoring Unit (IBPMU) so that the North AHB and South AHB performance can be observed. The events that can be monitored are provided in “Performance Monitoring Unit”.

The North AHB Arbiter is identical to the South AHB Arbiter in all respects except for the bus masters and targets in which they are connected.

5.1.1 Priority Mechanism

The arbiters allow the bus initiators access to the AHBs using a round-robin scheme. Table 97 illustrates a generic arbitration example for three AHB masters requesting the AHB. The functionality of the independent arbiters is identical.

Each bus initiator (X, Y, and Z) constantly requests the bus. The bottom row of Table 97 lists the current bus initiator/winner of the initiators. For example, when all the three masters are requesting access, X will be the winner, and then Y and Z will request. Next, Y wins the AHB and X returns with a new request. So ZX are still valid with Z being the oldest. Next, Z wins the bus, and so on.

Table 97. Bus Arbitration Example: Three Requesting Masters

	Initial	+1	+2	+3	+4	+5	+6	+7	+8	+9
Requesting Masters	XYZ	YZ	ZX	XY	YZ	ZX	XY	YZ	ZX	XY
Winning Bus Initiator	-	X	Y	Z	X	Y	Z	X	Y	Z

5.2 Memory Map

Table 98 shows the memory map of peripherals connected to the AHB.

Table 98. Memory Map (Sheet 1 of 2)

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 MB	Expansion Bus Data[1]
0000_0000	3FFF_FFFF	1 GB	DDR I-266/DDR II-400 SDRAM Data[1]
4000_0000	47FF_FFFF	128 MB	(Reserved)
4800_0000	4FFF_FFFF	128 MB	PCI Data
5000_0000	5FFF_FFFF	256 MB	Expansion Bus Data
6000_0000	63FF_FFFF	64 MB	Queue manager
6400_0000	BFFF_FFFF	1472 MB	(Reserved)
C000_0000	C3FF_FFFF	64 MB	PCI Controller Configuration and Status Registers
C400_0000	C7FF_FFFF	64 MB	Expansion Bus Configuration Registers
C800_0000	C800_0FFF	4 KB	High-Speed UART
C800_1000	C800_1FFF	4 KB	(Reserved)
C800_2000	C800_2FFF	4 KB	Internal Bus Performance Monitoring Unit
C800_3000	C800_3FFF	4 KB	Interrupt Controller
C800_4000	C800_4FFF	4 KB	GPIO Controller
C800_5000	C800_5FFF	4 KB	Timers
C800_6000	C800_6FFF	4 KB	NPE-A (Intel® IXP400 Software Definition)– Not User Programmable
C800_7000	C800_7FFF	4 KB	(Reserved)

Notes:

1. The lowest 256 Mbyte of address space is configurable based on the value of a configuration register located in the Expansion Bus Controllers.
2. When the configuration register is set to logic 1, the Expansion Bus occupies the lowest 256 Mbyte of address space.
3. When the configuration register is set to logic 0 the SDRAM occupies the lowest 256 Mbyte of address.
4. In both cases, the SDRAM occupies the 768 MB immediately following the lowest 256 Mbyte and the Expansion Bus can be accessed starting at address 5000_0000
5. On reset, the configuration register in the Expansion Bus is set to logic 1. This setting is required because the dedicated boot memory is flash memory located on the Expansion Bus

Table 98. Memory Map (Sheet 2 of 2)

Start Address	End Address	Size	Use
C800_8000	C800_8FFF	4 KB	NPE-C (IXP400 software definition) – Not User Programmable
C800_9000	C800_9FFF	4 KB	(Reserved)
C800_A000	C800_AFFF	4 KB	Ethernet MAC on NPE C
C800_B000	C800_BFFF	4 KB	(Reserved)
C800_C000	C800_CFFF	4 KB	Ethernet MAC on NPE A
C800_D000	C800_FFFF	12 KB	(Reserved)
C801_0000	C801_0FFF	4 KB	(Reserved)
C801_1000	C801_1FFF	4 KB	Reserved
C801_2000	C801_2FFF	4 KB	SSP Unit
C801_3000	CC00_E4FF	65517.25 KB	(Reserved)
CC00_E500	CC00_F5FF	4.25 KB	DDR I/II-266/400 SDRAM Configuration Registers
CC00_F600	CCFF_FFFF	16322.5 KB	(Reserved)
CD00_0000	CDFF_FFFF	16MB	USB2 Host Controller 1
CE00_0000	CEFF_FFFF	16MB	USB2 Host Controller 2
CF00_0000	FFFF_FFFF	768 MB	(Reserved)

Notes:

1. The lowest 256 Mbyte of address space is configurable based on the value of a configuration register located in the Expansion Bus Controllers.
2. When the configuration register is set to logic 1, the Expansion Bus occupies the lowest 256 Mbyte of address space.
3. When the configuration register is set to logic 0 the SDRAM occupies the lowest 256 Mbyte of address.
4. In both cases, the SDRAM occupies the 768 MB immediately following the lowest 256 Mbyte and the Expansion Bus can be accessed starting at address 5000_0000
5. On reset, the configuration register in the Expansion Bus is set to logic 1. This setting is required because the dedicated boot memory is flash memory located on the Expansion Bus

§ §

6.0 Ethernet MACs

The functionality supported by the MII Interfaces is tightly coupled with code written on the Network Processor Engine (NPE) core. This chapter explains hardware capabilities of the MII Interface contained within the Ethernet Coprocessor of the Intel® IXP43X Product Line of Network Processors. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and a subset of the features are described in this section.

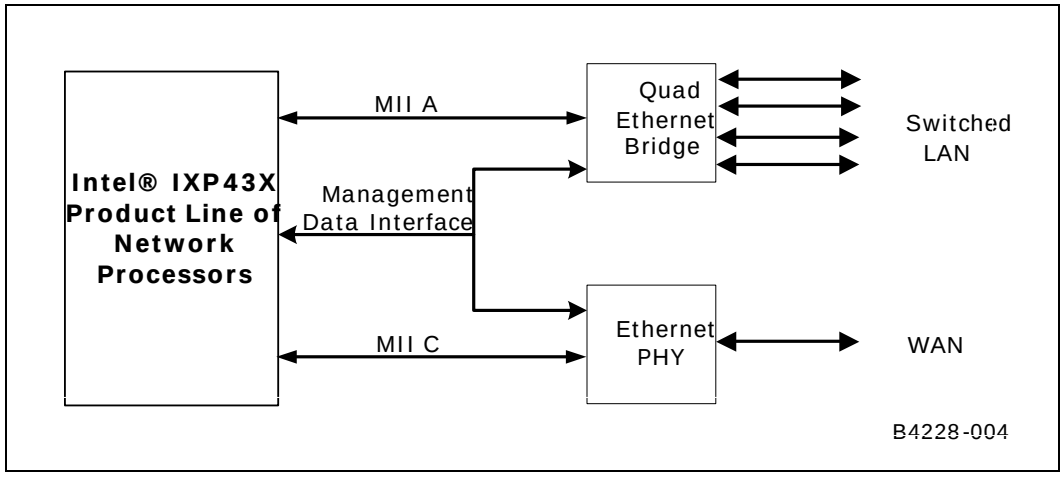
Note: Not all of the IXP43X network processors have this functionality. For details, see Table 95, "Network Processor Functions" on page 207.

The IXP43X network processors contain two NPEs. All the NPEs are used to process Ethernet traffic utilizing the MII interfaces. Each NPE core used for Ethernet traffic connects to a Transmit FIFO and Receive FIFO through the NPE Coprocessor interface. These Transmit and Receive FIFOs are used to buffer data between the Ethernet Media Access Controller (MAC) and the NPE core.

The Transmit FIFO, Receive FIFO, and MAC are contained in an NPE coprocessor unit called the Ethernet Coprocessor. The MAC contained in the Ethernet Coprocessor is compliant to the IEEE 802.3 specification and handling flow control for the IEEE 802.3Q VLAN specification.

One Management Data Interface is shared among all the MII interfaces. The single Management Data Interface is used to configure the physical devices attached to each of the MII interfaces. The Management Data Interface is accessible by manipulating the Management Data Registers associated with Ethernet MAC Coprocessor C, see "Register Descriptions Ethernet MACs" on page 225. Figure 30 shows a typical application that is used in connecting to the MII interface.

Figure 30. Multiple Ethernet PHYS Connected to Processor

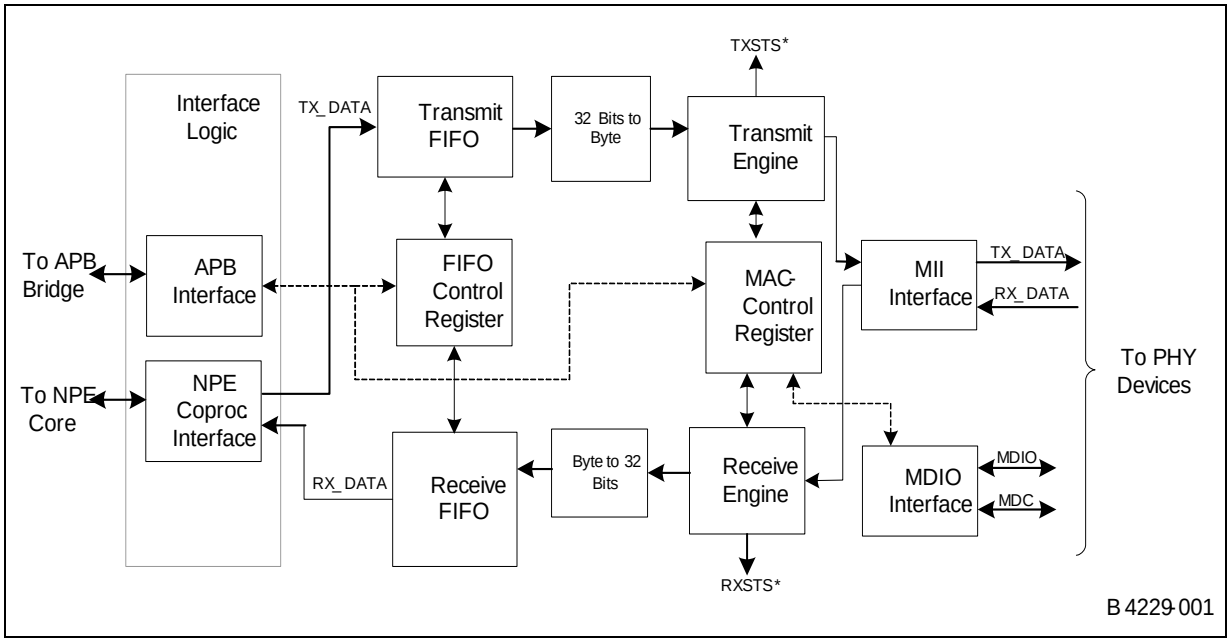


6.1 Ethernet Coprocessor

Each Ethernet Coprocessor contains a Media Access Controller, a transmit data FIFO, and a receive data FIFO.

Figure 31 displays block diagram of a single Ethernet Coprocessor.

Figure 31. Ethernet Coprocessor Interface



The Ethernet Coprocessor communicates to the peripheral devices and remainder of the IXP43X network processors over four interfaces:

- APB Interface
- NPE Coprocessor Interface
- MII Interface
- Management Data Interface

6.1.1 Ethernet Coprocessor APB Interface

The APB interface is used to allow the Intel XScale® Processor to communicate directly to configuration and control registers utilized by the Media Access Controller. The Ethernet coprocessor's APB interface is used to configure the Ethernet MAC, monitor Ethernet status, and configure the physical devices connected via the MII interfaces. The physical devices connected to the MII interface is configured using the shared Management Data Interface.

6.1.2 Ethernet Coprocessor NPE Interface

The NPE Coprocessor Interface is used to communicate between the Ethernet Coprocessor and the NPE core. The NPE coprocessor interface is used to transfer incoming and outgoing data traffic to and from the NPE core. The NPE core along with other coprocessor takes the Ethernet data and performs data manipulation, forwards the data to the SDRAM, and updates the Queue Manager.

6.1.3 Ethernet Coprocessor MDIO Interface

The Management Data Interface is a two-wire interface that supports configuration of an MII Interface on the IXP43X network processors. The Management Data Interface consists of the Management Data Input/Output (MDIO) signal and the Management Data Clock (MDC). The Management Data Input/Output signal is a bidirectional signal that is used to transfer control, configuration, and status information between the IXP43X network processors and any peripheral devices connected to the MII interfaces.

The Ethernet Coprocessor is initiated two times to support two Ethernet PHYs outside the device. However, only the MDC and MDIO pins of one of the coprocessors (Ethernet C) are brought out and they are used to program both the Ethernet PHYs.

The Management Data Clock is configured as an input or an output, enabling the IXP43X network processors to source the Management Data Clock or enable an external device to source the clock. The Management Data Clock is used to clock the data sent on the Management Data Input/Output Signal.

Data transfers are initiated over the MDIO using the MDIO Command Register (MDIOCMD). The MDIO Command Register is broken into four 8-bit registers that make up a full 32-bit command word.

If data is to be sent to the PHY over the MDIO interface, the Intel XScale processor writes a value to each of the four command words in sequential order:

- MDIO Command 1 (MDIOCMD1) Register and MDIO Command 2 (MDIOCMD2) Register contains the 16 bits of data that the destination PHY receives.
- MDIO Command 3 (MDIOCMD3) Register and MDIO Command 4 (MDIOCMD4) Register determines the PHY number that is to be addressed, the internal register of the addressed PHY, the direction of the access (read/write), and when to begin the access.

There is a limit of 32 physical ports with a limit of 32 registers per physical port that is addressed.

MDIOCMD3 makes up bits (23:16) of MDIOCMD and MDIOCMD4 make up bits (31:24):

- Bits (25:21) of MDIOCMD are used to select the physical interface that should accept the transmitted data or return the requested data.
- Bits (20:16) of MDIOCMD are used to select the register within the physical interface that is to accept the transmitted data or return the requested data.
- Bit 26 of MDIOCMD is used to determine if the requested command is a read or a write:
  - — Writing logic 0 to this bit causes the transaction to be a read
  - — Writing logic 1 to this bit causes the transaction to be a write

Setting Bit 31 of the MDIO Command (MDIOCMD) Register to logic 1 initiates the transfer. Bit 31 of MDIOCMD remains at logic 1 until the transaction is complete.

Figure 32 shows an example of data being written from the MII Management Master (IXP43X network processors) to a physical interface (PHY) using the MDIO interface.

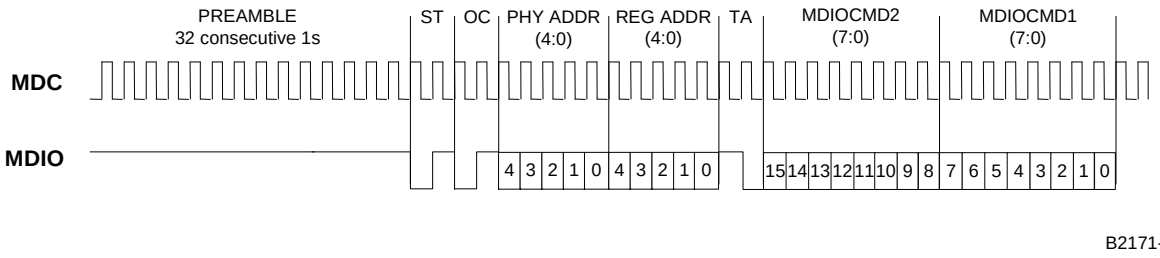
As stated previously, when bit 26 of the MDIO Command (MDIOCMD) Register is set to logic 0, the Intel XScale processor is requesting a read from a physical interface device using the MDIO interface. The data that the physical interface returns from the MDIO signal is captured in the MDIO STATUS (MDIOSTS) Register.

The MDIO Status Register is broken into four 8-bit registers. The data returned from the physical interface is captured in MDIO Status 0 (MDIOSTS0) Register and MDIO Status 1 (MDIOSTS1) Register:

- MDIO Status 0 (MDIOSTS0) Register corresponds to bits (7:0) of the MDIO Status (MDIOSTS) Register.
- MDIO Status 1 (MDIOSTS1) Register corresponds to bits (15:8) of the MDIO Status (MDIOSTS) Register.
- Bits (30:16) of the MDIO Status (MDIOSTS) Register are reserved and returns zeros when read.
- Bit 31 of the MDIO Status (MDIOSTS) Register indicates the condition of the read.
  - If logic 1 is read from this bit after a read transaction from the physical interface is complete, the read contained an error and should be disregarded.
  - If logic 0 is read from this bit after a read transaction from the physical interface is complete, the read is valid and error free.

Figure 33 shows an example of the data being read from the physical interface (PHY) by the MII Management Master (IXP43X network processors) using the MDIO interface. These registers should be manipulated using Intel supplied APIs.

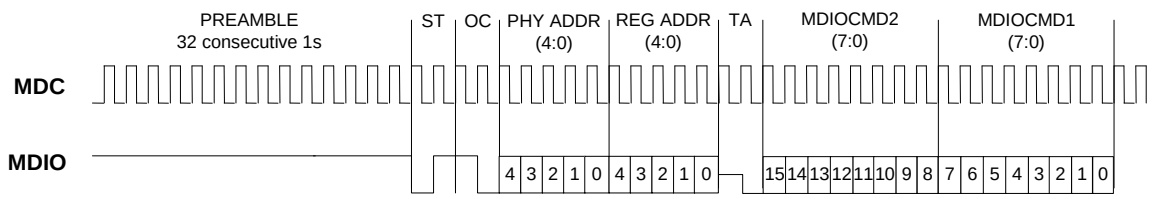
Figure 32. MDIO Write



Notes:

1. ST (Start Bits) is a signal that is logic 0 followed by logic 1 after a PREAMBLE stage.
2. OC (Op Code) is a two-bit signal that informs the destination PHYs if the current requested transaction is a read or a write. Logic 0 followed by logic 1 indicates a write transaction is requested. Logic 1 followed by logic 0 indicates a read transaction.
3. TA (Turn Around) is a two-bit, turn-around time used to allow the control of the MDIO to change directions. For write operations, the TA bits is logic 1 followed by logic 0. For read transactions, the TA bits is high-impedance (Z) followed by the selected PHY driving the MDIO signal with logic 0.
4. For write operations, the Management Interface Master drives the MDIO signal for the duration of the access.
5. For read operation, the Management Interface Master drives the MDIO signal until the turn around cycle. The PHY drives the MDIO signal for the second bit of the turn around and the remaining 16 data bits.

Figure 33. MDIO Read



Notes:

1. ST (Start Bits) is a signal that is logic 0 followed by logic 1 — after a PREAMBLE stage.
2. OC (Op Code) is a two-bit signal that informs the destination PHYs, if the current requested transaction is a read or a write. Logic 0 followed by logic 1 indicates a write transaction is requested. Logic 1 followed by logic 0 indicates a read transaction.
3. TA (Turn Around) is a two-bit, turn-around time used to allow the control of the MDIO to change directions. For write operations, the TA bits is logic 1 followed by logic 0. For read transactions, the TA bits is high-impedance (Z) followed by the selected PHY driving the MDIO signal with logic 0.
4. For write operations, the Management Interface Master drives the MDIO signal for the duration of the access.
5. For read operation, the Management Interface Master drives the MDIO signal until the turn around cycle. The PHY drives the MDIO signal for the second bit of the turn around and the remaining 16 data bits.

6.1.4 Transmitting Ethernet Frames with MII Interfaces

Using processor API calls, the Intel XScale processor can request the packets to be transmitted by the MII Interface. The Intel XScale processor prepares an Ethernet packet to be transmitted. On completion of the preparation, the Intel XScale processor uses Intel supplied API calls to inform the Ethernet NPE that a packet is ready to be transmitted. The Ethernet NPE fetches packet from the memory attached to the IXP43X network processors and forwards the data over the NPE coprocessor interface to the 256-byte Transmit FIFO contained in the Ethernet Coprocessor.

Once the data has reached a predefined trigger point, known as the Buffer Size for Transmit Register (TXBUFFSIZE) in the Transmit FIFO or the End of Frame signal is received, a data packet begins to transmit over the MII interface. For each frame transmitted, TXBUFFSIZE holds the minimum number of words that must be contained in the Transmit FIFO before the frame transmission starts. If total size of the frame is less than the Buffer Size for Transmit Register, the frame is transmitted when the end-of-frame signal is received.

When entries leave the bottom of the Transmit FIFO, the entries are 32 bits. The settings of the Buffer Size for Transmit Register (TXBUFFSIZE), the threshold for Partially Full (THRESHPF), and the threshold for Partially Empty (THRESHPE) are tightly coupled with the code written for the NPE core. Manipulation of the values results in unpredictable behavior.

- The threshold for Partially Full parameter sets a status flag going to the NPE core when the number of entries in the Transmit or Receive FIFOs is larger than the value programmed in this register.
- The threshold for Partially Empty parameter sets a status flag going to the NPE core when the number of entries in the Transmit or Receive FIFOs is smaller than the value programmed in this register.

After the data begins leaving the FIFO, the data is sent through a converter function that is used to convert the bits from 32-bits to byte-wide entries to supply to the Transmit Engine. The Transmit Engine takes the bytes supplied from the converter, manipulate the data as defined by MAC control registers and forward the data over the MII interface as 4-bit nibbles in the case of MII operation.

The Transmit Engine is configured using processor API calls to:

- Append a Frame Check Sequence to the end of a transmitted frame
- Autonomously append bytes to frames that are smaller than the minimum frame size (64 bytes)
- Enable/Disable transmit retries
- Set the number of times a frame is retried due to collision conditions before being dropped
- Select half- or full-duplex mode of operation
- Select a one- or two-part deferral to be used
- Enable/Disable the Transmit Engine

A Frame Check Sequence (FCS) is autonomously generated and appended to the end of each Ethernet frame. The Frame Check Sequence is used to ensure proper delivery of data between two Ethernet devices. The Frame Check Sequence consists of a 4-byte Cyclic Redundancy Generator that adheres to the polynomial:

$$G(x) = x\ 32 + x\ 26 + x\ 23 + x\ 22 + x\ 16 + x\ 12 + x\ 11 + x\ 10 + x\ 8 + x\ 7 + x\ 5 + x\ 4 + x\ 2 + x + 1$$

The Frame Check Sequence is computed over all fields beginning after the Start-of-Frame Delimiter (SFD) and up to the Frame Check Sequence (FCS) value. Autonomous insertion of the Frame Check Sequence into the transmitted frame is enabled/disabled by setting bit 4 of Transmit Control Register 1 (TXCTRL1):

- Setting this bit to logic 1 causes a CRC value to be generated and inserted into the Frame Check Sequence field of the transmitted frame.
- Setting this bit to logic 0 causes a transmitted frame to be sent without a Frame Check Sequence attached.

Transmit Control Register 1 is accessed directly, but it is recommended to manipulate this values through Intel supplied APIs. Failure to use Intel supplied APIs can result in unpredictable results.

The Transmit Engine can also be configured to append additional bytes to frames that are smaller than the 64-byte frame minimum. When the Transmit Engine observes that length field is smaller than 64 bytes and bit 3 of Transmit Control Register 1 is set to logic 1, the Transmit Engine appends additional bytes to equal the 64-byte minimum size. A CRC value is calculated over the appended bytes; the appended bytes is all zeros.

The IXP43X network processors also provide the capability to enable or disable transmit retries. When bit 2 of the Transmit Control Register 1 is set to logic 1 and collisions occur, the Transmit Engine attempts to retry sending the packet up to the maximum number of transmit retries specified in bits (3:0) of Transmit Control Register 2 (TXCTRL2). A maximum of 16 retries is attempted.

Note: Setting bit 1 of TXCTRL1 to logic 1 configures a device in half-duplex mode of operation. Setting the bit to logic 0 configures the device to full-duplex mode.

- If a packet is being transmitted and a collision is detected *prior* to 64 bytes (minimum packet size) of the packet are transmitted, the Transmit Engine rewinds the Transmit FIFO pointer to the beginning of the frame that is being transmitted and attempt to send the frame after a calculated back-off time, based upon the Slot Time (SLOTTIME) Register.
- If a packet is being transmitted and a collision is detected *after* 64 bytes (minimum packet size) of the packet are transmitted, the Transmit Engine forwards the

Transmit FIFO pointer to the beginning of the next packet that is to be transmitted and discard the current packet that is being transmitted.

In a properly-configured network, a collision should not occur after the first 64 bytes of a packet are sent. The back-off time algorithm adheres to the IEEE 802.3 specifications **truncated binary exponential algorithm**.

If the MII interface is configured in half-duplex mode of operation, the Transmit Interface must listen to the activity on the line for a defined wait period, called a transmit deferral period. If there is any transmit or receive activity on the medium, the Transmit Interface defers the transmission of the packet that the Transmit Engine has been requested to send.

When the Transmit Engine detects the medium has gone silent, the Transmit Engine continues to defer for a period of time equal to the inter-frame spacing. The deferral period is assigned by the transmit deferral parameters. The transmit deferral time is specified as a one-part, or optional two-part, deferral by manipulating bit 5 of Transmit Control Register 1:

- Setting this bit to logic 0 enables the one-part transmit deferral.
- Setting this bit to logic 1 enables the optional, two-part transmit deferral.

If the Transmit Engine has a frame ready to transmit and one-part transmit deferral is selected, the Transmit Engine waits for the medium to go silent and then wait for the time period specified in the Transmit Deferral Register (TXDEFPARS). The deferral period is the number of transmit clock cycles specified by the 8-bit Transmit Deferral Register minus three transmit clock cycles. The Single Transmit Deferral parameter specifies the Inter Frame Gap.

In the two-part deferral process, the deferral period is defined using the Transmit Two Part Deferral Parameters 1 Register (TX2PARTDEFPARS1) and Transmit Two Part Deferral Parameters 2 Register (TX2PARTDEFPARS2). The values specified in these two registers when added together should not be less than the minimum Inter Frame Gap to ensure fair access to the medium.

Transmit Two Part Deferral Parameters 1 Register is set to two-thirds of the Inter Frame Gap and Transmit Two Part Deferral Parameters 2 Register is set to the remaining one-third.

If the Transmit Engine has a frame ready to transmit and two-part transmit deferral is selected, the Transmit Engine waits for the medium to go silent and then wait for the time period specified in the Transmit Two Part Deferral Parameters 1 Register:

- If the medium is not silent during this first part of the deferral, the deferral counter is reset.
- If the medium is silent during the first part of the deferral, Transmit Engine continues to wait for the time period specified in the Transmit Two Part Deferral Parameters 2 Register.

When the MII interface of the IXP43X network processors is configured in full-duplex mode of operation, the two-part transmit deferral parameters and the back-off times is not utilized for data transmission.

Refer to IEEE 802.3 Section 4.2.3.2.1 for more information on deference.

The transmit deferral registers and the Transmit Control Register is accessed directly, but it is recommended that these register values be manipulated through Intel supplied APIs. Failure to use Intel supplied APIs can result in unpredictable results.

The value of TXCTRL1 is accessed directly, but it is recommended that TXCTRL1 values are manipulated through Intel supplied APIs. Failure to use Intel supplied APIs can result in unpredictable results.

6.1.5 Receiving Ethernet Frames with MII Interfaces

When data is received using the MII interface, the Receive engine is used to convert the data from 4-bit nibbles into 8-bit bytes when configured in MII mode of operation.

The receive interface sends the data to a byte-to-32-bit-word converter. The 32-bit word is written into the 256-byte receive FIFO.

A flag is generated to inform the NPE that the receive FIFO has new data to be removed after the threshold for Partial Full/Empty value has been reached. The Receive Engine implements the following functions:

- Enable/Disable the Receive Engine
- Implements uni-cast/multi-case/broadcast, single-address filtering
- Checks for runt frames
- Checks for valid length/type fields
- Removes padded bits added to frames
- Implements the Frame-Check Sequence Algorithm

When new receive data is detected, the Receive Engine looks to see if the address filtering checks are enabled. If the address filtering checks are enable by setting bit 5 of the Receive Control Register 1 (RXCTRL1) to logic 1, the Receive Engine obtains the destination address from the received frame, check the destination address against the address filtering parameters, and pass the frame to the 8-bit to 32-bit framing engine, if the test passed.

The Receive Engine is capable of filtering broadcast frames, multi-cast frames, and uni-cast frames. The frame filtering for multi-cast frames and uni-cast frames is controlled by registers called the Address Mask Register (ADDRMASK), the Address Register (ADDR), and the Uni-Cast Register (UNIADDR).

Determining if the 48-bit destination address of the received frame contains all logic 1s filters the broadcast frames. Setting bit 7 of Receive Control Register 1 (RXCTRL1) to logic 1 prevents received broadcast frames from being sent to the NPE. (For more information see, “Receive Control 1” on page 229.)

The multi-cast frame filtering and uni-cast frame filtering is enabled/disabled by setting bit 5 of Receive Control Register 1 (RXCTRL1). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all non-broadcast frames to be sent to the NPE. The NPE can then operate in promiscuous mode and implement a more comprehensive filtering algorithm if required.

Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 1 enables filtering for uni-cast frames that are received. When Uni-Cast frames are detected, the destination address of the received frame must match exactly the value contained in the Uni-Cast Address Register (UNIADDR). Setting bit 5 of Receive Control Register 1 (RXCTRL1) to logic 0 allows all uni-cast frames to be passed to the NPE.

For example, if the Uni-Cast Address Register, which is broken into six bytes, contains the value of 0x00ABCDEF1234 and uni-cast address filtering is enabled, then any uni-cast frames that are received (determined by bit 47 being set to logic 0) with anything other than 0x00ABCDEF1234 is discarded.

Multi-cast is when frames are sent to multiple destinations from a single source with a single-frame transmission. The Address Mask Register (ADDRMASK) and the Address Register (ADDR) Multi-Cast frames is used to filter multi-cast frames or frames with common addresses. The address mask is used to tell what each destination address has in common.

For example, bytes 3 and bytes 4 of every address in this group contain the same value. The Address Mask Register (ADDRMASK) would contain a hexadecimal value of 0x00FFFF000000. The Address Mask Register is a logical **AND** with the Address Register and then a logical **AND** with the destination address. The result of the logical **AND** values is compared to see if they match. If they match, the frame is passed to the NPE via the remaining logic.

The values seen were as follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D2-47-63-21

Notice that the underlined values are all that matters for the comparison because of the address mask. The frame in this example is forwarded to the next phase of the receive logic. An example of a frame that would be dropped due to address filtering is as follows:

- Address Mask Register (ADDRMASK) = 00-FF-FF-00-00-00
- Address Register (ADDR) = 00-C1-D2-38-72-00
- Destination Address = A1-C1-D3-47-63-21

Note: The **Address Filter Enable** field under the Receive Control 1 register (rxctrl1) must be set in order broadcast or multi-cast packets to be accepted.

To disable the multi-cast address filtering feature set the Address Mask Register to all logic 0s.

After the received frame has passed all address filtering checks, the Receive Engine captures the length/type field. If the length/type is determined to be a length value field and the length is less than 64 bytes, the Receive Engine removes any padded bytes (assuming bit 1 of Receive Control Register 1 is set to logic 1) and capture the remaining data. Padded bytes is not removed from the received packet when bit 1 of Receive Control Register is set to logic 0.

If the packet is less than 64 bytes and there are no padded bytes, the packet is determined to be a runt frame. The Receive Engine has the capability to discard these frames or forward the frames to the NPE via the remaining receive logic.

Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 1 informs the Receive Engine to allow the packet to be sent to the NPE. Setting bit 6 of Receive Control Register 1 (RXCTRL1) to logic 0 informs the Receive Engine to terminate the reception of the runt packet and purge the runt packet from the rest of the receive logic.

If the length/type is determined to be a type field, the field is looked at for validity. Invalid frames are purged from the receive logic.

Once the received frame has passed the frame validity checks, the received frame is checked for integrity using the Frame-Check Sequence Algorithm called out in the transmit-frame section. If the frame passes the Frame Check Sequence, the frame is forwarded on to the NPE via the remaining receive logic. If the frame fails the Frame Check Sequence, the frame is discarded along with purging all the remaining receive logic of the frames' contents.

Padded bytes are included in the calculation of the Receive Engine's Frame Check Sequence for frames that were transmitted and smaller than the 64-byte minimum frame size. A status flag is sent to the NPE to inform the NPE what to do with the received frame.

In addition to the above features, the MII receive interface allows some features to be used for test and debug. The transmit interface is looped back to the receive interface by setting bit 4 of Receive Control Register 1 (RXCTRL1) to logic 1. In loop-back mode, the Receive Engine receives all the data sent by the Transmit Engine.

The loopback feature allows software developers to develop their application code and test the code before they start dealing with physical interface problems. Setting bit 0 of Receive Control Register 1(RXCTRL1) to logic 1 enables the Receive Engine. This feature allows all initialization of the product to occur prior to bring the receive interface online. Bit 0 of Receive Control Register 2 (RXCTRL2) enables deferral checking on the receive side. The IXP43X network processors do not use the receive side deferral checking feature.

Bit 0 of Receive Control Register 2 (RXCTRL2) must be set to logic 0 for proper operation. Failure to do so results in unpredictable behavior.

It is recommended to manipulate the register values described in this section through Intel supplied APIs. Failure to use Intel supplied APIs can result in unpredictable results.

6.1.6 General Ethernet Coprocessor Configuration

The Ethernet Coprocessor contains various other registers that are used to configure the interface. Some of these registers are included due to the generic nature of the Ethernet Coprocessor. Other registers are added to allow greater flexibility in configuration of the Ethernet Coprocessor.

The threshold for Internal Clock (THRESH_INTCLK) Register is used to determine the frequency relationship between the MII interface and the host processor that is used to control the MII interface. The value in the threshold for Internal Clock (THRESH_INTCLK) Register is manipulated based upon the ratio of PHY_CLK_SPEED/ HOST_CLK_SPEED.

The physical interface clock speed is divided by the host-side clock speed and then rounded to the nearest whole number. The value from this calculation is written to the threshold for Internal Clock (THRESH_INTCLK) Register. The value contained in the threshold for Internal Clock (THRESH_INTCLK) Register must always be set to hexadecimal 0x01 for the IXP43X network processors. Failure to do so results in unpredictable behavior. The value of the threshold for Internal Clock (THRESH_INTCLK) Register is pre-programmed to proper value by Intel supplied APIs. Manipulation of this value can result in unpredictable behavior.

The Core Control (CORE_CONTROL) Register is added to allow the Intel XScale processor to have some control over the Ethernet Coprocessor. Configuring bit 4 of the Core Control (CORE_CONTROL) Register can configure as an input or as an output the MDC clock.

Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP43X network processors to drive the MDC clock. Setting bit 4 of the Core Control (CORE_CONTROL) Register to logic 0 enables an external source to drive the MDC clock. The IXP43X network processors become a recipient of that MDC clock.

Configuring bit 3 of the Core Control (CORE_CONTROL) Register can configure the Transmit Engine to send a JAM sequence if a new packet starts to be received. Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 1 enables the IXP43X

network processors to send a JAM sequence if a new packet is received. Setting bit 3 of the Core Control (CORE_CONTROL) Register to logic 0 allows the Transmit Engine to function in normal mode of operation.

Configuring bit 2 of the Core Control (CORE_CONTROL) Register causes the Transmit FIFO to be flushed. Packets that are currently in the Transmit FIFO are discarded. Setting bit 2 of the Core Control (CORE_CONTROL) Register to logic 1 clears the MII Interface Transmit FIFO. Setting bit 2 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Transmit FIFO to resume normal mode of operation.

Configuring bit 1 of the Core Control (CORE_CONTROL) Register causes the Receive FIFO to be flushed. Any packets that are currently in the Receive FIFO are discarded. Setting bit 1 of the Core Control (CORE_CONTROL) Register to logic 1 clears the MII Interface Receive FIFO. Setting bit 1 of the Core Control (CORE_CONTROL) Register back to logic 0 allows the Receive FIFO to resume normal mode of operation. Bit 0 of the Core Control (CORE_CONTROL) Register controls the reset state of the Media Access Controller (MAC) contained within the Ethernet Coprocessor.

Setting bit 0 of the Core Control (CORE_CONTROL) Register back to logic 1 causes the Media Access Controller to be reset. Deassertion (setting bit 0 to logic 0) allows the Media Access Controller to resume operation in a fully reset state.

This register must be manipulated using Intel supplied APIs. Failure to manipulate this register with Intel supplied APIs results in unpredictable behavior.

The Random-Seed Register is an 8-bit register used to support PHYs that support Auto MDI/MDI-X detection. The Random-Seed Register value is the value that is used to feed the Linear-Feedback Shift Register that is used to select the configuration to start initialization.

The Random-Seed Register must be manipulated using Intel supplied APIs. Failure to manipulate this can result in unpredictable behavior.

6.2 Register Descriptions Ethernet MACs

The internal registers shown below are accessible via the APB bus interface. Unspecified addresses are reserved and should not be written; if read, a zero value is returned.

All the Ethernet internal configuration and control registers are directly readable and writable by the Intel XScale processor via APB bus. All registers for all MII interfaces matches the detailed register descriptions contained in this chapter.

6.2.1 Ethernet MAC on NPE A

The table below shows the Registers of Ethernet MAC on NPE A.

Table 99. Ethernet MAC on NPE A (Sheet 1 of 3)

Address	Description
0xC800 C000	Transmit Control 1
0xC800 C004	Transmit Control 2
0xC800 C010	Receive Control 1
0xC800 C014	Receive Control 2
0xC800 C020	Random Seed

Table 99. Ethernet MAC on NPE A (Sheet 2 of 3)

Address	Description
0xC800 C030	Threshold For Partial Empty
0xC800 C038	Threshold For Partial Full
0xC800 C040	Buffer Size For Transmit
0xC800 C050	Transmit Single Deferral Parameters
0xC800 C054	Receive Deferral Parameters
0xC800 C060	Transmit Two Part Deferral Parameters 1
0xC800 C064	Transmit Two Part Deferral Parameters 2
0xC800 C070	Slot Time
0xC800 C080	MDIO Command 1†
0xC800 C084	MDIO Command 2†
0xC800 C088	MDIO Command 3†
0xC800 C08C	MDIO Command 4†
0xC800 C090	MDIO Status 1†
0xC800 C094	MDIO Status 2†
0xC800 C098	MDIO Status 3†
0xC800 C09C	MDIO Status 4†
0xC800 C0A0	Address Mask 1
0xC800 C0A4	Address Mask 2
0xC800 C0A8	Address Mask 3
0xC800 C0AC	Address Mask 4
0xC800 C0B0	Address Mask 5
0xC800 C0B4	Address Mask 6
0xC800 C0C0	Address 1
0xC800 C0C4	Address 2
0xC800 C0C8	Address 3
0xC800 C0CC	Address 4
0xC800 C0D0	Address 5
0xC800 C0D4	Address 6
0xC800 C0E0	Threshold For Internal Clock
0xC800 C0F0	Unicast Address 1
0xC800 C0F4	Unicast Address 2

Table 99. Ethernet MAC on NPE A (Sheet 3 of 3)

Address	Description
0xC800 C0F8	Unicast Address 3
0xC800 C0FC	Unicast Address 4
0xC800 C100	Unicast Address 5
0xC800 C104	Unicast Address 6
0xC800 C1FC	Core Control
† The MDI interface on NPE-A is inactive. All external PHYs are configured via NPE-C.	

6.2.2 Ethernet MAC on NPE C

The table below shows the Registers of Ethernet MAC on NPE C.

Table 100. Ethernet MAC on NPE C (Sheet 1 of 2)

Address	Description
0xC800 A000	Transmit Control 1
0xC800 A004	Transmit Control 2
0xC800 A010	Receive Control 1
0xC800 A014	Receive Control 2
0xC800 A020	Random Seed
0xC800 A030	Threshold For Partial Empty
0xC800 A038	Threshold For Partial Full
0xC800 A040	Buffer Size For Transmit
0xC800 A050	Transmit Single Deferral Parameters
0xC800 A054	Receive Deferral Parameters
0xC800 A060	Transmit Two Part Deferral Parameters 1
0xC800 A064	Transmit Two Part Deferral Parameters 2
0xC800 A070	Slot Time
0xC800 A080	MDIO Command 1
0xC800 A084	MDIO Command 2
0xC800 A088	MDIO Command 3
0xC800 A08C	MDIO Command 4
0xC800 A090	MDIO Status 1
0xC800 A094	MDIO Status 2
0xC800 A098	MDIO Status 3

Table 100. Ethernet MAC on NPE C (Sheet 2 of 2)

Address	Description
0xC800 A09C	MDIO Status 4
0xC800 A0A0	Address Mask 1
0xC800 A0A4	Address Mask 2
0xC800 A0A8	Address Mask 3
0xC800 A0AC	Address Mask 4
0xC800 A0B0	Address Mask 5
0xC800 A0B4	Address Mask 6
0xC800 A0C0	Address 1
0xC800 A0C4	Address 2
0xC800 A0C8	Address 3
0xC800 A0CC	Address 4
0xC800 A0D0	Address 5
0xC800 A0D4	Address 6
0xC800 A0E0	Threshold For Internal Clock
0xC800 A0F0	Unicast Address 1
0xC800 A0F4	Unicast Address 2
0xC800 A0F8	Unicast Address 3
0xC800 A0FC	Unicast Address 4
0xC800 A100	Unicast Address 5
0xC800 A104	Unicast Address 6
0xC800 A1FC	Core Control

6.2.3 Transmit Control 1

The tables below gives the detailed description of Transmit Control 1 Register.

Register Name:	txctrl1			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Transmit Control Register			

Access: Read/Write.

31	...	7	6	5	4	3	2	1	0
(Reserved)			MII CFG	2PRT DEF	APP FCS	PAD EN	RET EN	HALFDUP	TX EN

Register		txctrl1
Bits	**Name**	**Description**
31:7		(Reserved)
6	MII config	0 = Configures the PHY interface as a MII.
5	Two-part deferral	1 = Causes the optional two part deferral to be used.
4	Append FCS	1 = Causes FCS to be computed and appended to transmit frames before they are sent to the PHY.
3	Pad enable	1 = Causes transmit frames less than to minimum frame size to be padded before they are sent to the PHY.
2	Retry enable	1 = Causes transmit frames to be retried until the maximum retry limit shown in the Transmit Control 2 Register is reached, when collisions occur.
1	Half duplex	1 = Half-duplex operation 0 = Full-duplex
0	Transmit enable	1 = Causes transmission to be enabled.

6.2.4 Transmit Control 2

The tables below gives the detailed description of Transmit Control 2 Register.

Register Name:	**txcrtl2**			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		**Reset Value:**	**0x00000000**
Register Description:	Transmit Control Register			
Access: Read/Write.				

31																									**4**	**3**			**0**
(Reserved)																										Maximum Retries			

Register		txcrtl2
Bits	**Name**	**Description**
4:31	(Reserved)	
3:0	Maximum retries	Maximum number of retries for a packet when collisions occur.

6.2.5 Receive Control 1

The tables below provide detailed description of the Receive Control 1 Register:

Register Name:	**rxctrl1**			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		**Reset Value:**	0x00000000
Register Description:	Receive Control Register			
Access: Read/Write.				

31																						**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(Reserved)																							BCDIS	RX RP	ADD FILT	LOOP EN	PSE EN	CRC	PAD STRP	RX EN

Register		rxcrtl1
Bits	**Name**	**Description**
31:8	(Reserved)	
7	Broadcast disable	Broadcast packets is dropped if broadcast disable is set to **1**, the address mask register is NOT 00 00 00 00 00 00 and the address filter is enabled. Setting the address mask register to all 00 (don't care about the address) and setting the Broadcast disable bit to **1** (checking the address) at the same time is a contradiction and results in broadcast packets still being received and the packet status (MCST_PKT / BCST_PKT) is read back accordingly.
6	Receive runt packet	1 = Causes runt packets to be passed to the application logic. 0 = Runt packets are dropped.
5	Address filter enable	1 = Causes address filtering to take place. Non-broadcast packets are only passed to the application logic if they pass the address filter.
4	Loopback enable	1 = Causes loop-back operation. ***Note:*** In order for the loop-back operation to operate correctly, the Ethernet Coprocessor requires synchronous and in-phase clocks to be provided on the rx_clk and tx_clk pins.
3	Pause enable	1 = Enables detection of Pause frames. Upon detecting a pause frame, data transmission is halted based on the data in the pause frame. The 2-byte data of the received pause frame indicates the time, as a number of 512 bit times, to halt the transmission.
2	Send CRC	1 = Causes the CRC data to be sent to the application logic.
1	Pad strip	1 = Causes the pad bytes to be stripped from receive data.
0	Receive enable	1 = Causes reception to be enabled.

6.2.6 Receive Control 2

The tables below provide detailed description of the Receive Control 2 Register:

Register Name:	rxctrl2			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		**Reset Value:**	**0x00000000**
Register Description:	Receive Control Register			
Access: Read/Write.				

31																											1	0
(Reserved)																												RX DEF EN

Register		rxctrl2
Bits	**Name**	**Description**
31:1	(Reserved)	
0	Receive deferral enable	1 = Enables receive deferral checking.

6.2.7 Random Seed

The tables below provide detailed description of the Random Seed Register:

Register Name:	rndmseed		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Random Seed Register		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			Random Seed		

Register		rndmseed
Bits	Name	Description
31:8	(Reserved)	
7:0	Random Seed	Random seed used for LFSR initialization in the back-off block.

6.2.8 Threshold For Partially Empty

The tables below provide detailed description of the Threshold for Partially Empty Register.

Register Name:	threshpe		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	FIFO partially full/empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			Partial Empty		

Register		threshpe
Bits	Name	Description
31:8	(Reserved)	
7:0	Partial Empty	Marks the partial empty thresholds of the Transmit FIFO and Receive FIFO. When the number of entries in the Transmit FIFO is less than or equal to the contents of this register, tx_fifo_p_empty is asserted. When the number of entries in the Receive FIFO is less than or equal to the contents of this register, rx_fifo_p_empty is asserted.

6.2.9 Threshold For Partially Full

The tables below gives the detailed description of Threshold for Partially Full Register.

Register Name:	threshpf		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	FIFO Partially Full/Empty Threshold Register. The threshold is the number of 32-bit words in the FIFO.		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			Partial Full		

Register		threshpf
Bits	**Name**	**Description**
31:8	(Reserved)	
7:0	Partial Full	Marks the partial full thresholds of the Transmit FIFO and Receive FIFO. When the number of entries in the Transmit FIFO is greater than or equal to the contents of this register, tx_fifo_p_full is asserted. When the number of entries in the Receive FIFO is greater than or equal to the contents of this register, rx_fifo_p_full is asserted.

6.2.10 Buffer Size For Transmit

The tables below gives the detailed description of Buffer Size for Transmit Register.

Register Name:	txbuffsize			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Transmit Buffer Size. The size is the number of 32-bit words in the FIFO. Note* Only 6 out of 8 bits value in this register is valid since the maximum FIFO size is 64 words.			
Access: Read/Write.				

31																					8	7							0
(Reserved)																						Tx Buffer Size							

Register		txbuffsize
Bits	**Name**	**Description**
31:8	(Reserved)	
7:0	Tx Buffer size	Holds minimum number of bytes of each frame that must be in the Transmit FIFO for that frame's transmission to start. If a complete frame is less than this minimum, it is always transmitted.

6.2.11 Transmit Deferral Parameter

The tables below gives the detailed description of Transmit Deferral Parameter Register.

Register Name:	txdefpars			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Transmit/Receive Deferral Parameters			
Access: Read/Write.				

31																					8	7							0
(Reserved)																						Transmit Deferral							

Register		txdefpars
Bits	**Name**	**Description**
31:8	(Reserved)	
7:0	Single deferral	Number of transmit clock cycles (tx_clk) in the transmit deferral period minus three, when single deferral is used for transmission (Transmit Control[15] = 0).

6.2.12 Receive Deferral Parameter

The tables below provide detailed description of the Receive Deferral Parameter Register:

Register Name:	rxdefpars		
Hex Offset Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Transmit/Receive Deferral Parameters		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			Receive Deferral		

Register		rxdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Receive Deferral	Number of receive clock cycles (rx_clk) in the receive deferral period minus three, when checking the Inter Frame Gap for packets received (Receive Control 2[0] = 0).

6.2.13 Transmit Two Part Deferral Parameters 1

The tables below provide detailed description of the Transmit Two Part Deferral Parameters 1 Register:

Register Name:	tx2partdefpars1		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Transmit Two Part Deferral Parameters Register.		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			First Deferral Period		

Register		txdefpars
Bits	Name	Description
31:8	(Reserved)	
7:0	First deferral period	Number of transmit clock cycles (tx_clk) in the first deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

6.2.14 Transmit Two Part Deferral Parameters 2

The tables below provide detailed description of the Transmit Two Part Deferral Parameters 2 Register:

Register Name:	tx2partdefpars2			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Transmit Two Part Deferral Parameters Register			

Access: Read/Write.

31																					8	7							0
(Reserved)																						Second Deferral Period							

Register		tx2defpars
Bits	Name	Description
31:8	(Reserved)	
7:0	Second Deferral Period	Number of transmit clock cycles (tx_clk) in the second deferral period minus three, when two-part deferral is used for transmission (Transmit Control 1[5] = 1) and half-duplex mode.

6.2.15 Slot Time

The tables below provide detailed description of the Slot Time Register:

Register Name:	slottime				
Hex Offset Address:	0xC8009070		Reset Hex Value:	0x00000000	
Register Description:	Slot Time Register				

Access: Read/Write.

31																					8	7							0
(Reserved)																						Slot Time							

Register		slottime
Bits	Name	Description
31:8	(Reserved)	
7:0	Slot Time	Slot time for back-off algorithm Expressed in number of tx_clk cycles. 128 in MII mode.

6.2.16 MDIO Commands Registers

Following four registers make up the 32-bit MDIO Command:

- MDIO Command[31:24] — MDIO Command 4
- MDIO Command[23:16] — MDIO Command 3
- MDIO Command[15:8] — MDIO Command 2
- MDIO Command[7:0] — MDIO Command 1

The detailed bit descriptions follow the bit maps of the four commands.

6.2.17 MDIO Command 1

The tables below provide detailed description of the MDIO Command 1 Register:

Register Name:	mdiocmd1		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	MDIO Command 1 (8 Bits of 32-Bit Register).		
Access: Read/Write.			

7						0
MDIO Command [7:0]						

6.2.18 MDIO Command 2

The tables below provide detailed description of the MDIO Command 2 Register:

Register Name:	mdiocmd2		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	MDIO Command Register		
Access: Read/Write.			

31																						8	7							0
(Reserved)																							MDIO_COMMAND [15:8]							

6.2.19 MDIO Command 3

The table below provide detailed description of the MDIO Command 3 Register:

Register Name:	mdiocm3		
Hex t Address:	See Table 99 for NPE-A and Table 100 for NPE-C	reset value:	0x00000000
Register Description:	MDIO Command Register		
Access: Read/Write.			

31																						8	7							0
(Reserved)																							MDIO_COMMAND [23:16]							

6.2.20 MDIO Command 4

The tables below provide detailed description of the MDIO Command 4 Register:

Register Name:	mdiocm4		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	MDIO Command Register		
Access: Read/Write.			

31	30																					8	7							0
GO	(Reserved)																						MDIO_COMMAND[31:24]							

Register		MDIO Command
Bits	**Name**	**Description**
31	Go	Application logic sets this to 1 to start the MDIO access. This bit remains 1 during the access. When the access is finished, the Ethernet core resets this bit to 0.
30:27	(Reserved)	
26	MDIO Write	1 = MDIO write access 0 = MDIO read access.
25:21	PHY address	Physical address of the PHY to be accessed.
20:16	PHY Register	Register number of the PHY Register to be accessed.
15:0	Write data	Write data on MDIO write accesses.

6.2.21 MDIO Status Registers

Following four registers make up the 32-bit MDIO status:

- MDIO Status[31:24] — MDIO Status 4
- MDIO Status[23:16] — MDIO Status 3
- MDIO Status[15:8] — MDIO Status 2
- MDIO Status[7:0] — MDIO Status 1

The detailed bit descriptions follow the bit maps of the four registers.

6.2.22 MDIO Status 1

The table below provides detailed description of the MDIO Status 1 Register:

Register Name:	**mdiosts1**			
Hex Address:	See **Table 99** for NPE-A and **Table 100** for NPE-C		**Reset Value:**	**0x00000000**
Register Description:	MDIO Status Register			
Access: Read Only.				

31																					**8**	**7**							**0**
(Reserved)																						MDIO_STATUS[7:0]							

6.2.23 MDIO Status 2

The table below provides detailed description of the MDIO Status 2 Register:

Register Name:	**mdiosts2**			
Hex Address:	See **Table 99** for NPE-A and **Table 100** for NPE-C		**Reset Value:**	**0x00000000**
Register Description:	MDIO Status Register			
Access: Read Only.				

31																					**8**	**7**							**0**
(Reserved)																						MDIO_STATUS[15:8]							

6.2.24 MDIO Status 3

The table below provides detailed description of the MDIO Status 3 Register:

Register Name:	mdiosts3			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	MDIO Status Register			
Access: Read Only.				

31																				8	7							0
(Reserved)																					MDIO_STATUS[23:16]							

6.2.25 MDIO Status 4

The table below provides detailed description of the MDIO Status 4 Register:

Register Name:	mdiosts4			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	MDIO Status Register			
Access: Read Only.				

31																				8	7							0
(Reserved)																					MDIO_STATUS[31:24]							

Register		MDIO Status
Bits	Name	Description
31	Successful read	0 = A successful read 1 = A read error. Read only.
30:16	(Reserved)	Read only.
15:0	Read data	Read only.

6.2.26 Address Mask Registers

Following six registers make up the 48-bit Address Mask:

- Address Mask[47:40] — Address Mask 1
- Address Mask[39:32] — Address Mask 2
- Address Mask[31:24] — Address Mask 3
- Address Mask[23:16] — Address Mask 4
- Address Mask[15:8] — Address Mask 5
- Address Mask[7:0] — Address Mask 6

Example: Address Mask is 00-A0-24-D1-7F-02

- Address Mask 1 = 0x00
- Address Mask 2 = 0x00
- Address Mask 3 = 0x00

- Address Mask 4 = 0xFF
- Address Mask 5 = 0xFF
- Address Mask 6 = 0x00

The detailed bit descriptions follow the six registers bit maps.

6.2.27 Address Mask 1

The table below provides detailed description of the Address Mask 1 Register:

Register Name:	addrmask1			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Mask Register #1. First register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted			
Access: Read/Write.				

31																						8	7							0
(Reserved)																							ADDRESS MASK [47:40]								

6.2.28 Address Mask 2

The table below provides detailed description of the Address Mask 2 Register:

Register Name:	addrmask2			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Mask Register #1. Second register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																						8	7							0
(Reserved)																							ADDRESS MASK [39:32]								

6.2.29 Address Mask 3

The table below provides detailed description of the Address Mask 3 Register:

Register Name:	addrmask3			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Mask Register #1. Third register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																						8	7							0
(Reserved)																							ADDRESS MASK [31:24]								

6.2.30 Address Mask 4

The table below provides detailed description of the Address Mask 4 Register:

Register Name:	addrmask4		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Address Mask Register #1. Forth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			ADDRESS MASK [23:16]		

6.2.31 Address Mask 5

The table below provides detailed description of the Address Mask 5 Register:

Register Name:	addrmask5		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Address Mask Register #1. Fifth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.		

Access: Read/Write.

31	...	8	7	...	0
(Reserved)			ADDRESS MASK [15:8]		

6.2.32 Address Mask 6

The tables below provides detailed description of the Address Mask 6 Register:

Register Name:	addrmask6		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Address Mask Register #1. Sixth register of six that makes up the Address Mask. Address Mask is used with Address for multicast address filtering. Bits set to 1 in Address Mask represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.		

Access: Read/Write.

31	...	16	15	...	8	7	...	0
RESERVED			ADDRESS MASK [7:0]					

Register		Address Mask
Bits	Name	Description
47:0	Address Mask1-6	Address Mask 1-6 is used with Address 1-6 for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.

6.2.33 Address Registers

Following six registers make up the 48-bit Address:

- Address[47:40] — Address 1

- Address[39:32] — Address 2
- Address[31:24] — Address 3
- Address[23:16] — Address 4
- Address[15:8] — Address 5
- Address[7:0] — Address 6

Example: Address is 00-A0-24-D1-7F-02

- Address 1 = 0x00
- Address 2 = 0xA0
- Address 3 = 0x24
- Address 4 = 0xD1
- Address 5 = 0x7F
- Address 6 = 0x02

The detailed bit descriptions follow the six registers' bit maps.

6.2.34 Address 1

The table below provides detailed description of the Address 1 Register:

Register Name:	addr1			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. First register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																						8	7							0
(Reserved)																							ADDRESS[47:40]								

6.2.35 Address 2

The table below provides detailed description of the Address 2 Register:

Register Name:	addr2			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. Second register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																						8	7							0
(Reserved)																							ADDRESS[39:32]								

6.2.36 Address 3

The table below provides detailed description of the Address 3 Register:

Register Name:	addr3			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. Third register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																					8	7							0
(Reserved)																						ADDRESS[31:24]							

6.2.37 Address 4

The table below provides detailed description of the Address 4 Register:

Register Name:	addr4			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. Forth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																					8	7							0
(Reserved)																						ADDRESS[23:16]							

6.2.38 Address 5

The table below provides detailed description of the Address 5 Register:

Register Name:	addr5			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. Fifth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			
Access: Read/Write.				

31																					8	7							0
(Reserved)																						ADDRESS[15:8]							

6.2.39 Address 6

The tables below provides detailed description of the Address 6 Register:

Register Name:	addr6			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Address Register #1. Sixth register of six that makes up the Address. Address Mask is used with Address for multicast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.			

Access: Read/Write.

31																					8	7							0
(Reserved)																						ADDRESS[7:0]							

Register		Address
Bits	Name	Description
47:0	Address	Address Mask 1-6 is used with Address 1-6 for multi-cast address filtering. Bits set to 1, in Address Mask, represent bits of the Address Register that must match the corresponding bits in incoming destination addresses for packets to be accepted.

6.2.40 Threshold for Internal Clock

The tables below provides detailed description of the Threshold for Internal Clock Register:

Register Name:	thresh_intclk			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Threshold for Internal Clock Register			

Access: Read/Write.

31																					8	7							0
(Reserved)																						CLOCK RATIO							

Register		thresh_intclk
Bits	Name	Description
31:8	(Reserved)	
7:0	Clock ratio	Holds ratio of PHY side clock (tx_clk or rx_clk) to sys_clk. PHY side clock frequency/application clock frequency. Round up for value to be written to this register. Example: 25 MHz PHY clock, 133 MHz application clock — Sets this register to 1. Any application, clock frequency greater than the tx_clk or rx_clk frequency has a clock ratio of 1. **Always set to 1** for the IXP43X network processors**.**

6.2.41 Unicast Address Registers

Following six registers make up the 48-bit Unicast Address:

- Unicast Address[47:40] — Unicast Address 1
- Unicast Address[39:32] — Unicast Address 2
- Unicast Address[31:24] — Unicast Address 3
- Unicast Address[23:16] — Unicast Address 4
- Unicast Address[15:8] — Unicast Address 5

- Unicast Address[7:0] — Unicast Address 6

Example: Unicast Address is 00-A0-24-D1-7F-02

- Unicast Address 1 = 0x00
- Unicast Address 2 = 0xA0
- Unicast Address 3 = 0x24
- Unicast Address 4 = 0xD1
- Unicast Address 5 = 0x7F
- Unicast Address 6 = 0x02

The detailed bit descriptions follow the bit maps of the six registers.

6.2.42 Unicast Address 1

The table below provides detailed description of the Unicast Address 1 Register:

Register Name:	uniaddr1		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. First register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.		

Access: Read/Write.

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
(Reserved)																								UNICAST ADDRESS[47:40]							

6.2.43 Unicast Address 2

The table below provides detailed description of the Unicast Address 2 Register:

Register Name:	uniaddr2		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. Second register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.		

Access: Read/Write.

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
(Reserved)																								UNICAST ADDRESS[39:32]							

6.2.44 Unicast Address 3

The table below provides detailed description of the Unicast Address 3 Register:

Register Name:	uniaddr3		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. Third register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.		

Access: Read/Write.

31																						8	7							0
(Reserved)																							UNICAST ADDRESS[31:24]							

6.2.45 Unicast Address 4

The table below provides detailed description of the Unicast Address 4 Register:

Register Name:	uniaddr4		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. Forth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.		

Access: Read/Write.

31																						8	7							0
(Reserved)																							UNICAST ADDRESS[23:16]							

6.2.46 Unicast Address 5

The table below provides detailed description of the Unicast Address 5 Register:

Register Name:	uniaddr5		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. Fifth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.		

Access: Read/Write.

31																						8	7							0
(Reserved)																							UNICAST ADDRESS[15:8]							

6.2.47 Unicast Address 6

The tables below provides detailed description of the Unicast Address 6 Register:

Register Name:	uniaddr6		
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C	Reset Value:	0x00000000
Register Description:	Unicast Address Register #1. Sixth register of six that makes up the Unicast Address. Matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE		

Access: Read/Write.

31																						8	7							0
(Reserved)																							UNICAST ADDRESS[7:0]							

Register		Unicast Address
Bits	**Name**	**Description**
47:0		Unicast Address 1-6 are matched with destination address of receive packets for unicast address filtering. (Receive Control Address Filter bit is 1.) If a match occurs, the frame is passed to the NPE.

6.2.48 Core Control

The tables below provides detailed description of the Core Control Register:

Register Name:	core_control			
Hex Address:	See Table 99 for NPE-A and Table 100 for NPE-C		Reset Value:	0x00000000
Register Description:	Controls key functions of the core			

Access: Read/Write.

31																							5	4	3	2	1	0
(Reserved)																								mdc en	send jam	clr tx err	clr rx err	rst_mac

Register		core_control
Bits	**Name**	**Description**
31:5	(Reserved)	
4	Mdc_en	1 = Configures the MDC as an output clock. Set to 1 for the IXP43X network processors.
3	Send_jam	1 = Causes a jam sequence to be sent if reception of a packet begins.
2	clr_tx_err	Assertion (**1**) causes the Transmit FIFO to be flushed. Data in the Transmit FIFO is discarded.
1	clr_rx_err	Assertion (**1**) causes the Receive FIFO to be flushed. Data in the Receive FIFO is discarded.
0	rst_mac	Assertion (**1**) causes the MAC to be reset.

§ §

7.0 UTOPIA Level 2

The functionality supported by the UTOPIA Level 2 interface is tightly coupled with the code written on the Network Processor Engine (NPE). This chapter covers hardware capabilities of the UTOPIA Level 2 interface contained within UTOPIA Level 2 coprocessor of the Intel® IXP43X Product Line of Network Processors. The accessible features are described in the *Intel® IXP400 Software Programmer's Guide* and a subset of the features is described in this chapter.

Note: Not all of the IXP43X network processors have this functionality. For details, see Table 95, "Network Processor Functions" on page 207.

7.1 Introduction

UTOPIA Level 2 is an industry standard interface that is used to provide connection between Asynchronous Transmission Mode (ATM) and physical layer (PHY) of an ATM network. The UTOPIA Level 2 coprocessor is an entity within the IXP43X network processors that provides the UTOPIA Level 2 interface.

The UTOPIA Level 2 coprocessor implemented on the IXP43X network processors provides an 8-bit, UTOPIA Level 2 interface operating at speeds up to 33 MHz. The UTOPIA Level 2 interface can be configured to operate in a single-PHY (SPHY) or a multiple-PHY (MPHY) environment.

The interface contains five transmit and five receive address lines for multi-PHY selection. The UTOPIA Level 2 coprocessor comprises the following three functional modules:

- UTOPIA Transmit Module
- UTOPIA Receive Module
- Network Processor Engine (NPE) Core Interface Module

Two 128-byte-deep FIFOs are contained in each direction of data flow, one FIFO for transmit, and one FIFO for receive. Each FIFO is organized into two cell buffers; each being 64 bytes deep. This FIFO arrangement allows the receive module to process a cell and store it away at the same time the Network Processor Engine core processes a previously received cell.

In the transmit direction, the Network Processor Engine core can place a cell into one transmit buffer as the Transmit Module removes the cell from the other transmit buffer.

While operating in single-PHY (SPHY) mode, the UTOPIA Level 2 interface supports octet or cell level handshaking as defined by the UTOPIA Level 2 specification. When configured in multiple-PHY (MPHY) mode, only cell level handshaking is supported.

The hardware interface allows connection up to 31 physical interface devices as defined in the UTOPIA Level 2 specification. The ATM Adaptation Layers (AAL) supports only 31 physical devices for the IXP43X network processors.

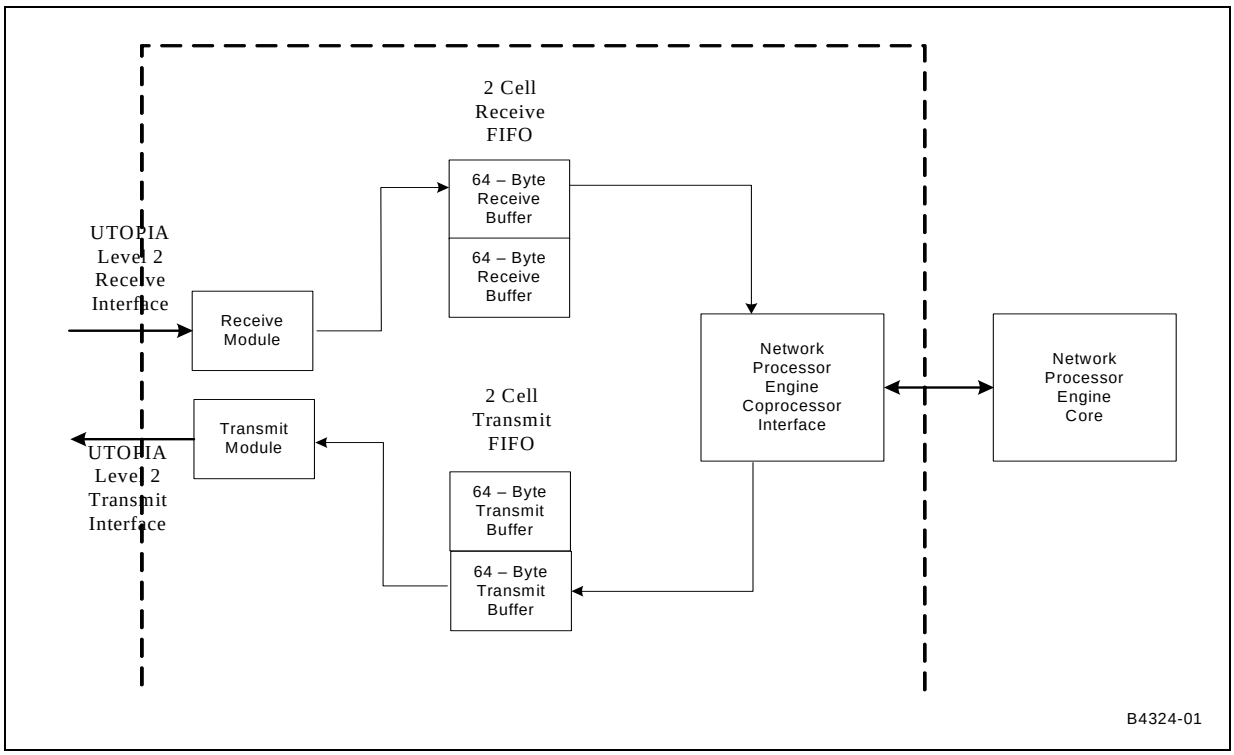
Additionally, the two-cycle polling routine defined by the UTOPIA Level 2 specification limits the number of physical devices polled during an ATM cell transfer to 26.

The ATM Adaptation Layers (AAL) implemented by the Network Processor Engine can have some further limitations on the number of physical interfaces supported. For more details on the number of physical interfaces supported, see the *Intel® IXP400 Software Programmer's Guide*.

On the Network Processor Engine side, the UTOPIA Level 2 coprocessor interfaces with the Network Processor Engine Core through the Network Processor Engine Coprocessor Bus Interface. The Network Processor Engine Coprocessor Bus Interface is used to transfer data to and from the Network Processor Engine core. The Network Processor Engine Coprocessor Bus Interface is also used to access status and configuration information for the UTOPIA Level 2 coprocessor.

Figure 34 shows the various modules within the UTOPIA Level 2 coprocessor.

Figure 34. UTOPIA Level 2 Coprocessor



7.2 UTOPIA Interface

The UTOPIA interface is a UTOPIA Level 1 and UTOPIA Level 2 compliant interface that is capable of operating as a PHY or ATM interface in SPHY or MPHY modes. The interface consists of the following signals:

Table 101. UTOPIA Transmit Interface (Sheet 1 of 2)

Signal(s)	Signal Type	Reset Value	UTOPIA Interface Modes				Description
			ATM Octet	PHY Octet	ATM Cell	PHY Cell	
UTP_OP_CLK	Input	1	TxClk	RxClk	TxClk	RxClk	The rising-edge of UTP_OP_CLK is used to clock cell data from the UTOPIA coprocessor.

Table 101. UTOPIA Transmit Interface (Sheet 2 of 2)

Signal(s)	Signal Type	Reset Value	UTOPIA Interface Modes				Description
			ATM Octet	PHY Octet	ATM Cell	PHY Cell	
UTP_OP_FCO	Output Tristate	1	TxEnb*	RxEmp*	TxEnb*	RxClav	UTOPIA flow control output signal. Refer to UTOPIA Level 2 specification for a detailed explanation of use depending on the mode of operation. In MPHY mode **RxClav** is an active high **tri-stateable** signal.
UTP_OP_FCI	Input	1	TxFull*	RxEnb*	TxClav	RxEnb*	UTOPIA flow control input signal. In MPHY mode, **RxEnb*** is **used** to tristate **RxData** and **RxSOC**.
UTP_OP_SOC	Output Tristate	1	TxSOC	RxSOC	TxSOC	RxSOC	Start of Cell. Active high signal asserted when UTP_OP_DATA contains the first valid byte of the cell. To support MPHY mode, **RxSOC** must be **tri-stateable**, enabled only in cycles following those with **RxEnb*** asserted.
UTP_OP_PRTY	Output Tristate	1	TxPrty	RxPrty	TxPrty	RxPrty	Data bus parity. UTP_OP_PRTY is the odd parity bit over UTP_OP_DATA[7:0]. To support MPHY mode, RxPrty must be tri-stateable, enabled only in cycles following those with RxEnb* asserted.
UTP_OP_DATA [7:0]	Output Tristate	00H	TxData [7:0]	RxData [7:0]	TxData [7:0]	RxData [7:0]	UTOPIA output data. UTP_OP_DATA[7] is the MSB. To support MPHY mode, **RxData** must be **tri-stateable**, enabled only in cycles following those with **RxEnb*** asserted.
UTP_OP_ADDR [4:0]	Bidir	ZZZZZ			TxAddr [4:0] output	RxAddr [4:0] input	PHY address bus. It is used by the UTOPIA master to poll and select the appropriate PHY. UTP_OP_ADDR[4] is the MSB. **Not supported in UTOPIA Level 1**.

Table 102. UTOPIA Receive Interface (Sheet 1 of 2)

Signal(s)	Signal Type	Reset Value	UTOPIA Interface Modes				Description
			ATM Octet	PHY Octet	ATM Cell	PHY Cell	
UTP_IP_CLK	Input	N/A	RxClk	TxClk	RxClk	TxClk	The rising-edge of UTP_IP_CLK is used to clock cell data into the UTOPIA coprocessor.
UTP_IP_FCO	Output Tristate	1	RxEnb*	TxFull*	RxEnb*	TxClav	Flow control output signal. In MPHY mode: • **TxClav** is an active high, **tristateable** signal. • **RxEnb*** is **used** to tristate **RxData** and **RxSOC**.

Table 102. UTOPIA Receive Interface (Sheet 2 of 2)

Signal(s)	Signal Type	Reset Value	UTOPIA Interface Modes				Description
			ATM Octet	PHY Octet	ATM Cell	PHY Cell	
UTP_IP_FCI	Input	1	RxEmp*	TxEnb*	RxClav	TxEnb*	UTOPIA flow control input signal. Refer to UTOPIA Level 2 specification for a detailed explanation of use depending on the mode.
UTP_IP_SOC	Input	1	RxSOC	TxSOC	RxSOC	TxSOC	Start of cell. Active high signal asserted if UTP_IP_DATA contains the first valid byte of a cell.
UTP_IP_PRTY	Input	1	RxPrty	TxPrty	RxPrty	TxPrty	Data bus parity. UTP_IP_PRTY is the ODD parity bit over UTP_IP_DATA[7:0].
UTP_IP_DATA [7:0]	Input	8	RxData [7:0]	TxData [7:0]	RxData [7:0]	TxData [7:0]	UTOPIA input data. UTP_IP_DATA[7] is the MSB.
UTP_IP_ADDR [4:0]	Bidir	5			RxAddr [4:0] output	TxAddr [4:0] input	PHY address bus. ATM master uses it to poll/select the appropriate PHY. UTP_IP_ADDR[4] is the MSB. **Not supported in UTOPIA Level 1.**

7.3 UTOPIA Transmit Module

The functionality supported by the Transmit Module is tightly coupled with the code written on the Network Processor Engine. This section explains hardware capabilities of the Transmit Module contained within the UTOPIA Level 2 coprocessor of the IXP43X network processors' control plane processors. The module's user-accessible features are described in the *Intel® IXP400 Software Programmer's Guide* and a subset of the features are described in this section.

The UTOPIA Level 2 Transmit interface transfers ATM cells to one or more UTOPIA-compliant physical devices. In multiple-PHY (MPHY) mode, the UTOPIA Level 2 transmit interface uses a round-robin polling routine to poll various physical interfaces using the five transmit address lines (UTP_OP_ADDR), to determine which physical interfaces are ready to accept data transfers. The result of the polling is provided as status to the Network Processor Engine.

The Transmit Module is the entity within the UTOPIA coprocessor, that implements this functionality. The Transmit Module polls a programmable number of physical interfaces as defined by the Transmit Address Range (TXADDRRANGE) register.

If five physical interfaces are connected to the UTOPIA Level 2 interface, a value of four can be programmed into the Transmit Address Range (TXADDRRANGE) register by the Network Processor Engine Core. The polling begins at logic address 0 and moves sequentially to the value contained in the Transmit Address Range (TXADDRRANGE) register.

To allow more flexibility, a logical address to physical address table is provided. The look-up table makes it possible for the five addresses that were called out above, not to be in sequential order. For example, the following logical to physical address map could be used for the above example of five physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_OP_ADDR lines = 00011
- Logical Address 1 => Physical Address 5 => UTP_OP_ADDR lines = 00101

- Logical Address 2 => Physical Address 7 => UTP_OP_ADDR lines = 00111
- Logical Address 3 => Physical Address 9 => UTP_OP_ADDR lines = 01001
- Logical Address 4 => Physical Address 22 => UTP_OP_ADDR lines = 10110

Once the physical address is driven to all physical interfaces, using the UTP_OP_ADDR signals, the physical interface is ready to accept a cell. The physical interface is configured to the address signals that match the values contained on the UTP_OP_ADDR signals and responds to the UTOPIA Level 2 interface on the IXP43X network processors by driving the UTP_OP_FCI, also known as TX_FULL_N/TX_CLAV) signal to inform the UTOPIA Level 2 Interface that the physical interface is ready to receive a cell.

The Transmit Port Status (TXPORTSTAT) register contained within the Transmit Module stores the polling result for each physical interface. The Network Processor Engine core uses the values stored in the Transmit Port Status (TXPORTSTAT) Register to select a physical interface that is ready to complete a transfer and loads the Transmit FIFO.

The Transmit FIFO informs the Transmit Module that a cell is ready to be transmitted to a specific physical interface. The Transmit Module then removes the cell information from the Transmit FIFO and begins transmitting data to the specified physical interface.

Note: The NPE code sends to the Transmit Module the logical port address of the physical interface to be selected along with the cell data. This feature allows the NPE to have full control over transmitted data based on polling status returned from the hardware.

While transmitting the data, an optional head-error correction (HEC) value can be calculated from the header and inserted into the data stream. The HEC generation unit takes the header data and uses the data with an internal, 8-bit HEC cyclical redundancy check (CRC) residue register to produce the value for the next HEC CRC residue. The HEC is generated new for every cell transmitted and has no dependencies on previous transmitted cells.

The HEC residue can be inserted directly into the data stream being transmitted over the UTOPIA Level 2 interface or optionally, the HEC residue can be exclusive-ORed with hexadecimal 0x55 to generate a COSET value before inserting into the data stream.

The HEC value is available one clock period after the last byte of the header information is transmitted. Therefore, a complete stream of cell data (H0, H1, H2, H3, HEC, D0, D1, ...) can be transmitted in successive clock cycles without interruption to the data stream.

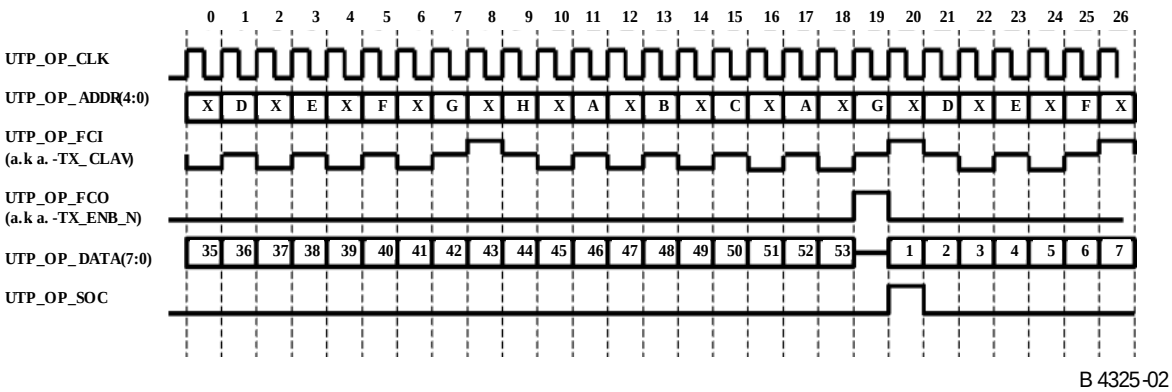
When the Transmit HEC (TxHEC**)** configuration bit is enabled, the UTOPIA transmit interface inserts an extra byte (valid HEC) into the cell being transmitted. In normal operation where TxHEC is enabled, the UTOPIA 2 Coprocessor Transmit Module expects 52 bytes from the Transmit FIFO and the Transmit Module inserts a valid HEC field into the data stream.

Figure 35 shows the transmission of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for this figure:

- There are eight active physical interfaces connected, named A through H, that map to logical address 0 through 7
- Physical Interface A is the currently selected physical interface for clock cycles 0 through18
- Notice on clock 8 that the result from Physical Interface G is that Physical Interface G, which is ready to receive a cell. The UTP_OP_FCI signal flags that a full cell can be sent to Physical Interface G by the Physical Interface asserting the UTP_OP_FCI to logic 1 one clock after Physical Interface G has been polled.

- On clock 17, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that was polled previously prior to this location.
- Notice on clock cycles 19 and 20 that Physical Interface G is selected as the next Physical Interface to which the IXP43X network processors transmit data.

Figure 35. UTOPIA Level 2 MPHY Transmit Polling



In cell level single-PHY (SPHY) mode the physical interface indicates that it can accept a cell by asserting UTP_OP_FCI, also known as TX_FULL_N/TX_CLAV signal. The UTOPIA Level 2 Interface subsequently transmits a cell to the PHY at the same time asserting UTP_OP_FCO, also known as TX_ENB_N.

For more timing diagrams and more information on operation in single-PHY (SPHY) mode of operation, see the *UTOPIA Level 2, Revision 1.0 Specification*.

While using the UTOPIA Level 2 Interface in single-PHY (SPHY) mode, the UTP_OP_ADDR is driven with an address of all logic 1s and the UTP_IP_ADDR must be driven with all logic 1s.

In octet-level single-PHY (SPHY) mode, the physical interface indicates to the UTOPIA Level 2 interface on the IXP43X network processors that the physical interface can accept data by deasserting UTP_OP_FCI, also known as TX_FULL_N/TX_CLAV signal. The UTOPIA Level 2 Interface subsequently transmits data to the PHY at the same time, asserting UTP_OP_FCO, also known as TX_ENB_N.

When the physical interface deasserts UTP_OP_FCI also known as TX_FULL_N/ TX_CLAV, it indicates to the UTOPIA Level 2 Interface that the physical interface accepts only four more bytes of data.

For more timing diagrams and information on the single-PHY (SPHY) mode of operation, see the *UTOPIA Level 2, Revision 1.0 Specification*.

In addition to supporting data transmission and HEC generation, the Transmit Module maintains some statistical values. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters maintains the following counts:

- The number of cells transmitted
- The number of idle cells transmitted

The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values. There is an overflow bit for each counter to indicate that

the count has **rolled-over**. A maskable interrupt mechanism is used to enable the UTOPIA Level 2 coprocessor to flag to the Network Processor Engine core where the roll over has occurred.

7.4 UTOPIA Receive Module

The functionality supported by the Receive Module is tightly coupled with the code written on the Network Processor Engine core. This section explains full hardware capabilities of the Receive Module contained within the UTOPIA Level 2 Coprocessor of the IXP43X network processors. The user-accessible features of this module are described in the *Intel® IXP400 Software Programmer's Guide* and a subset of the features is described in this section.

The UTOPIA Level 2 Receive interface receives ATM cells from one or more UTOPIA-compliant physical devices.

In multiple-PHY (MPHY) mode, the UTOPIA Level 2 receive interface uses a round-robin polling routine to poll the various physical interfaces by using the five receive address lines (UTP_IP_ADDR) to determine which physical interfaces are ready to send data. The result of the polling is provided as a status to the Network Processor Engine core. The Receive Module is the entity within the UTOPIA coprocessor that implements this functionality.

The Receive Module polls a programmable number of physical interfaces as defined by the Receive Address Range (RXADDRRANGE) register. If three physical interfaces are connected to the UTOPIA Level 2 interface, a value of two can be programmed into the Receive Address Range (RXADDRRANGE) register by the Network Processor Engine core. The polling always begins at address 0 and polled sequentially to the value contained in the Receive Address Range (RXADDRRANGE) register. For example, if two is programmed into the Receive Address Range (RXADDRRANGE) register, the external physical interfaces is configured to respond to the first three physical addresses produced by the UTOPIA Level 2 UTP_IP_ADDR signals.

To allow more flexibility a logical address to physical address table is provided. The look-up table makes it possible for the three addresses that were called out above not to be in sequential order.

For example, the following logical to physical address map can be used for the above example of three physical interfaces.

- Logical Address 0 => Physical Address 3 => UTP_IP_ADDR lines = “00011”
- Logical Address 1 => Physical Address 5 => UTP_IP_ADDR lines = “00101”
- Logical Address 2 => Physical Address 7 => UTP_IP_ADDR lines = “00111”

Once the physical address is driven to all physical interfaces using the UTP_IP_ADDR signals. The physical interface that is prepared to send a cell and configured to the address signals that match the values contained on the UTP_IP_ADDR signals responds to the UTOPIA Level 2 Interface on the IXP43X network processors by driving the UTP_IP_FCI, also known as RX_EMPTY_N/RX_CLAV signal to inform the UTOPIA Level 2 Interface that the physical interface is ready to send a cell.

The Receive Port Status (RXPORTSTAT) register contained within the Receive Module stores the polling result of each physical interface. The UTOPIA Level 2 hardware uses the values stored in the Receive Port Status (RXPORTSTAT**)** Register to determine the physical interface from which the received cell is originated. The Receive Module stores the received cell along with the physical interface address that the cell has received from the Receive FIFO with some basic filtering capability if desired. The Network Processor Engine then reads the address from which the cell originated and then the cell.

The Receive Module performs an optional cell level filtering that can cause a cell to be discarded prior to being placed into the Receive FIFO. Some of these features that can be enabled or disabled include the following:

- Received cells that are too short are discarded
- Excess bytes of received cells that are too long are discarded.
- Detection of HEC errors in the cell header causes the cell to be discarded. Can be enabled/disabled.
- Detection of idle cells is discarded. This can be enabled/disabled. The definition of an idle cell is programmable by setting the appropriate values in the Receive Define Idle [RxDefineIdle] registers.

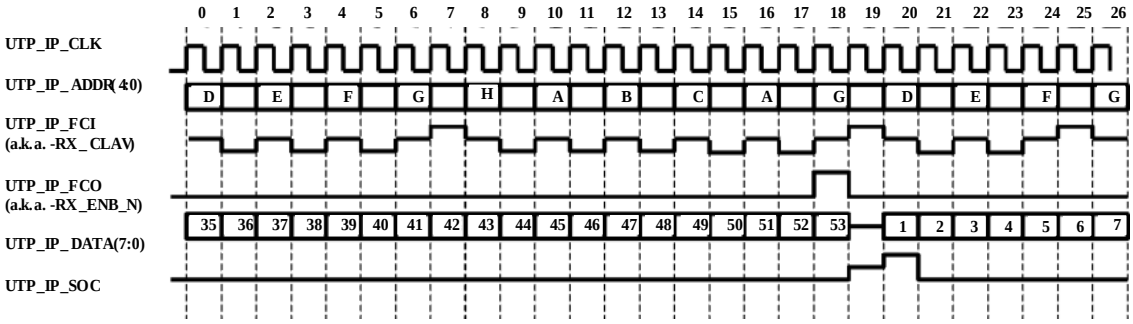
As the Receive Module places data into the Receive FIFO, the header information is passed to the Receive Pre-Hash Unit.

The UTOPIA Coprocessor Receive Pre-Hash module provides a mechanism for allowing incoming UTOPIA cells to have the ATM header looked up in a hash table to achieve faster address recognition. The hash unit takes the incoming ATM header combines it with the arriving port information and produces a header that can be read by the Network Processor Engine core. The Receive Pre-Hash Function can be enabled or disabled and can be used in single-PHY or multiple-PHY modes of operation. When used in single-PHY mode of operation, the port address should always be zero.

Figure 36 shows the reception of a cell in multiple-PHY (MPHY) mode. The following assumptions are made for the figure:

- There are eight active physical interfaces connected named A through H, which map to logical address 0 through 7.
- Physical Interface A is the currently selected physical interface for clock cycles 0 through18.
- Notice on clock 7 that the result from Physical Interface G is that Physical Interface G, which has a full cell ready for the IXP43X network processors. The UTP_IP_FCI signal flags that a full cell is ready to be sent by Physical Interface G to the IXP43X network processors, asserting the UTP_IP_FCI to logic 1 one clock after Physical Interface G has been polled.
- On clock 16, the final Physical Interface polled is the Physical Interface that is currently selected. This polling is irrelevant to the Physical Interface that is polled previously prior to this location.
- Notice on clock cycles 18 and 19 that Physical Interface G is selected as the next Physical Interface that the IXP43X network processors will receive data from that Physical Interface.

Figure 36. UTOPIA Level 2 MPHY Receive Polling



In cell level single-PHY (SPHY) mode, the physical interface indicates that a cell is ready to be sent by asserting the UTP_IP_FCI, also known as RX_EMPTY_N/RX_CLAV signal. The UTOPIA Level-2 Interface on the IXP43X network processors subsequently initiates the transfer of a cell from the physical interface by asserting UTP_IP_FCO, also known as RX_ENB_N.

In octet-level single-PHY (SPHY) mode, the UTOPIA Level 2 Interface on the IXP43X network processors indicates to the physical interface that the UTOPIA Receive interface is ready to receive bytes by asserting UTP_IP_FCO also known as RX_ENB_N signal. The physical interface indicates a valid byte is on the UTOPIA data bus by deasserting UTP_IP_FCI, also known as RX_EMPTY_N/RX_CLAV signal.

The Receive Module maintains various statistical counters. The statistics that can be maintained are on a single physical port address on a specified VPI/VCI address value. The 32-bit counters maintains the following counts:

- The number of cells received
- The number of cells with an incorrect cell size
- The number of cells containing HEC errors.
- The number of idle cells received

The counters are not cleared when read by the Network Processor Engine core. The Network Processor Engine core must perform an explicit write to the specified register to clear the counter values.

There is an overflow bit per counter to indicate that the count has **rolled over**. A maskable interrupt mechanism is used to allow the UTOPIA Level 2 Coprocessor to flag to the Network Processor Engine Core that a **roll over** has occurred.

7.5 UTOPIA Level 2 Coprocessor / NPE Coprocessor: Bus Interface

The Network Processor Engine Coprocessor Interface Module provides the necessary interface logic required for configuration, monitoring, control, and test of the UTOPIA 2 coprocessor. All the UTOPIA 2 coprocessors internal configuration and control registers, instruction registers, and FIFOs are directly accessible by the Network Processor Engine core.

7.6 MPHY Polling Routines

The UTOPIA 2 coprocessor implements a round-robin polling algorithm. The Receive and Transmit modules use a logical-to-physical address-translation table to determine the actual physical interface that is to be polled. This feature allows the designer complete control over the physical address polling sequence.

The multiple-PHY (MPHY) address translation is used by the UTOPIA Level 2 Interface on the IXP43X network processors to poll physical addresses that are not contiguous or do not start at 0.

There are two translation tables implemented. One translation table is used for receive interface polling and the other translation table is used for transmit interface polling. Each translation table is implemented as 31, 5-bit registers. Each register is addressed from 0 to 30, corresponding to one of 31 logical addresses.

The five bits of each register are used to designate a physical interface number. Therefore, if a binary value of 00101 is written to address location 0 (logical port 0) of the transmit translation table, the polling sequence asserts a five on the transmit (UTP_OP_ADDR) address lines of the UTOPIA Level 2 interface during logical port 0's turn in the polling algorithm.

For example, make the following assumptions:

- A design requires eight physical interfaces to be connected, which are configured to respond to addresses 0 through 7.
- The polling order of the physical interfaces is required to be 1, 3, 5, 7, 0, 2, 4, and 6 for both transmit and receive.

To accomplish this polling sequence:

1. The Network Processor Engine core sets the TXADDRRANGE and the RXADDRRANGE to a hexadecimal value of 0x7. This identifies that there are eight physical interfaces attached and involved in the polling sequence.
2. Define the values in both the transmit and receive translation tables as follows:
   - Address 0 (logical address 0) = A binary value of 00001
   - Address 1 (logical address 1) = A binary value of 00011
   - Address 2 (logical address 2) = Aa binary value of 00101
   - Address 3 (logical address 3) = Aa binary value of 00111
   - Address 4 (logical address 4) = A binary value of 00000
   - Address 5 (logical address 5) = A binary value of 00010
   - Address 6 (logical address 6) = A binary value of 00100
   - Address 7 (logical address 7) = A binary value of 00110

The polling sequence at the physical interface rotates from logical address 0 through 7. This event causes the physical address polling values on the UTOPIA Level 2 physical interface to be 1, 3, 5, 7, 0, 2, 4, 6.

7.7 UTOPIA Level 2 Clocks

The UTOPIA Level 2 interface on the IXP43X network processors characterizes the interface for clock speeds of 25 MHz and 33 MHz.

The UTOPIA Level 2 interface requires both transmit and receive clock inputs to be supplied from an external source. The transmit module and receive module of the UTOPIA Level 2 interface can have independent clocks running at separate clock speeds.

§ §

8.0 HSS Coprocessor

8.1 Overview

This document outlines the functional description of the HSS coprocessor.

The functionality supported by the high-speed serial (HSS) interface is tightly coupled with the code written on the Network Processor Engine (NPE) core. This chapter explains the hardware capabilities of the HSS interface contained in the Intel® IXP43X Product Line of Network Processors. The features accessible by the user are described in the *Intel® IXP400 Software Programmer's Guide* and may be a subset of the features described below.

The HSS coprocessor enables the IXP43X network processors to communicate in a Time Divisible Multiplexed (TDM) bit serial fashion with external chips. The HSS interface is a six-wire, serial interface that can operate at speeds from 512 KHz to 8.192 MHz.

The NPEs core controls the HSS interface. By programming certain parameters to the HSS interface, such as frame length/offset, frame signal polarity, and data endianness, the interfaces can be configured to support a variety of bit serial protocols. The bit streams protocols that the hardware supports are T1, E1, GCI, MVIP. The HSS hardware also has the ability to interface with certain xDSL framers. The HSS is the interface between the NPE Core and an external device (usually classified as a framer), that uses one of the above protocols.

For a list of protocols supported by the current software release, see the *Intel® IXP400 Software Programmer's Guide*.

The HSS core contains 5 Rx FIFOs and 5 Tx FIFOs, 4 Rx FIFOs and 4 Tx FIFOs are for HDLC and are each two words in length, these are further divided into two buffers, and are each 1 word in length. These two buffers act in a ping-pong fashion.

The 4 Tx/Rx FIFOs for HDLC can be configured to behave like two or one Tx/Rx FIFO, see Section 8.3.1, "FIFOs and Lookup Tables" for more details. The Tx FIFOs and the Rx FIFOs are identical in size and numbers.

The fifth Rx FIFO is for VOICE and is four words in length, this is also split into two buffers, and are each 2 words in length. These buffers also behave in a ping-pong fashion, so the software processes two words (one buffer) at a time.

The HSS core can be programmed by the NPE Core to perform the byte/frame interleaving in the Tx direction (MVIP), each core can also split up an incoming Rx MVIP stream into its constituent voice/HDLC streams (using the lookup tables described below).

Two lookup tables 128*2 bits in size are present in the HSS core, 1 for the Tx direction and the other for the Rx direction. These tables indicate if an incoming/outgoing stream consists of HDLC/VOICE/56K or unassigned timeslots (bytes), the length of the lookup table can be programmed as needed. Selecting a timeslot as HDLC tells the HSS core

that the timeslot should be stored in a HDLC FIFO; the particular HDLC FIFO depends on what location in the lookup table was programmed, see Figure 37 on page 264 for more details.

In the case where dual MVIP or normal E1/T1 protocols are operating, not all of the lookup table space is used. The frame size programmed indicates the number of locations used in lookup table.

In the case of GCI, the lookup tables should be programmed to VOICE.

To support software ATM-TC (Transmission Convergence) operations, a HEC generator and an ATM scrambler are located within the HSS, note that only two are employed in total, 1 Tx and 1 Rx.

8.1.1 High-Speed Serial Interface Receive Operation

The HSS interface contains five receive FIFOs and four receive FIFOs intended for facilitating the use of HDLC. Each is two 32-bit word in length. The four receive FIFOs are further divided into two buffers, each one 32-bit word in length. These FIFOs have hardware support added to facilitate the implementation of four HDLC channels and do not preclude the implementation of more HDLC channels using the Network Processor Engine core and the HDLC coprocessor connected.

The HSS interface is filling one buffer, as the NPE Core empties the other buffer. The fifth receive FIFO is intended for voice-processing support and is four 32-bit word in length. This receive FIFO is split into two buffers, each buffer is two 32-bit word in length.

These buffers also behave in a ping-pong fashion, so the NPE Core reads two 32-bit word at a time for processing. The location that each received byte is placed into these FIFOs is a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table characterizes each received byte as one of the following four types:

- Unassigned
- HDLC
- Voice
- 56-K mode

This characterization is assigned on a time-slot basis using Intel supplied APIs. For example, time slot 0 may be defined as a voice cell, time slot 1 as an HDLC wrapped packet, time slot 2 as an undefined time slot, and time slot 3 defined as an 56-K mode cell.

When the HSS receive interface processes the first byte (time slot 0), the look-up table indicates that this received byte is a voice cell and must be placed into the voice FIFO. Likewise, when the HSS receive interface processes the second byte (time slot 1), the look-up table indicates that this received byte is an HDLC cell and must be placed into one of the HDLC FIFOs. The actual FIFO the byte is placed in is dependent on the protocol implemented and the FIFO arrangement. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the HSS receive interface processes the third byte (time slot 2), the look-up table indicates that this received byte is an unassigned cell and must be discarded. When the HSS receive interface processes the fourth byte (time slot 3), the look-up table indicates that this received byte is a 56-K mode cell and is placed into the voice FIFO.

8.1.2 High-Speed Serial Interface Transmit Operation

The HSS interface contains five transmit FIFOs, organized exactly as the receive FIFOs for transmission. For additional details on the FIFO organization, see Section 8.1.1, "High-Speed Serial Interface Receive Operation".

Each transmitted byte is placed into these FIFOs. The data is transmitted using the HSS interface as a function of a user programmable look-up table (LUT) and the protocol that is being implemented.

The look-up table characterizes each byte to be transmitted as one of four types:

- Unassigned
- Voice
- HDLC
- 56-K mode

This characterization is assigned on a time-slot basis using Intel supplied APIs.

Assume the same example as before, time slot 0 is defined as a voice cell, time slot 1 is defined as an HDLC wrapped packet, time slot 2 is defined as an undefined time slot, and time slot 3 is defined as an 56-K mode cell.

When the HSS transmit interface is ready to process the first byte (time slot 0), the look-up table indicates that the byte to be transmitted is a voice cell and must be extracted from the voice FIFO and placed onto the HSS interface. Likewise, when the HSS transmit interface is ready to process the second byte (time slot 1), the look-up table indicates that the byte to be transmitted is an HDLC cell and must be extracted from one of the HDLC FIFOs and placed onto the HSS interface.

In the actual FIFO, the byte is extracted from is dependent upon the protocol implemented and the FIFO arrangement. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

When the HSS transmit interface processes the third byte (time slot 2), the look-up table indicates that the byte to be transmitted is an unassigned cell. Using Intel supplied APIs, the Intel XScale® Processor can program the HSS interface to transmit one of three values in an unassigned time slot:

- All zeros
- All ones
- High-impedance

When the HSS transmit interface processes the fourth byte (time slot 3), the look-up table indicates that the byte to be transmitted is a 56-K mode cell and is located in the voice FIFO. When the transmit interface detects from the transmit look-up table that the slot to be transmitted is a 56-K mode byte, only seven of the eight bits in a time slot is valid. The most-significant bit or the
least-significant bit is invalid.

Using Intel supplied APIs, the Intel XScale processor can program the invalid bit location and the value to be placed into the invalid bits location when data is transmitted. The data inserted into the invalid bit location can be programmed to be logic 0, logic 1, or tri-state. For more details, see the *Intel® IXP400 Software Programmer's Guide*.

8.2 Feature List

The features of the HSS coprocessor are listed below:

- T1/E1/GCI/MVIP protocols supported in hardware.

- HSS coprocessor split into **cores**, each core houses a set of FIFOs, stream generator and stream parser. This accommodates re-use of HSS design.
- Ten FIFOs, 5 for Tx, 5 for Rx (1 voice, 4 HDLC).
- The four HDLC FIFOs can be programmed to operate as 2 FIFOs or 1 FIFO.
- Two lookup tables are employed (1 for Tx, the other for Rx), this allows voice/HDLC channelization, also removes unassigned timeslots.
- Condition signals are used to indicate the status of the FIFOs with the HSS.
- An ATM-TC scrambler and HEC generator are supplied within the HSS.
- The line clock speed is programmable (allowing differing protocols speeds).
- Loopback facility available for debug purposes.
- Very flexible frame generation options (allows HSS to interface to many framers). The following are the programmable options:
  - The frame pulse can be set active low, active high, falling edge or rising edge.
  - The frame pulse and synchronization clock can be set as an input or an output.
  - The synchronization clock can be can be set to positive or negative edge with respect to (w.r.t.) the data.
  - The synchronization clock can be set to positive edge or negative edge with respect to the frame pulse.
  - The data polarity can be set low or high.
  - The data endianness can be programmed (w.r.t. what's transmitted or received first).
  - Open drain mode on the Tx data pins.
  - The clock can run at the data rate or double the data rate.
  - The output data can be high impedance, high or low when not driving data.
  - The NPE Core can select to use FBit or not.
  - The frame size can be programmed; the maximum value is 1,024 bits.
  - The frame pulse offset can also be programmed; the maximum value is 1,023 bits.
  - The NPE Core can detect if an unexpected frame pulse has been received (in both Tx and Rx) by the HSS.
  - The clock speed can run at 512 KHz, 1.536 MHz, 1.544 MHz, 1.568 MHz, 2.048 MHz, 4.096 MHz, 8.192 MHz depending on the protocol that is in use.

8.3 Theory of Operation

The external device communicates with the HSS coprocessor using three signals per direction, namely a frame pulse, a clock, and data bit.

The data stream consists of frames, and the number of frames depends on the protocol in use. Each frame is composed of timeslots, each of these timeslots consists of 8 bits, the number of the timeslot indicates its location within the frame, this timeslot count value is maintained within the HSS core and the NPE Core has access to this information.

Some protocols use an FBit (frame bit) at the beginning of the frame (T1). In the case where T1 is running at 1.544 MHz, the HSS reads in the received FBit, add on 7 padding bits (zeros in the 7 lsbs) and place the byte into the voice FIFO. The software reads the FIFO location like any other timeslot. When MVIP is used where T1s are

located within the MVIP frame (see “MVIP” on page 278), the HSS does not process the FBits any differently to other timeslots, and treats them as dictated by the lookup tables (assign them as voice/HDLC or unassigned).

The number of timeslots expected by the HSS core is programmable by the NPE Core. The maximum frame size is 1024 bits, the maximum frame pulse offset is 1,023 bits.

To support ATM-TC, a LFSR scrambler is also located in the HSS. The NPE Core writes an initial value to the LFSR register. The NPE Core then reads back the register. The MSb (bit 31 of the bus to the NPE Core) is an XOR operation of bits 30 and 27 of the LFSR register. The NPE Core also indicates when the LFSR register should be updated.

A HEC generator is also included. The NPE Core is used to initialize the HEC register.

8.3.1 FIFOs and Lookup Tables

This section describes the FIFOs and the lookup tables present in the HSS Core.

8.3.1.1 FIFOs

The HSS core instantiated within the HSS coprocessor contains 5 Rx FIFOs and 5 Tx FIFOs as described in Section 8.1, “Overview” on page 259.

In the Tx direction, a buffer is emptying into the external device as the other buffer is filled by the NPE Core. The HSS keeps count of the timeslot value w.r.t. the programmed frame length. When the software reads the Tx timeslot value, that value represents the timeslot due to be written by the NPE Core into the Tx buffer.

The NPE Core need not know the locations in the buffer that contain the relevant data, as the HSS automatically increments the buffer location. When the HSS detects that the Tx buffer location has been filled, the hss_tx_va_empty / hss_tx_vb_empty / hss_tx_h_empty signal goes low indicating that the buffer is now full. The NPE Core should not attempt to write more data until the HSS indicates an empty buffer.

In the case of a full Rx buffer, the software is expected to read to the end of the buffer. If more read instruction are issued after the end of the buffer is reached, the pointer does not go back to the start of the buffer. The pointer goes back to the beginning of the Rx buffer when that Rx buffer has been re-filled with received data.

When the HSS must process byte interleaved dual MVIP(2 E1s or 2 T1s are present in the stream), the 4 HDLC FIFOs should be programmed (by the NPE Core) to resemble 2 FIFOs, each of size 4 words (2 words per buffer). The software services two words when the hdlc full/empty hdlc flag asserts.

If E1/T1/GCI operation is desired, then the four HDLC FIFOs is programmed to act as one FIFO of size eight words (software services four words). In frame interleaved MVIP, the HDLC FIFOs should be programmed to resemble one FIFO also.

In byte interleaved quad MVIP, the hdlc buffers must be programmed to act like four FIFOs. Each FIFO contains two words, the software services one word only.

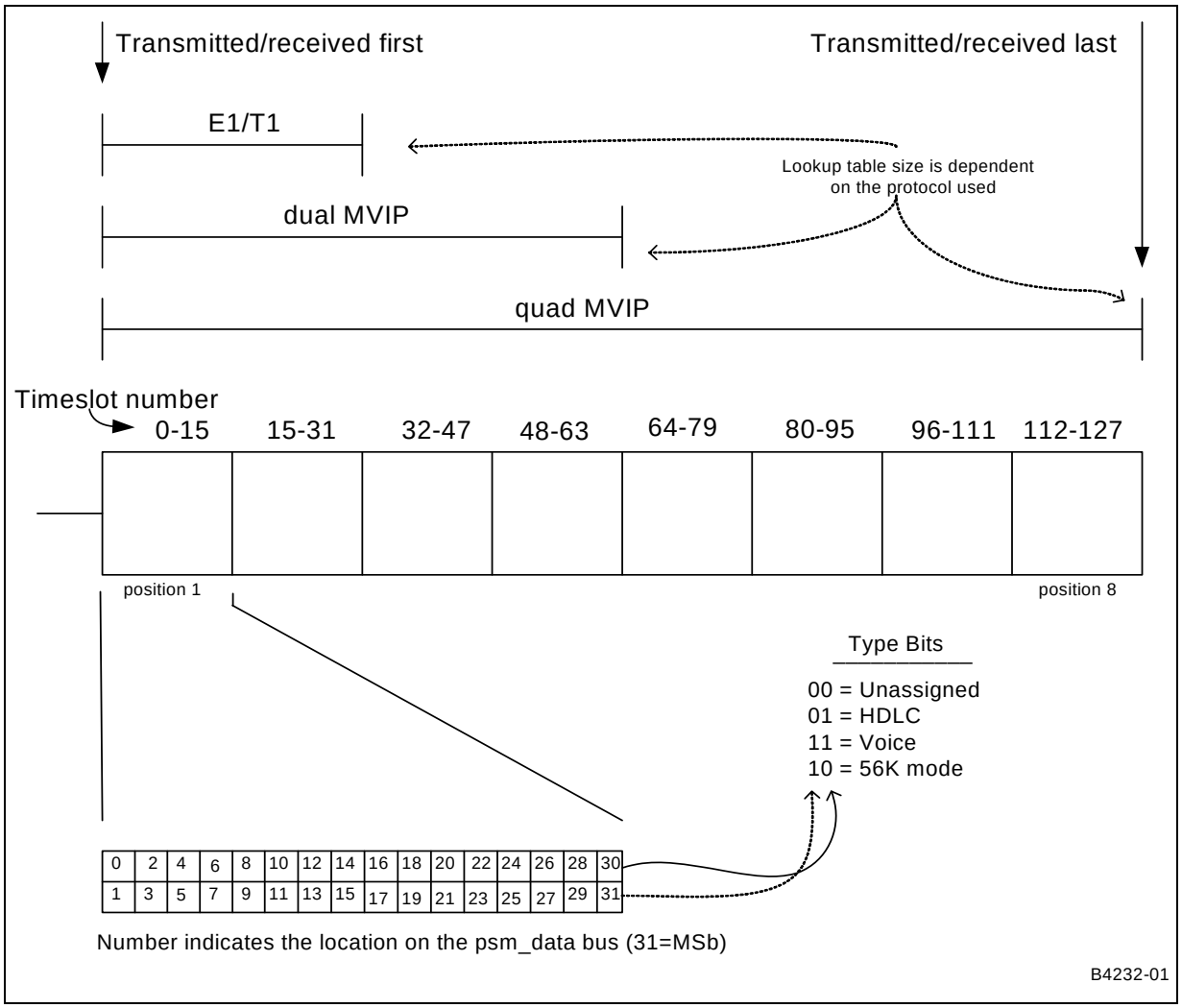
The voice FIFO contains four words, that is, two words per buffer. The software services two words when the appropriate voice condition flag asserts.

8.3.1.2 Lookup Tables

Two lookup tables (each 2*128 bits) are present in the HSS core (one for Tx, the other for Rx). The tables tell the HSS core what timeslots are HDLC/voice or unassigned within a stream.

Each lookup table requires eight write instructions (32 bit) to be completely filled by the NPE Core. This allows each timeslot to be programmed by the NPE Core to be unassigned/voice/HDLC/56Kmode. For more details on 56Kmode, see “56K Mode” on page 269. Figure 37 illustrates how each lookup table is organized.

Figure 37. Look-up Table Organization



In Figure 37, the first NPE Core write to the LUT is to position 1, the last write is to position 8.

Each FIFO controller communicates to the arbitrator about any full Rx buffers and any empty Tx buffers, when Tx FIFO (both buffers) is empty (under-run), and when the Rx FIFO (2 buffers) is full (over-run). It is up to the NPE Core to prevent the Tx FIFO from underflowing and prevent the Rx FIFO from overflowing.

In the case where the HSS has finished transmitting the first Tx buffer and the second buffer is due for transmission, the second buffer must have been completely filled previously by the NPE Core to prevent underflow. The same system applies to the Rx FIFOs.

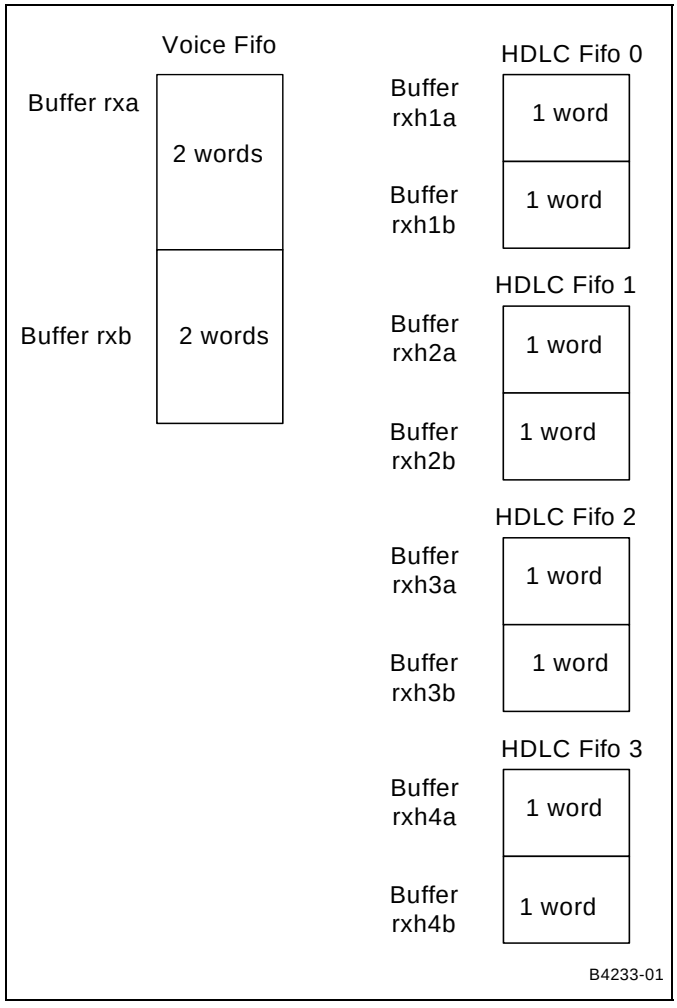
In the case of 1 or more simultaneous Rx voice 'A' buffers becoming full, the arbitrator generates the hss_rx_va_full signal as needed. The NPE Core issues the HSSrdRxVaCond instruction (result indicates the FIFO/core that activated the signal) and issue instructions as appropriate. As the HSS knows the core/buffer that the NPE Core is going to service. The HSS ensures that the FIFO related instructions is directed towards the correct buffer.

It is necessary to have eight buffer pointers per direction in the HSS core. Their usage is described as follows:

- Four of these pointers allow external devices to write to 4 HDLC buffers (Rx HDLC).
- One is for allowing an external device to write to the voice buffer (Rx voice).
- Three are for allowing the NPE Core to read from a HDLC/voice buffer (Rx).
- Four of these pointers allow external devices to read from 4 HDLC buffers (Tx HDLC).
- One is for allowing an external device to read from the voice buffer (Tx voice).
- Three are for allowing the NPE Core to write to a HDLC/voice buffer (Tx).

None of these pointers are read/writable by the NPE Core.

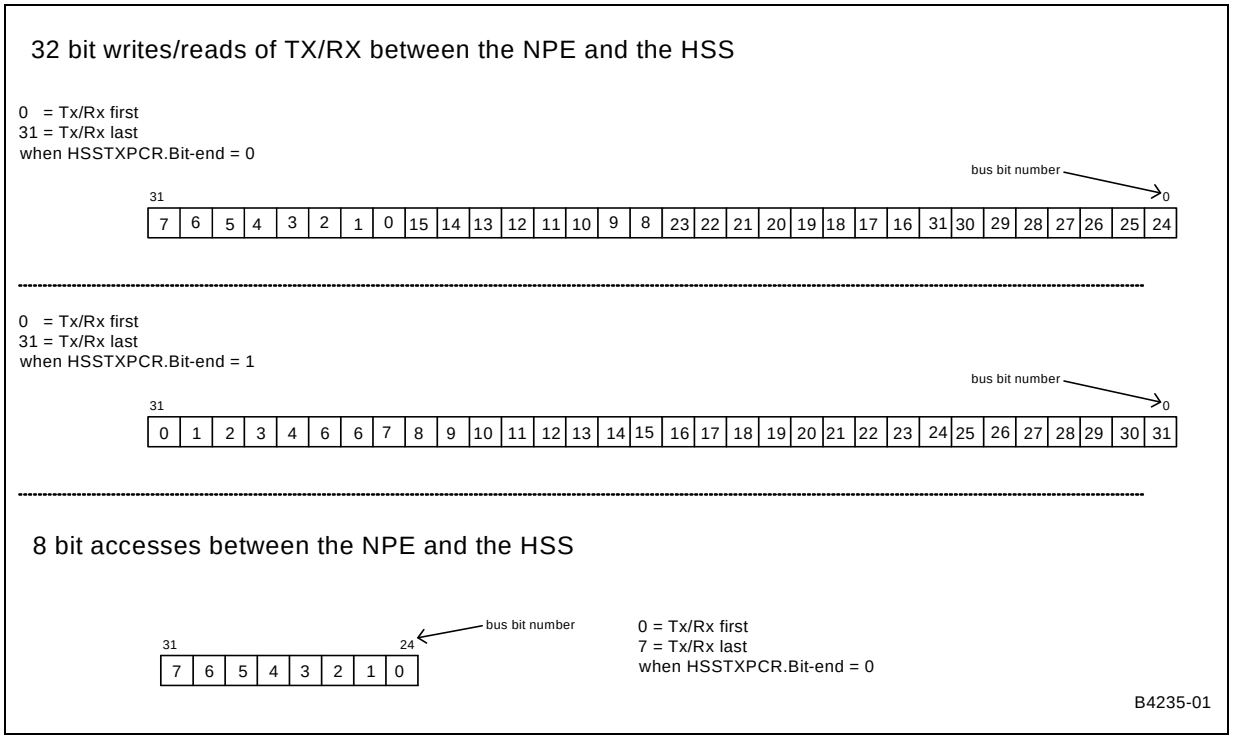
Figure 38. HSS Core Rx Buffer Structure (Identical to Tx Buffer Structure)



8.3.2 Endianness

The endianness of the data bus must be taken into account when writing data to the HSS for transmitting. The same goes for reading back received data.

Figure 39. HSS Endianness Examples



Changing the endianness only affects data to/from the FIFOs within the HSS; writes to the lookup tables are not affected.

When an 8-bit read is performed, the byte read is always the byte received first. The same principle applies to Tx data written to the HSS by the NPE Core.

8.3.3 Programmable Frame Pulse Offset and Frame Synchronization

An offset can be programmed by the NPE Core between the frame pulse and the start of the frame.

Before the HSS indicates to the NPE Core that it has received data, it must synchronize to the frame pulse.

The HSS must detect two frame pulses (in the case of gapped frame pulses, then on the correct detection of the second frame pulse) to gain synchronization. Rx frame pulses must be synchronized before any data is placed into the Rx FIFOs.

If there is no Rx offset (programmed value of 0), then the data at the start of the second frame is considered valid data and is placed into the appropriate FIFOs (LUT dependent). If there is an offset programmed, then the first data at the beginning of the third frame is considered as the start of valid data. Either way, the first data the NPE Core receives begins at the start of a frame. Therefore, no timeslot counters are needed within the HSS.

The HSS does not request data from the NPE Core until it has synchronized up to the Tx frame pulse. Meaning that the HSS must detect two consecutive frame pulses (in the case of gapped frame pulses, then synchronization is assumed on the detection of the second frame pulse). Once data has been sent from the NPE Core, the data is transmitted at the start of the next frame. If an offset is programmed, the data transmitted is the offset number of clock cycles before the frame pulse.

If the HSS core has not synchronized to the frame pulse, it does not request Tx/Rx servicing nor it indicates under/overflow conditions. Data received before synchronization is lost, regardless of what caused the loss of synchronization (reset/ frame pulse error/incorrect programming, and so on), is dropped. When not synchronized, the Tx data pin is set to high impedance.

If under-run/over-run occurs, frame synchronization is maintained if the frame pulse is still generated (if sourced internally or externally).

The behavior of the HSS is indifferent to the source of the frame pulse (in terms of synchronization), be it sourced externally or generated internally.

Figure 40. Tx Frame Synchronization Example (Presuming Zero Offset)

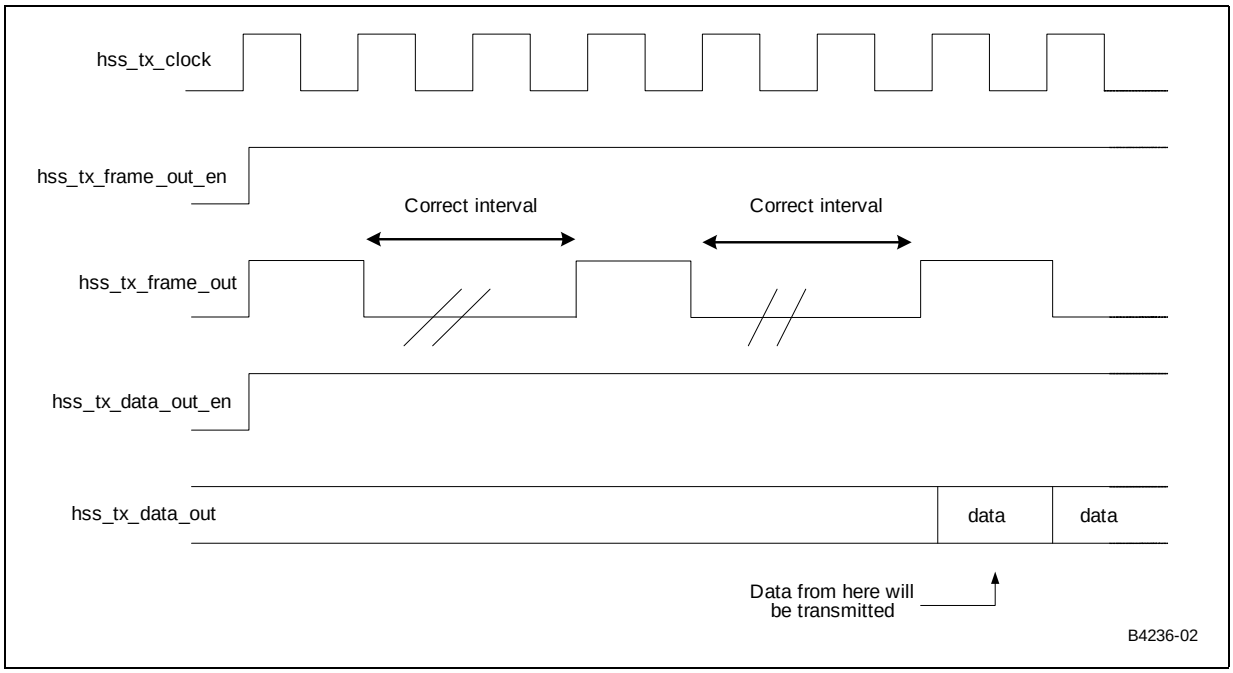
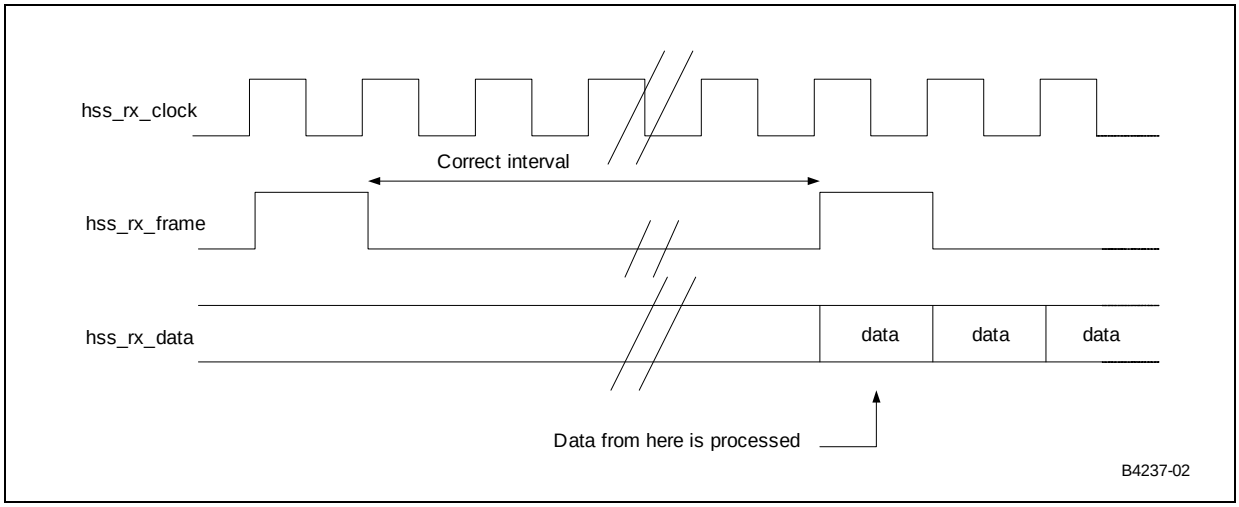


Figure 41. FRx Frame Synchronization Example (Presuming Zero Offset)



8.3.4 Underflow / Overflow / Unexpected Frame Pulse

Underflow occurs if the HSS attempts to read from a FIFO that has not been filled by the NPE Core.

If this occurs in the HSS core, then all transmission in the core in that direction is halted and an error condition is indicated to the NPE Core. Once the NPE Core has acknowledged that error condition (hss_error condition signal), the HSS core waits until the beginning of the next frame to request for data. That data is transmitted on the start of the following frame.

Within the core is multiple FIFOs, 5-2 FIFOs (4 hdlc+2voice to 1 hdlc + 1 voice). In the case where a number of these FIFOs underflow in around the same time, the first FIFO to underflow is the one indicated to the NPE Core. When the NPE Core reads the error register, all FIFO errors and the FIFOs themselves in the HSS core for that direction are cleared. Upon reading the error register, the Tx condition signals from the HSS core are cleared. The conditions signals goes high again when data is needed for transmission.

If synchronization is attained, and the NPE Core withholds sending data to the HSS Tx FIFOs for many frames, the HSS does not indicate that as an error, and the Tx condition flag remains asserted.

Overflow occurs if the HSS attempts to write to a FIFO that has not been emptied by the NPE Core. If this occurs, then all reception in the HSS core is halted until the NPE Core acknowledges this condition by reading the error register. Once this has occurred, the HSS starts filling the Rx FIFOs at the start of the next frame (offset compensated).

Within the core is multiple FIFOs, 5-2 FIFOs (4 hdlc+2voice to 1 hdlc + 1 voice). In the case where a number of these FIFOs overflow in around the same time, the first FIFO to overflow is the one indicated to the NPE Core. When the NPE Core reads the error register, all FIFO errors and the FIFOs themselves in the HSS core for that direction are cleared. When the NPE Core reads the error register, the Rx condition signals are cleared.

The HSS can indicate Tx and Rx errors in the HSS core simultaneously. Meaning that the HSS can indicate an HSS core Rx and HSS core Tx error simultaneously. During under/overflow the Tx data pin is set to the programmed unassigned value.

Unexpected frame pulse error is defined as a frame pulse that occurs where one should not exist (incorrect frame length, frame pulse offset incorrectly programmed, and so on). If this happens simultaneously with an over/underflow error, the unexpected frame pulse is given priority.

After the NPE Core reads the error register, the FIFOs in that direction and the condition flags to the NPE Core are cleared.

A frame pulse not present where one is expected is not considered as an error and normal operations are maintained.

Note: Error conditions are not signaled to the NPE Core as the core is not synchronized to the frame pulse.

8.3.5 56K Mode

Certain protocols require **bit stealing**, meaning certain bits are dropped (do not carry information). In E1 mode for example, the HSS transports at 2.048Kbps and gives 64K per channel (32 channels). 56K mode places a value (default = 0) in the MSb of each timeslot transmitted (depending on the endianness it is at the left or right side of the byte).

As this value in the MSb does not carry information about any of the channels, it therefore reduces the capacity per channel to 56 Kbps,
(8000 frames * 32 timeslots per frame * 7 bits per timeslot / 32 channels = 56 Kbyte per channel).

The value placed by the HSS into the MSb is programmable by the NPE Core. The HSS performs this by overriding FIFO control for that bit in that timeslot, thus preventing FIFO data from been accessed.

The 56K mode has no effect on Rx data, except that it is presumed that the transmitter at the far end of the line has performed 56K operations on the stream. Received **stolen** bits are placed in the Rx FIFO and are treated as data by the HSS.

8.3.6 Frameless Data Protocol Support

The HSS need not have a frame pulse supplied to transmit/receive data. In this case, the HSS should have its lookup tables/frame lengths/programmed as if it was using the 512-KHz GCI protocol. The HSS immediately assumes synchronization and uses internal logic that predicts frame pulses (for gapped frame pulses) to coordinate transmission and reception of data. The internal frame generator is not used nor is it sent out on the frame pin (for Tx/Rx).

8.3.7 Loopback

Loopback is a debug function that is used to deduce and debug problems observed with the system. It can help isolate a problem by eliminating the HSS coprocessor and all blocks further up the line as a source of a problem, if the system works correctly in loopback mode this indicates that a problem (if one exists) is within blocks further down the system.

All the Tx and Rx FIFOs should be empty before loopback is attempted. The HSS must synchronize to the frame pulse before commencing any operations. The user is free to select an internal or external frame pulse and clock.

When using the internal frame pulse/clock, the Tx frame pulse/clock is internally looped over to the Rx side. If using the external frame pulse/clock, the clock/frame pulse must be supplied on the Tx frame pulse/clock pins, this is then automatically sent to the Rx logic also, there is no need to supply any signals on the Rx pins.

No data is sent or received to/from the outside world when in loopback mode. This mode can only be directly activated by the NPE Core.

8.4 HSS Registers and Clock Configuration

There are numerous Control and Status Registers (CSRs) contained within the NPE/HSS interface of the IXP43X network processors that are used to configure the many unique HSS settings discussed in this manual. The functional details of these registers are not exposed within any documentation for the IXP43X network processors and are reserved for use by NPE firmware only. But the IxHssAcc API, provides indirect access to the these registers to allow the complete control and configuration of all HSS features enabled by a particular Intel® IXP400 Software. The *Intel® IXP400 Software Programmer's Guide* should be referenced for specific information regarding use of the IxHssAcc API.

There is one register titled the HSS Clock Divider Register that provides a means to generate a unique data clock for the HSS interface of the IXP43X network processors. The IxHssAcc API configures the HSS clock divider register with the appropriate values depending on the clock frequencies that is selected, being 512 KHz, 1.536 MHz, 1.544 MHz, 2.048 MHz, 4.096 MHz, and 8.192 MHz. This is discussed further in the next section:

8.4.1 HSS Clock and Jitter

The high-speed serial (HSS) interface on the IXP43X network processors is configured to generate an output clock on the HSS_TXCLK and HSS_RXCLK pins. But this output clock, is not as accurate and stable as using an external oscillator. That is because the HSS clock is based on the internal, 133.32MHz AHB bus. This frequency does not divide down easily. Subsequently, jitter and error are introduced into the resultant HSS output clock.

If developers are clocking a framer, DAA, or other device with a sensitive input PLL, they should use an external clock.

To provide developers with additional data, this chapter contains five tables:

- Table 103, "HSS Tx/Rx Clock Output" on page 271
- Table 104, "HSS Tx/Rx Clock Output Frequencies and PPM Error" on page 271
- Table 105, "HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization" on page 271
- Table 106, "HSS Frame Output Characterization" on page 272
- Table 107, "Jitter Definitions" on page 272

The jitter and error calculations in the following tables are valid only for Intel® IXP400 Software release 1.4 and later.

8.4.2 Overview of HSS Clock Configuration

The HSS clock is configured by using HSS API in the IXP400 software releases 1.4 and later. The clock speed is set by using the HSS API.

It is important to be aware of the jitter and frequency error on the output clock of the HSS. The six clock values in Table 103 have been pre-defined by the IXP400 software release 1.4 to minimize jitter and PPM error. The six clock values can only be selected by using the HSS API.

Table 103. HSS Tx/Rx Clock Output

HSS Clock Output Frequency	Protocol	Frame Size (Bits)	Notes
512 KHz	GCI	32	1
1.536 MHz	GCI	96	1
1.544 MHz	1 T1	193	
2.048 MHz	1 T1/E1	256	2
4.096 MHz	2 T1/E1	512	2
8.192 MHz	4 T1/E1	1,024	2

Notes:
1. These clock speeds are supported using the HSS API. But, the GCI protocol is not supported by the Intel® IXP400 Software.
2. When the frequencies of 2.048, 4.096, or 8.192 MHz are used for T1, the data rate for each T1 remains at 1.544 MHz by making certain time slots within a frame unassigned and with no data. The HSS for the IXP43X network processors can be configured to discard unassigned time slots.

Table 104. HSS Tx/Rx Clock Output Frequencies and PPM Error

HSS Tx/Rx Frequency	Min. Frequency (MHz)	Avg. Frequency (MHz)	Max. Frequency (MHz)	Avg. Frequency Error (PPM)	Notes
512 KHz	0.508855	0.512031	0.512769	-60.0096	1, 2, 3
1.536 MHz	1.515	1.536	1.55023	-60.0096	1, 2, 3
1.544 MHz	1.515	1.5439	1.55023	+60.0024	1, 2, 3
2.048 MHz	2.01998	2.0481	2.08313	-60.0096	1, 2, 3
4.096 MHz	3.92118	4.0962	4.16625	-60.0096	1, 2, 3
8.192 MHz	7.4066667	8.1925	8.3325	-60.0096	1, 2, 3

Notes:
1. These HSS Tx/Rx clock output frequencies are based on the set-up parameters in Table 103.
2. Characterization data of the HSS Tx/Rx clock output frequency data was determined by silicon simulation.
3. The parts-per-million (PPM) error rate is calculated using the average output frequency versus the ideal frequency.

Table 105. HSS Tx/Rx Clock Output Frequencies and Associated Jitter Characterization

HSS Tx/Rx Clock Output Frequency	Pj Max. (ns)	Cj Max. (ns)	Aj Max. (ns)
512 KHz	12.189	15	18.283
1.536 MHz	9.063	15	86.102
1.544 MHz	12.359	15	210.099
2.048 MHz	-8.204	15	118.957
4.096 MHz	10.9	15	190.742
8.192 MHz	12.951	15	226.634

Note: For jitter definitions, see Table 107 on page 272.

Table 106. HSS Frame Output Characterization

HSS Tx/Rx Frequency	Frame Size (Bits)	Actual Frame Length (µS)	Frame Length Error (PPM)
512 KHz	32	62.496249	-60.0096
1.536 MHz	96	62.496249	60.016
1.544 MHz	193	125.007499	60.0024
2.048 MHz	256	124.9925	-60.0096
4.096 MHz	512	62.496	-60.0096
8.192 MHz	1024	62.49624	-60.0096
Note: PPM frame length error is calculated from ideal frame frequency.			

Table 107. Jitter Definitions

Jitter Type	Jitter Definition
Period Jitter (Pj):	$Pj_{(i)} = Period_{(i)} - Period_{average}$
Cycle-to-Cycle Jitter (Cj):	$Cj_{(i)} = Pj_{(i+1)} - Pj_{(i)}$
Wander-or-Accumulated Jitter (Aj):	$Aj_{(i)} = \sum_{i} Pj$

8.5 HSS Supported Framing Protocols

The following sections provide an overview of Framing protocols supported by the HSS interface for the IXP43X network processors in addition to the recommended HSS interface setting for each protocol that are configured using the IxHssAcc API defined in the *Intel® IXP400 Software Programmer's Guide*.

8.5.1 T1

T1 is a serial data stream operating at 1.544 MHz. The stream is composed of frames and they are 8000 a second. Each frame consists of 1 frame bit and 24 time slots, each time slot is a byte in size. As there are 24 time slots per frame, T1 can carry 24 channels.

Figure 42 and Figure 43 illustrate a typical transmitted T1 frame with an active high frame synchronization (level) and a positive edge clock for generating data. The HSS clock and frame pulse is programmed to be HSS outputs or HSS inputs. Figure 42 shows an example with an internally generated frame pulse. Figure 43 shows an example with an externally generated frame pulse. An offset is programmed indicating when the Tx frame is to be transmitted. The Polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

Figure 42. T1 Tx Frame, HSS Generating Frame Pulse

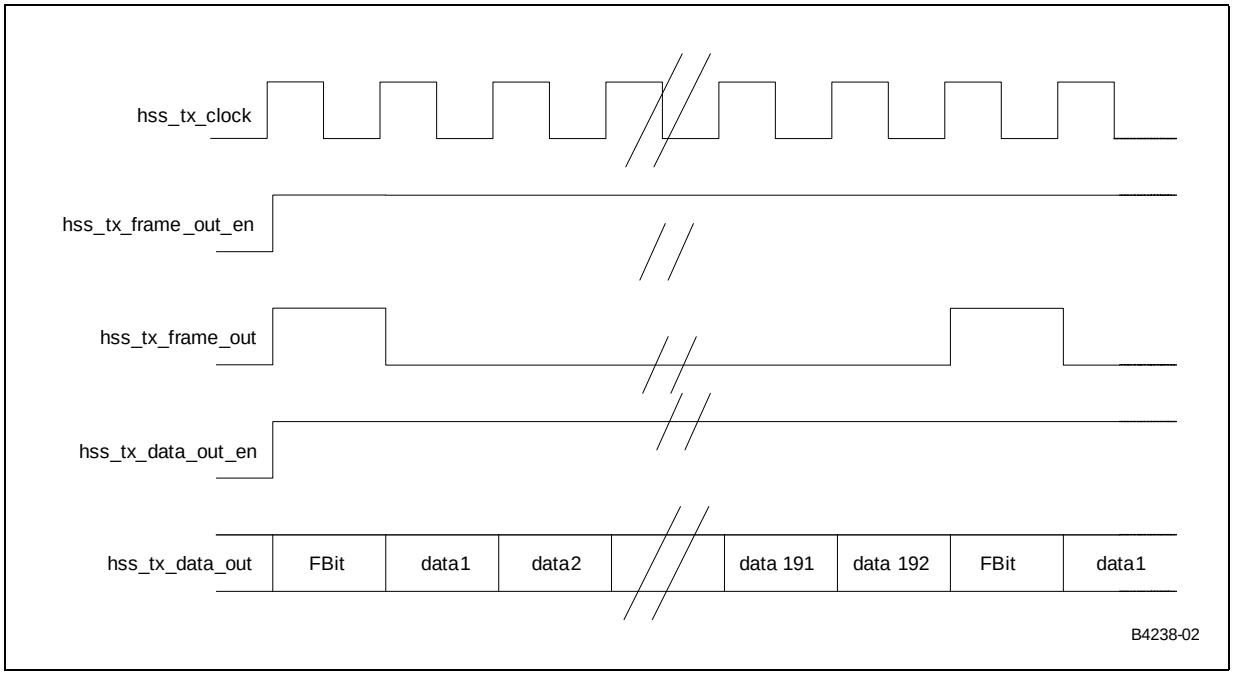


Figure 42 illustrates a typical T1 frame with active high frame synchronization (level) and a posedge clock for generating data. If the frame pulse was generated with a negedge clock, the frame pulse in Figure 42 would be located one half clock space to the right. The same location applies to the data when being generated on the negedge of the clock.

In Figure 42 and Figure 43, the FBit to be transmitted is stored in the HSS Transmit FIFO. The HSS knows the time slot in the FIFO that is holding the F Bit, as it knows from the time slot counter and frame offset, when the F Bit should be transmitted.

Figure 43. T1 Tx Frame Using External Frame Pulse

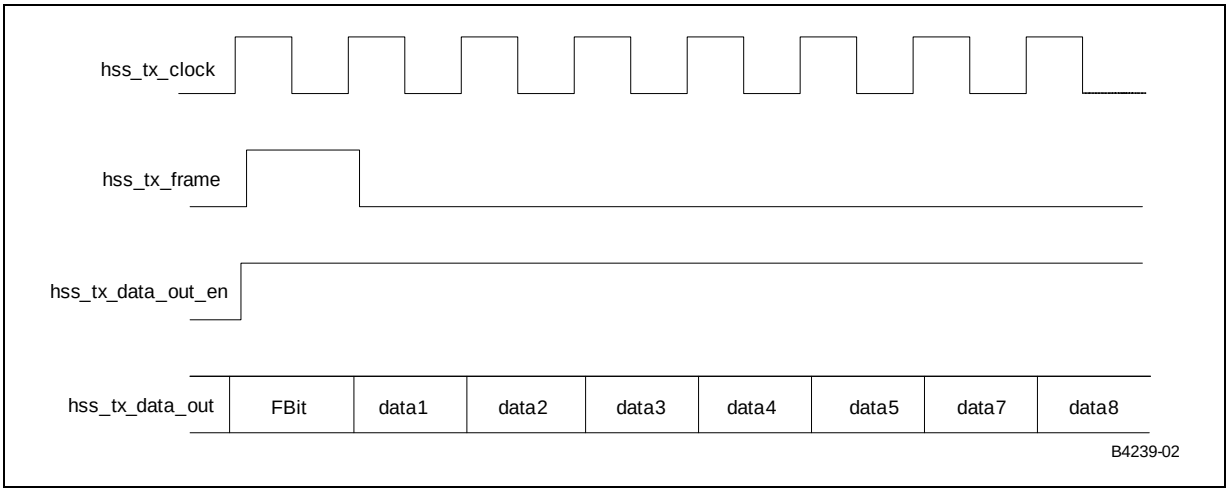
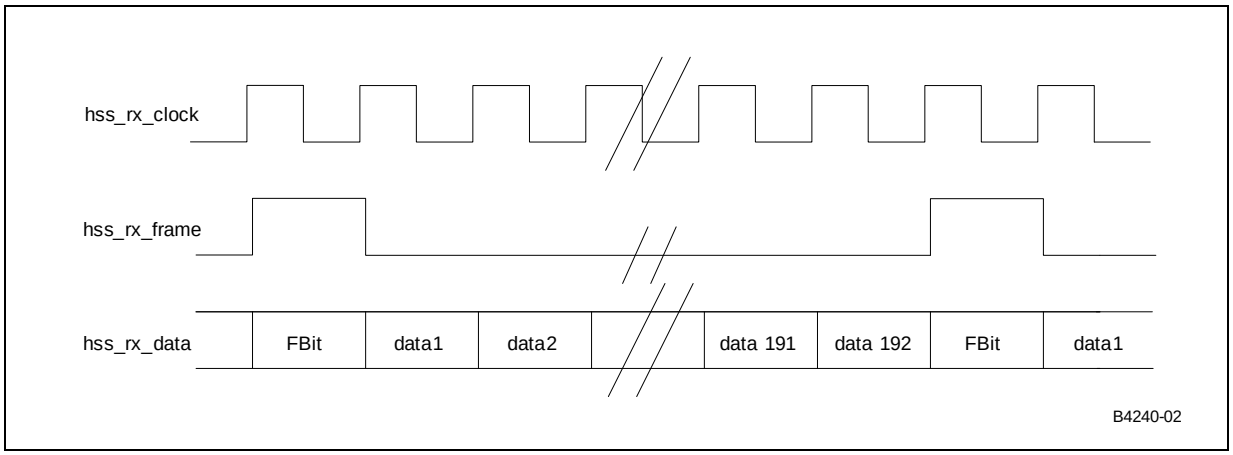


Figure 44 illustrates a typical T1 received frame with an active high frame synchronization (level) and a positive edge clock for sampling data.

In Figure 44, as stated in Section 8.3, “Theory of Operation” on page 262, the F Bit to be received is padded with 7 other bits (zeros) and placed into the HSS Receive FIFO.

Figure 44. T1 Rx Frame Using External Frame Pulse



By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for T1 operation:

- Frame size 193 bits (for T1).
- Frame synchronization simultaneous with F Bit – Tx frame offset and Rx frame offset should be set due to HSS logic; various values due to external device is accommodated.
- Select use of input/output Tx/Rx frame synchronization.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame synchronization active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to use FBit (not data for T1) at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

8.5.2 E1

This is a HSS stream operating at 2.048 MHz. The stream is composed of frames and they are 8000 a second. Each frame consists of 32 slots, each slot is a byte in size. As there are 32 slots per frame, E1 can carry 32 channels. There are no frame bits in this protocol.

Figure 45, Figure 46, and Figure 47 illustrate a typical E1 frame with an active high frame synchronization (level) and a positive edge clock for generating or sampling data. The HSS clock and frame pulse is programmed to be HSS outputs or HSS inputs. An offset is programmed indicating when the Tx frame is to be transmitted. The polarity of the received data and the level of the frame can also be programmed using the IxHssAcc API.

Figure 45. E1 Tx Frame, HSS Generating Frame Pulse

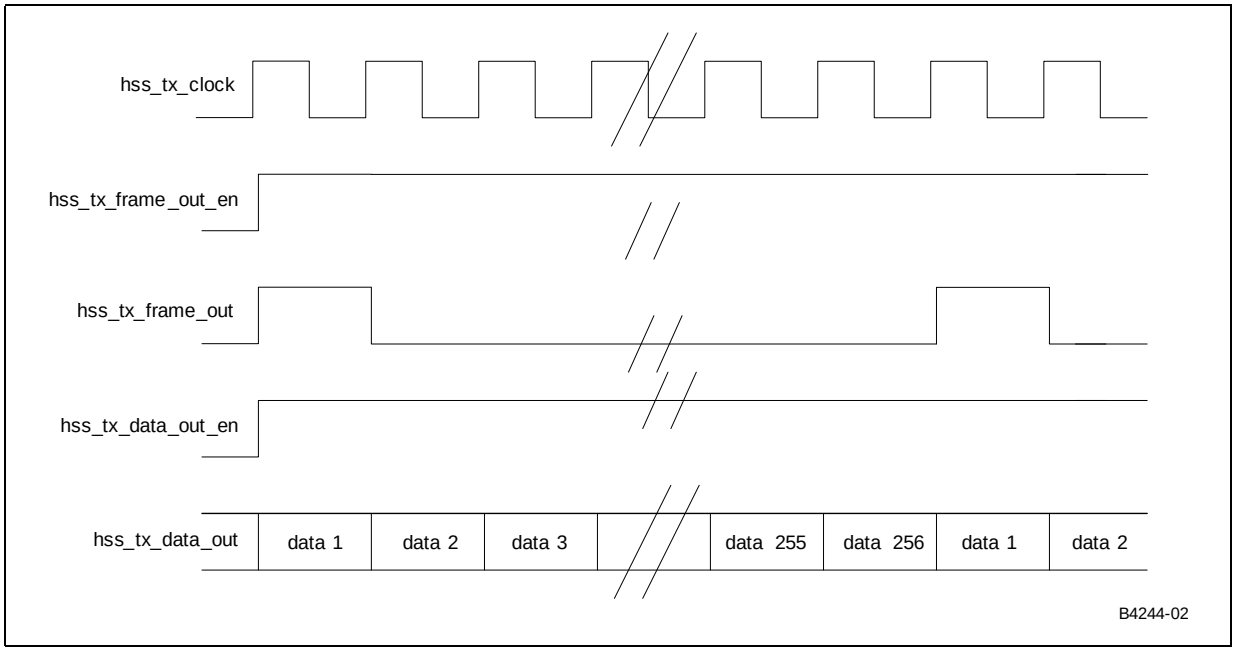
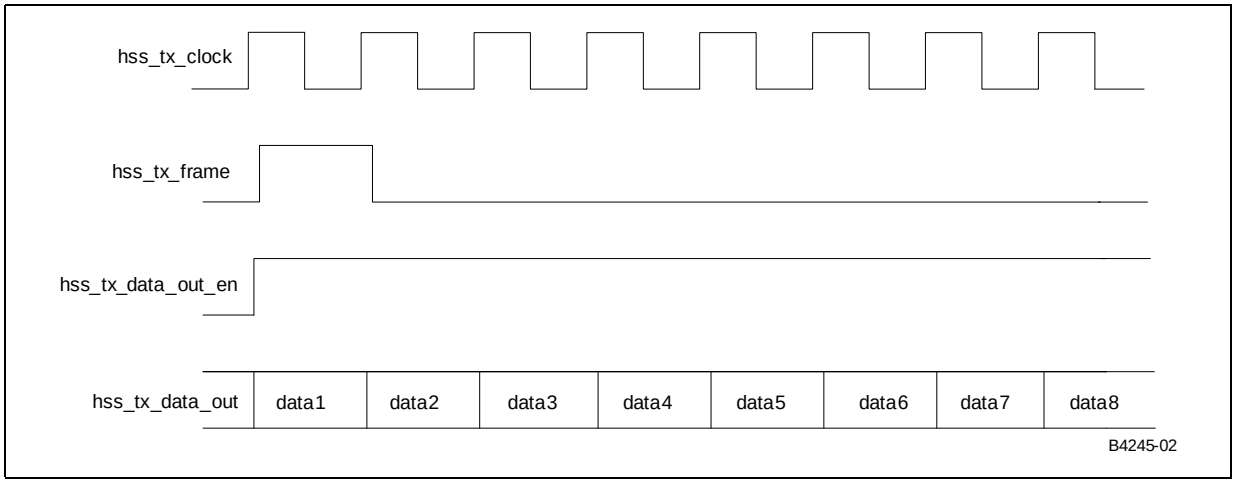
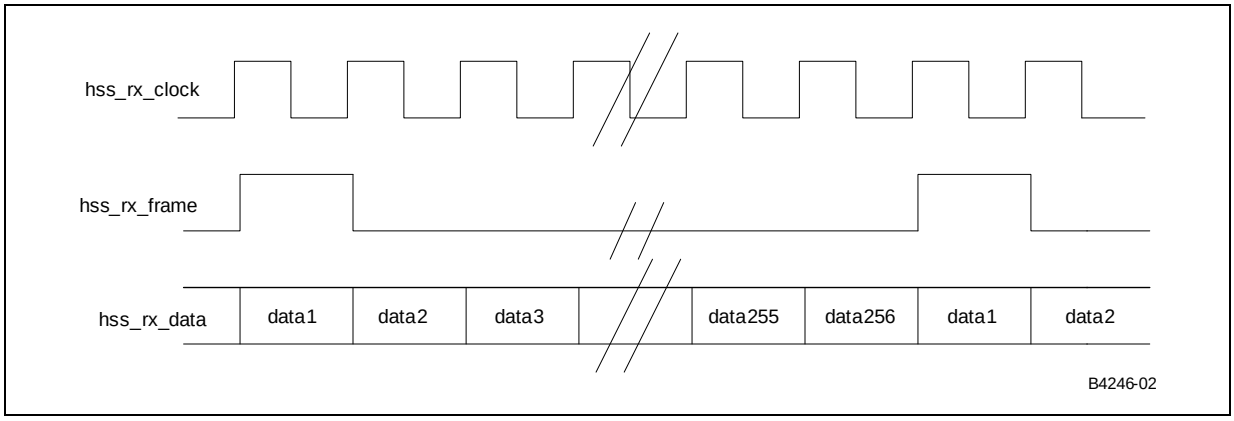


Figure 46. E1 Tx Frame, Externally Generated Frame Pulse



As with the T1 protocol, the Tx frame synchronization is generated externally. When dealing with the externally sourced frame synchronization signal, the logic on the data path within the HSS must be accounted for, thus an offset must be programmed within the HSS.

Figure 47. E1 Rx Frame, Externally Generated Frame Pulse



By using the IxHssAcc API, the following settings should be considered when configuring HSS interface for E1 operation:

- Frame size 256 bits (for E1).
- Frame synchronization simultaneous with first bit in first timeslot – Tx frame offset and Rx frame offset should be set due to HSS logic, various values due to external device is accommodated.
- Select use of input/output Tx/Rx frame synchronization.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame synchronization active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to not use FBit at frame start.
- Select value for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Configure lookup tables.

8.5.3 GCI

The HSS hardware has support to allow connection to a General Circuit Interface (GCI). This interface is also sometimes referred to as an IOM (ISDN Oriented Modular) interface. The HSS hardware supports two basic modes of operation for GCI protocol:

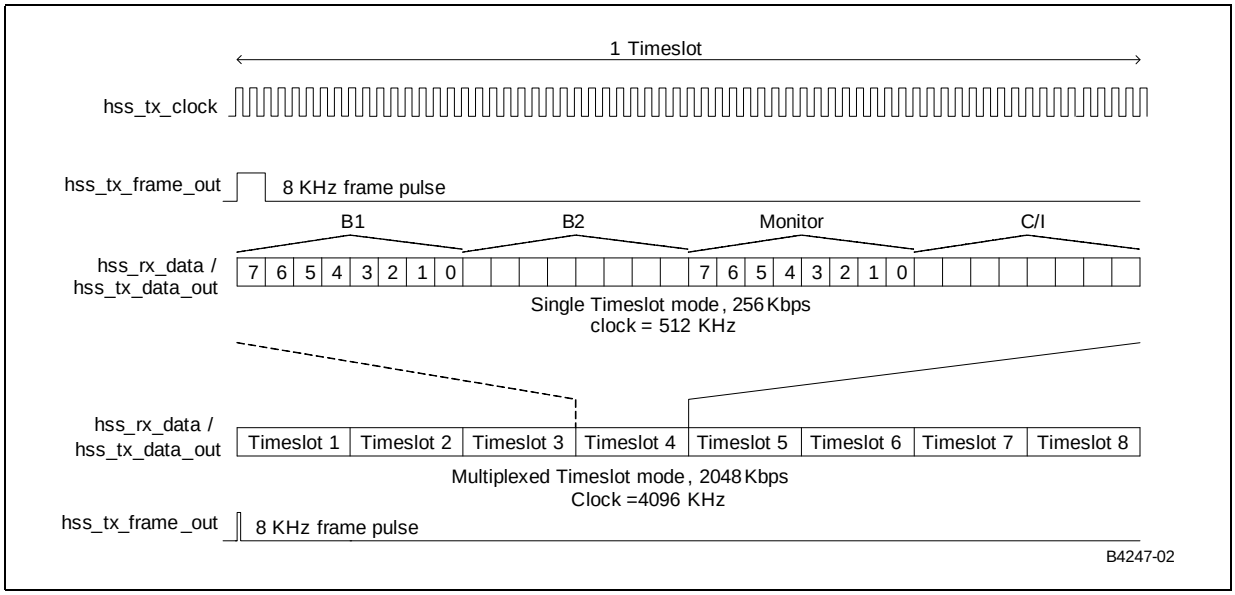
- Line-Card Mode
- Termination Mode

8.5.3.1 Line-Card Mode

The General Circuit Interface (also know as IOM) is a serial data stream consisting of 8000 frames, 4 slots per frame and each slot is a byte wide. It is used to exchange Bearer channel data (PCM-coded voice) and control information.

The HSS also supports GCI multiplexed mode, that is, a number of GCI-compatible devices is connected to the same serial bus, the numbers allowed are 1, 3, 4 and 8.

Figure 48. GCI Frames, Internally Generated Frame Pulse (Line-Card Mode)



When in multiplex mode and using 8 timeslots, GCI can support 8 ISDN transceivers (1 per timeslot).

Figure 48 illustrates the frame structure used in GCI when in line-card mode.

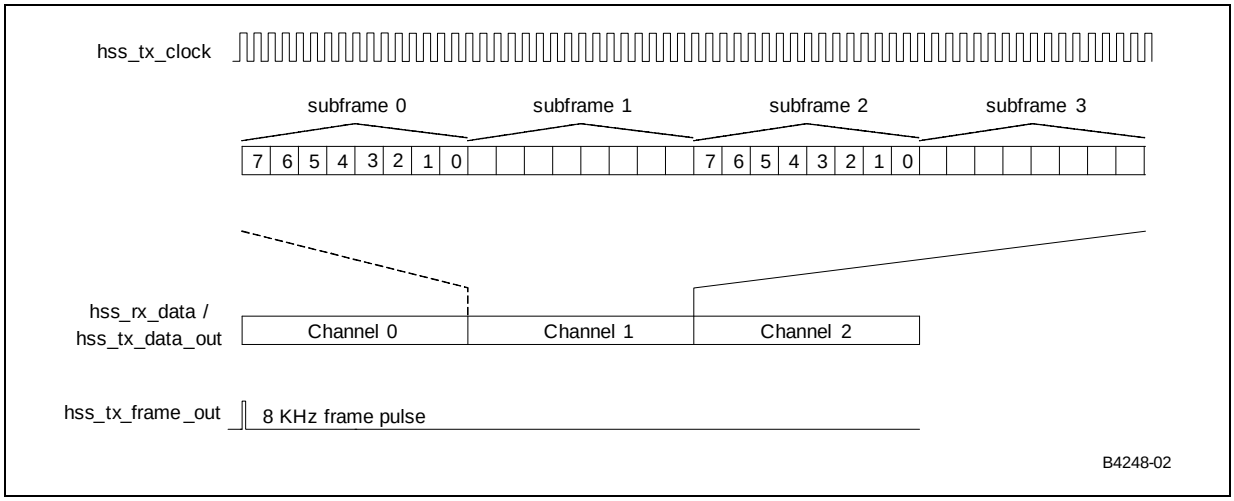
Table 108. Timeslot Configurations

Timeslot Mode	Clock Frequency	Frame Size
1	512 KHz	32 bits
3	1.536 MHz	96 bits
4	2.048 MHz	128 bits
8	4.096 MHz	256 bits

8.5.3.2 Termination Mode

This mode is a variation of Line-card mode. The frame comprises 12 sub-frames, each containing eight bits. Each 12 byte frame is repeated at 8 KHz, giving an aggregate data rate of 768 Kbps.

Figure 49. GCI Frames, Internally Generated Frame Pulse (Termination Mode)



The clock runs at 0.768 MHz, unlike other GCI protocols the clock rate is equal the data rate.

The following settings should be used when programming the HSS for GCI,

- Set frame size for GCI.
- Frame synchronization simultaneous with first bit – set Tx frame offset and Rx frame offset set to due to HSS logic, various values due to external device is accommodated.
- Select use of input/output Tx/Rx frame synchronization.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame synchronization active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select to not use FBit in the frame.
- Select value for idle timeslots on transmit.
- Configure buffer size.

8.5.4 MVIP

MVIP provides a method of interlacing E1 streams onto a single E1 line, and multiple T1 streams onto a single T1 line. Increasing the clock speed and alternating various timeslots helps provide this functionality. A single T1 line can also be mapped into an E1 line.

The NPE Core can program the HSS in the following ways:

Consider the following when programming the HSS for MVIP:

- Set frame size for MVIP.

- Frame synchronization simultaneous with first data nibble – set Tx frame offset and Rx frame offset due to HSS logic, various values due to external device is accommodated.
- Select use of input/output Tx/Rx frame synchronization.
- Select use of input/output clock, and clock speed.
- Select negative/positive clock for generating/sampling frame in transmit/receive.
- Select negative/positive clock for generating/sampling data in transmit/receive.
- Frame synchronization active level (high/low).
- MSb/LSb-first ordering for transmit and receive.
- Data polarity, maintain or invert.
- Select not to use F Bit in the frame
- Select level for idle timeslots on transmit and unused bit in 56k timeslots.
- Select buffer size.
- Set interlace mode (byte/frame)
- Set lookup tables.

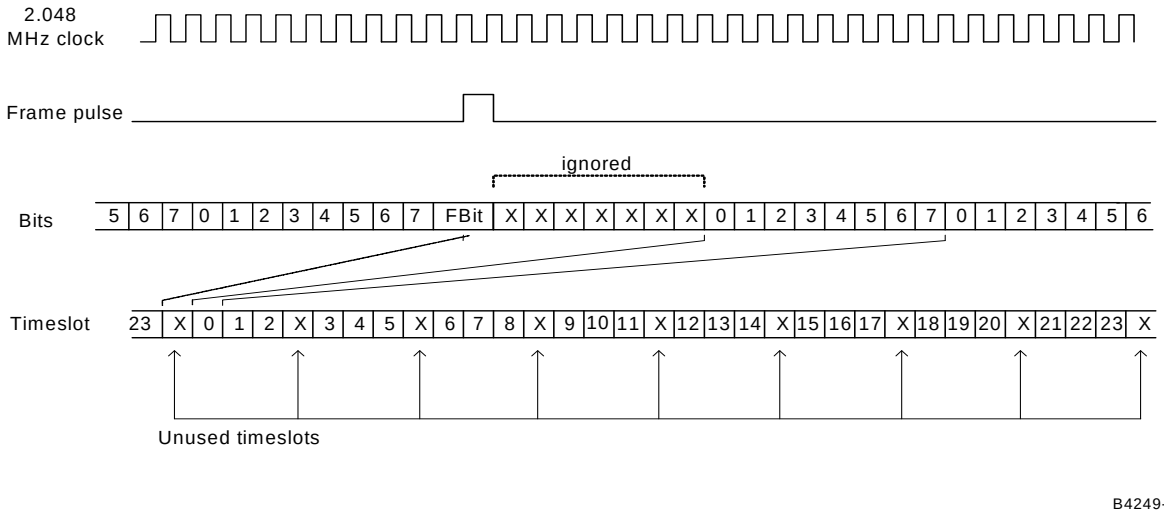
The clocks must be capable of running at 1.544 MHz, 2,048 MHz, 4.096 and 8.192 MHz for MVIP mode.

The following three sections describe the three ways how the protocol can operate. The Tx and Rx are not explicitly described below, as the Rx side of the protocol is identical to the Tx side of the protocol.

8.5.4.1 2.048-Mbps Backplane

The clock rate for this form of T1 is 2.048Mbps as opposed to the usual T1 rate of 1.544 Mbps. The data rate remains at 1.544Mbps, meaning that certain timeslots within the frame carry no data and thus are discarded by the HSS (unassigned timeslots, configured by the look up tables). Figure 50 illustrates the point.

Figure 50. MVIP, Interleaved Mapping of a T1 Frame to an E1 Frame



Note: In timeslot 0, the frame timeslot is not ignored

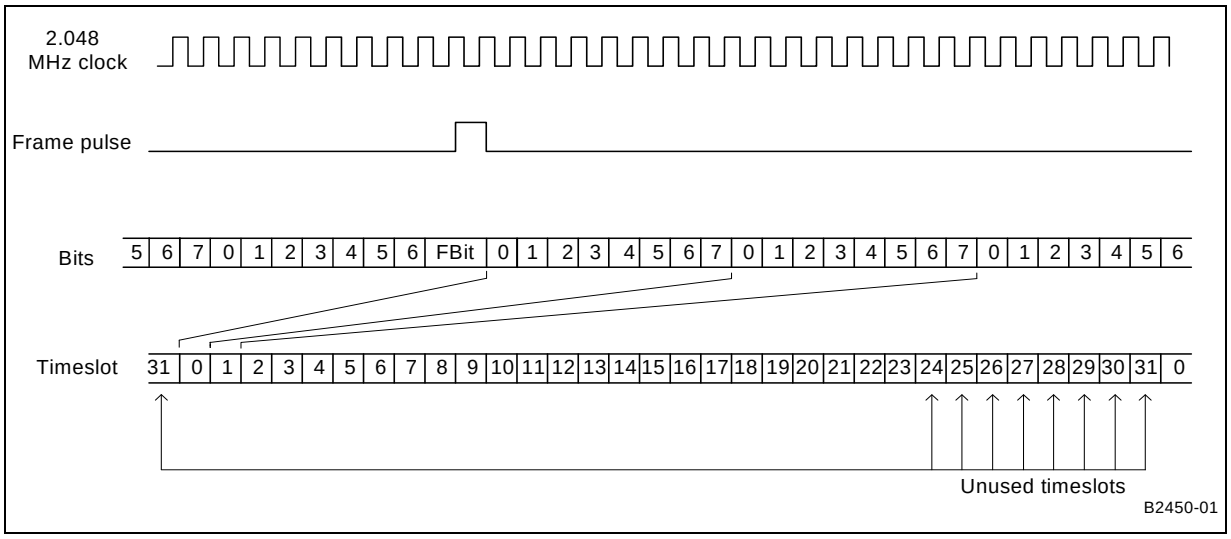
Every fourth timeslot received by the HSS is discarded, meaning it is not loaded into the FIFO and is therefore not sent to the NPE Core.

The HSS can transmit all zeros/ones (NPE Core programmable) for the duration of the unassigned timeslots.

Received unassigned timeslots are not sent to the NPE Core as they are discarded by the HSS.

Another method of sending a T1 line over an E1 line called frame mapping is illustrated in Figure 51. This method groups the 24 timeslots together and places unassigned timeslots towards the end of the frame. The F Bit is located at the last bit of the 32nd timeslot, the HSS does not treat this timeslot any differently to other timeslots, it is up the software to detect the F Bit. The NPE Core can select the method that it desires to use when implemented in NPE software.

Figure 51. MVIP, Frame Mapping a T1 Frame to an E1 Frame



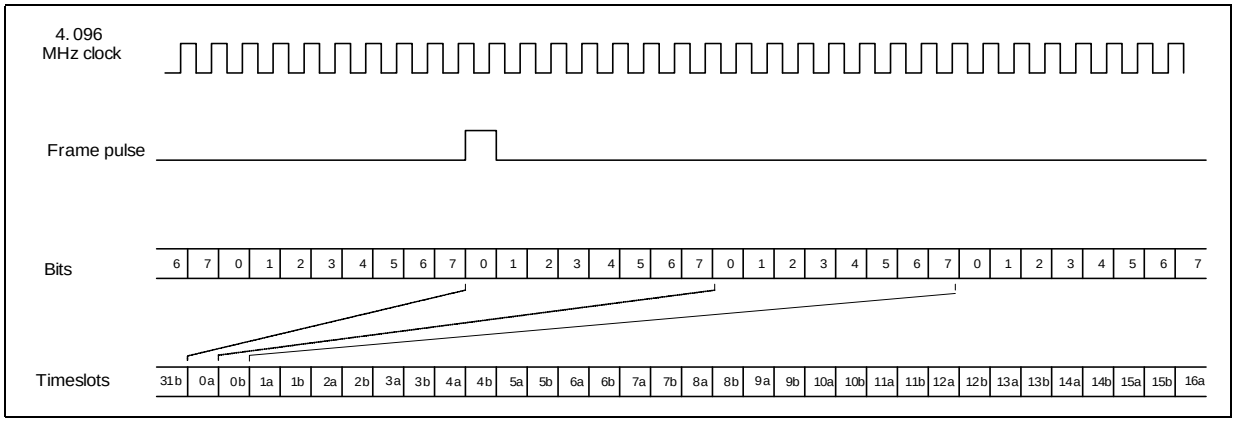
The NPE Core can program the HSS to automatically ignore (lookup table assigned) the last eight timeslots. Meaning the NPE Core does not receive the contents of the last eight timeslots. When timeslot 23 is transmitted, the next data from the NPE Core is not transmitted until timeslot zero occurs. The HSS transmits all zeros/ones for the duration of the empty timeslots (NPE Core programmable).

The NPE Core must program the HSS to indicate the method of T1 mapping that is used (if any).

8.5.4.2 4.096-Mbps Backplane

This backplane is used to transport two E1s or two T1s on a single line that allows a clock rate of 4.096 MHz. Two complete E1 frames fills this frame, therefore unassigned timeslots are not compulsory. When transporting T1 frames, unassigned timeslots are used for padding the frame due to the shorter length of the T1 frame.

Figure 52. MVIP, Byte Interlacing Two E1 Streams onto a 4.096-Mbps Backplane



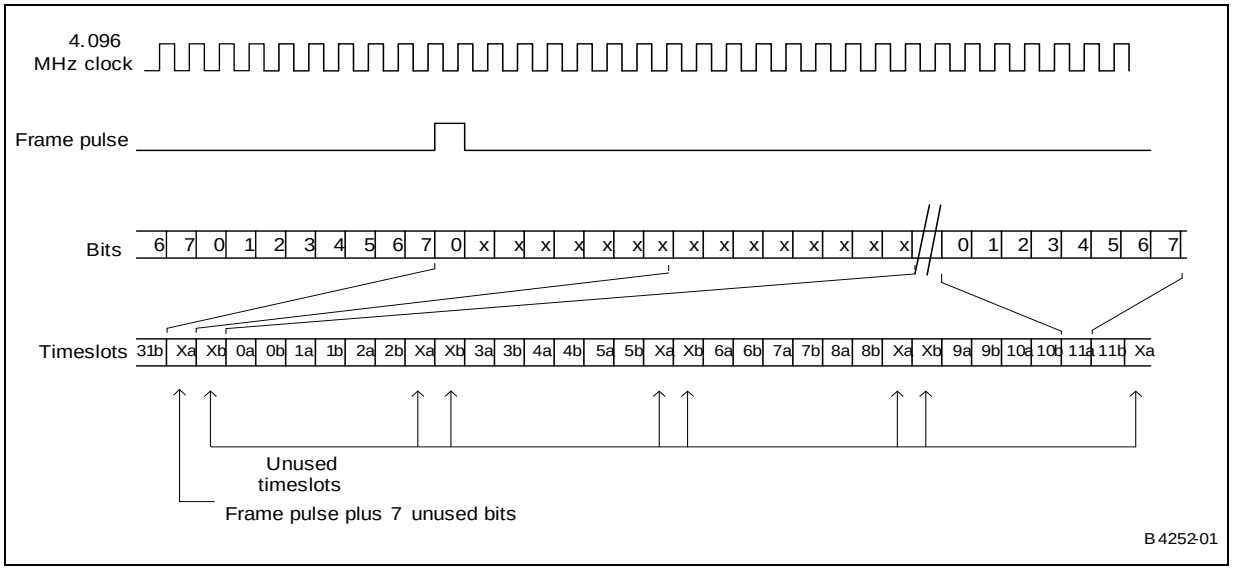
In Figure 52, the ‘a’ denotes the first E1 stream, the ‘b’ denotes the second E1 stream, the two streams are interlaced byte wise. Another method of placing E1 stream on this backplane is to process the first entire E1 stream followed by the second complete E2 stream (frame interleaving).

Figure 53 illustrates a method where two T1 frames are placed on a 4.096-Mbps backplane. It is seen that the first two timeslots are unassigned with the exception of the frame pulse. The first three timeslots of each T1 stream are then placed (interlaced) in succession on the bus, then one unassigned timeslot per T1 stream present are placed on the bus. Unassigned Rx bytes do not pass through the HSS FIFO (lookup tables give unassigned timeslots).

The HSS can also transmit unassigned timeslots, and the value is programmable. The NPE Core should only supply the contents of the T1 frames; it need not transmit unused timeslots to the HSS. The location of these unassigned timeslots are defined by the lookup table.

The backplane can contain the 2 T1 streams byte interlaced as shown in Figure 53 or the T1 stream is placed in its entirety first followed by eight unassigned timeslots (frame pulse at the last bit of the 32nd timeslot. The second T1 stream then commences followed by eight unassigned timeslots. The frame pulse is coincidental with the last bit in the 32nd timeslot. The second timeslot follows the format of the first timeslot and together take up the 64 timeslots available. Once again the HSS is programmed by the NPE Core to ignore the eight unassigned timeslots when taking the frame bit into account.

Figure 53. MVIP, Byte Interleaving Two T1 Streams onto a 4.096-Mbps Backplane

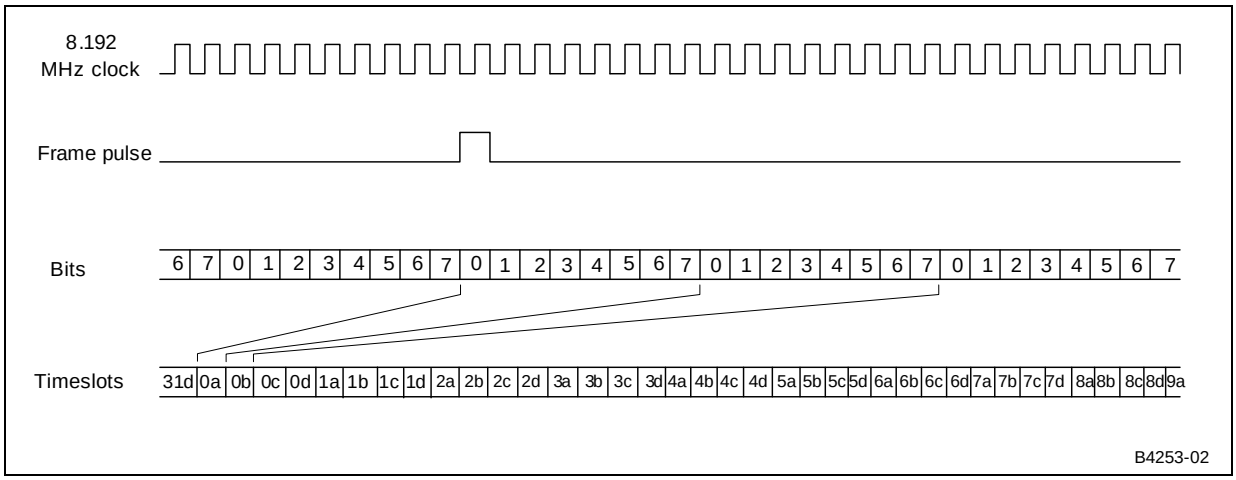


8.5.4.3 8.192-Mbps Backplane

This backplane supports 4 E1 or 4 T1 streams on a single line and requires that the interface operates at 8.192 MHz.

Figure 54 illustrates that 4 E1 streams can be byte interleaved. The frame pulse occurs at the first byte of the first E1 stream. No unassigned timeslots are necessary here as the E1 frames can completely fill all the timeslots available.

Figure 54. MVIP, Byte Interleaving Four E1 Streams on a 8.192-Mbps Backplane Bus



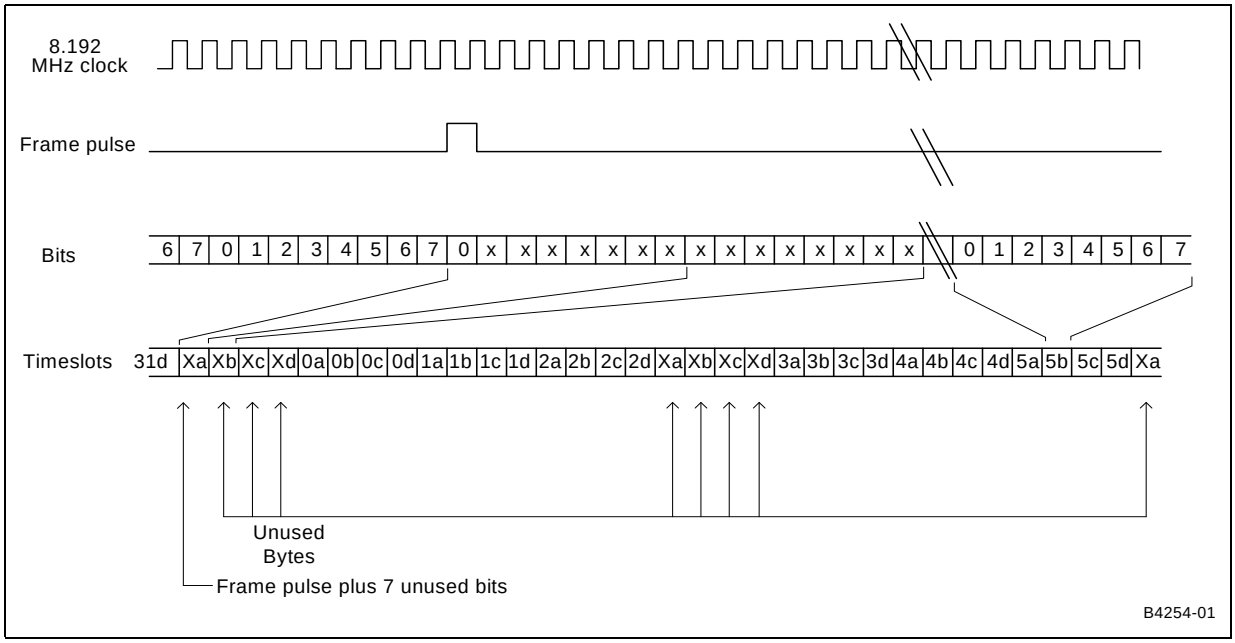
When T1 frames are placed on this backplane bus, then unassigned timeslots are required as a T1 frame is 24 timeslots wide unlike E1, that is 32 timeslots wide.

Byte interleaving involves disregarding four in every 16 timeslots (as shown in Figure 55). The first four timeslots are unassigned except for the framing pulse. The following 4 bytes are byte 0 of each T1 frame. The next 4 bytes are byte 1 of each T1 frame. The next 4 bytes are byte 2 of each T1 frame. The following 4 bytes are again unassigned. Unassigned timeslots are dictated by the look up tables.

Frame interleaving T1 frames onto this backplane bus would be to process the first T1 frame in its entirety starting with the first timeslot and finishing on the 24th timeslot, the frame bit/pulse is located on the last bit of the 32nd timeslot.

The second T1 frame is then processed and so on until all 4 frames are processed, this fills the entire 128 timeslots available. This mode is also programmable by the NPE Core.

Figure 55. MVIP, Byte Interleaving Four T1 Streams on a 8.192-Mbps Backplane Bus



§ §

9.0 PCI Controller

9.1 Introduction

The Intel® IXP43X Product Line of Network Processors contains a 32-bit, 33.33 MHz PCI interface compatible with PCI Version 2.2.

Note: The Intel® IXP42X and IXP46X Network Processors support up to 66 MHz PCI operation while the Intel® IXP43X Product Line of Network Processors supports up to 33 MHz PCI operation.

9.2 Overview

The PCI interface is capable of operating as a host or as an option. The PCI Controller supports these modes of operation by enabling access to configuration register space of the IXP43X network processors from the Intel XScale® Processor when operating as a host or from an external PCI device using the PCI bus when configured as an option. Whether configured as a PCI host or PCI option, initiator or target operations are fully supported by the PCI interface of the IXP43X network processors.

When the PCI Controller is configured as a host, an internal PCI arbiter is utilized to allow up to four devices to be connected to the IXP43X network processors without the need for an external arbiter. But, even though the internal PCI arbiter exists, the internal PCI arbiter is not required to be used when the PCI Controller is configured in host or for that matter option mode of operation.The arbiter functionality is completely independent from the PCI mode of operation. An example connection of this configuration is contained in Figure 56. The PCI arbiter function allows access to the PCI bus in a round-robin fashion.

The PCI Controller operating as an initiator can generate Memory, I/O, or Configuration PCI bus cycles. Operating as a target, the PCI Controller can accept Memory, I/O, or Configuration PCI bus cycles. The PCI Controller also contains two DMA engines to allow data movement between the PCI bus and the DDRI/II SDRAM without the aid of the Intel XScale processor.

When the PCI Controller is configured as an option, the internal PCI arbiter is disabled and REQ0/GNT0 are used to connect to an external arbiter on the PCI bus. An example of the IXP43X network processors connected in this configuration is shown in Figure 57. The PCI arbiter is enabled/disabled independently from the PCI host/option configuration.

Figure 56. PCI Bus Configured as a Host

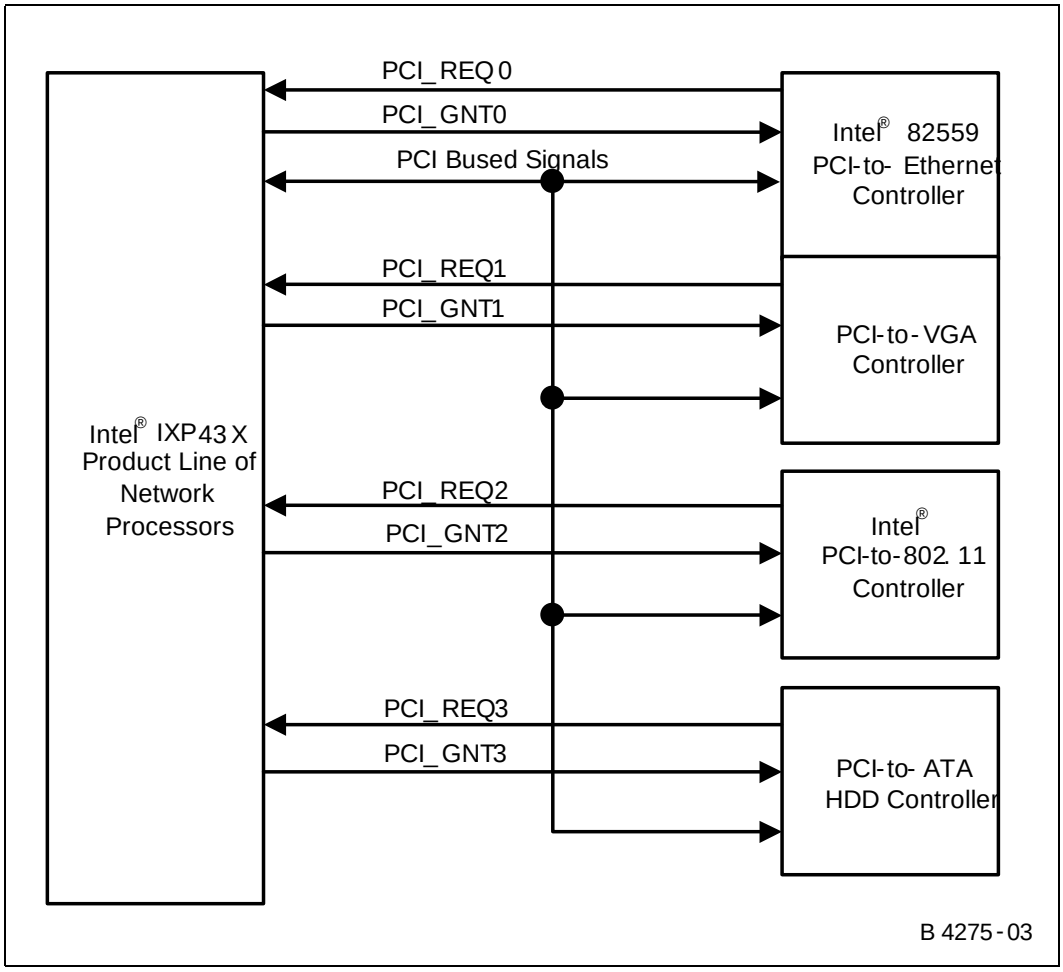
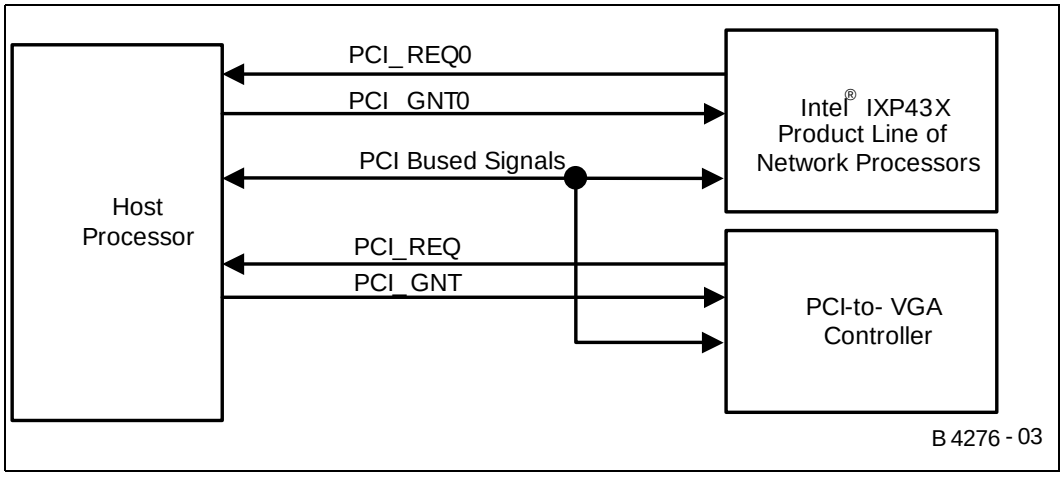
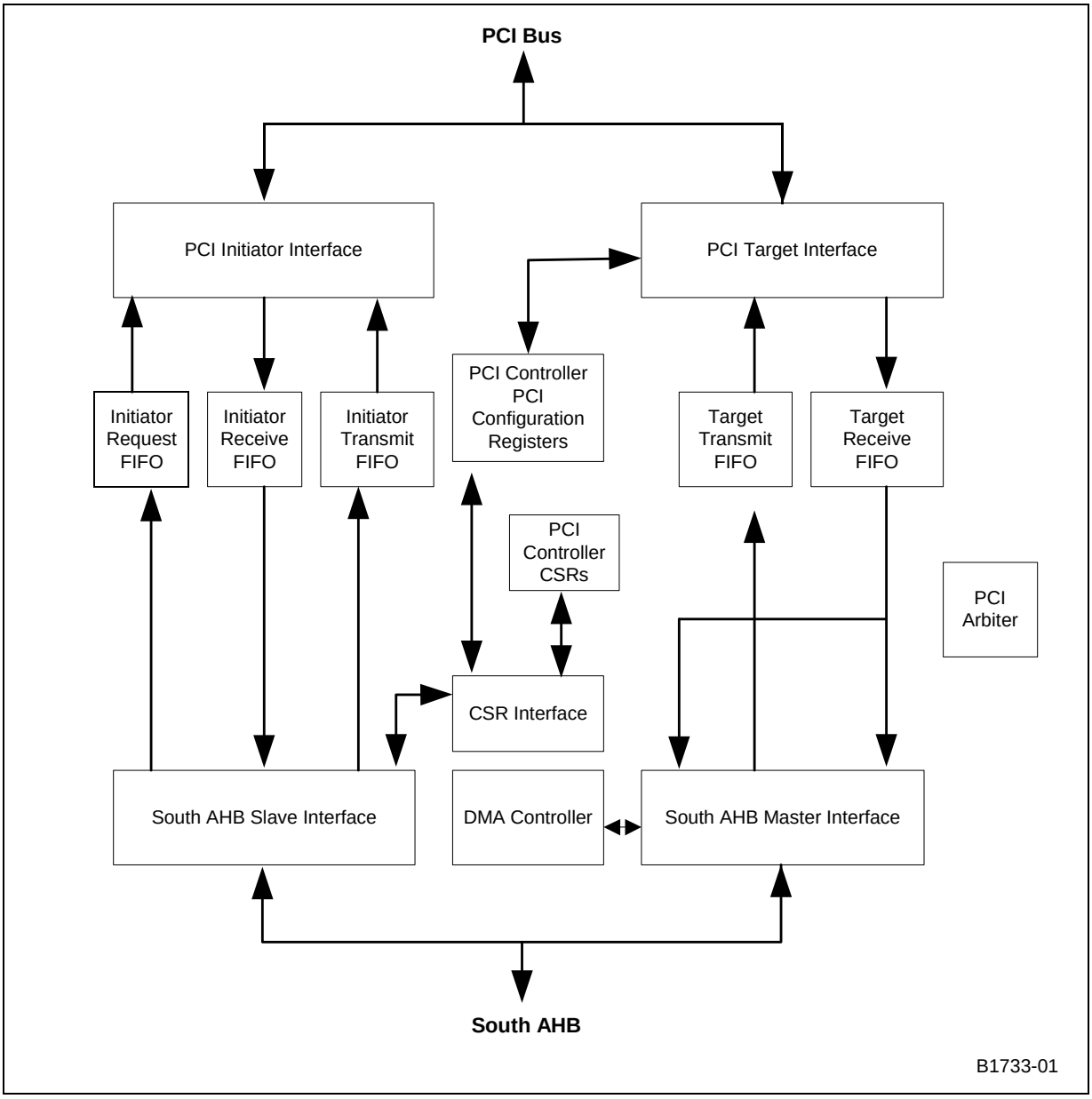


Figure 57. PCI Bus Configured as an Option



The IXP43X network processors' PCI Controller block diagram is shown in Figure 58.

Figure 58. PCI Controller Block Diagram



The PCI Controller has two main interfaces that allow interconnection between the external PCI bus and the South AHB. The external PCI bus has both a target and an initiator controller that interfaces to the PCI bus.

When an external PCI device wants to use the IXP43X network processors as the target of a PCI transfer, the PCI Controller Target Interface interprets the data and forwards the appropriate information (data/address/control) to the target interface FIFOs. The target interface FIFOs are 8 words deep. The PCI-Target Interface, in conjunction with the target interface FIFOs, uses the South AHB Master interface of the PCI Controller to provide read and write access to AHB agents, PCI Controller Configuration and status

registers. The PCI Controller Configuration Registers are accessed through configuration cycles. Target transactions is accessed directly from the PCI-Target Interface.

Table 109 lists the supported command types when the IXP43X network processors is used as a target of a PCI transaction. The PCI Target Interface does not support the following features:

- Lock cycles
- VGA palette snoop
- Dual address cycles
- Cache-line wrap mode addressing (Disconnected after first data phase of transaction)
- Type 1 Configuration space

Table 109. PCI Target Interface Supported Commands

PCI Command/Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Not Supported
0x1	Special Cycle	Not Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Converted To Memory Read
0xD	Dual Address Cycle	Not Supported
0xE	Memory Read Line	Converted To Memory Read
0xF	Memory Write and Invalidate	Converted To Memory Write

When the IXP43X network processors use an external PCI device as the target of a PCI transfer, the PCI Controller Initiator interface is used to generate the appropriate PCI bus cycles and forward the information to the PCI bus. There are three ways how PCI bus cycles is initiated:

— The DMA channels generate PCI Memory cycles. Refer to Section 9.3.3, “PCI Controller DMA” for additional details.

— AHB masters generate PCI Memory cycles using memory-mapped direct access on the AHB bus. Refer to “Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register” on page 296 for additional details.

— Interrupt Acknowledge, Special Cycle, I/O, Configuration, and single-data-phase Memory cycles are generated indirectly by AHB masters using a Non-pre-fetch CSR mechanism. Refer to Section 9.2.2, “PCI Controller Configured as Host” for additional details.

For PCI bus memory cycles, the PCI Initiator interface receives requests for PCI transfer from the PCI Controller DMA channels. For PCI bus I/O cycles and configuration cycles, the PCI Initiator interface receives requests for PCI transfer from an AHB master (a single word PCI Bus Memory Cycle is produced using this method).

Requests for PCI transfers using the PCI Controller Initiator interface are buffered in the Initiator Request FIFO and handled by the PCI Controller Initiator interface when appropriate. The Initiator interface receives the appropriate transfer information from the Initiator Request FIFO, which are the address, word count, byte enables, and PCI command type.

The Initiator Request FIFO is a four-entry FIFO allowing up to four requests to be buffered. If a request is issued that generates an initiator transaction and the Initiator Request FIFO is already full, a retry is issued to the AHB master that initiated the request.

After gathering the appropriate information, the PCI Initiator interface performs the specified transaction on the PCI bus, handling all bus protocol and any retry/disconnect situations. The data is moved from the South AHB to the PCI bus using the Initiator Data FIFOs. The Initiator Data FIFOs are eight words deep. Table 110 lists the supported PCI transaction types produced by the PCI Controller Initiator Interface of the IXP43X network processors.

Table 110. PCI Initiator Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	Supported
0x1	Special Cycle	Supported
0x2	I/O Read	Supported
0x3	I/O Write	Supported
0x4	(Reserved)	
0x5	(Reserved)	
0x6	Memory Read	Supported
0x7	Memory Write	Supported
0x8	(Reserved)	
0x9	(Reserved)	
0xA	Configuration Read	Supported
0xB	Configuration Write	Supported
0xC	Memory Read Multiple	Supported
0xD	Dual Address Cycle	Not supported
0xE	Memory Read Line	Supported
0xF	Memory Write and Invalidate	Not supported

It is important to note that the target interface and the DMA channels used for supporting the initiator interface, can contend for the use of the South AHB Master Controller. When this contention occurs, arbitration for control of South AHB Master Controller is carried out on two levels.

On the first level, the PCI Target Interface requests and the DMA requests alternate for priority access. On the first transaction the PCI Target Interface would gain access to the South AHB Master Controller's services, followed by one of the DMA channels gaining access to the South AHB Master Controller's services, then the PCI Target

interface would gain access to the South AHB Master Controller's services again, followed by the second DMA channel gaining access to the South AHB Master Controller's services, and so on.

On the second level of arbitration for the South AHB Master Controller's services, the two DMA channels alternates for priority access. DMA channel 0 would gain access to the South AHB Master Controller's services first, followed by DMA channel 1, and then DMA channel 0, and so on. This arbitration scheme balances the high-bandwidth DMA traffic with what should be lower bandwidth PCI Target Interface traffic and is only used in cases where contention exists. For instance, if there are only PCI Target Interface requests being received by the South AHB Master Controller. The PCI Target would continually get access to the South AHB Master Controller until a DMA request is detected.

The PCI Controller also contains two configuration spaces. The PCI Controller Control and Status Register (CSR) configuration space is used to configure the PCI Controller, initiate single cycle PCI transactions using the non-pre-fetch registers, operate the DMA channels, report PCI Controller status, and allow access to the PCI Controller PCI Configuration Registers. The PCI Configuration Space is a 64-byte, PCI type-0 configuration space that supports a single function.

The PCI Configuration Space is written or read using registers defined in the Control and Status Registers when the IXP43X network processors is configured as a PCI Host. An external PCI Master using PCI Configuration Cycles can write or read the PCI Configuration Space when the IXP43X network processors is configured as an option. The PCI Configuration Space is accessed by the Intel XScale processor or the PCI bus but never by both at the same time.

9.2.1 List of Features

The features of PCI Controller are listed below:

- Conforms to PCI Local Bus Specification Revision 2.2
- 32-bit, 0-33MHz PCI bus operation
- Provides initiator (master) and target (slave) PCI interfaces
- Provides AHB-to-PCI and PCI-to-AHB DMA channels
- Includes PCI bus arbiter supporting up to 4 external PCI masters using round-robin arbitration.
- Access to PCI Configuration registers from PCI or AHB busses
- Provides interrupt to processor to indicate transaction errors on the PCI or AHB bus
- Provides interrupt to processor for DMA complete and DMA error
- Provides doorbell interrupt generation capability to PCI and AHB agents
- Byte, half word, word single reads/writes, burst word reads/writes supported on AHB and PCI busses
- Generates memory, I/O, and configuration cycles as PCI master
- Provides AHB masters with full access to the 4Gbyte PCI address space, via DMA channels and/or non-prefetch transactions.
- Provides PCI-to-AHB address translation to map PCI accesses to AHB address space

9.2.2 PCI Controller Configured as Host

The IXP43X network processors can be configured as a host function on the PCI bus. Configuring the IXP43X network processors as a host does not require the internal PCI arbiter function in the PCI Controller to be enabled.

The first step to using the PCI interface in any mode of operation is to determine the mode of operation and then configure the interface. The PCI bus mode of operation is obtained by reading bit 0 of the PCI Controller Control and Status Register (PCI_CSR). If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the IXP43X network processors are required to function as an Option on the PCI bus. If bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the IXP43X network processors are required to function as the Host on the PCI bus.

Bit 0 of the PCI Controller Control and Status Register (PCI_CSR) is set by the logic level contained on Expansion Bus Address Bus bit 1 at the de-assertion of the reset signal supplied to the IXP43X network processors. The internal arbiter is enabled/ disabled based on the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of the reset signal supplied to the IXP43X network processors. The PCI Controller Control and Status Register (PCI_CSR) bit 1 captures the logic level contained on Expansion Bus Address Bus bit 2 at the de-assertion of reset.

If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 1, the internal arbiter is enabled on the IXP43X network processors. If bit 1 of the PCI Controller Control and Status Register (PCI_CSR) is set to logic 0, the internal arbiter is disabled on the IXP43X network processors.

Once the PCI Controller has determined that the mode of operation is to be host, the IXP43X network processors are required to configure the rest of the PCI bus. But, the PCI Controller must be configured before the IXP43X network processors configure the rest of the PCI bus.

The Configuration and Status Registers must be initialized and the PCI Controller Configuration and Status Registers must be initialized. (For more detail on initializing the Configuration and Status Registers, see Section 9.2.4, “Initializing PCI Controller Configuration and Status Registers for Data Transactions”. For more detail on initializing the PCI Controller Configuration and Status Registers, see “Initializing the PCI Controller Configuration Registers” on page 298.)

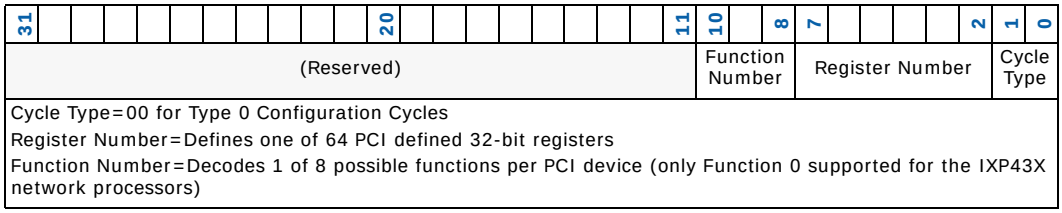
After the local Configuration and Status Register and PCI Controller Configuration and Status Registers have been initialized, the remainder of the PCI bus is ready to be configured by the hosting network processor. The IXP43X network processors will begin to initiate configuration cycles to all of the potential devices on the PCI bus.

The order and nature in which the devices are learned is up to the individual application. But, one bit must be configured prior to initiating PCI Configuration Cycles with the IXP43X network processors. Bit 2 of the PCI Control Register/Status (PCI_SRCR) Register must be set to logic 1 using the methods described in “Initializing the PCI Controller Configuration Registers” on page 298. The setting of bit 2 to logic 1 enables PCI bus-mastering capability.

Two types of PCI Configuration Cycles is generated using the IXP43X network processors: Type 0 and Type 1 Configuration Cycles. Type 0 Configuration Cycles are use to communicate to a PCI device that is contained on the same local segment that the generator of the Configuration Cycles. Type 1 Configuration Cycles are use to communicate to a PCI device that is contained on another segment of the PCI bus other than the PCI bus segment that is generating the Configuration Cycles, that is, a segment on the other side of a PCI bridge.

A PCI bus can have up to 32 devices (logically but there are loading restrictions that limit this number) per segment and up to 256 segments. Figure 59 shows the address makeup for Type 0 PCI bus configuration cycles and Figure 60 shows the address makeup Type 1 PCI bus configuration cycles.

Figure 59. Type 0 Configuration Address Phase

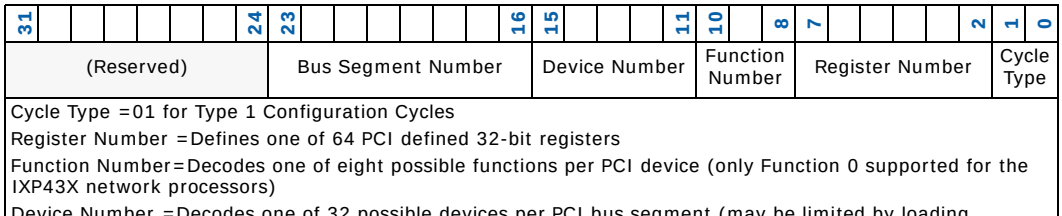


Cycle Type=00 for Type 0 Configuration Cycles

Register Number=Defines one of 64 PCI defined 32-bit registers

Function Number=Decodes 1 of 8 possible functions per PCI device (only Function 0 supported for the IXP43X network processors)

Figure 60. Type 1 Configuration Address Phase



Cycle Type =01 for Type 1 Configuration Cycles

Register Number =Defines one of 64 PCI defined 32-bit registers

Function Number=Decodes one of eight possible functions per PCI device (only Function 0 supported for the IXP43X network processors)

Device Number =Decodes one of 32 possible devices per PCI bus segment (may be limited by loading restrictions, refer to the *PCI Local Bus Specification*, Rev. 2.2)

Bus Segment Number = Decodes one of 256 possible bus segments per PCI Bus (refer to the *PCI Local Bus Specification*, Rev. 2.2)

Configuration cycle is produced by the IXP43X network processors using four 32-bit Configuration and Status Registers referred to as the Non-Pre-fetch Registers. These registers are:

- PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register
- PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register
- PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register
- PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register

The Non-Pre-fetch Register accesses can also be used to produce memory and I/O PCI bus cycle. But, these cycles can only be single cycle accesses.

Non-Pre-fetch Read Cycles is implemented by placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register and then placing the PCI Command Type and Byte Enables for the desired read cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register. The PCI Controller then initiates the proper transaction on the PCI bus to read the requested data. Then the returned data is placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.

To avoid incorrect data from being read by the initiator of this transaction, retries is issued to any AHB master that attempts to read the PCI Controller Configuration and Status Registers prior to the Non-Pre-fetch PCI read data being placed into the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register. This action assures that the next read of the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register does not contain stale data.

Non-Pre-fetch Write Cycles are implemented by:

- Placing a 32-bit PCI address in the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register
- Placing the PCI Command Type and Byte Enables for the desired write cycle in the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register
- Writing the data that is to be placed onto the PCI bus into the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register

The PCI Controller then initiates the proper transaction on the PCI bus to place the requested write data onto the PCI bus. To avoid the write data from being corrupted by new request from an AHB master, retries are issued to any AHB master that attempts to write the PCI Controller Configuration and Status Registers prior to the completion of the requested PCI transaction.

It is also noteworthy to mention that the PCI Controller does not interpret or manipulate the contents of the Non-Pre-fetch Registers. The address, command, byte enables, and write data are passed to the PCI bus as-is. For example, I/O read and I/O write requests must be set-up such that the byte-enables are consistent with the 2 LSBs of the address in accordance with the PCI local-bus specification.

9.2.2.1 Example: Generating a PCI Configuration Write and Read

This example examines the initializing of the Base Address Register.

1. Assume a PCI device has been located and now the Base Address Register configuration of this PCI device is going to be initialized. The first step is to write all logic 1s to the PCI Base Address Registers.

   Base Address Register 0 is located at hexadecimal offset of 0x10 when the ID_SEL of this device is active and the access is a PCI Bus Configuration Cycle. The intent of this exercise is to initialize this Base Address Register.

2. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.

   This value allows a write to a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1. This bit is set, assuming that ID_SEL for a given device on the local segment is selected using address bit 16. This value chosen for PCI_NP_AD follows the convention outlined in Figure 59, “Type 0 Configuration Address Phase” on page 292.

3. Write a hexadecimal value of 0x0000000B to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.

   Bits 7:4 of this register specify the byte enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be written. Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1011b signifies that a Configuration Write Cycle is being requested.

4. Write a hexadecimal value of 0xFFFFFFFF to the PCI Non-Pre-fetch Access Write Data (PCI_NP_WDATA) Register.

   This write to the Configuration and Status Registers causes a PCI Configuration Write Cycle with all byte-enables active to be initiated on the PCI bus.

5. Base Address Register 0 has been written with all logic 1s. But, only some of these bits is set to logic 1.

   Logic 1s are written to the bits corresponding to a given address space defined for the PCI device. For instance, assume that the PCI device being configured requires a 64-Mbyte address space for Base Address Register 0 used for memory transactions with no adverse side effects to reads. Only bits (31:26) would be written.

   The IXP43X network processors must now read Base Address Register 0 to determine the Address Space, Address space type (memory or I/O), and any limitations to reading this address space.

6. Write a hexadecimal value of 0x00010010 to the PCI Non-Pre-fetch Access Address (PCI_NP_AD) Register.

   This value allows a read from a Type 0 PCI configuration space address location 0x10. Notice also that address bit 16 is set to logic 1.

   This bit is set assuming that ID_SEL for a given device on the local segment is selected, using address bit 16. This device is the one that is attempting to be accessed.

7. Write a hexadecimal value of 0x0000000A to the PCI Non-Pre-fetch Access Command/Byte Enables (PCI_NP_CBE) Register.

   Bits 7:4 of this register specify the byte-enables for the data transfer. The selection of all bits to logic 0 signifies that all bytes are to be read.

   Bits 3:0 of this register specify the PCI Command Type to be used for the data transfer. A logic value of 1010b signifies that a Configuration Read Cycle is being requested. This action causes the PCI Controller to initiate the read transaction.

8. The data returned is placed in the PCI Non-Pre-fetch Access Read Data (PCI_NP_RDATA) Register.

   The returned value looks like hexadecimal 0xFC000008. This value signifies that an address space of 64 Mbyte is being requested by Base Address Register 0 of the PCI device to be mapped anywhere into the PCI address map, the address space is a Memory Space, and there are no special read conditions that apply to this address space. (See the *PCI Local Bus Specification*, Rev. 2.2 for more details.)

9. The IXP43X network processors must specify the PCI address space that this Base Address Register is going to occupy. This is done by executing a Configuration Write to bits 31:26 of Base Address Register 0, with the logical value where the address is going to reside.

   Assume we want the address to reside at PCI location 0xA0000000. A Configuration Write of 0xA0000000 is written to Base Address Register 0 of the external PCI device. No other PCI assignment is placed between PCI addresses 0xA0000000 and 0xA3FFFFFF.

When the IXP43X network processors function in Host mode of operation, all other PCI Configuration Registers contained on external PCI devices are configured or used to configure the PCI Bus using PCI Configuration Read/Write Cycles produced from the IXP43X network processors for each device on the PCI bus. Some examples of these parameters are Device Identifications, Vendor Identifications, Base Address Register, and Grant Latencies. For more details on exact settings/usage of these parameters for a given application, see the *PCI Local Bus Specification*, Rev. 2.2.

The IXP43X network processors are successfully configured as a PCI host and the PCI bus. PCI memory and PCI I/O transaction can now take place.

For more details on generating PCI Memory and PCI I/O transactions using IXP43X network processors, see "PCI Controller Functioning as Bus Initiator" on page 302. For more details on accepting PCI Memory and PCI I/O transactions using IXP43X network processors, see "PCI Controller Functioning as Bus Target" on page 310.

9.2.3 PCI Controller Configured as Option

The IXP43X network processors are configured as an option function on the PCI bus without requiring the Internal Arbiter function in the PCI Controller to be enabled. Therefore, the Internal Arbiter is enabled independently and the host/option configuration is selected independently. The option function is selected similar to the manner in which the host function is selected. For more details, see "PCI Controller Configured as Host" on page 290.

If the IXP43X network processors are configured as an option, then an external PCI Host wants to access the PCI Configuration Space of the IXP43X network processors. The PCI Host completes these accesses using PCI Configuration Cycles. But, if the IXP43X network processors receive Configuration Cycles prior to being initialized, improper PCI bus configuration may occur.

To prevent this event from occurring, the IXP43X network processors can refuse to accept configuration cycles from an external source by programming bit 15 of the PCI Controller Control and Status (PCI_CSR) Register. Bit 15 of the PCI Controller Control

and Status (PCI_CSR) Register is the Initialization Complete bit. When bit 15 is set to logic 0, the PCI Controller Target Interface issues retries to PCI Configuration cycles. When bit 15 is set to logic 1, PCI Configuration Cycles are accepted.

The Initialization Complete bit allows time for the IXP43X network processors to configure the chip prior to accepting cycles from an external PCI device. If initialization is not completed in the first 2^{25} PCI clocks after the PCI reset signal is de-asserted, the possibility exists for the external PCI Host to assume that no PCI device is resident or active at this particular IDSEL.

An access to the PCI Controller PCI Configuration Registers of the IXP43X network processors occurs when the PCI_IDSEL input is asserted, the PCI command field as represented by the PCI Command/Byte enable signals is a configuration read or write, PCI_AD[1:0] = 00 indicating a type 0 configuration cycle, and the PCI Controller Target Interface is allowed to accept Type 0 Configuration Cycles by asserting the Initialization Complete bit. The PCI Configuration Register accessed is determined by the value contained on the PCI_AD[7:2] pins during the address phase of the PCI Configuration Transaction. Accesses to the PCI Configuration Register is a single-word only. The PCI Controller Target Interface disconnects any burst longer than one word.

During reads of the PCI Configuration Registers, byte-enables are ignored and the full 32-bit register value is always returned. Read accesses to unimplemented registers complete normally on the bus and return all zeroes.

During PCI Configuration Register writes, the PCI byte-enables determine the byte(s) that are written within the addressed register. Write accesses to unimplemented PCI Configuration Registers complete normally on the bus but the data is discarded. The PCI Configuration Space supported by the IXP43X network processors is a single-function, Type 0 configuration space. For more information on the PCI Configuration Space and additional configuration details, see “PCI Configuration Registers” on page 338 and the *PCI Local Bus Specification*, Rev. 2.2.

9.2.4 Initializing PCI Controller Configuration and Status Registers for Data Transactions

To use the PCI Controller for transactions, other than single-word initiator transactions implemented by Non-Pre-fetch transactions, various registers must be set in the PCI Controller Configuration and Status Registers. The registers that must be initialized are:

- AHB Memory Base Address Register (PCI_AHBMEMBASE)
- AHB I/O Base Address Register (PCI_AHBIOBASE)
- PCI Memory Base Address Register (PCI_PCIMEMBASE).

The AHB Memory Base Address Register (PCI_AHBMEMBASE) is used to map the address of a PCI Memory Cycle Target transfers from the address of the PCI Bus to the address of the South AHB. The AHB I/O Base Address Register (PCI_AHBIOBASE) is used to map the address of a PCI I/O Cycle Target transfers from the address of the PCI Bus to the address of the South AHB. The PCI Memory Base Address Register (PCI_PCIMEMBASE) is used to map the address of direct access PCI memory-mapped transfers from the address of the South AHB to the address of the PCI Bus.

When the IXP43X network processors are the target of a PCI bus transaction, the values written or read by external PCI Bus Initiators using the Base Address Registers contained within the IXP43X network processors must be translated to an address location within the processors. The configuration of the internal memory allocation is implemented differently for each of the Base Address Registers (BAR). The following paragraphs describe the implementation for each of the Base Address Registers.

For Base Address Registers 0 through 3, that are used to complete PCI Bus Memory Cycles Target transactions, the AHB Memory Base Address (PCI_AHBMEMBASE) register is used to translate PCI Memory Cycle accesses to their appropriate AHB locations. The AHB Memory Base Address (PCI_AHBMEMBASE) register is used to determine the upper 8 AHB address bits when an external Initiator on the PCI bus accesses the memory spaces of the IXP43X network processors. The PCI Controller is configured to support four 16-Mbyte locations for PCI Target Memory Cycle transactions using the AHB Memory Base Address (PCI_AHBMEMBASE) register and the PCI Base Address Registers.

The AHB Memory Base Address (PCI_AHBMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a PCI Base Address Register

- Bits 31:24 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 0 and the first 16-Mbyte AHB memory location (AHB base 0)
- Bits 23:16 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 1 and the second 16-Mbyte AHB memory location (AHB base 1)
- Bits 15:8 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 2 and the third 16-Mbyte AHB memory location (AHB base 2)
- Bits 7:0 of the AHB Memory Base Address (PCI_AHBMEMBASE) register corresponds to PCI Base Address 3 and the fourth 16-Mbyte AHB memory location (AHB base 3).

Base Address Register 4 is used to complete accesses to internal PCI Controller Configuration and Status registers. (These registers are not the PCI Controller PCI Configuration Registers.) PCI Base Address Register 4 is used to decode that an access has been made to the Configuration and Status Register Space. There are no AHB cycles produced for this type of an access, as all accesses to this Base Address Register are internal to the PCI Controller. Therefore, an address translation register is not required.

For Base Address Register 5, that is used to complete PCI bus I/O cycles, the AHB I/O Base Address (PCI_AHBIOBASE) register is used to translate I/O PCI accesses to their appropriate AHB locations. The PCI Controller is configured to support a single 256-Byte location for PCI target I/O cycle transactions, using the AHB I/O Base Address (PCI_AHBIOBASE) register and PCI Base Address Register 5.

The AHB I/O Base Address (PCI_AHBIOBASE) register consists of a single 24-bit field. The AHB I/O Base Address (PCI_AHBIOBASE) register is used to determine the upper 24 AHB address bits, when an external initiator on the PCI bus accesses the I/O space of the IXP43X network processors.

9.2.4.1 Example: AHB Memory Base Address Register, AHB I/O Base Address Register, and PCI Memory Base Address Register

The following example is used to understand the operation of the AHB Memory Base Address Register (PCI_AHBMEMBASE), AHB I/O Base Address Register (PCI_AHBIOBASE), and PCI Memory Base Address Register (PCI_PCIMEMBASE).

1. Assume that PCI_AHBMEMBASE = 0x04010506 and PCI_AHBIOBASE = 0x000A1200.
2. Assume that the PCI Bus has gone through configuration and the Base Address Registers (BAR0 – BAR5) are set as follows:
   - BAR0 = 0xA0000000
   - BAR1 = 0xA1000000

- BAR2 = 0xA2000000
- BAR3 = 0xA3000000
- BAR4 = 0xA4000000
- BAR5 = 0xA5123400

3. An external PCI device initiates a PCI bus transfer to BAR1 of the IXP43X network processors. The PCI address looks like the following: PCI Address = 0xA100402C. The address placed on the South AHB is 0100402C.

   Notice that the third byte from the right, of the PCI_AHBMEMBASE = 0x04010506, is substituted for the A1 located in the fourth byte from the right of the PCI Address = 0xA100402C.

4. Next, an external PCI device initiates a PCI bus transfer to BAR3 of the IXP43X network processors. The PCI address looks like the following: PCI Address = 0xA3004014. The address placed on the South AHB is 06004014.

   Notice that the first byte from the right of the PCI_AHBMEMBASE = 0x04010506 is substituted for the A3 located in the fourth byte from the right of the PCI Address = 0xA3004014.

5. PCI I/O space example is an external PCI device initiates a PCI bus transfer to BAR5 of the IXP43X network processors. The PCI address looks like the following PCI Address = 0xA5123418. The address placed on the South AHB is 0x0A120018.

   Notice that the first three bytes from the right of the PCI_AHBIOBASE = 0x000A1200 is substituted for the A51234 located in the PCI Address = 0xA5123418.

6. The final example is an external PCI device initiates a PCI bus transfer to BAR4 of the IXP43X network processors. This allows access to the PCI Controller Configuration and Status Register. The PCI address looks like the following PCI Address = 0xA4000038. There is no address placed on the South AHB. This causes an access of the PCI Doorbell Register on the IXP43X network processors.

   The PCI Doorbell Register is used to generate an interrupt to the Intel XScale processor.

When the IXP43X network processors are the initiator of a PCI Bus transaction and desires the transaction to produce PCI Memory Transactions, the values can be written or read by providing a transfer to the PCI Memory Cycle Address Space defined for the IXP43X network processors. The 64-Mbyte address space defined for the PCI Memory Cycle Address Space is from AHB address location 0x48000000 to 0x4BFFFFFF.

Only four 16-Mbyte windows is enabled. The four 16-Mbyte windows are divided among the addresses as shown in Table 111.

Table 111. PCI Memory Map Allocation

Description	Starting Address	Ending Address
First 16-Mbyte window	0x48000000	0x48FFFFFF
Second 16-Mbyte window	0x49000000	0x49FFFFFF
Third 16-Mbyte window	0x4A000000	0x4AFFFFFF
Fourth 16-Mbyte window	0x4B000000	0x4BFFFFFF

The four 16-Mbyte windows translate their South AHB addresses to the PCI Bus addresses using the PCI Memory Base Address Register (PCI_PCIMEMBASE).

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register consists of four 8-bit fields. Each of these fields corresponds to a given 16-Mbyte window:

- Bits 31:24 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the first 16-Mbyte window from South AHB address 0x48000000 to 0x48FFFFFF
- Bits 23:16 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the second 16-Mbyte window from South AHB address 0x49000000 to 0x49FFFFFF
- Bits 15:8 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the third 16-Mbyte window from South AHB address 0x4A000000 to 0x4AFFFFFF
- Bits 7:0 of the PCI Memory Base Address Register (PCI_PCIMEMBASE) register correspond to the fourth 16-Mbyte window from South AHB address 0x4B000000 to 0x4BFFFFFF.

The PCI Memory Base Address Register (PCI_PCIMEMBASE) register is used to determine the upper eight PCI address bits when the IXP43X network processors access the memory spaces of external Targets on the PCI bus.

9.2.4.2 Example: PCI Memory Base Address Register and South-AHB Translation

The following example discusses the operation of the PCI Memory Base Address Register (PCI_PCIMEMBASE) and the South AHB translation.

1. Assume that PCI_PCIMEMBASE = 0xC3A24169.
2. The next example shows an access to the first 16-Mbyte window.
   The South AHB address is for the access is 0x48123450. The address presented on the PCI bus is 0xC3123450.
3. The next example shows an access to the second 16-Mbyte window.
   The South AHB address is for the access is 0x49123450. The address presented on the PCI bus is 0xA2123450.
4. The next example shows an access to the third 16-Mbyte window.
   The South AHB address is for the access is 0x4A123450. The address presented on the PCI bus is 0x41123450.
5. The next example shows an access to the fourth 16-Mbyte window.
   The South AHB address is for the access is 0x4B123450. The address presented on the PCI bus is 0x69123450.

9.2.5 Initializing the PCI Controller Configuration Registers

The PCI Base Address Registers along with any other pertinent PCI Configuration Registers, located in the PCI Controller PCI Configuration Register space, must be initialized by the Intel XScale processor when the IXP43X network processors is configured as the PCI host. The PCI Base Address Registers must be initialized by an external PCI device when the IXP43X network processors are configured as a PCI option.

The PCI Base Address Registers, along with any other registers in the PCI Configuration Space are accessed by the Intel XScale processor using three Configuration and Status Registers:

- PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register
- PCI Configuration Port Write Data (PCI_CRP_WDATA) Register
- PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.

The IXP43X network processors are a single-function, Type 0 Configuration space when functioning as a PCI option. For detailed information on the values to program the PCI Controller Configuration and Status Registers, see the *PCI Local Bus Specification*, Rev. 2.2.

The PCI Configuration Port Write Data (PCI_CRP_WDATA) Register is a 32-bit register that is used to place the data that is to be written into the PCI Configuration Space. The PCI Configuration Port Read Data (PCI_CRP_RDATA) Register is a 32-bit register that is used to capture the data that is returned from the PCI Configuration Space. The PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register provides the address, byte enables, and control for the read and write access to the PCI Configuration Space from the internal side of the IXP43X network processors.

- Bits 23:20 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the byte enables for the access to the PCI Configuration Space

  These bits directly correspond to the four - byte field associated with the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register. Table 112 on page 300 shows the mapping of the byte enables of the PCI Configuration Port Address/ Command/Byte Enables (PCI_CRP_AD_CBE) Register to the byte lane fields of the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register.

- Bits 7:2 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the address for the register access within the 64 32-bit Word PCI Configuration Space.

  The 64 32-bit Word PCI Configuration Space is shown in Table 113 on page 301.

- Bits 19:16 of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register specify the command to execute on the PCI Configuration Space. The only two commands currently defined are read and write.

  Table 114 on page 301 shows valid command codes for accessing the PCI Configuration Space. When a read command is written into the command field of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate address of the PCI Configuration register to be accessed, the data from the address requested is returned to the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register.

  A master on the AHB bus can then read the PCI Configuration Port Read Data (PCI_CRP_RDATA) Register. For example:

1. PCI_CRP_AD_CBE is written with hexadecimal 0x00300004, which causes the contents of the PCI Control Register/Status Register (PCI_SRCR) to be written into the PCI_CRP_RDATA register.

   Note that bits 23:20 are set to hexadecimal 3. For read accesses, byte-enables are ignored. Bits 19:16 are set to hexadecimal 0, which denotes a read command. Bits 7:0 are set to hexadecimal 04.

2. PCI_CRP_RDATA is read by the AHB master that requested the PCI_SRCR to be returned to the PCI_CRP_RDATA register.

When a write to the PCI Configuration Space is desired, the AHB master requesting the write must send a write command to the command field of the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register along with the appropriate byte enables and address of the PCI Configuration register to be accesses. Once the PCI_CRP_AD_CBE Register has been updated, the data that is to be written to the PCI Configuration Register is written into the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register.

The data contained in the PCI Configuration Port Write Data (PCI_CRP_WDATA) Register is written to the PCI Configuration Register specified by the address and byte enables contained in the PCI Configuration Port Address/Command/Byte Enables (PCI_CRP_AD_CBE) Register. For Example:

1. An AHB master that wants to write a particular PCI Configuration Register writes PCI_CRP_AD_CBE register first. Assume that the AHB master wants to write a hexadecimal value of 0x85008086 to the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register. The PCI_CRP_AD_CBE register is written with a hexadecimal 0x00010040.

   Note that bits 23:20 are set to hexadecimal 0. For write accesses byte enables are active low. Bits 19:16 are set to hexadecimal 1, that denotes a write command. Bits 7:0 are set to hexadecimal 40, and that addresses the PCI_RTOTTO register.

2. Next, the hexadecimal value of 0x85008086 is written to the PCI Configuration Register PCI_CRP_WDATA register, that causes the contents of the Retry Timeout/ TRDY Timeout (PCI_RTOTTO) Register to be written with a hexadecimal value of 0x85008086.

One more example demonstrates the effects of the byte-enables on write accesses to the PCI Configuration Space:

1. Assume that the objective is to update the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) without updating the TRDY terminal count value of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 7:0). The Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register is located at hexadecimal address 0x40. Also assume the value currently contained in the Retry Timeout/ TRDY Timeout (PCI_RTOTTO) Register is a hexadecimal 0x00008080.
2. The PCI_CRP_AD_CBE is written with hexadecimal 0x00D10040.

   Note that bits 23:20 are set to hexadecimal D. For write accesses this allows only byte 1 to be written (bits 15:8). Bits 19:16 are set to hexadecimal 1, that denotes a write command. Bits 7:0 are set to hexadecimal 40, and that addresses the PCI_RTOTTO register.

3. Assume that the AHB master wants to write a hexadecimal value of 0x0000AB00 to the second byte of the retry section of the Retry Timeout/TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8). The PCI_CRP_WDATA is loaded with a value of 0x0000AB00, that causes the contents of the retry section of the Retry Timeout/ TRDY Timeout (PCI_RTOTTO) Register (Bits 15:8) to be written with a hexadecimal value of 0x0000AB00. The value that is now contained within the Retry Timeout/ TRDY Timeout (PCI_RTOTTO) Register is 0x0000AB80. Notice that only one byte of data was manipulated.

Table 112 shows the PCI Byte Enables Byte Lane Mapping (accesses to the PCI Configuration Space from within the IXP43X network processors) when using the CRP access mechanism.

Table 112. PCI Byte Enables Using CRP Access Method (Sheet 1 of 2)

PCI_CRP_AD_CBE(23:20)	PCI_CRP_WDATA [31:24]	PCI_CRP_WDATA [24:16]	PCI_CRP_WDATA [15:8]	PCI_CRP_WDATA [7:0]
0000	X	X	X	X
0001	X	X	X	
0010	X	X		X
0011	X	X		
0100	X		X	X
0101	X		X	
0110	X			X
0111	X			
1000		X	X	X
1001		X	X	

Table 112. PCI Byte Enables Using CRP Access Method (Sheet 2 of 2)

PCI_CRP_AD_CBE(23:20)	PCI_CRP_WDATA [31:24]	PCI_CRP_WDATA [24:16]	PCI_CRP_WDATA [15:8]	PCI_CRP_WDATA [7:0]
1010		X		X
1011		X		
1100			X	X
1101			X	
1110				X
1111				

Table 113. PCI Configuration Space

Offset	Register Name	Description
0x00	pci_didvid	Device ID/Vendor ID
0x04	pci_srcr	Status Register/Control Register
0x08	pci_ccrid	Class Code/Revision ID
0x0C	pci_bhlc	BIST/Header Type/Latency Timer/Cache Line
0x10	pci_bar0	Base Address 0
0x14	pci_bar1	Base Address 1
0x18	pci_bar2	Base Address 2
0x1C	pci_bar3	Base Address 3
0x20	pci_bar4	Base Address 4
0x24	pci_bar5	Base Address 5
0x28	reserved	(Reserved)
0x2c	pci_sidsvid	Subsystem ID/Subsystem Vendor ID
0x30-38	reserved	(Reserved)
0x3C	pci_latint	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line
0x40	pci_rtotto	Defines retry timeout and trdy timeout parameters

Table 114. Command Type for PCI Controller Configuration and Status Register Accesses

Command Value pci_crp_ad_cbe[19:16]	Command Type	Description
0x0	Read	Initiates a read of the PCI Controller Configuration and Status Register Accesses
0x1	Write	Initiates a write to the PCI Controller Configuration and Status Register Accesses
0x2 – 0xF	(Reserved)	Reserved for future use. Use of these values produce unpredictable results.

9.2.6 PCI Controller South AHB Transactions

The PCI Controller provides access to internal functionality within the IXP43X network processors. The PCI Controller provides access to the South AHB through the AHB Target Interface and the AHB Master Interface. The AHB Target Interface is used to

accept transaction request from other AHB Masters. The AHB Master Interface is used to initiate transaction requests to other AHB Targets. The two DMA channels and the PCI Target Interface uses the AHB Master Interface.

The AHB Target Interface can accept 8-bit (1 Byte) transactions, 16-bit transactions, and 32-bit transactions. Due to the South AHB not using byte enables, all 16-bit transactions to the PCI Controller AHB Target Interface must be implemented as consecutive-byte addresses. AHB protocol requires that 16-bit accesses be half-word address aligned, address bit 0 must be 0. Inability to do this results in multiple byte wide transactions.

The AHB Master interface initiates 8-bit (1 Byte) transactions and 32-bit (word) transactions only. The DMA engines initiates only 32-bit transactions. PCI Target Interface initiated transactions are 32-bit transactions. Sub 32-bit transactions, initiated by the PCI Target Interface are implemented as multiple 8-bit transactions initiated by the PCI Controller AHB Master on the AHB. For information on prioritization of the three functional blocks that use the PCI Controller AHB Master Interface, see next section that describes "PCI Controller Functioning as Bus Initiator" on page 302.

9.2.7 PCI Controller Functioning as Bus Initiator

The IXP43X network processors are used to initiate PCI transactions in one of the following three ways:

- Using the Non-Pre-fetch Registers described in "PCI Controller Configured as Host" on page 290

  The Non-Pre-fetch Registers allow various single 32-bit word PCI Cycles to be produced and also 8 and 16 bit transfers. The Non-Pre-fetch Registers is used to initiate Type 0 Configuration Cycles, Type 1 Configuration Cycles, Memory Cycles, I/O Cycles, and Special Cycles.
- Writing to the PCI Memory Cycle Address Space located between AHB address 0x48000000 and 0x4BFFFFFF as described in "Initializing PCI Controller Configuration and Status Registers for Data Transactions" on page 295
- Using the PCI Controller DMA channels, described in "PCI Controller DMA" on page 325

The remainder of the section shows example of each cycle type that is initiated.

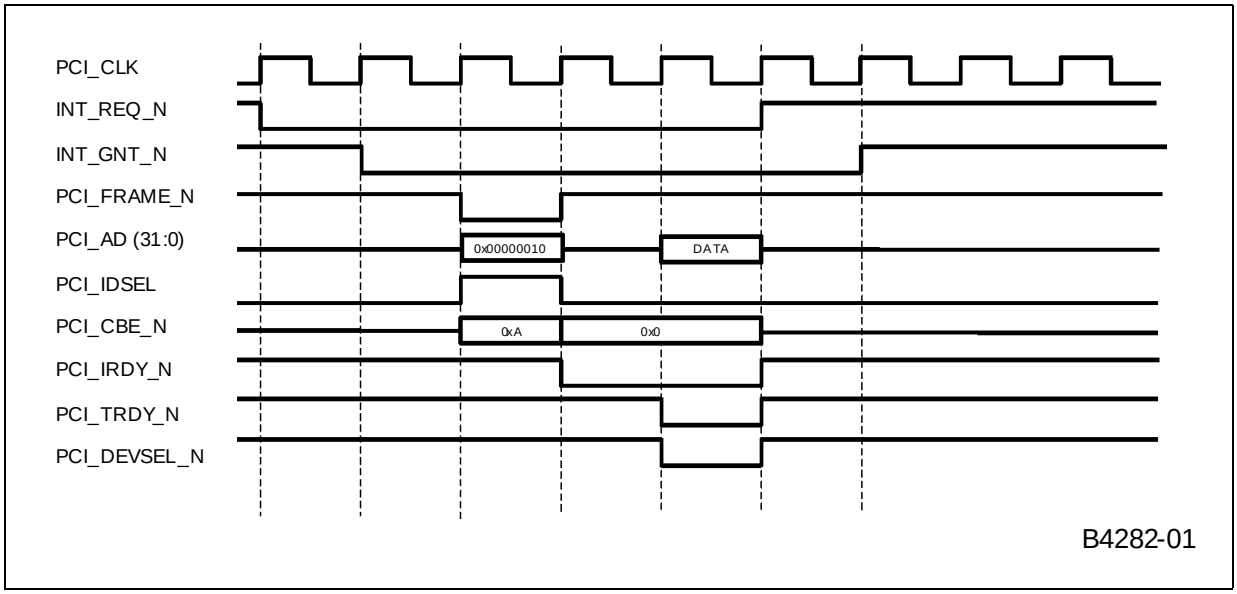
The details in this section are provided to understand some functional aspects of the PCI Controller on the IXP43X network processors. Refer to the *PCI Local Bus Specification*, Rev. 2.2. for complete information.

9.2.7.1 Initiated Type-0 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The Configuration transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function 0, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It is driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xA, written on the PCI_CBE_N bus during the PCI Bus address phase, signifies that this is a PCI Bus Configuration Read Cycle.

Figure 61. Initiated PCI Type-0 Configuration Read Cycle

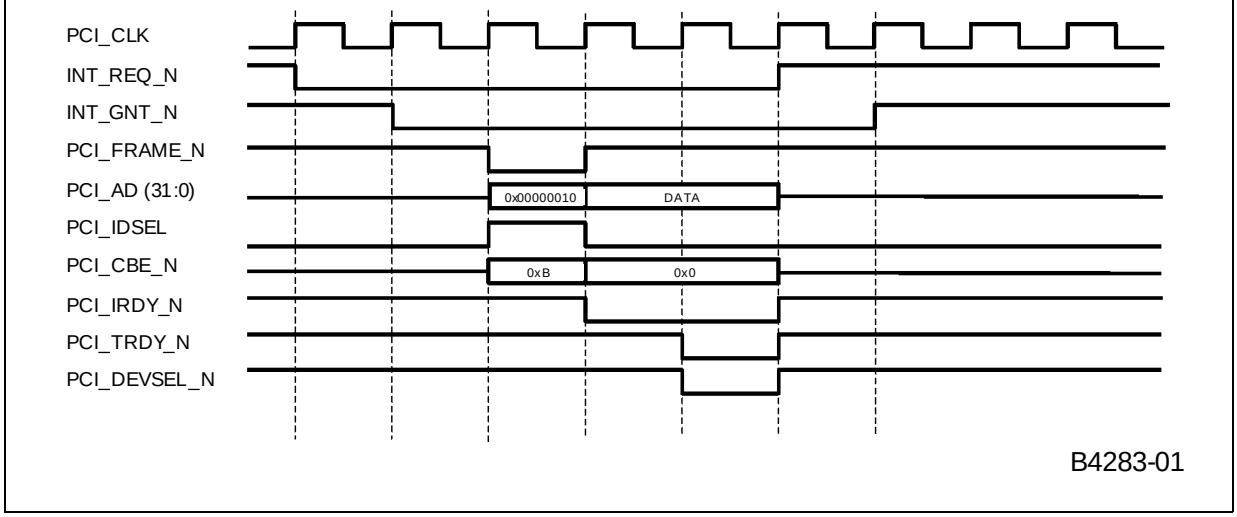


9.2.7.2 Initiated Type-0 Write Transaction

The following transaction is a PCI Configuration Write Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to the local PCI bus segment, Device number (chosen by IDSEL), Function number 2, and Base Address Register 0.

The IDSEL signal is left up to the user to determine how to drive this signal. It is driven from one of the upper address signals on the PCI_AD bus. A hexadecimal value of 0xB written on the PCI_CBE_N bus, during the address phase, signifies that this is a PCI Bus Configuration Write Cycle.

Figure 62. Initiated PCI Type-0 Configuration Write Cycle



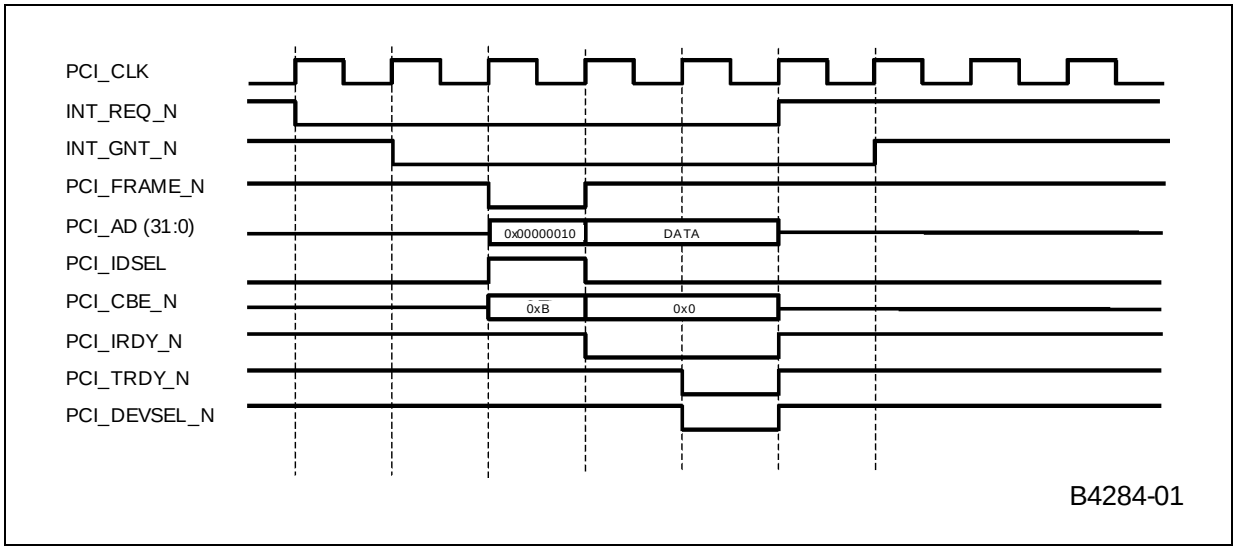
9.2.7.3 Initiated Type-1 Read Transaction

The following transaction is a PCI Configuration Read Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to PCI bus segment 0, Device number 0, Function 0, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01 being located in bits 1:0 of the PCI_AD bus during the address phase denotes a Type 1 PCI Configuration cycle.

A hexadecimal value of 0xA, written on the PCI_CBE_N bus during the address phase, signifies that this is a PCI Bus Configuration Read Cycle. Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore is extended by several clocks and is not shown here.

Figure 63. Initiated PCI Type-1 Configuration Read Cycle



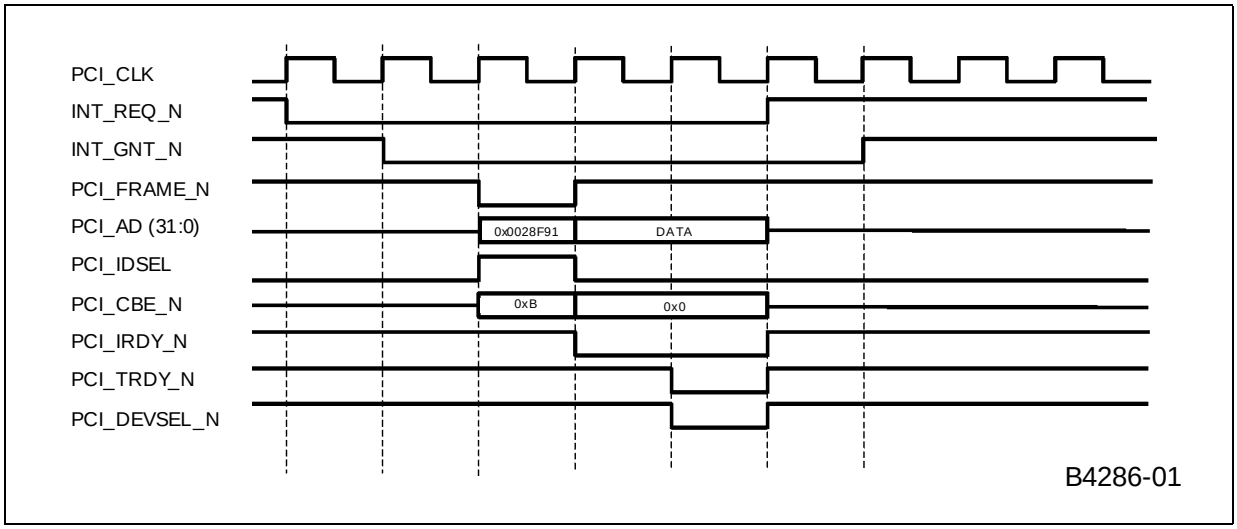
9.2.7.4 Initiated Type-1 Write Transaction

The following transaction is a PCI Configuration Write working-site cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to PCI bus segment 5, Device number 3, Function number 7, and Base Address Register 0.

This configuration cycle is a Type 1 configuration cycle and is intended for another PCI bus segment. Binary 01, being located in bits 1:0 of the PCI_AD bus during the address phase, denotes a Type 1 PCI Configuration cycle. A hexadecimal value of 0xB, written on the PCI_CBE_N bus during the address phase, signifies that this is a PCI Bus Configuration Write Cycle.

Due to the fact that the access is on another PCI Bus Segment, the PCI_TRDY_N signal may take longer to respond and therefore is extended by several clocks and is not shown here.

Figure 64. Initiated PCI Type-1 Configuration Write Cycle

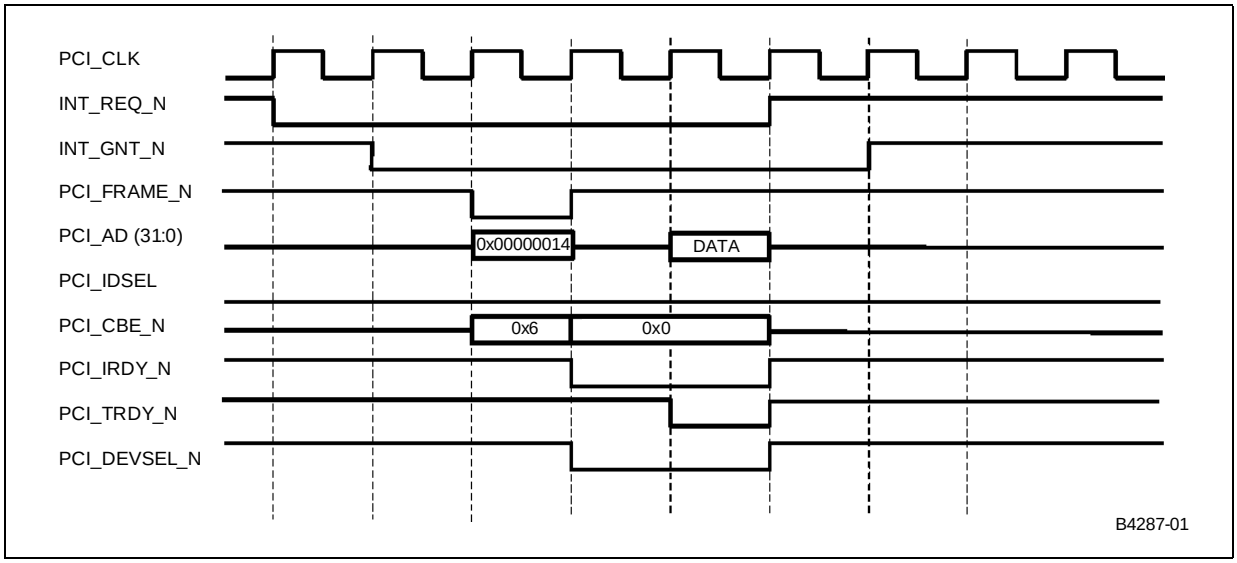


9.2.7.5 Initiated Memory Read Transaction

The following transaction is a PCI Memory Read Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type.

A hexadecimal value of 0x6, written on the PCI_CBE_N bus during the address phase, signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Figure 65. Initiated PCI Memory Read Cycle

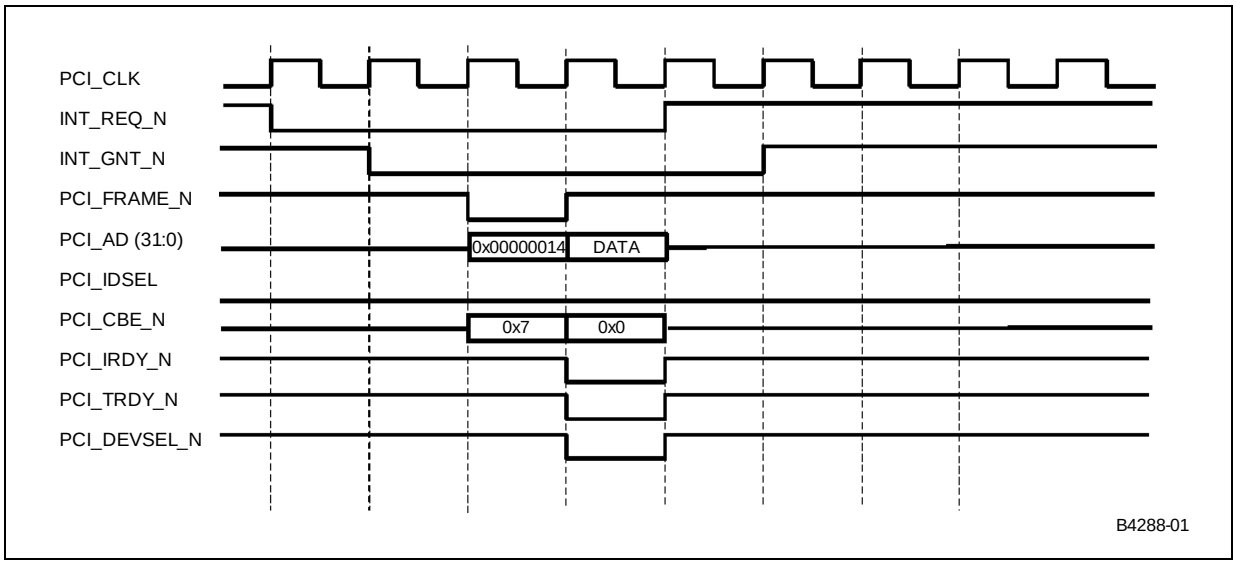


9.2.7.6 Initiated Memory Write Transaction

The following transaction is a PCI Memory Write Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear-increment transfer type. A hexadecimal value of 0x7 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Read Cycle. All byte enables are asserted for the transaction.

Notice that on this transaction the PCI_DEVSEL_N signal timing is different. This signal-timing differential is due to the fact that the PCI_DEVSEL_N signal must become active within the first three clocks after the PCI_FRAME_N becoming active. This requirement could be different for every device that is on the PCI Bus. There is also no relationship to when PCI_TRDY_N becomes active other than the PCI_TRDY_N signal must not become active prior to the PCI_DEVSEL_N signal becoming active.

Figure 66. Initiated PCI Memory Write Cycle

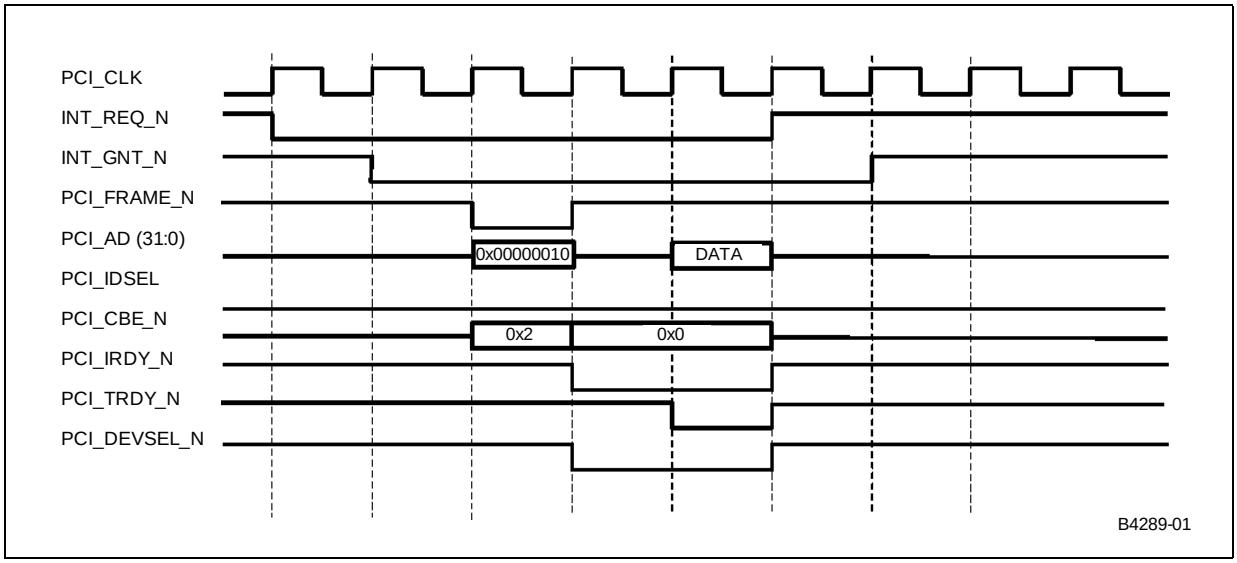


9.2.7.7 Initiated I/O Read Transaction

The following transaction is a PCI I/O Read Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to address location hexadecimal 0x00000010.

A hexadecimal value of 0x2 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus I/O Read Cycle.

Figure 67. Initiated PCI I/O Read Cycle

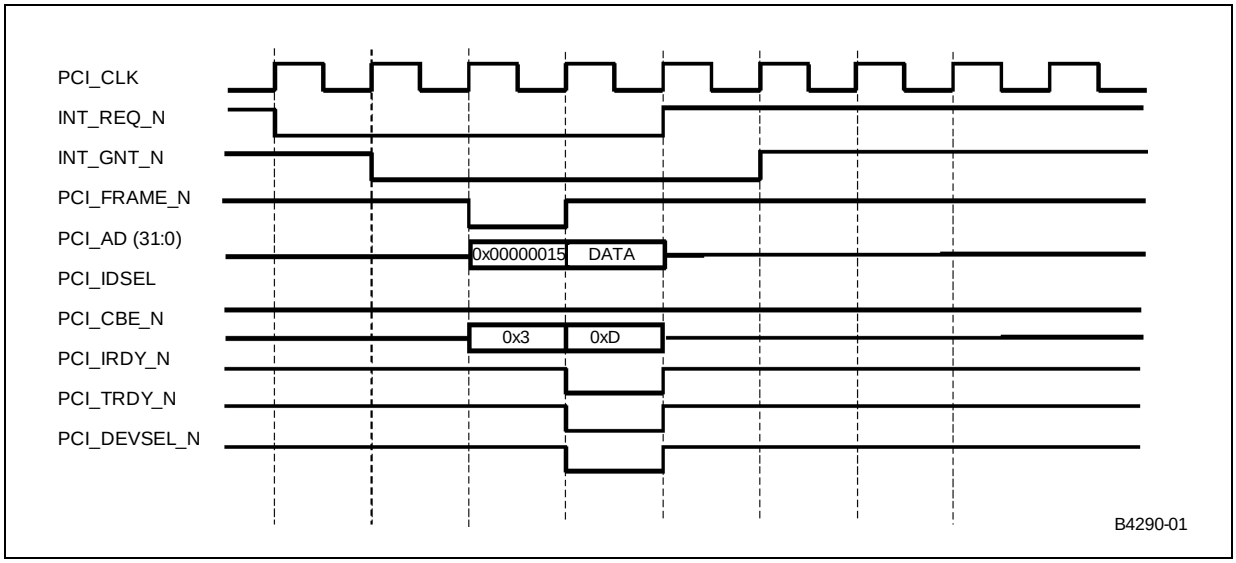


9.2.7.8 Initiated I/O Write Transaction

The following transaction is a PCI I/O Write Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to address location hexadecimal 0x00000015. The value of binary 01 in PCI_AD (1:0) indicates that the transfer is a valid byte address of the first byte of 32-bit word address 0x00000014 (0x00000014 + 0x00000001 = 0x00000015).

The byte-enables being 0xD, during the data transfer, signify that the transfer is a byte transfer to the above-mentioned address. A hexadecimal value of 0x3 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus I/O Write Cycle.

Figure 68. Initiated PCI I/O Write Cycle



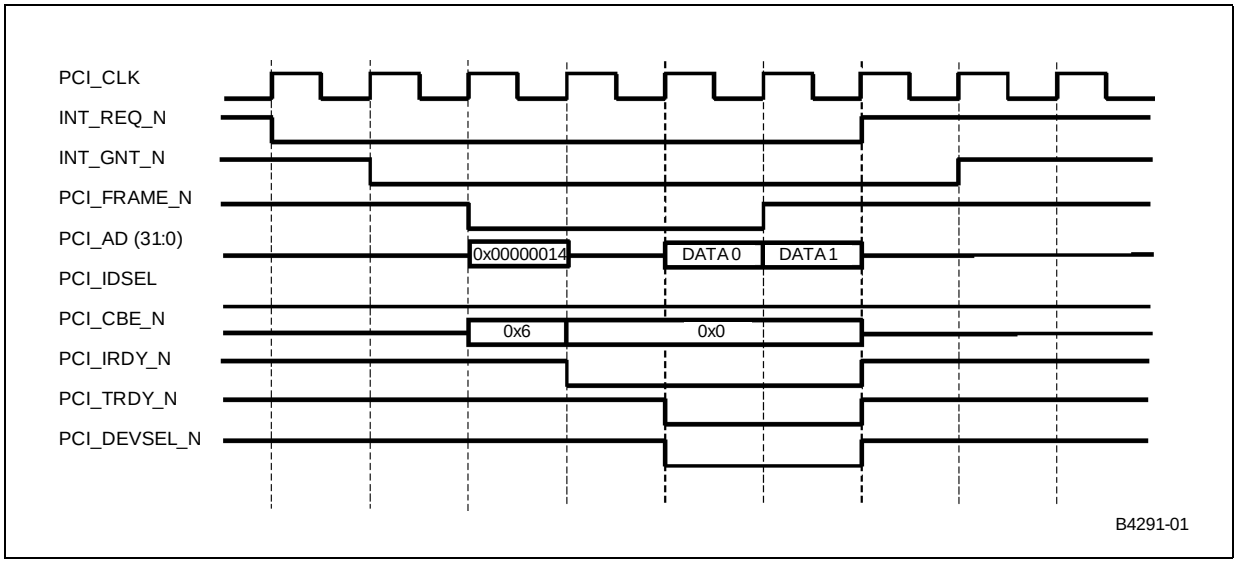
9.2.7.9 Initiated Burst Memory Read Transaction

The following transaction is a two word bursting PCI Memory Read Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to initial address location hexadecimal 0x00000014. The value of binary 00 in PCI_AD (1:0) indicates that this is a linear increment transfer type. The second data word transferred is from address hexadecimal 0x00000018.

A hexadecimal value of 0x6 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Read Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP43X network processors.

Figure 69. Initiated PCI Burst Memory Read Cycle



9.2.7.10 Initiated Burst Memory Write Transaction

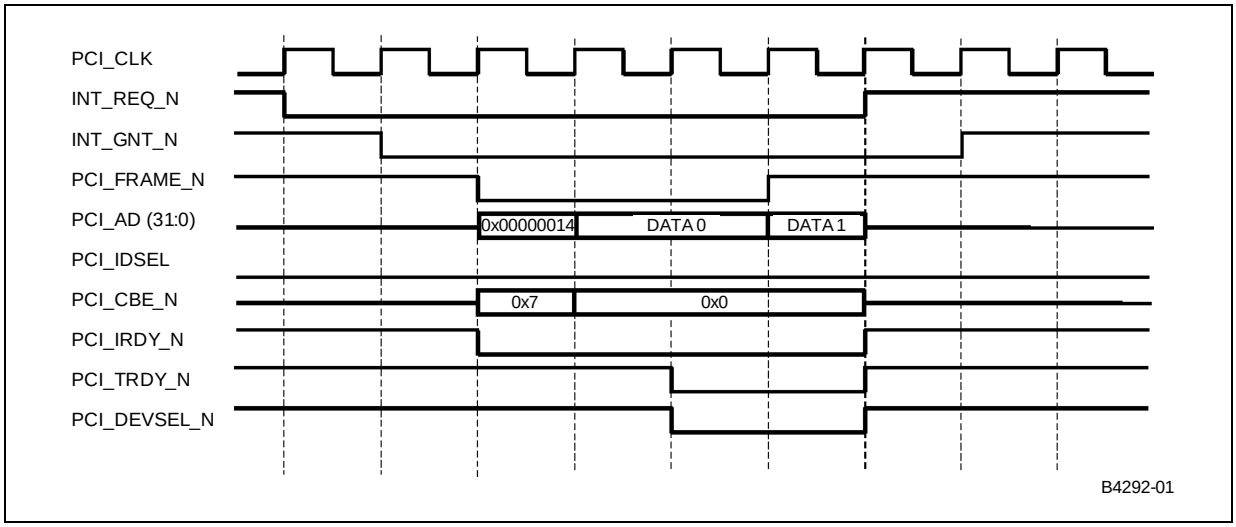
The following transaction is a two word bursting PCI Memory Write Cycle initiated from the IXP43X network processors. This diagram is to understand the inner workings of PCI transfers and may not reflect actual operation of the PCI Controller implemented on the IXP43X network processors. The transaction is initiated to initial address location hexadecimal 0x00000014.

The value of binary 00 in PCI_AD (1:0) indicates that this is an linear increment transfer type. The second data word transferred is from address hexadecimal 0x00000018.

A hexadecimal value of 0x7 written on the PCI_CBE_N bus during the address phase signifies that this is a PCI Bus Memory Write Cycle. All byte-enables are active for the transaction.

A maximum burst length of eight 32-bit words is supported for initiated Memory Cycle transactions from the IXP43X network processors.

Figure 70. Initiated PCI Burst Memory Write Cycle



9.2.8 PCI Controller Functioning as Bus Target

The IXP43X network processors can be the target of PCI transactions. Operating as a PCI target, the PCI bus can accept Memory Cycles, I/O Cycles, or Configuration Cycles. Target transactions can take place independent of the Host/Option configuration of the IXP43X network processors. Refer to Table 109, “PCI Target Interface Supported Commands” on page 288 for additional information on supported commands.

Only Type 0 Configuration Cycles are supported.

Timing diagrams are not shown for the target transactions because they are similar to initiated transactions. The only differences are the PCI devices that source/sink the various PCI signals.

For target-read transactions, a retry is issued upon the receipt of a request to transfer data. Between the time that the retry occurs and the access to the IXP43X network processors reoccurs, the PCI Controller retrieves the data from the previously requested location.

Refer to the *PCI Local Bus Specification*, Rev. 2.2 for additional information.

9.2.9 PCI Controller Door Bell Register

The PCI Controller has two registers that make up the Door Bell register logic on the IXP43X network processors. These two registers are the AHB Door Bell (PCI_AHBDOORBELL) register and the PCI Door Bell Register (PCI_PCIDOORBELL).

An external PCI device writes the AHB Door Bell (PCI_AHBDOORBELL) register to generate an interrupt signal to the Intel XScale processor. If the AHB doorbell interrupt is enabled (PCI_INTEN.ADBEN = 1) in the PCI interrupt registers, any bit set to logic 1 in the AHB Door Bell (PCI_AHBDOORBELL) forces the interrupt signal to occur.

The AHB Door Bell (PCI_AHBDOORBELL) register is set from the PCI bus only by writing logic 1 to the register. Writing logic 1 to the set bits from the SOUTH AHB clears bits that are set in the AHB Door Bell (PCI_AHBDOORBELL).

An example of using the AHB Door Bell (PCI_AHBDOORBELL) is as follows:

1. An external PCI device writes logic 1 to a bit or pattern of bits to generate an interrupt to the Intel XScale processor.
2. The AHB agent reads the AHB Door Bell (PCI_AHBDOORBELL) register and writes logic 1(s) to all set bits clear the set bit(s). This in turn de-assert the interrupt generated to the Intel XScale processor.

Using the South AHB, the Intel XScale processor writes the PCI Door Bell Register (PCI_PCIDOORBELL) to generate an interrupt to an external PCI device over the PCI_INTA_N signal. Any bit set to logic 1 in the PCI Door Bell Register (PCI_PCIDOORBELL) generates the PCI interrupt, if the PCI doorbell interrupt is enabled in the PCI Interrupt Enable Register (PCI_INTEN).

The PCI Door Bell Register (PCI_PCIDOORBELL) register can only be written by the AHB. The external PCI device must write logic 1 to all set bits in the PCI Door Bell Register (PCI_PCIDOORBELL) to clear the bits set by the Intel XScale processor.

An example of using the PCI Door Bell (PCI_PCIDOORBELL) is as follows:

1. The Intel XScale processor writes logic 1 to a bit or pattern of bits in the PCI Door Bell Register (PCI_PCIDOORBELL) to generate an interrupt on the PCI bus using PCI_INTA_N.
2. An external PCI device reads the PCI Door Bell Register (PCI_PCIDOORBELL) and writes logic 1(s) to all set bits to clear the set bit(s). This causes the interrupt that is asserted to de-assert.

9.3 Functional Description

The following section gives the functional description of the PCI Controller:

9.3.1 PCI Byte-Enable Generation

The byte enables for single PCI transactions are generated based on the type of AHB access (direct read/write or non-prefetch read/write), address, transfer size (8-bit, 16-bit, 32-bit), and the settings in effect for AHB endianness and data swapping modes. All 32-bit accesses obviously assert all byte enables when the transaction is sent to the PCI Initiator. Since the AHB Slave Interface only supports bursts with a size of 32-bits, PCI burst transactions always has all byte enables asserted. Table 118 shows the byte enables for 8-bit and 16-bit single PCI read and write cycles generated by direct access of the AHB Slave Interface. PCI cycles generated by the non-prefetch CSRs use the value of the NP_BE field of the pci_np_cbe CSR for the byte enables.

9.3.2 PCI Core

This block generates the PCI compliant protocol logic. It operates as an initiator or a target device on the PCI Bus. As an initiator, all bus cycles are generated by the core. As a PCI target, the core responds to bus cycles that have been directed towards it.

On the PCI Bus, the core supports interrupts, 32 bit data path, 32 bit addressing, and a single configuration space. The local configuration registers (CSRs) are accessible from the PCI Bus or from the Intel XScale processor through the AHB bus.

There are four 8 word deep data FIFOs and a 4 word deep address FIFO in the core. The separate slave and master data FIFOs allow simultaneous operations and multiple outstanding PCI bus transfers. The initiator address FIFO can accumulate up to four addresses that are PCI reads or writes.

9.3.2.1 PCI Target Interface

The PCI Target Interface provides read/write access to AHB agents, PCI Controller PCI Configuration registers (through Configuration cycles), and some PCI Controller CSRs. The interface performs the PCI target actions in response to external PCI Master transactions. Table 115 lists the supported command types.

The interface does not support the following features:

- Lock cycles
- VGA palette snoop
- Dual address cycles
- Cache-line, wrap-mode addressing
- Configuration Type 1 read/write cycles

Table 115. PCI Target Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	not supported
0x1	Special Cycle	not supported
0x2	I/O Read	supported
0x3	I/O Write	supported
0x4	Reserved	
0x5	Reserved	
0x6	Memory Read	supported
0x7	Memory Write	supported
0x8	Reserved	
0x9	Reserved	
0xa	Configuration Read	supported
0xb	Configuration Write	supported
0xc	Memory Read Multiple	Converted to Memory Read
0xd	Dual Address Cycle	not supported
0xe	Memory Read Line	Converted to Memory Read
0xf	Memory Write and Invalidate	converted to Memory Write

9.3.2.1.1 PCI Bus Access to PCI Configuration Registers

An access to PCI Configuration registers occurs when the PCI_IDSEL input is asserted, the PCI command is a configuration read or write, and PCI_AD[1:0] = 00 indicating a type 0 configuration cycle. The selected register is indicated by PCI_AD[7:2]. Accesses are single-word only, the Target Interface disconnects any burst longer than 1 word. During reads, byte enables are ignored and the full 32-bit register value is returned. Read accesses to unimplemented registers complete normally on the bus and return all zeroes. During writes, the PCI byte enables determine the byte(s) to be written within the addressed register. Write accesses to unimplemented registers complete normally on the bus but the data is discarded.

9.3.2.1.2 PCI Bus Access to PCI Controller CSRs

PCI Controller CSRs are accessed from PCI through read or write transactions whose address matches the PCI base address register pci_bar4. Pci_bar4 is written by the PCI Host during the bus configuration process to map the PCI Controller CSRs into the

external PCI bus memory map. Currently, only two of the PCI Controller CSRs are write accessible from the PCI bus when not in PCI test mode (exp_pcitest = 1): pci_pcidoorbell and pci_ahbdoorbell. When exp_pcitest = 0, all CSRs are write accessible from the PCI bus. All registers are read from PCI.

9.3.2.1.3 PCI Bus Access to AHB Address Space

Internal AHB address space is accessed from the PCI bus through read or write transactions whose address matches one of the implemented PCI base address registers pci_bar0,1,2,3, or 5. These registers are written by the PCI Host during the bus configuration process to map five AHB address regions into the external PCI bus address map.

9.3.2.1.4 PCI Target Write Accesses

A PCI target memory write occurs if the PCI address matches one of the PCI base address registers pci_bar0,1,2,3 and the PCI command is Memory Write or Memory Write and Invalidate. The address, data, byte enables, and BAR identifier (indicating the PCI base address register that was hit) are written to the Target Receive FIFO. If the FIFO fills up during the transfer, the PCI Controller signals a target disconnect to the external master. If the FIFO is full at the start of a write access, a retry is signaled on the bus. All combinations of data phase byte enables are supported.

A PCI target I/O write occurs if the PCI address matches the PCI base address register pci_bar5 and the PCI command is a I/O Write. In this case, the PCI Target Interface disconnects the transfer after the 1st data phase.

9.3.2.1.5 PCI Target Read Accesses

A PCI target memory read occurs if the PCI address matches one of the PCI base address registers and the PCI command is Memory Read, Memory Read Line, or Memory Read Multiple. The address and BAR identifier (indicating the PCI base address register that was hit) are written to the Target Receive FIFO and the transaction is retried on the PCI bus. After AHB read data is written into the Target Transmit FIFO by the AHB Master Interface, the PCI Target Interface delivers the data to the initiator when the initiator retries the transfer. If the Target Transmit FIFO becomes empty, the PCI Target Interface issues a target disconnect to the initiator until it becomes non-empty again. The interface ignores byte enables for PCI memory reads and delivers all addressed words to the initiator.

A PCI target I/O read occurs if the PCI address matches the PCI base address register pci_bar5 and the PCI command is an I/O Read. In this case, the PCI Target Interface disconnects the transfer after the 1st data phase.

When the read data is returned into the Target Transmit FIFO, the PCI Controller begins to decrement a discard timer. If the external PCI bus initiator has not repeated the read by the time the timer reaches zero, the PCI Controller discards the read data and invalidates the delayed read address. The discard timer counts 2^{15} (32768) PCI clocks. Once the timer expires, the next PCI target read is treated as a new operation.

9.3.2.2 PCI Initiator Interface

The Master Interface provides bus mastering capability in response to PCI requests received from an AHB master or from the PCI Controller DMA channels. Requests are buffered in the Initiator Request FIFO and handled by the PCI Core Initiator Interface. This interface receives the address, word count, byte enables and PCI command type from the AHB side and performs the specified transaction on the PCI bus, handling all bus protocol and retry/disconnect situations. Table 116 lists the supported transaction types.

Table 116. PCI Initiator Interface Supported Commands

PCI Byte Enables	Command Type	Support
0x0	Interrupt Acknowledge	supported
0x1	Special Cycle	supported
0x2	I/O Read	supported
0x3	I/O Write	supported
0x4	Reserved	
0x5	Reserved	
0x6	Memory Read	supported
0x7	Memory Write	supported
0x8	Reserved	
0x9	Reserved	
0xa	Configuration Read	supported
0xb	Configuration Write	supported
0xc	Memory Read Multiple	supported
0xd	Dual Address Cycle	not supported
0xe	Memory Read Line	supported
0xf	Memory Write and Invalidate	not supported

9.3.2.2.1 Initiator Write Transactions

The PCI Master Interface receives write requests from the AHB Slave Interface or AHB-to-PCI DMA Controller via the Initiator Request FIFO. Write data is supplied using the Initiator Transmit FIFO. The Master Interface transfers the indicated number of words from the FIFO to the PCI bus. The following rules apply to the write transfers:

- If the Transmit FIFO becomes empty, the Master Interface terminates the cycle on the PCI bus. When more data becomes available, the transfer resumes using the address of the next word to be delivered.
- If the master's Latency Timer expires, the Master Interface terminates the cycle on the PCI bus (see "Master Latency Timer" on page 315 for a description of the Latency Timer). The transfer resumes at the first opportunity using the address of the next word to be delivered.
- If a retry or target disconnect is received before the transfer ends, the Master Interface resumes the transfer at the first opportunity using the address of the next data word to be delivered.
- If a master abort occurs, all of the write data is discarded and the Received Master Abort bit is set in the PCI Configuration status register.
- If a target abort is received, all of the write data is discarded and the Received Target Abort bit is set in the PCI Configuration status register.

9.3.2.2.2 Initiator Read Transactions

The PCI Master Interface receives read requests from the AHB Slave Interface or PCI-to-AHB DMA Controller via the Initiator Request FIFO. Read data is supplied to the AHB side using the Initiator Receive FIFO. The Master Interface transfers the indicated number of words from the PCI bus to the FIFO. The following rules apply to the read transfers:

- For memory read operations, if the cache line size indicated in the PCI Configuration Register pci_bhlc is zero, the Memory Read command is used on PCI.

For non-zero cache-line sizes, the starting address and word count of the transfer are used to determine how many cache lines are read. If one full cache line is to be read, the Memory Read Line command is used. If two or more full cache lines, the Memory Read Multiple command is used. Note that read commands may change from transaction to transaction during long burst reads that are broken up by retries or disconnects.

- If the Receive FIFO becomes full, the Master Interface terminates the cycle on the PCI bus using a disconnect. When room becomes available, the transfer resumes using the address of the next word to be read.
- If the master's Latency Timer expires, the Master Interface terminates the cycle on the PCI bus. The transfer resumes at the first opportunity using address of the next word to be read.
- If a retry or target disconnect is received before the transfer ends, the Master Interface resumes the transfer at the first opportunity using the address of the next data word to be read.
- If a master abort occurs, the AHB side transaction is terminated with an ERROR response and the Received Master Abort bit is set in the PCI Configuration status register.
- If a target abort is received, the AHB side transaction is terminated with an ERROR response and the Received Target Abort bit is set in the PCI Configuration status register.

9.3.2.2.3 Master Latency Timer

When the Initiator Interface begins PCI transaction by asserting pci_frame_n, it begins to decrement its master latency timer. When the timer value reaches zero, the PCI grant input is checked. If the grant is deasserted, the Initiator Interface deasserts PCI_FRAME_N (if it is still asserted) at the earliest opportunity (normally the next data phase for all transactions). The transfer is resumed as soon as possible at the address of the next word.

9.3.2.2.4 Initiator Special Cycles

Special Cycles are broadcast to all PCI targets on the bus and are terminated by Master Abort. No error is signaled in this case.

9.3.2.3 PCI Host Functions

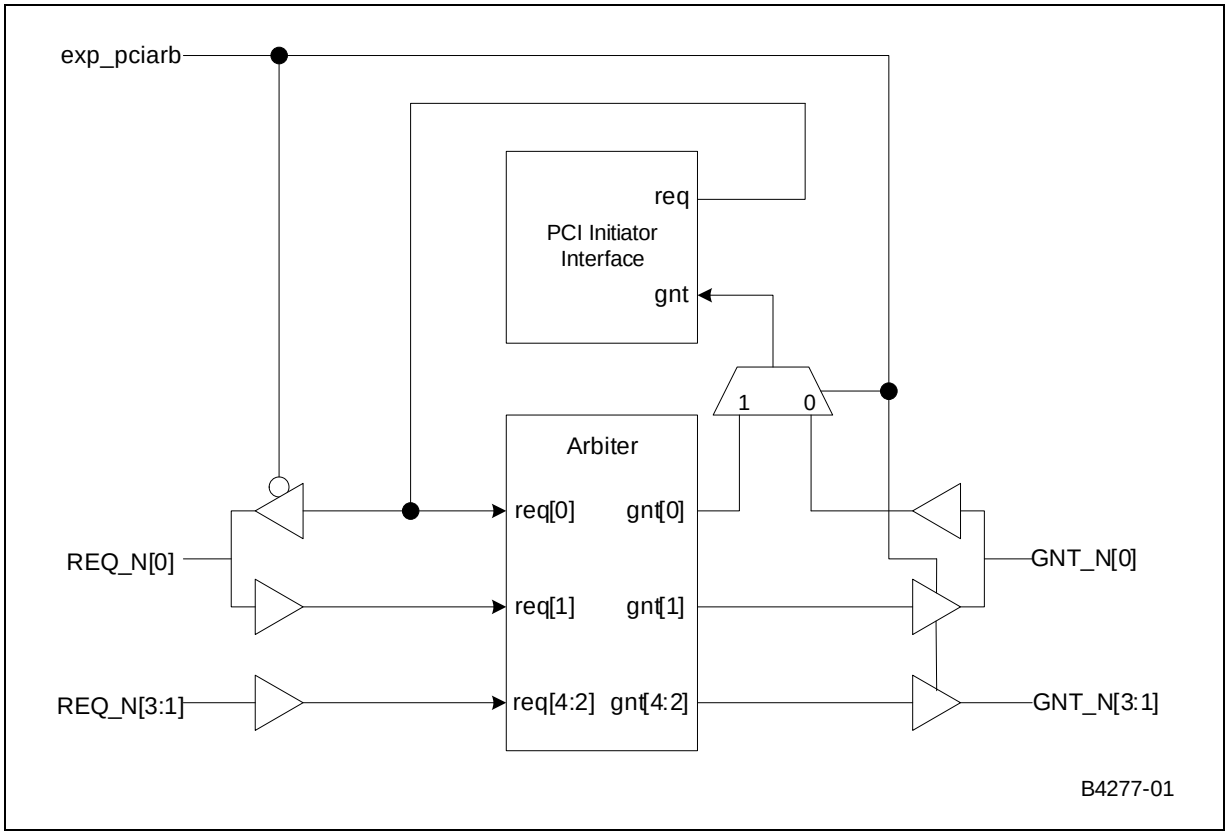
A logic high state on the exp_pcihost input port indicates that the PCI Controller should function in host mode. In this mode, access to the PCI Configuration registers is only allowed from the AHB via the CSR-based configuration access port. In non-host mode (exp_pcihost = 0), access is only allowed from the external PCI bus.

The following sections describe PCI host functions that do not directly affect PCI Controller functionality.

9.3.2.3.1 PCI Arbiter

The PCI Controller contains a PCI bus arbiter that supports four external masters in addition to the PCI Controller's Initiator Interface. To enable the arbiter, the exp_pciarb pin must be a logic 1. When the arbiter is enabled, the PCI Controller PCI request and grant signals are routed internal to the chip and cannot be viewed externally. Figure 71 shows a conceptual view of the arbiter request and grant signal wiring. The PCI Master Interface, Arbiter, and 2:1 multiplexer are part of the PCI Controller.

Figure 71. PCI Controller Arbiter Configuration



The arbiter uses a simple round-robin priority algorithm. The arbiter asserts the grant signal corresponding to the next request in the round-robin during the current executing transaction on the PCI bus (that is, arbitration is hidden). If the arbiter detects that an initiator has failed to assert the PCI_FRAME_N signal after 16 cycles of both grant assertion and a PCI bus idle condition, the arbiter deasserts the grant. That master does not receive any more grants until it deasserts its request for at least one PCI clock cycle. Bus parking is implemented in that the last bus grant stays asserted if no request is pending. To prevent bus contention, if the PCI bus is idle, the arbiter never asserts one grant signal in the same PCI cycle in which it deasserts another. It deasserts one grant, then asserts the next grant after one full PCI clock cycle has elapsed to provide for bus driver turnaround.

9.3.2.3.2 PCI Interrupt Inputs

If used as a PCI host, up to 4 PCI interrupt signals are wired as GPIO inputs to the GPIO Controller function (not a part of the PCI Controller) and further presented to the Interrupt Controller function (again, not a part of the PCI Controller) to generate an interrupt to the Intel XScale processor. The PCI Controller provides no hardware support for these interrupts. By specification, PCI interrupts are level-sensitive and asserted/deasserted asynchronously to the PCI clock.

9.3.2.4 PCI Controller Clock and Reset Generation

The PCI reset and PCI clock signals are provided using general-purpose input/output (GPIO) outputs or from an external source. The GPIO block is not part of the PCI controller. See below for particular signals that is used on various products for these functions:

Table 117. PCI CLOCK and RESET Sourcing

Function	Intel® IXP42X and IXP46X Network Processors	Intel® IXP43X Product Line of Network Processors
Clock Source	GPIO14	GPIO14
Reset Source	GPIO[13:0]	GPIO[13:0]

Both signals are sourced from an external device as well. The Intel XScale processor can generate the PCI reset and PCI clock outputs to satisfy the reset timing requirements of the PCI bus.

A PCI startup sequence is as follows:

1. Power-on reset occurs to the IXP43X network processors, the Intel XScale processor starts execution (internal PLL assumed locked and internal clocks stable).
2. Software configures PCI reset and PCI clock GPIOs as outputs driving 0. A pull-down on the GPIO pin chosen to drive the PCI reset signal is required. This pull-down is required because the GPIO are at a tri-stated value until the device comes completely out of reset and the PCI reset must be low from the start.
3. Wait 1ms to satisfy minimum reset assertion time of the PCI specification.
4. Configure the PCI clock GPIO for the proper PCI bus frequency (defined in the section GPIO).
5. Enable the PCI clock GPIO to drive the PCI clock
6. Wait 100 µs to satisfy the **minimum reset assertion time from clock stable** requirement of the PCI specification.
7. Set the PCI reset GPIO output to drive a 1. This releases the PCI bus.

Note: The PCI reset are asserted and de-asserted asynchronously with respect to the PCI clock. It is also important to note the PCI reset signal cannot be the same signal as the RESET_IN_N signal going to the IXP43X network processors due to PCI reset timing and PCI initialization requirements.

9.3.2.5 PCI Configuration Register Access

As a PCI host performing configuration of the bus, the Intel XScale processor must have read/write access to all of the PCI Configuration registers in the PCI Controller. This access is provided via a set of three CSRs accessible from the AHB bus as described in “AHB Accesses of Local PCI Configuration Registers” on page 323.

9.3.2.5.1 PCI Configuration

Configuration of the PCI bus is performed by a PCI agent using configuration read and write cycles. All devices on the bus have their configuration space read by the agent to determine the attributes of the device. The agent then writes the appropriate configuration registers in each device to, for example, set up the PCI memory map, and enable/disable certain bus signalling characteristics depending on the capabilities of the devices on the bus. The PCI Controller supports bus configuration both as a host performing configuration transactions, and as a non-host responding to configuration transactions from an external agent.

In non-host mode, select registers must be initialized by the Intel XScale processor before the PCI Controller can respond to PCI configuration cycles from an external agent. The Subsystem ID and Subsystem Vendor ID are two such registers. The Initialization Complete bit in the control and status register, when cleared, forces a retry on the PCI bus in response to configuration cycles. This bit is cleared at reset thus holding off configuration by an external agent until the PCI Controller configuration registers are properly initialized. When the Intel XScale processor has completed its initialization of the necessary registers, it sets the bit to allow PCI configuration to proceed. A special test mode, indicated by a logic 0 level on the exp_pcitest input port, overrides this process and tells the PCI Controller to respond to PCI configuration cycles regardless of the state of Initialization Complete.

When the Intel XScale processor is the PCI host performing system configuration, it is responsible for setting up the PCI Controller configuration registers and those of all the other devices on the bus. To support this, access to local PCI configuration registers is provided via a CSR-based configuration access port as described in "AHB Accesses of Local PCI Configuration Registers" on page 323. Likewise, configuration read/write cycles are generated on the PCI bus using a CSR-based PCI access port as described in "AHB Non-Prefetch PCI Accesses" on page 324.

9.3.2.6 PCI Pad Drive Strength Compensation Support

The PCI Core supports PCI pad strength compensation via the exp_rcomp_complete input. When exp_rcomp_complete is at a logic 0 state, the PCI Core Initiator and Target interfaces are inhibited from bus operations thus preventing any switching activity on the PCI bus from this device. In addition, the PCI arbiter grant outputs pcc_gnt_n[3:0] remains in the de-asserted state regardless of the state of the request inputs. This feature allows an external drive strength compensation circuit to inhibit any switching of the PCI output drivers until an optimum drive for the PCI pads is determined based on temperature and voltage levels on the die. When this drive is determined, exp_rcomp_complete is asserted to a logic 1 state to allow normal PCI functioning of the PCI Core.

9.3.2.6.1 Effect on Initiator Interface

When exp_rcomp_complete is logic 0, the PCI Controller Initiator interface is inhibited from starting a PCI master cycle that is queued in the Initiator Request FIFO. The PCI bus request is not asserted. When exp_rcomp_complete is a logic 1, normal operation of the Initiator Interface is enabled. If one or more requests is waiting in the Request FIFO when exp_rcomp_complete changes state from 0 to 1, those requests are processed in a normal manner.

9.3.2.6.2 Effect on Target Interface

When exp_rcomp_complete is logic 0, the PCI Controller Target interface is inhibited from responding to PCI cycles initiated from PCI masters. The interface does not drive any PCI signals and the cycle terminates with a Master Abort. When exp_rcomp_complete asserts to a logic 1, the Target Interface responds normally to the next PCI cycle that targets this device, usually a Configuration Read.

9.3.2.6.3 Effect on Internal PCI Arbiter

If the internal PCI arbiter is enabled, a low level on exp_rcomp_complete blocks all requests from external PCI masters and cause the arbiter to park the bus on the local PCI Initiator Interface. This has the effect of preventing any switching on the arbiter grant outputs. When exp_rcomp_complete is a logic 1, the requests from external masters are enabled and the arbiter functions normally.

9.3.2.7 AHB Master Interface

The AHB Master Interface provides read/write access to AHB slaves and local CSRs for transactions generated by external PCI Initiators. It also services read/write requests from the DMA Controller channels to effect DMA transfers between the AHB and PCI busses. Both big-endian and little-endian addressing are implemented depending on the state of the pci_csr.ABE bit. Three virtual AHB masters can initiate transfers using the AHB Master Interface:

1. Transfers originating from an external PCI master are presented to the AHB Master Interface by the PCI Target Interface via the Target FIFOs in the PCI Core. The AHB Master Interface reads the PCI address and control information from the Target Receive FIFO and translates the PCI transaction to an appropriate AHB transaction. For writes, data is written to the Target Receive FIFO by the PCI Core and read out to the AHB bus by the AHB Master Interface. During reads, the AHB Master Interface performs an appropriate read transaction and writes the data to the Target Transmit FIFO.
2. The AHB-to-PCI DMA channel uses the AHB Master Interface to read data from an AHB agent and send a PCI write request to the PCI Core via the Initiator Request and Initiator Transmit FIFOs.
3. The PCI-to-AHB DMA channel uses the AHB Master Interface to read previously requested PCI read data from the Initiator Receive FIFO and write this data to an AHB agent.

Arbitration for control of AHB mastering resources is carried out on two levels. On the first level, PCI requests and DMA requests alternate for priority access. On the second level, the two DMA channels alternate for priority access. This arbitration scheme balances the high bandwidth DMA traffic with the lower bandwidth PCI traffic.

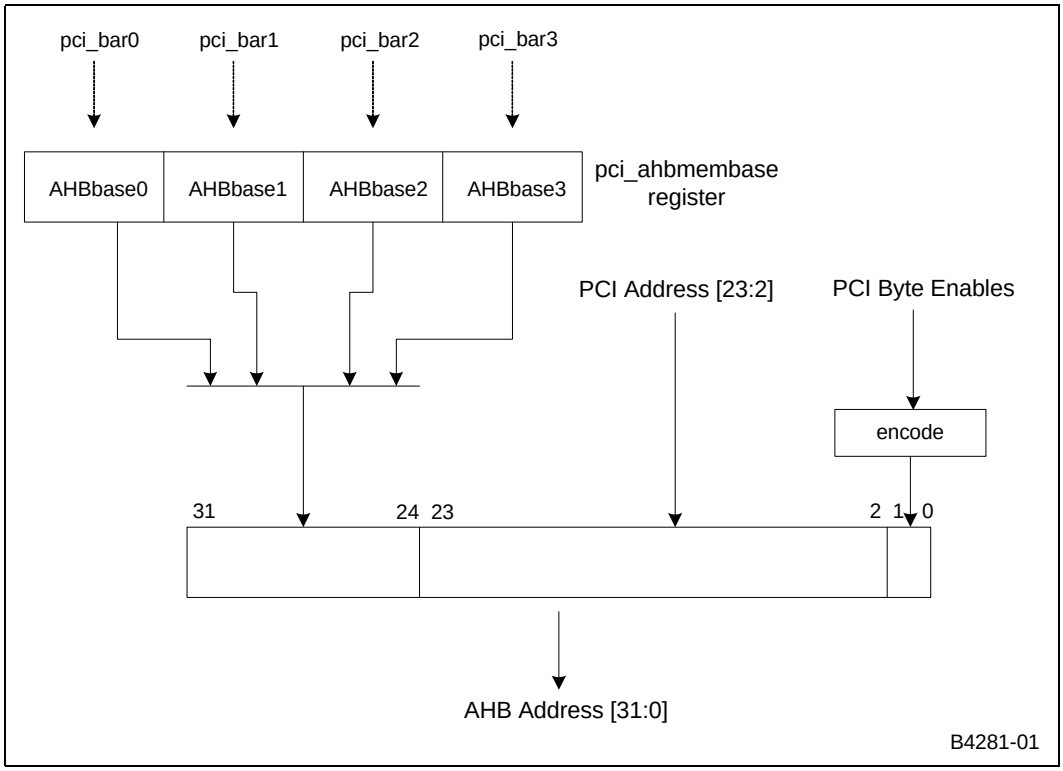
9.3.2.7.1 PCI Address Translation to AHB Address

The PCI address received from the PCI Core is translated to an AHB address as follows:

- The AHB Master Interface reads a BAR indicator (BAR ID) from the Core to determine the 6 PCI Configuration BARs that was targeted.
- If one of the lower four BARs was targeted (PCI_BAR0/1/2/3), the bits 23:2 of the PCI address map to bits 23:2 of the AHB address bits and an 8-bit field in the PCI_AHBMEMBASE register provides the upper 8 address bits of the AHB address. In this manner, each 16MB address range on the PCI bus is mapped to any 16MB region in the full 4GB AHB address space by appropriately initializing the four base address fields in pci_ahbmembase. Bits 1:0 of the AHB address are derived from the PCI byte enables. Figure 72 shows the address translation mechanism.
- If PCI_BAR4 was targeted, the access is directed to the PCI accessible CSRs in the PCI Controller and no AHB transaction is performed.
- If PCI_BAR5 was targeted, bits 7:2 of the PCI address map to bits 7:2 of the AHB address bits and the 24-bit IOBase field of the PCI_AHBIOBASE register provides the upper 24 bits of the AHB address. Bits 1:0 of the AHB address are derived from the PCI byte enables.

Since the PCI controller performs pre-fetches when doing reads to the PCI_BAR0/1/2/3 address windows, the PCI_AHBMEMBASE must not be programmed to access any AHB I/O space or the AHB Queue Manager. To access AHB I/O space or the AHB Queue Manager, PCI_BAR5 must be used.

Figure 72. PCI-to-AHB Address Translation



9.3.2.8 AHB Master Writes

If the AHB Master Interface receives a Memory Write or Memory Write and Invalidate command from the PCI target, the BAR ID and byte enables are examined to determine the appropriate operation:

- If the BAR ID indicates a CSR access (BAR 4), CSR access is requested and, when granted, the indicated write is performed using the supplied address, data, and byte enables. An AHB master operation is not performed.
- If the BAR ID indicates a memory access (BARs 0-3), the byte enables corresponding to each word of the transfer are examined to determine the AHB address and burst size to use during the transfer. If all byte enables are asserted, an INCR word operation is started on the bus and continues until the last-data-word indicator is detected, or a word with at least one byte enable de-asserted is encountered. Words received from the Initiator Receive FIFO without all byte enables asserted produce a single byte write operation for each byte that is enabled. PCI Memory writes produce SINGLE or INCR writes on the AHB bus.
- If the BAR ID indicates an I/O access (BAR5), a single word write is performed if all byte enables are asserted, otherwise, a single byte write operation is performed for each byte that is enabled.

All AHB burst write transfers are disconnected at the 8-word address boundary. If the Write FIFO becomes empty during the transfer, the AHB Master Interface disconnects the transfer by driving IDLE cycles on the bus. When data is available again in the FIFO, the bus is re-acquired and the transfer continues.

Data parity errors received from the Receive FIFO are ignored as far as AHB master operations are concerned, but, the pci_isr.PPE CSR bit is set to capture the error.

9.3.2.9 AHB Master Reads

If the AHB Master Interface receives a Memory Read, Memory Read Line or Memory Read Multiple command from the PCI target, the BAR ID and byte enables are examined to determine the appropriate operation:

- If the BAR ID indicates a CSR access (BAR 4), CSR access is requested and, when granted, a single word read is performed using the supplied address. An AHB master operation is not performed.
- If the BAR ID indicates a memory access (BARs 0-3), an INCR word read operation is started on the bus and continues until the PCI Initiator Interface indicates that the requesting master has terminated the transfer. Any unused data in the Initiator Read FIFO is discarded.
- If the BAR ID indicates an I/O access (BAR5), a single word read is performed if all byte enables are asserted, otherwise, a single byte read operation is performed for each byte that is enabled.

All AHB burst read transfers are disconnected at the 8-word address boundary. If the Read FIFO becomes full during the transfer, the AHB Master Interface disconnects the transfer by driving IDLE cycles on the bus. When room is available in the FIFO again, the bus is re-acquired and the transfer continues.

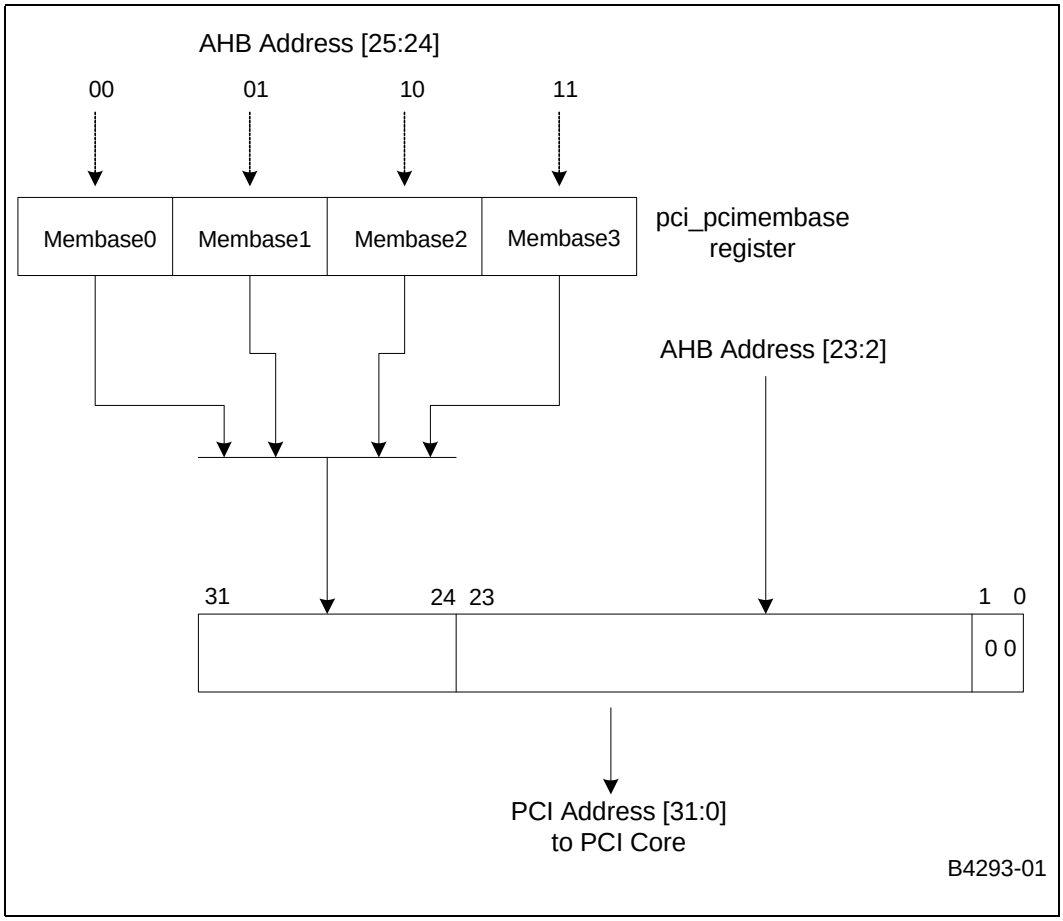
9.3.2.10 AHB Slave Interface

The AHB Slave Interface provides external AHB masters with read/write access to external PCI targets, PCI Controller PCI Configuration registers, and local PCI Controller CSRs. The interface supports both little-endian and big-endian addressing. AHB Slave accesses are processed by converting the AHB transaction to a PCI transaction with PCI address, Command, and byte enables provided to the PCI Core. The PCI Core then executes the requested PCI transaction and provides status and read data to the AHB Slave.

9.3.2.10.1 AHB-to-PCI Address Translation

For each of the four 16MB memory ranges, the lower 24 bits of the AHB address map to the lower 24 PCI address bits and an 8-bit field in the PCI_PCIMEMBASE register provides the upper 8 address bits of the PCI address. Bits 24 and 25 of the AHB address determine the 8-bit fields in PCI_PCIMEMBASE that is used. In this manner, the Intel XScale processor can map each range to any 16MB region in the full 4GB PCI address space by appropriately initializing the four base address fields in pci_pcimembase. See Figure 73 for details.

Figure 73. AHB-to-PCI Address Translation – Memory Cycles



9.3.2.10.2 AHB Slave Read Accesses

An AHB Slave Read occurs when an external AHB master performs a read operation addressing one of the PCI Controller AHB address regions. The AHB Slave Interface services one PCI read request at a time. When an AHB Slave Read occurs, the Slave Interface checks the status of the Initiator Request FIFO to determine the course of action:

- If the Initiator Request FIFO is not full and a previous direct PCI read is not currently pending, the Slave Interface immediately issues a retry for direct PCI reads and writes the PCI address to the Request FIFO. CSR reads complete immediately with one wait state.
- If the Request FIFO is full or if a previous read request is still pending (has not completed yet on AHB), a retry is issued in response to direct PCI reads and CSR reads. The address of the read is not written to the Request FIFO in the direct PCI read case.
- If the DMA Controller has been granted access to the Request FIFO, a retry is issued in response to direct PCI reads and reads of the four pci_np_xxx CSRs. The address of the read is not written to the Request FIFO in the direct PCI read case. Reads of any of the other CSRs complete immediately with one wait state.

When the PCI read data is available in the Initiator Receive FIFO, the Slave Interface provides the data in response to the retried read request from the same master. As the read is pending, any accesses attempted by other AHB masters are retried with no request sent to the PCI Core. The Slave Interface only issues the retry on the cycle following the address phase on AHB, never in the middle of a burst. Bursts of 1 to 255 words are supported.

9.3.2.10.3 AHB Slave Write Accesses

An AHB Slave Write occurs when an external AHB master performs a write operation addressing one of the PCI Controller AHB address regions (indicated by the assertion of arbs_hsel_pcc or arbs_hsel_pcc_mmr by the AHB arbiter). When an AHB Slave Write occurs, the Slave Interface checks the status of the Initiator Request FIFO and Initiator Transmit FIFOs to determine the course of action:

- If a PCI read is currently pending, a retry is issued on AHB. The request is not written to the Request FIFO.
- If the DMA Controller has been granted access to the Request FIFO, a retry is issued on AHB for direct writes to PCI (arbs_hsel_pcc_asserted) or writes to any of the four pci_np_xxx registers. Writes to any of the other CSRs complete immediately. The request is not written to the Request FIFO in the direct PCI write case.
- If the operation is a direct write to PCI and the Request FIFO is full or the Transmit FIFO has insufficient storage available for the entire burst (INCR bursts are assumed to be up to 8 words), a RETRY is issued on AHB and the request is not written to the Request FIFO.
- If the operation is a direct write to PCI and is a SINGLE, INCR with burst length of 8 or less, INCR4 or INCR8, and the Request FIFO is not full and the Transmit FIFO has sufficient storage for all of the data in the entire burst, the request is immediately posted in the Request FIFO, the data is written to the Transmit FIFO and the transfer completes on AHB. In the case of an INCR write of more than eight words, if the Transmit FIFO becomes full during the transfer, wait states are inserted on the AHB bus until the FIFO becomes not full again.

9.3.2.10.4 AHB Accesses of Local PCI Configuration Registers

The PCI Controller local PCI Configuration registers (listed in Table 119 on page 338) in the PCI Core are accessed from the AHB bus via a set of configuration port CSRs: pci_crp_ad_cbe, pci_crp_wdata, and pci_crp_rdata.

A read access is processed as follows:

1. An AHB master writes the PCI function number, PCI configuration register word address, and read command to the PCI_CRP_AD_CBE register. Note: the PCI Controller has one PCI function so the function number must always be 0.
2. The hardware reads the addressed register in the PCI Core and loads the data into the PCI_CRP_RDATA register. During this operation, any access of CSR space from the AHB is retried.
3. The AHB master reads the pci_crp_rdata register to retrieve the data. Note that this AHB read operation can immediately follow the write in step 1. The hardware retries the read until the data is valid thus making any data validity handshaking transparent to software.

A write access is processed as follows:

1. An AHB master writes the PCI function number, PCI configuration register word address, write command and active-low byte enables to the PCI_CRP_AD_CBE register. Note: the PCI Controller has one PCI function so the function number must always be 0.

2. The AHB master writes the data to be written to the PCI_CRP_WDATA register.
3. The hardware writes the data in PCI_CRP_WDATA to the enabled bytes in the addressed register of the PCI Core. During this operation, any access of CSR space from the AHB is retried.

A burst CSR write operation is used to write PCI_CRP_AD_CBE and PCI_CRP_WDATA to initiate the PCI configuration register write operation.

9.3.2.10.5 AHB Non-Prefetch PCI Accesses

The PCI Controller can generate **non-prefetch** (single) PCI cycles such as I/O Read and Write and Configuration Read and Write cycles from the AHB bus via the non-prefetch CSRs PCI_NP_AD, PCI_NP_CBE, PCI_NP_WDATA, and PCI_NP_RDATA.

A read access is processed as follows:

1. An AHB master writes the 32-bit address of the PCI read cycle to the PCI_NP_AD register.
2. The AHB master writes the PCI Command Type and data byte enables for the desired read cycle to the PCI_NP_CBE register.
3. The hardware sends the read request (address, command, byte enables) to the PCI Core that performs the indicated read cycle and returns the read data. The data is loaded into the pci_np_rdata register. As the read operation is pending, any access of CSR space from the AHB is retried.
4. The AHB master reads the PCI_NP_RDATA register to retrieve the data. Note that this AHB read operation can immediately follow the write in step 2. The hardware retries the read until the data is valid thus making any data validity handshaking transparent to software.

A write access is processed as follows:

1. An AHB master writes the 32-bit address of the PCI write cycle to the PCI_NP_AD register.
2. The AHB master writes the PCI Command Type and data byte enables for the desired write cycle to the PCI_NP_CBE register. Byte enables conform to the PCI little-endian convention.
3. The AHB master writes the data to be written to the PCI_NP_WDATA register.
4. The hardware sends the write request (address, command, byte enables, data) to the PCI Core that performs the indicated write cycle. As the write operation is in progress, any access of CSR space from the AHB is retried.

It should be noted that the AHB Slave Interface and CSR hardware do not interpret the contents of the non-prefetch registers. The address, command, byte enables, and write data are passed to the PCI Core as-is. Thus, for example, I/O read and write requests must be set-up such that the byte enables are consistent with the 2 LSBs of the address in accordance with the PCI Local Bus Specification.

9.3.2.11 PCI Byte Enable Generation

The byte enables for single PCI transactions are generated based on the type of AHB access (direct read/write or non-prefetch read/write), address, transfer size (8-bit, 16-bit, 32-bit), and the settings in effect for AHB endianness and data swapping modes. All 32-bit accesses obviously assert all byte enables when the transaction is sent to the PCI Initiator. Since the AHB Slave Interface only supports bursts with a size of 32-bits, PCI burst transactions always has all byte enables asserted. Table 118 shows the byte enables for 8-bit and 16-bit single PCI read and write cycles generated by direct access of the AHB Slave Interface. PCI cycles generated by the non-prefetch CSRs use the value of the NP_BE field of the PCI_NP_CBE CSR for the byte enables.

Table 118. PCI Byte Enables for Sub-word Single AHB Read/write Cycles

AHB addr[1:0]	pci_csr.ABE	pci_csr.ADS	PCI Byte Enable [3:0]	
			8-bit transfer	16-bit transfer
00	0	0	1110	1100
01	0	0	1101	1100
10	0	0	1011	0011
11	0	0	0111	0011
00	0	1	0111	0011
01	0	1	1011	0011
10	0	1	1101	1100
11	0	1	1110	1100
00	1	0	0111	0011
01	1	0	1011	0011
10	1	0	1101	1100
11	1	0	1110	1100
00	1	1	1110	1100
01	1	1	1101	1100
10	1	1	1011	0011
11	1	1	0111	0011

9.3.3 PCI Controller DMA

The IXP43X network processors contain two channels that are used for DMA (Direct Memory Accesses) to/from the PCI bus and the AHB. The DMA Controller function provides two channels of DMA capability to off load, from the Intel XScale processor, large data transfers between the PCI bus and AHB.

The DMA channels are unidirectional: one DMA channel is used for PCI-to-AHB transfers and one DMA channel is used for AHB-to-PCI transfers. The DMA transfers are implemented using three of the PCI Controller Configuration and Status Registers to specify the PCI address, the AHB address, and the transfer length/control. Each DMA channel has two sets of three registers to provide buffering for consecutive transfers.

For each direction, when a DMA channel is executing one transfer using the active DMA register set, the other DMA register set is set-up by the Intel XScale processor to specify the next transfer. Both DMA channels can run concurrently so that individual PCI-to-AHB transfers and AHB-to-PCI transfers that make up the DMA transfers are interleaved on the AHB and PCI bus.

Individual DMA-complete and DMA-error status indication is provided for each channel using the DMA Control Register (PCI_DMACTRL) with an interrupt that is optionally generated in each case.

The DMA channels share resources with the AHB Master and Slave interfaces and therefore must arbitrate for these resources. AHB-to-PCI DMA transfers use the AHB Master Interface, the PCI Initiator Request, and Initiator Transmit FIFOs. PCI-to-AHB DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Receive FIFO. Use of the AHB Master Interface revolves between the two DMA channels and PCI requests that appear in the Target Receive FIFO.

As a particular channel is accessing the Initiator Request FIFO, accesses to the PCI bus coming in to the AHB Slave Interface from the AHB is retried. The access is flagged by the hardware to signal the DMA Controller channels that a PCI access is pending (AHB masters must attempt retried transfers until complete). This enables the DMA channels to permit the AHB initiated PCI access to go through to the PCI bus.

Additionally, as the AHB Master Interface is in use by a DMA channel, PCI requests that appear in the Target Receive FIFO are flagged to allow these received requests to gain access of the AHB bus.

Access to CSRs from the AHB bus is unrestricted as the DMA channels are operating. Ability to access the PCI Controller Control and Status Registers is provided to allow the Intel XScale processor to set up the off-line DMA Register set while the on-line DMA Register set is operating.

The register sets associated with the DMA channels are as follows:

1. PCI-to-AHB Transfers
   a. Register Set 0
   — PCI-to-AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR)
   — PCI-to-AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR)
   — PCI-to-AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH)
   b. Register Set 1
   — PCI-to-AHB DMA AHB Address Register 1 (PCI_PTADMA1_AHBADDR)
   — PCI-to-AHB DMA PCI Address Register 1 (PCI_PTADMA1_PCIADDR)
   — PCI-to-AHB DMA Length Register 1 (PCI_PTADMA1_LENGTH)
2. AHB-to-PCI Transfers
   a. Register Set 0
   — AHB-to-PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR)
   — AHB-to-PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR)
   — AHB-to-PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH)
   b. Register Set 1
   — AHB-to-PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR)
   — AHB-to-PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR)
   — AHB-to-PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH)

The PCI Address Registers described above are used to specify the beginning 32-bit word address for the PCI side of the DMA transfers. The AHB Address Registers, described above, are used to specify the beginning 32-bit word address for the AHB side of the DMA transfers. The least significant two bits of both addresses are hard-wired to logic 0. Thus, all transfers are word-aligned.

The Length Registers are used for three purposes:

- Sixteen bits to define a word count
  Bits 15:0 of the Length Registers define the word count.
- One bit to enable the DMA transfer
  Bit 31 of the Length Registers enables the DMA transfer to execute. When bit 31 of the Length Registers is set to logic 1, the DMA transfer executes until a word count of zero is reached. When the word count reaches zero and bit 31 of the Length Registers is set to logic 1, bit 31 of the Length Register is cleared to logic 0. When bit 31 of the Length Register is set to logic 0, the register set associated with the

DMA channel is disabled. The second register set may be active and using the DMA channel when the first DMA has finished.

- One bit to define the byte order of the data transferred.
  Bit 28 of the Length Register is used to provide a byte swap on the DMA data as data is transferred from the AHB to the PCI bus or from the PCI Bus to AHB, depending upon the direction of the DMA transfer. When bit 28 is set to logic 1, a byte swap occurs on the DMA data. Figure 74 and Figure 75 demonstrates the DMA transfer byte lane swapping.

Figure 74. AHB-to-PCI DMA-Transfer Byte Lane Swapping

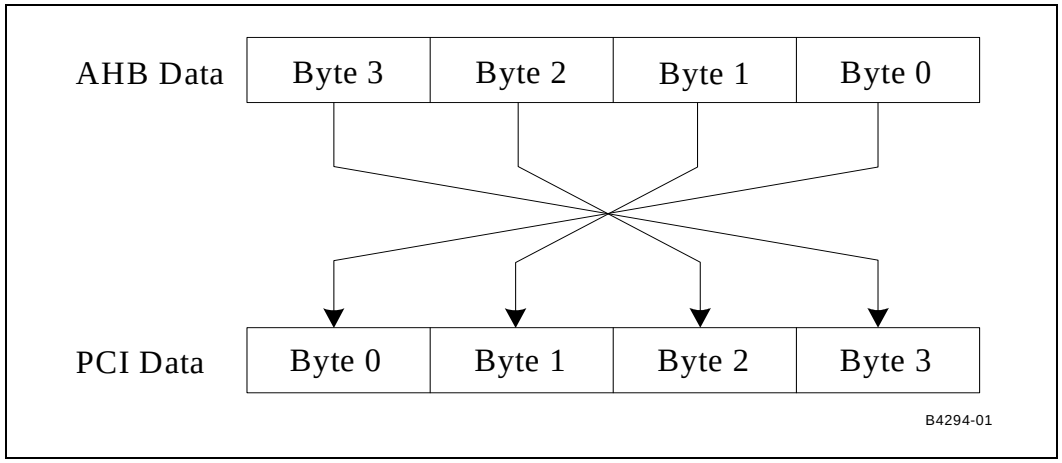
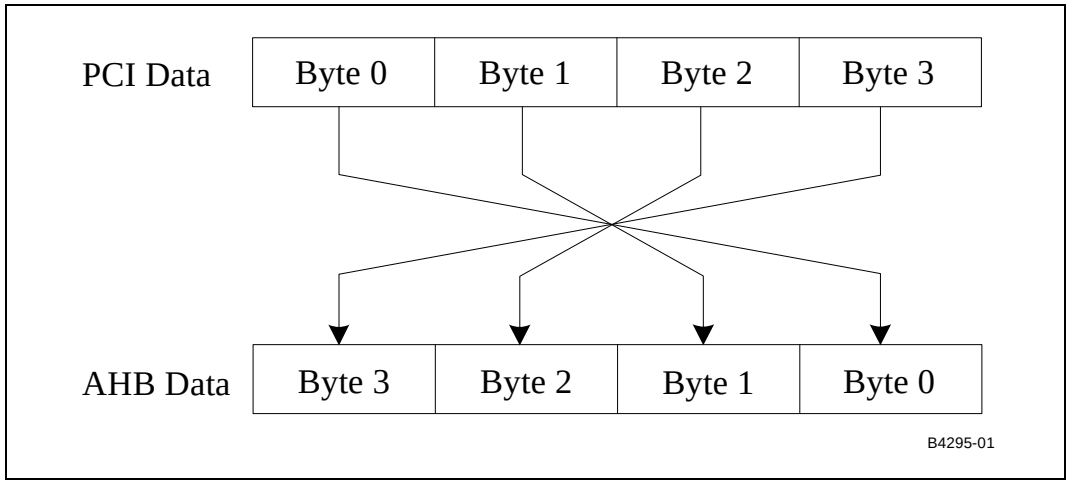


Figure 75. PCI-to-AHB DMA-Transfer Byte Lane Swapping



The DMA channels share resources with the AHB Master and Target interfaces and therefore must arbitrate for these resources. AHB-to-PCI DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Transmit FIFO. PCI-to-AHB DMA transfers use the AHB Master Interface, the PCI Initiator Request FIFO, and Initiator Receive FIFO. Use of the AHB Master Interface revolves between the two DMA channels and PCI requests that appear in the Target Receive FIFO.

A DMA transfer is started on a particular channel by writing the PCI start address, AHB start address, and length to one set of DMA CSRs. If the channel enable bit is set in the length register, that DMA channel is enabled and the transfer is executed. As the transfer is taking place, the other set of DMA CSRs is set-up to specify the next

transfer. When the current transfer is complete, the DMA complete bit is set in the status register and the channel enable bit is cleared in the length register. Execution then starts using the second set of DMA CSRs, if the channel is enabled in the length register. Transfers continue in this fashion until the channel enable bits in both sets of DMA length registers are cleared.

The DMA channels use 8-word burst accesses on the PCI and AHB busses (PCI-to-AHB and AHB -to-PCI) whenever possible. In the general case, a transfer issues a starting burst from 1 to 7 words to align the AHB word address to an 8-word boundary. Then issue 8-word bursts until the words remaining to be transferred is less than 8. Then complete the transfer with a burst of from 1 to 7 words. Eight-word bursts are the maximum sustained length that the AHB and the PCI bus can transfer. Every eight words, the PCI bus disconnects and reconnect later.

This implementation allows fairness among all devices on the PCI bus. In the general case, a transfer issues a beginning burst transfer from one to eight words to align the AHB word address of the DMA to an eight-word boundary.

The subsequent transfers are issued as eight-word bursts until the words remaining to be transferred are eight words or less. The final transfer completes the DMA with a burst of one to eight words.

The example below demonstrates how to use the DMA channels.

The goal is to:

- Write a 16-word burst to the PCI Bus with no byte-swapping, using the AHB-to-PCI DMA channel
- Initialize a 16-word burst read from the PCI Bus, using the PCI-to-AHB DMA channel
- Initialize a six-word burst write to the PCI Bus using the AHB-to-PCI DMA channel.

  The AHB-to-PCI DMA channel is used to complete PCI Memory Cycle write accesses and the PCI-to-AHB DMA channel is used to complete PCI Memory Cycle read accesses always.

1. Update the AHB-to-PCI DMA AHB Address Register 0 (PCI_ATPDMA0_AHBADDR) with PCI_ATPDMA0_AHBADDR = 0x00004000 and the AHB-to-PCI DMA PCI Address Register 0 (PCI_ATPDMA0_PCIADDR) with PCI_ATPDMA0_PCIADDR = 0xFC000004.
2. Update the AHB-to-PCI DMA AHB Address Register 1 (PCI_ATPDMA1_AHBADDR) with PCI_ATPDMA1_AHBADDR = 0x00004F00 and the AHB-to-PCI DMA PCI Address Register 1 (PCI_ATPDMA1_PCIADDR) with PCI_ATPDMA1_PCIADDR = 0xA2000004.
3. Update the PCI-to-AHB DMA AHB Address Register 0 (PCI_PTADMA0_AHBADDR) with PCI_PTADMA0_AHBADDR = 0x00004A00 and the PCI-to-AHB DMA PCI Address Register 0 (PCI_PTADMA0_PCIADDR) with PCI_PTADMA0_PCIADDR = 0x10000004.
4. Update the AHB-to-PCI DMA Length Register 0 (PCI_ATPDMA0_LENGTH) with PCI_ATPDMA0_LENGTH = 0x80000010.

   The DMA write transfer to the PCI bus begins.
5. Update the PCI-to-AHB DMA Length Register 0 (PCI_PTADMA0_LENGTH) with PCI_PTADMA0_LENGTH = 0x80000010.

   Assume that this DMA channel is enabled prior to the end of the first eight-word burst of the first write DMA transfer ending. The DMA read transfer to the PCI bus becomes interleaved with the first write transfer. So the first eight words of the read starts towards completion.

6. Update the AHB-to-PCI DMA Length Register 1 (PCI_ATPDMA1_LENGTH) with PCI_ATPDMA1_LENGTH = 0x90000006.

   Assume this is set while the above read DMA transaction is occurring.

7. The next PCI transfer is completing the last eight words of the initial 16-word write DMA transfer. That is followed by the last eight words of the 16-word read DMA transfer and the execution of the six-word write transfer with the data byte lanes swapped.

9.3.3.1 AHB-to-PCI DMA Channel Operation

The AHB-to-PCI (ATP) channel uses the PCI Core Initiator Request and Initiator Transmit FIFOs. The channel reads data from the AHB bus and writes it to a PCI target on word aligned boundaries.

A DMA transfer from AHB-to-PCI is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_ATPDMA0/1_PCIADDR, PCI_ATPDMA0/1_AHBADDR, PCI_ATPDMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_ATPDMA0/1_LENGTH register, the DMA transfer commences.
2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for the Transmit FIFO to become empty.
3. The DMA Controller requests access to the AHB Master Interface which it shares with the other DMA channel and accesses from the PCI bus.
4. When access is obtained, data is read from AHB and loaded into the Initiator Transmit FIFO. A PCI write request is loaded into the Initiator Request FIFO.
5. The AHB Master and Slave Interfaces are released.
6. When the transfer completes on the PCI bus, the DMA address and length registers are updated.
7. Steps 2-6 are repeated until all words are transferred. When done, the channel enable bit in the PCI_ATPDMA0/1_LENGTH register is cleared, the DMA complete status bit is set.
8. In response to the interrupt, an AHB agent may read the DMA Control register (pci_dmactrl) to determine the status of the transfer.

9.3.3.2 PCI-to-AHB DMA Channel Operation

The PCI-to-AHB (PTA) channel uses the PCI Core Initiator Request and Initiator Receive FIFOs. The channel reads data from the PCI bus and writes it to an AHB slave on word aligned boundaries. A DMA transfer from PCI-to-AHB is processed as follows:

1. An AHB master writes the PCI starting address, AHB starting address, and word count to the PCI_PTADMA0/1_PCIADDR, PCI_PTADMA0/1_AHBADDR, PCI_PTADMA0/1_LENGTH registers respectively. If the channel enable bit is set in the PCI_PTADMA0/1_LENGTH register, the DMA transfer commences.
2. The DMA Controller signals the AHB Slave Interface to retry all access attempts from the AHB bus and waits for non-full indication from the Initiator Request FIFO.
3. The DMA Controller issues a read request to the PCI Core via the Request FIFO and releases the AHB Slave Interface.
4. When the PCI data arrives in the Initiator Receive FIFO, the DMA Controller requests access to the AHB Master Interface which it shares with the AHB-to-PCI DMA channel and accesses from the PCI bus.
5. When access is obtained, data is read from the Target Receive FIFO and written to the AHB agent.

6. When the transfer completes on the AHB bus, the DMA address and length registers are updated.
7. Steps 2-6 are repeated until all words are transferred. When done, the channel enable bit in the PCI_PTADMA0/1_LENGTH register is cleared, the DMA complete status bit is set.
8. In response to the interrupt, an AHB agent may read the DMA Control register (PCI_DMACTRL) to determine the status of the transfer.

9.3.4 Data Byte Alignment and Addressing — PCI Endianness

The *PCI Local Bus Specification*, Rev. 2.2 defines the byte-addressing convention on the PCI Bus as little-endian. Since the byte addressing convention on the PCI bus is little-endian and the convention used on the AHB bus is big-endian, data passing from the PCI Core FIFOs to and from the AHB bus can go through an optional CSR-bit controlled byte lane reversal. The AHB bus interfaces can also be configured to operate in big-endian or little-endian addressing conventions.

As shown in Figure 76, when an external PCI device accesses an AHB address with the AHB in big-endian mode (pci_csr.ABE = 1), the pci_csr.PDS (PCI Data Swap) bit controls the byte lane swapping between the two busses. When pci_csr.PDS is 1, the PCI data bytes are treated as little-endian addressed and are swapped to the corresponding big-endian byte lanes of the AHB bus. In the figure, PCI bytes are numbered according to the corresponding PCI byte enables and AHB bytes are numbered according to the corresponding byte address on the AHB bus. For example, a PCI write with byte enable 2 asserted and pci_csr.PDS = 1 produces an AHB byte write with AHB address bits [1:0] = 10b. When pci_csr.PDS is 0, the PCI data bytes are treated as big-endian addresses and are not swapped. In this case, a PCI write with PCI byte enable 2 asserted produces an AHB byte write with AHB address bits [1:0] = 01b.

Figure 77 shows the byte alignment in AHB little-endian mode (pci_csr.ABE = 0).

Figure 76. Byte Lane Routing during PCI Target Accesses of the AHB Bus – Big-Endian AHB Bus

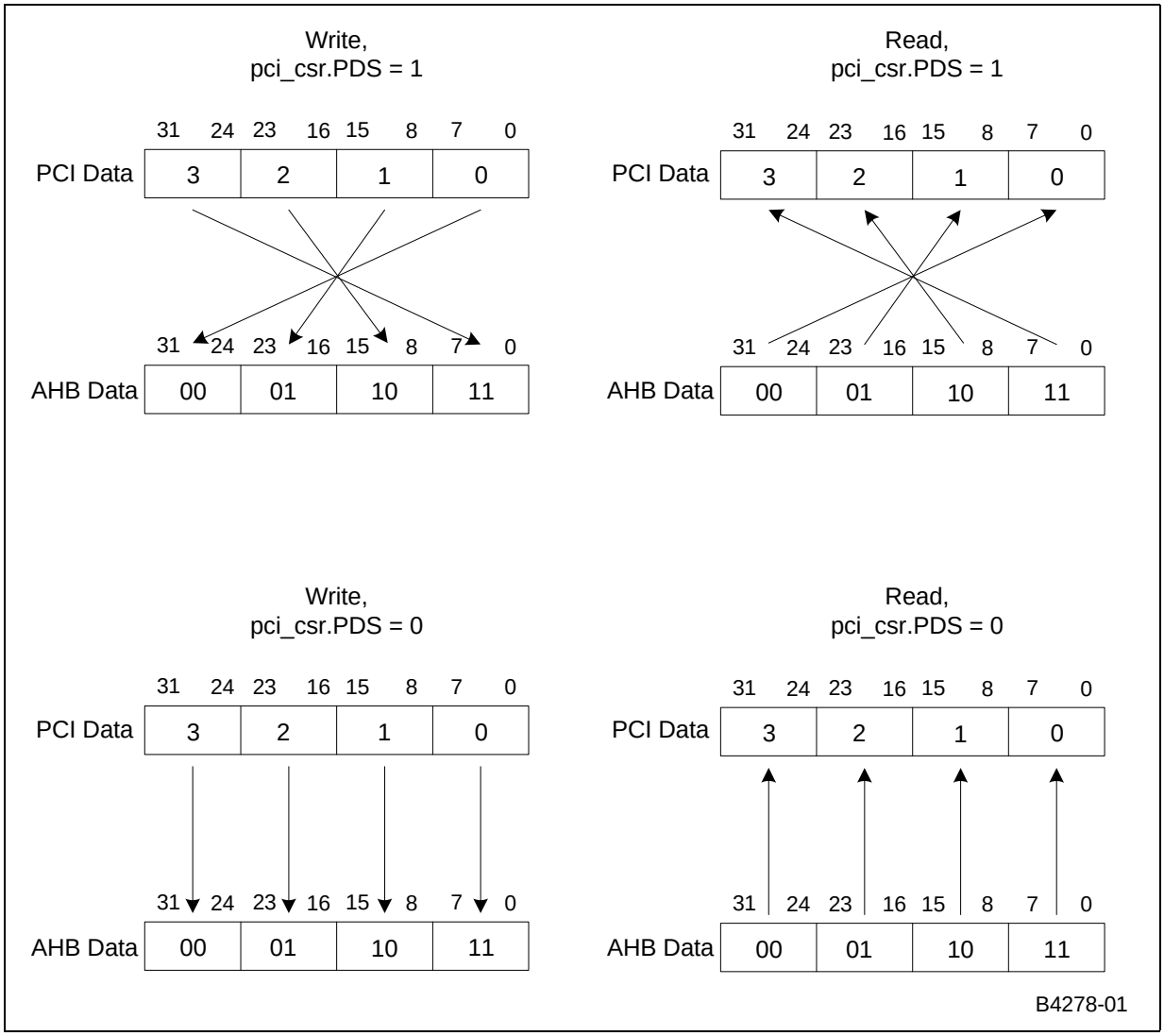
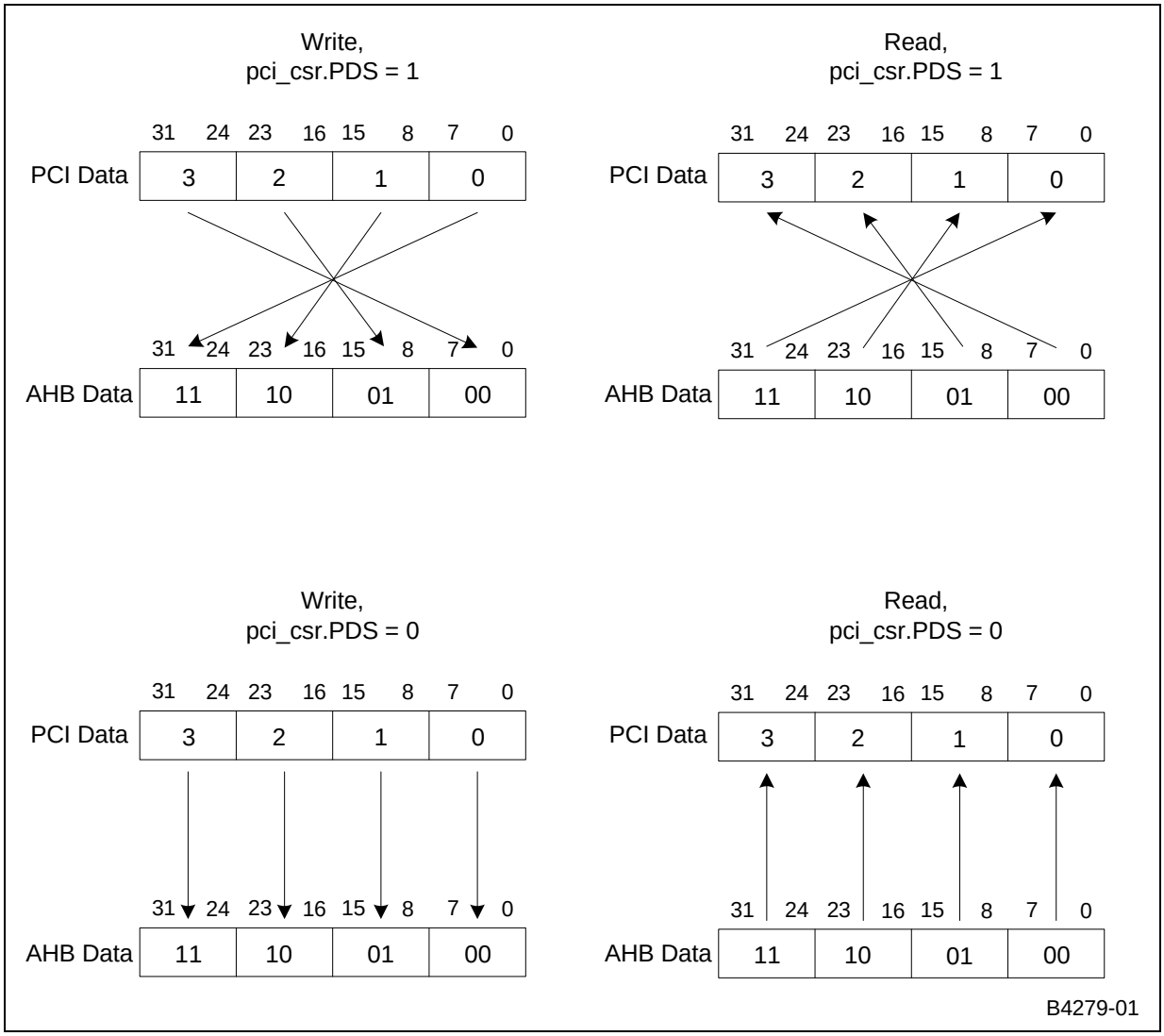


Figure 77. Byte Lane Routing during PCI Target Accesses of the AHB Bus – Little-Endian AHB Bus



In a similar manner, the pci_csr.ADS bit controls byte lane routing when an AHB agent accesses an external PCI device using memory-mapped accesses (no swapping is performed during the CSR-controlled non-prefetch cycles). Figure 78 and Figure 79 illustrate the data routing in AHB big-endian and little-endian modes respectively.

Figure 78. Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Big-Endian AHB Bus

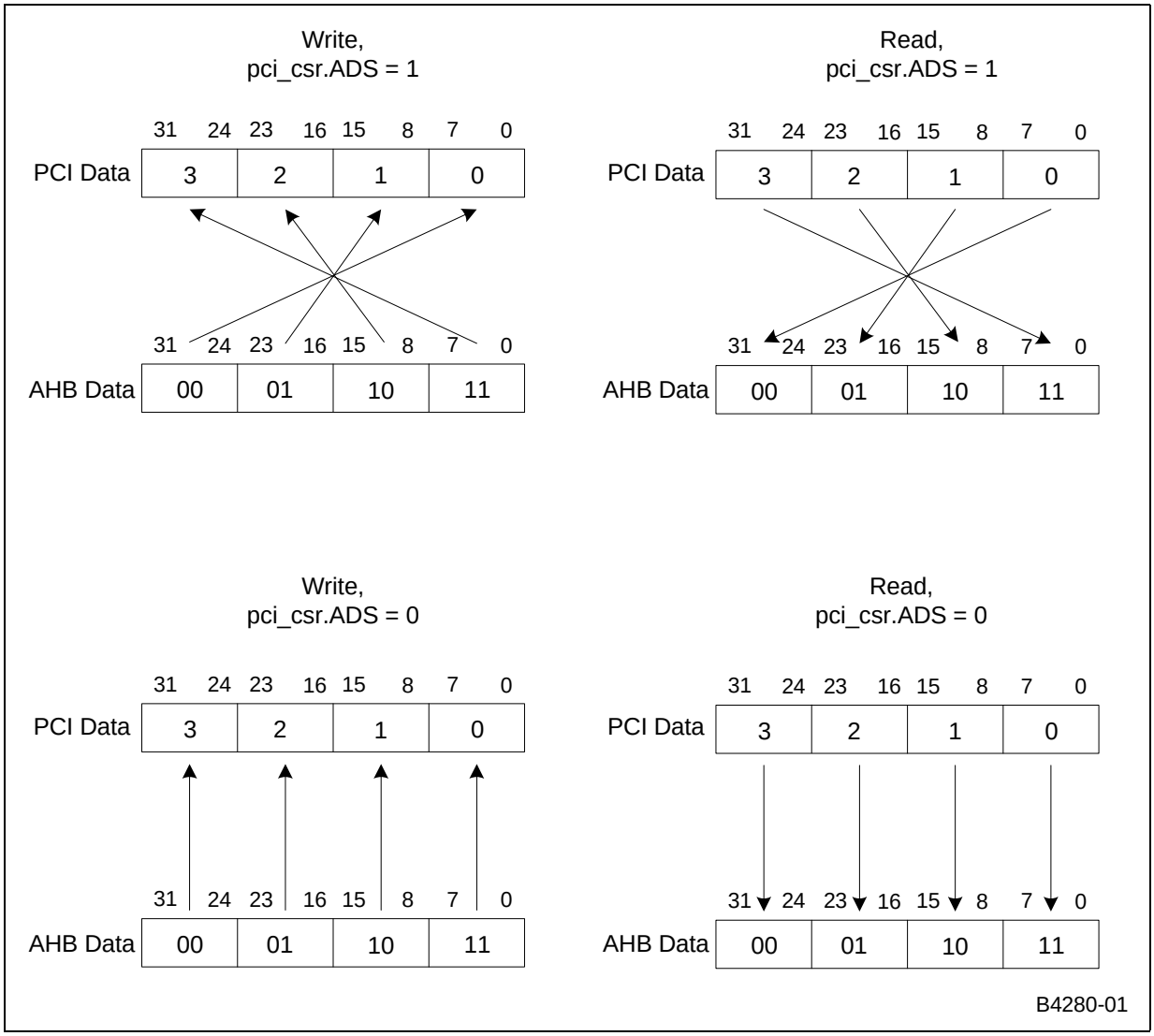
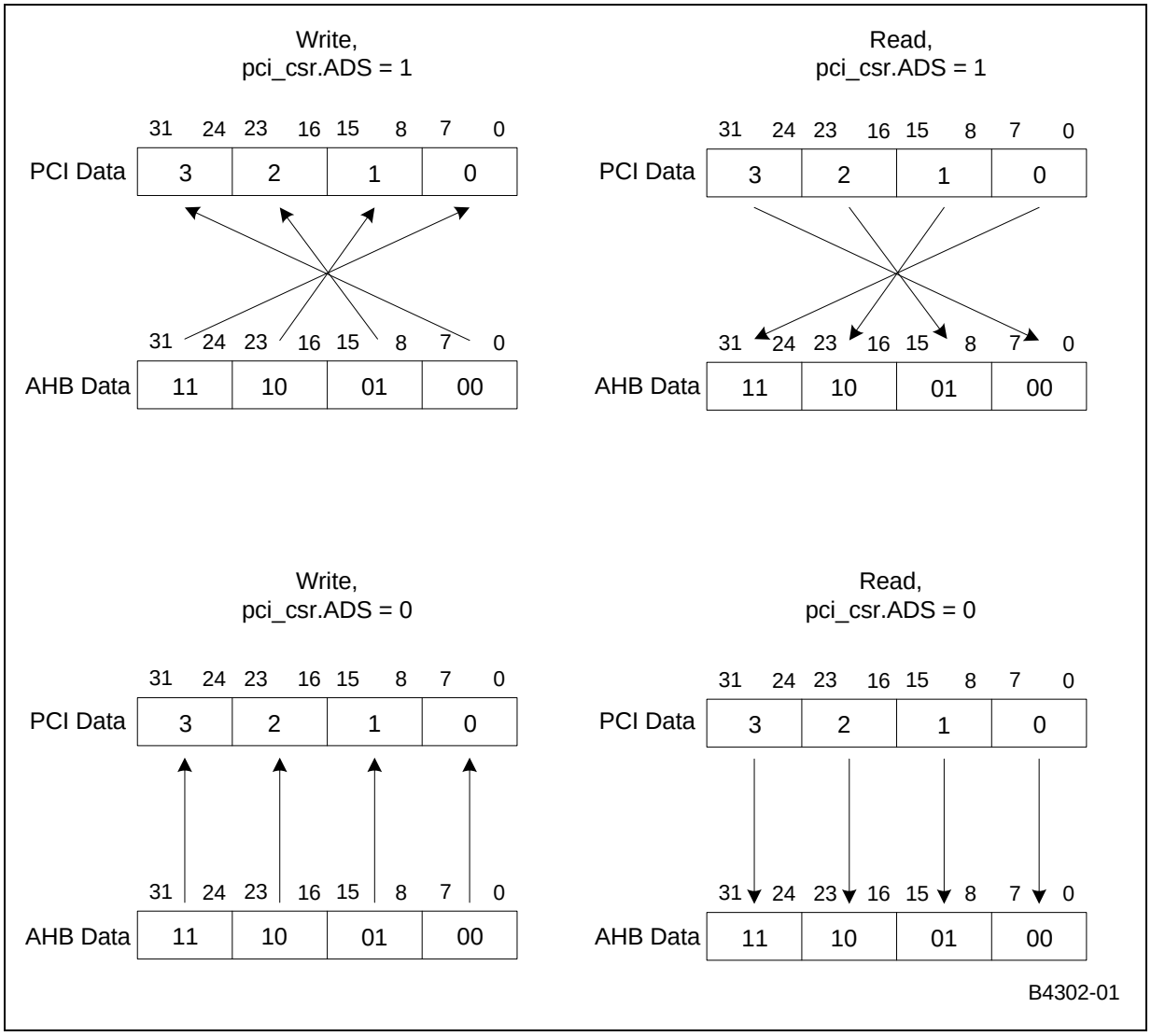
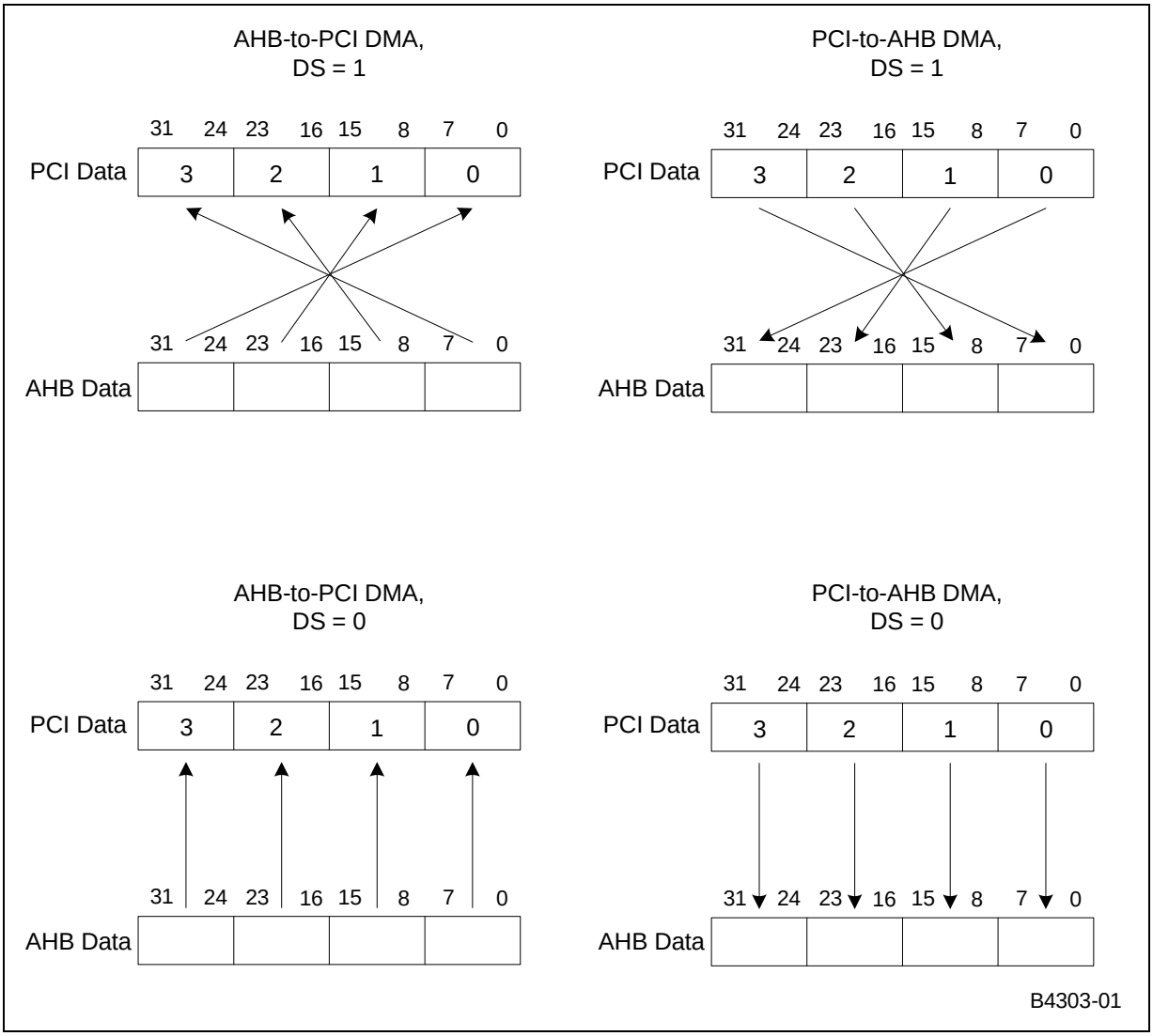


Figure 79. Byte Lane Routing During AHB Slave Accesses of the PCI Bus – Little-Endian AHB Bus



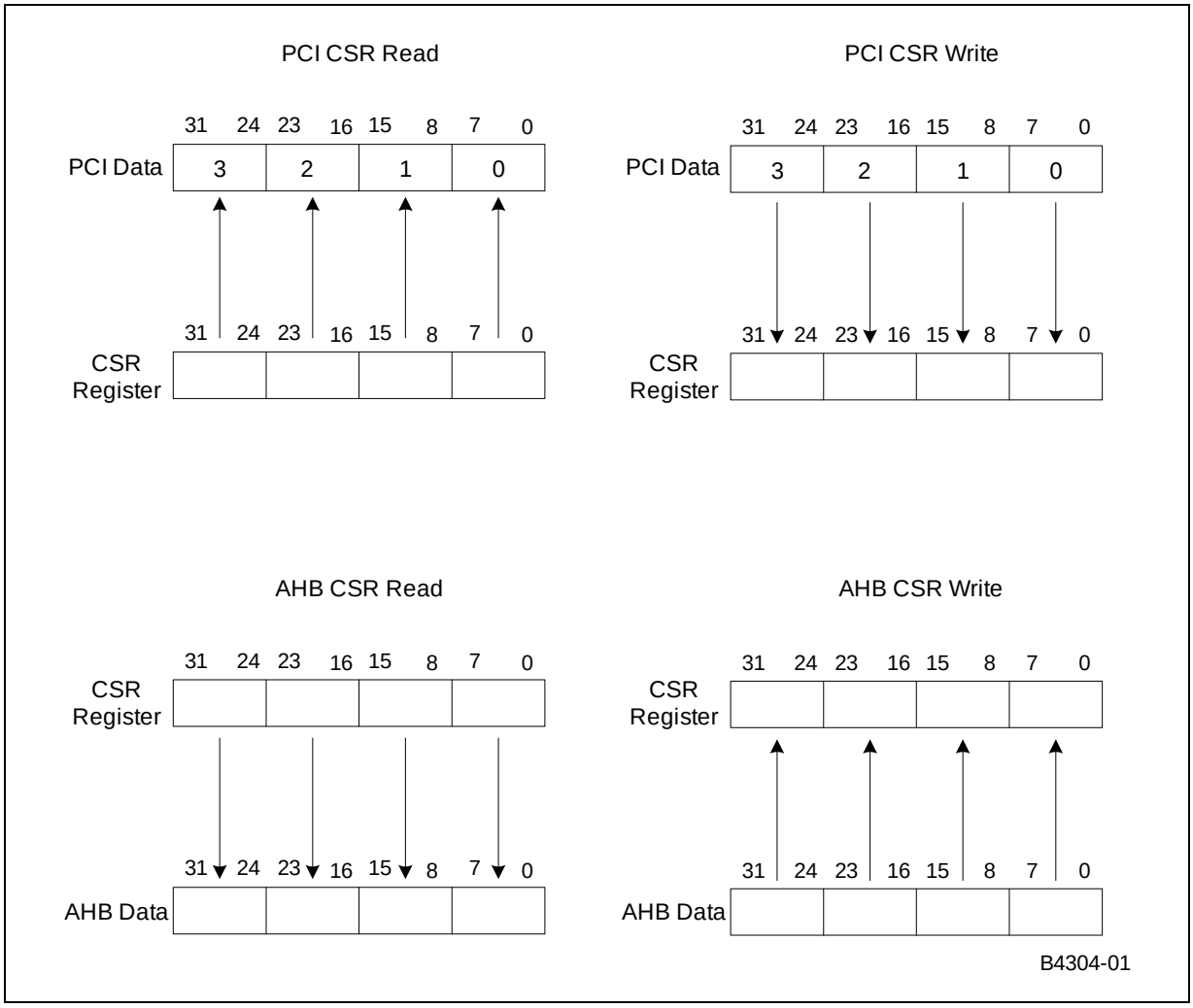
During DMA transfers, byte lane routing is controlled by the DS bit in the DMA length register as shown in Figure 80.

Figure 80. Byte Lane Routing During DMA Transfers



There is no byte lane reversal for accesses to PCI Controller CSRs or PCI Configuration registers. Figure 81 shows the byte lane routing for these types of accesses. A PCI CSR write with byte enable 2 asserted, for example, write to bits 23:16 of the register. An AHB write with Address bits [1:0] = 10b in big-endian mode (pci_csr.ABE = 1) writes bits 15:8 of the register. In little-endian mode (pci_csr.ABE = 0), bits 23:16 are written.

Figure 81. Byte Lane Routing During CSR Accesses



9.3.5 PCI Controller Interrupts

The PCI Controller supports generation of PCI interrupts and interrupts to internal AHB agents. Complete control of the interrupt sources and enabling is provided using two registers: the PCI Interrupt Status Register (PCI_ISR) and PCI Interrupt Enable Register (PCI_INTEN).

9.3.5.1 PCI Interrupt Generation

The PCI Door Bell Register (PCI_PCIDOORBELL) is used to generate a PCI interrupt on the PCI Bus using the PCI_INTA_N signal. This register is read/write-1-to-set from the AHB bus, and read/write-1-to-clear from the PCI bus. All bits are ORed together to generate the PCI interrupt. The sequence is:

- An AHB agent writes a pattern of ones to the PCI_PCIDOORBELL register, setting the corresponding bits in the register.
- The interrupted PCI device reads the bit pattern in the doorbell register and writes the same pattern back to clear the bits and deassert the interrupt.

The PCI interrupt is enabled by the PDB bit of the Interrupt Enable Register (pci_inten). When this bit is set and at least 1 bit is set in the pci_pcidoorbell register, an interrupt is asserted on the PCI_INTA_N open-drain output.

The PDB bit, Bit 7 of the PCI Interrupt Enable Register (PCI_INTEN) is used to enable the external PCI Interrupt. When bit 7 is set to logic 1, the external PCI Interrupt logic is enabled, an interrupt is asserted on the PCI_INTA_N open-drain output. When bit 7 is set to logic 0 the external PCI Interrupt logic is disabled.

Bit 7 of the PCI Interrupt Status Register (PCI_ISR) displays the status of the external PCI Interrupt. This bit is set to logic 1 when any of the PCI_PCIDOORBELL bits are set to logic 1.

9.3.5.2 Internal Interrupt Generation

The PCI Controller employs three internal interrupt outputs to signal AHB agents of the occurrence of various events. The PCC_INT signal is a general-purpose interrupt, whereas PCC_ATPDMA_INT, and PCC_PTADMA_INT are DMA interrupts. All interrupts are high-active and remain asserted until an AHB agent clears the interrupting source by writing appropriate CSR register bits in the PCI Controller.

The general-purpose interrupt pcc_int is asserted when:

- A PCI error occurs
- An AHB error occurs
- An AHB-to-PCI DMA transfer is complete or terminates due to an error
- A PCI-to-AHB DMA transfer is complete or terminates due to an error
- A doorbell is **pushed** by an external PCI device

The PCI_ISR register indicates the source(s) of the PCC_INT interrupt. The PCC_INTEN register provides an enable for each of the sources in PCI_ISR. If a bit is set in PCI_ISR and its corresponding enable is set in PCI_INTEN, the PCC_INT output is asserted high-active. The interrupt remains asserted until the source in PCI_ISR or the enable in PCI_INTEN is cleared. Clearing an interrupt source may involve clearing bits in other CSR registers.

The PCI_AHBDOORBELL register is used to generate the doorbell interrupt to an AHB agent. This register is read/write-1-to-set from the PCI bus, and read/write-1-to-clear from the AHB bus. All bits are ORed together to generate the interrupt. The sequence is:

- an external PCI agent writes a pattern of ones to the PCI_AHBDOORBELL register, setting the corresponding bits in the register and asserting the interrupt to the AHB agent.
- the AHB agent reads the bit pattern in the doorbell register and writes the same pattern back to clear the bits and deassert the interrupt.

The DMA interrupt PCC_ATPDMA_INT is asserted when the APDCEN of the DMA Control register is set and the DMA transfer completes or terminates due to an error. DMA completion, with or without error, is indicated by APDC0 or APDC1 bits being set, where 0/1 corresponds to the DMA register set used for the transfer (PCI_ATPDMA0/1_xxx).

Likewise, the DMA interrupt PCC_PTADMA_INT is asserted when the PADCEN of the DMA Control register is set and the DMA transfer completes or terminates due to an error. DMA completion, with or without error, is indicated by PADC0 or PADC1 bits being set, where 0/1 corresponds to the DMA register set used for the transfer (PCI_PTADMA0/1_xxx).

Note that DMA completions can generate an interrupt on the DMA interrupt outputs or on the general-purpose interrupt PCC_INT. Separate DMA interrupt enables are provided for each interrupt output signal but there remains just one interrupt source for each DMA channel in the DMA Control register.

9.4 Register Descriptions

The following sections describe the internal PCI Configuration registers. Registers or fields defined as reserved return 0s when read and are not affected by writes. The access key is as follows:

Access Type	Description
RO	Read-only
RW	Read/write
RW1C	Read/write 1 to clear – normal read access. Writing a 1 to a bit position clears the bit, writing a 0 has no effect.
RW1S	Read/write 1 to set – normal read access. Writing a 1 to a bit position sets the bit, writing a 0 has no effect.

9.4.1 PCI Configuration Registers

These registers comprise the configuration registers as defined in the PCI 2.2 specification with the exception of the pci_rtotto register which is device specific. They are accessible from the PCI bus using configuration read and write transactions and from the Intel XScale processor by accessing the PCI Controller CSR-based PCI Configuration register port. Table 119 lists the registers.

Table 119. PCI Configuration Register Map

Offset Address	Register Name	Description	Reset Value	Page
0x00	pci_didvid	Device ID/Vendor ID	0x85028086	339
0x04	pci_srcr	Status Register/Control Register	0x02800000	339
0x08	pci_ccrid	Class Code/Revision ID	0x0b4000XY	341
0x0c	pci_bhlc	BIST/Header Type/Latency Timer/Cache Line	0x00000000	341
0x10	pci_bar0	Base Address 0	0x00000008	342
0x14	pci_bar1	Base Address 1	0x00000008	342
0x18	pci_bar2	Base Address 2	0x00000008	343
0x1c	pci_bar3	Base Address 3	0x00000008	343
0x20	pci_bar4	Base Address 4	0x00000008	344
0x24	pci_bar5	Base Address 5	0x00000001	344
0x28	(Reserved)			
0x2c	pci_sidsvid	Subsystem ID/Subsystem Vendor ID	0x00000000	345
0x30-38	(Reserved)			
0x3c	pci_latent	Defines Max_Lat, Min_Gnt, Interrupt Pin, and Interrupt Line	0x00000100	345
0x40	pci_rtotto	Defines retry timeout and trdy timeout parameters	0x00008080	346

9.4.2 PCI Configuration Register Descriptions

The following sections describes the internal PCI Configuration registers.

9.4.2.1 Device ID/Vendor ID Register

The tables below describes the Device ID/Vendor ID Register:

Register Name:	pci_didvid				
Block Base Address:	0xC00000	Offset Address	0x00	Reset Value	0x85028086
Register Description:	Provides Device ID and Vendor ID values as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
DeviceID																VendorID															

Register	pci_didvid				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	DeviceID	Unique device identifier assigned by Intel. The state of the tlu_pci_id_sel input determines the device Id that is used.	0x8501	RO	RO
15:0	VendorID	Unique vendor identifier assigned to Intel by PCISIG	0x8086	RO	RO

9.4.2.2 Status Register/Control Register

The tables below describes the Status Register/Control Register:

Register Name:	pci_srcr				
Block Base Address:	0xC00000	Offset Address	0x04	Reset Value	0x02800000
Register Description:	Contains the Command and Status registers as specified in the PCI 2.2 Local Bus Specification			Access:	(See below.)

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
DPE	SSE	RMA	RTA	STA	DEVSEL		MDPE	FBBC	UDF	66MHZ	CLI	(Reserved)										FBBE	SER	SC	PER	PSE	MWIE	SCE	BME	MAE	IOAE

Register	pci_srcr (Sheet 1 of 2)				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	DPE	Detected Parity Error. Set when this device detects a parity error on the bus even when parity handling is disabled. Writing a 1 to this bit clears it.	0	RW1C	RW1C
30	SSE	Signaled System Error. Set when this device generates a System Error PCI_SERR_N. Writing a 1 to this bit clears it.	0	RW1C	RW1C
29	RMA	Received Master Abort. Set by this device as a Master when its transaction terminates due to a master abort (except for special cycles). Writing a 1 to this bit clears it.	0	RW1C	RW1C
28	RTA	Received Target Abort. Set by this device as a Master when its transaction is terminated due to a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C

Register		pci_srcr (Sheet 2 of 2)			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
27	STA	Signaled Target Abort. Set by this device as a Target when it terminates a transaction with a target abort. Writing a 1 to this bit clears it.	0	RW1C	RW1C
26:25	DEVSEL	Defines the DEVSEL speed for this device. Set to medium.	01	RO	RO
24	MDPE	Master Data Parity Error. Set by this device as a Master if PER (bit 6) is set and this device asserted the PCI_PERR_N signal or saw PCI_PERR_N asserted for one of its data phases.	0	RW1C	RW1C
23	FBBC	Fast Back-to-Back Capable.	1	RO	RO
22	UDF	User Definable Features supported. 0 = not supported	0	RO	RO
21	66MHz	66MHZ capable. Indicates if this device is capable of 66MHz operation. 1 = 66MHz capable 0 = 33MHz capable ***Note:*** The IXP43X network processors support 33MHz only. Users are advised NOT to overwrite this bit to '1' to avoid unpredictable results.	0	RO	RW
20	CLI	Capabilities List Indicator, Not supported	0	RO	RO
19:10	-	reserved	00	RO	RO
9	FBBE	Fast Back-to-Back Enable. When set to a 1 enables the device to generate fast back-to-back cycles to different targets as a Master.	0	RW	RW
8	SER	System Error Enable. When set to a 1, enables the PCI_SERR_N output driver. 0 disables the driver.	0	RW	RW
7	SC	Stepping Control. When set to a 1, enables address stepping on the bus. This feature not supported.	0	RO	RO
6	PER	Parity Error Response. When set to a 1, enables reporting of parity errors on PCI_PERR_N. When set to 0, parity errors not reported on PCI_PERR_N but the DPE bit (bit 31) is still set.	0	RW	RW
5	PSE	Palette Snoop Enable. When set to a 1, enables VGA palette snooping. This feature not supported.	0	RO	RO
4	MWIE	Memory Write and Invalidate Enable. When set to a one, enables this device to generate the memory write and invalidate command.	0	RW	RW
3	SCE	Special Cycle Enable. When set, enables this device to monitor for Special Cycles. This feature not supported.	0	RO	RO
2	BME	Bus Master Enable. When set, enables this device to act as a bus Master.	0	RW	RW
1	MAE	Memory Access Enable. When set to a 1, enables memory accesses as a target.	0	RW	RW
0	IOAE	I/O Access Enable. When set to a 1, enables I/O accesses as a target.	0	RW	RW

9.4.2.3 Class Code/Revision ID Register

The tables below describes the Class Code/Revision ID Register:

Register Name:	pci_ccrid				
Block Base Address:	0xC00000	Offset Address	0x08	Reset Value	0x0b4000XY XY = pci_revision_id[7:0]
Register Description:	Provides Class Code and Revision ID values as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
Class Code								Sub-Class Code								Interface								RevisionID							

Register	pci_ccrid			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	Class Code	Class/Sub-Class identifier for the device as defined in the PCI specification. 0x0b = processor	0x0b	RO	RW
23:16	Sub-Class	Sub-Class identifier for Class Code 0x0b. 0x40 = coprocessor	0x40	RO	RW
15:8	Interface	Programming Interface code. Always 0x00 for this class.	0x00	RO	RW
7:0	RevisionID	Silicon revision for the device.	0x01	RO	RW

9.4.2.4 BIST/Header Type/Latency Timer/Cache Line Register

The tables below describe the BIST/Header Type/Latency Timer/Cache Line Register:

Register Name:	pci_bhlc				
Block Base Address:	0xC00000	Offset Address	0x0c	Reset Value	0x00000000
Register Description:	Provides BIST, Header Type, Latency Timer, and Cache Line Size registers as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
BIST								HeaderType								LatencyTimer								cache line							

Register	pci_bhlc				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	BIST	BIST control register, not supported	0x00	RO	RO
23	Header Type[7]	Single Function/Multi-Function Device. Set to 0 to identify this device as a single-function PCI device.	0	RO	RO
22:16	Header Type[6:0]	Configuration Header Type for this device. Set to 00	0x00	RO	RO
15:10	Latency Timer[7:2]	Latency Timer value in units of four PCI bus clocks.	0x00	RW	RW
9:8	Latency 1 Timer[1:0]	Hard-wired low order Latency Timer bits	0x0	RO	RO
7:0	Cache Line	Cache Line Size in units of 32-bit words.	0x00	RW	RW

9.4.2.5 Base Address 0 Register

The tables below describe the Base Address 0 Register:

Register Name:	pci_bar0				
Block Base Address:	0xC00000	Offset Address	0x10	Reset Value	0x00000008
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase								FixedBase								FixedBase												PREF	Type		MSI

Register	pci_bar0				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

9.4.2.6 Base Address 1 Register

The tables below describe the Base Address 1 Register:

Register Name:	pci_bar1				
Block Base Address:	0xC00000	Offset Address	0x14	Reset Value	0x00000008
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase								FixedBase								FixedBase												PREF	Type		MSI

Register	pci_bar1				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

9.4.2.7 Base Address 2 Register

The tables below describe the Base Address 2 Register:

Register Name:	pci_bar2				
Block Base Address:	0xC00000	Offset Address	0x18	Reset Value	0x00000008
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase								FixedBase								FixedBase												PREF	Type		MSI

Register		pci_bar2			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

9.4.2.8 Base Address 3 Register

The tables below describe the Base Address 3 Register:

Register Name:	pci_bar3				
Block Base Address:	0xC00000	Offset Address	0x1c	Reset Value	0x00000008
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase								FixedBase								FixedBase												PREF	Type		MSI

Register		pci_bar3			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

9.4.2.9 Base Address 4 Register

The tables below describe the Base Address 4 Register:

Register Name:	pci_bar4				
Block Base Address:	0xC00000	Offset Address	0x20	Reset Value	0x00000008
Register Description:	PCI Base Address register for AHB memory space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase								FixedBase								FixedBase												PREF	Type		MSI

Register	pci_bar4				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	RWBase	Read/Write bits of Base Address register.	0x00	RW	RW
23:4	FixedBase	Read-only bits of Base Address register. Specifies fixed 16MB address range for this BAR.	0x000	RO	RO
3	PREF	Prefetchable memory indicator.	1	RO	RO
2:1	Type	Relocatable anywhere in 32-bit address space.	00	RO	RO
0	MSI	Memory space indicator. Hard-wire to 0 for memory space.	0	RO	RO

9.4.2.10 Base Address 5 Register

The tables below describe the Base Address 5 Register:

Register Name:	pci_bar5				
Block Base Address:	0xC00000	Offset Address	0x24	Reset Value	0x00000001
Register Description:	PCI Base Address register for AHB I/O space access. Format as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
RWBase																									FixedBase					(Rsv'd)	IOSI

Register	pci_bar5				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	RWBase	Read/Write bits of Base Address register.	0x000000	RW	RW
7:2	FixedBase	Read-only bits of Base Address register. Specifies fixed 256 byte address range for this BAR.	0x00	RO	RO
1	reserved	Reserved	0	RO	RO
0	IOSI	I/O space indicator. Hard-wire to 1 for I/O space.	1	RO	RO

9.4.2.11 Subsystem ID/Subsystem Vendor ID Register

The tables below describe the Subsystem ID/Subsystem Vendor ID Register:

Register Name:	pci_sidsvid				
Block Base Address:	0xC00000	Offset Address	0x2c	Reset Value	0x00000000
Register Description:	Provides Subsystem Device ID and Susbsystem Vendor ID values as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
SDeviceID																SVendorID															

Register	pci_sidsvid				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	SDeviceID	Subsytem Device ID	0x0000	RO	RW
15:0	SVendorID	Subsystem Vendor ID	0x0000	RO	RW

9.4.2.12 Max_Lat, Min_gnt, Interrupt Pin and Interrupt Line Register

The tables below describe the Max_Lat, Min_gnt, Interrupt Pin and Interrupt Line Register:

Register Name:	pci_latent				
Block Base Address:	0xC00000	Offset Address	0x3c	Reset Value	0x00000100
Register Description:	Miscellaneous register provides Max Latency, Min Grant, Interrupt Pin and Interrupt Line parameters as specified in the PCI 2.2 Local Bus Specification.			Access:	(See below.)

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
MaxLat								MinGnt								IntPin								IntLine							

Register	pci_latent				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	MaxLat	Indicates how often this device needs access to the bus, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
23:16	MinGnt	Indicates the time interval required for a burst operation, in units of 0.25us. Used by configuration software to set the value of the Latency Timer.	0x00	RO	RW
15:8	IntPin	Indicates the interrupt pin that this device connects to. Set to connect to INT_A#.	0x01	RO	RW
7:0	IntLine	Indicates interrupt line routing information.	0x00	RW	RW

9.4.2.13 Retry Timeout/trdy Timeout Register

The tables below describe the Retry Timeout/trdy Timeout Register:

Register Name:	pci_rtotto				
Block Base Address:	0xC00000	Offset Address	0x40	Reset Value	0x00008080
Register Description:	Specifies values for the Retry and TRDY timeout timers.			Access:	(See below.)

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
Reserved																RetryTO								TRDYTO							

Register	pci_rtotto				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:16	reserved	Reserved	0x00	RO	RO
15:8	RetryTO	Specifies value for the Retry timer. Specifies the maximum number of retries the Master Interface accepts before terminating the transaction. A value of 0 disables the timer and allows an unlimited number of Retry responses.	0x80	RW	RW
7:0	TRDYTO	Specifies value for the TRDY timer. Specifies the number of PCI clocks the Master Interface waits before terminating a transfer with Master Abort when a target accepts a transaction by asserting PCI_DEVSEL_N but does not assert PCI_TRDY_N or PCI_STOP_N. A value of 0 disables the timer and the Master Interface waits indefinitely for the target to respond.	0x80	RW	RW

9.4.3 PCI Controller Configuration and Status Registers (CSRs)

These registers are accessible from the AHB bus and are memory mapped in the AHB address space. When the arbs_hsel_ppc_mmr is asserted, the lower AHB address bits select the accessed register. Registers are byte writable with individual bytes addressed according to the endianness setting of the AHB bus in pci_csr.ABE. Table 120 shows the address map for the CSRs. The AHB offset is relative to the base address for PCI Controller memory mapped registers in the AHB address space. The PCI offset is relative to the base address in pci_bar4 for accesses from the PCI bus. Byte addressing from PCI uses the PCI byte enable convention.

Table 120. CSR Address Map (Sheet 1 of 2)

AHB Offset	PCI Offset	Register Name	Description	Reset Value	Page
0x00	0x00	pci_np_ad	PCI non-prefetch address register	0x00000000	347
0x04	0x04	pci_np_cbe	PCI non-prefetch command/byte enables register	0x00000000	348
0x08	0x08	pci_np_wdata	PCI non-prefetch write data register	0x00000000	348
0x0c	0x0c	pci_np_rdata	PCI non-prefetch read data register	0x00000000	349
0x10	0x10	pci_crp_ad_cbe	PCI configuration port address/cmd/byte enables register	0x00000000	349
0x14	0x14	pci_crp_wdata	PCI configuration port write data register	0x00000000	350
0x18	0x18	pci_crp_rdata	PCI configuration port read data register	0x00000000	350
0x1c	0x1c	pci_csr	PCI Controller Control and Status register	0x0000000x	351
0x20	0x20	pci_isr	PCI Controller Interrupt Status register	0x00000000	352
0x24	0x24	pci_inten	PCI Controller Interrupt Enable register	0x00000000	353

Table 120. CSR Address Map (Sheet 2 of 2)

AHB Offset	PCI Offset	Register Name	Description	Reset Value	Page
0x28	0x28	pci_dmactrl	DMA control register	0x00000000	353
0x2c	0x2c	pci_ahbmembase	AHB Memory Base Address Register	0xc0000000	354
0x30	0x30	pci_ahbiobase	AHB I/O Base Address Register	0x00000000	355
0x34	0x34	pci_ahbmembase	PCI Memory Base Address Register	0x00000000	354
0x38	0x38	pci_ahbdoorbell	AHB Doorbell Register	0x00000000	356
0x3c	0x3c	pci_pcidoorbell	PCI Doorbell Register	0x00000000	356
0x40	0x40	pci_atpdma0_ahbaddr	AHB-to-PCI DMA AHB Address Register 0	0x00000000	357
0x44	0x44	pci_atpdma0_pciaddr	AHB-to-PCI DMA PCI Address Register 0	0x00000000	357
0x48	0x48	pci_atpdma0_length	AHB-to-PCI DMA Length Register 0	0x00000000	358
0x4c	0x4c	pci_atpdma1_ahbaddr	AHB-to-PCI DMA AHB Address Register 1	0x00000000	358
0x50	0x50	pci_atpdma1_pciaddr	AHB-to-PCI DMA PCI Address Register 1	0x00000000	359
0x54	0x54	pci_atpdma1_length	AHB-to-PCI DMA Length Register 1	0x00000000	359
0x58	0x58	pci_ptadma0_ahbaddr	PCI-to-AHB DMA AHB Address Register 0	0x00000000	360
0x5c	0x5c	pci_ptadma0_ahbaddr	PCI-to-AHB DMA PCI Address Register 0	0x00000000	360
0x60	0x60	pci_ptadma0_length	PCI-to-AHB DMA Length Register 0	0x00000000	360
0x64	0x64	pci_ptadma1_ahbaddr	PCI-to-AHB DMA AHB Address Register 1	0x00000000	361
0x68	0x68	pci_ptadma1_pciaddr	PCI-to-AHB DMA PCI Address Register 1	0x00000000	361
0x6c	0x6c	pci_ptadma1_length	PCI-to-AHB DMA Length Register 1	0x00000000	362

9.4.3.1 PCI Controller Non-Prefetch Address Register

The tables below describe the PCI Controller Non-Prefetch Address Register:

Register Name:	pci_np_ad				
Block Base Address:	0xC00000	Offset Address	0x00	Reset Value	0x00000000
Register Description:	PCI non-prefetch access address register. Provides address for CSR-initiated non-prefetch PCI accesses.			Access:	(See below.)

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
np_address																															

Register	pci_np_ad				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_address	Address of the non-prefetch PCI cycle. The format of the address depends on the PCI command type used.	0x00000000	none	RW

9.4.3.2 PCI Controller Non-Prefetch Command/Byte Enables Register

The tables below describe the PCI Controller Non-Prefetch Command/Byte Enables Register:

Register Name:	pci_np_cbe				
Block Base Address:	0xC00000	Offset Address	0x04	Reset Value	0x00000000
Register Description:	PCI non-prefetch access command/byte enables register. Provides PCI command and data byte enables for CSR-initiated non-prefetch PCI accesses.			Access:	(See below.)

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
(Reserved)																								NP_BE				NP_CMD			

Register	pci_np_cbe				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	reserved	reserved – read as 0	0x000000	none	RW
7:4	NP_BE	Byte enables driven onto the PCI_CBE_N[3:0] lines of the PCI bus during the data phase of the non-prefetch PCI access.	0x0	none	RW
3:0	NP_CMD	PCI command driven onto the PCI_CBE_N[3:0] lines of the PCI bus during the address phase of the non-prefetch PCI access.	0x0	none	RW

9.4.3.3 PCI Controller Non-Prefetch Write Data Register

The tables below describe the PCI Controller Non-Prefetch Write Data Register:

Register Name:	pci_np_wdata				
Block Base Address:	0xC00000	Offset Address	0x08	Reset Value	0x00000000
Register Description:	PCI non-prefetch access write data register. Provides write data for CSR-initiated non-prefetch PCI write access.			Access:	(See below.)

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
np_wdata																															

Register	pci_np_wdata				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_wdata	Write data for the non-prefetch PCI write cycle.	0x00000000	none	RW

9.4.3.4 PCI Controller Non-Prefetch Read Data Register

The tables below describe the PCI Controller Non-Prefetch Read Data Register:

Register Name:	pci_np_rdata				
Block Base Address:	0xC00000	Offset Address	0x0c	Reset Value	0x00000000
Register Description:	PCI non-prefetch access read data register. Holds read data from the CSR-initiated non-prefetch PCI read access.			Access:	(See below.)

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
np_rdata																															

Register	pci_np_rdata				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:0	np_rdata	Read data from the non-prefetch PCI read cycle.	0x00000000	none	RO

9.4.3.5 PCI Controller Configuration Port Address/Command/Byte Enables Register

The tables below describe the PCI Controller Configuration Port Address/Command/ Byte Enables Register:

Register Name:	pci_crp_ad_cbe				
Block Base Address:	0xC00000	Offset Address	0x10	Reset Value	0x00000000
Register Description:	PCI configuration port address/command/byte enables register. Provides address, command, and data byte enables for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. A write to this register that sets the CRP_CMD[16] bit to a 0 (read) initiates a read of the PCI Controller PCI configuration register addressed by CRP_AD[7:2]. The resultant read data is written to the pci_crp_rdata register.			Access:	(See below.)

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
(Reserved)								CRP_BE				CRP_CMD				(Reserved)					CRP_AD										

Register	pci_crp_ad_cbe (Sheet 1 of 2)				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	reserved	reserved – read as 0	0x00	none	RO
23:20	CRP_BE	Active-low byte enables for a PCI configuration port write access. This field corresponds to byte lanes in the pci_crp_wdata register and addressed PCI configuration register as follows: CRP_BE[0] –> bits 7:0 CRP_BE[1] –> bits 15:8 CRP_BE[2] –> bits 23:16 CRP_BE[3] –> bits 31:24	0x0	none	RW

Register	pci_crp_ad_cbe (Sheet 2 of 2)				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
19:16	CRP_CMD	Command for the PCI configuration port access. xxx0 = read, xxx1 = write	0x0	none	RW
15:11	reserved	reserved – read as 0	00000	none	RO
10:0	CRP_AD	Byte address for the PCI configuration port access. PCI configuration registers are word-addressed so bits 1 and 0 should always be 0. Bits 10:8 specify the function number and must always be 000.	0x000	None	RW

9.4.3.6 PCI Controller Configuration Port Write Data Register

The tables below describe the PCI Controller Configuration Port Write Data Register:

Register Name:	pci_crp_wdata				
Block Base Address:	0xC00000	Offset Address	0x14	Reset Value	0x00000000
Register Description:	PCI configuration port write data register. Provides write data for CSR-initiated accesses of the PCI Controller PCI configuration registers in the PCI Core. If the pci_crp_ad_cmd_be.CRP_CMD[16] bit is 1 (write), a write to pci_crp_w data initiates a write to the PCI Controller PCI configuration register addressed by pci_crp_ad_cmd_be.CRP_AD[7:2]. The pci_crp_ad_cmd_be.CRP_BE field determines the bytes that are affected.			Access:	(See below.)

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
CRP_WDATA																															

Register	pci_crp_wdata				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	CRP_WDATA	Write data for the configuration port write access.	0x0000 0000	none	RW

9.4.3.7 PCI Controller Configuration Port Read Data Register

The tables below describe the PCI Controller Configuration Port Read Data Register:

Register Name:	pci_crp_rdata				
Block Base Address:	0xC00000	Offset Address	0x18	Reset Value	0x00000000
Register Description:	PCI configuration port read data register. Provides read data for CSR-initiated read accesses of the PCI Controller PCI configuration registers in the PCI Core.			Access:	(See below.)

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
CRP_RDATA																															

Register	pci_crp_rdata				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	CRP_RDATA	Read data for the configuration port read access.	0x0000 0000	none	RO

9.4.3.8 PCI Controller Control and Status Register

The tables below describe the PCI Controller Control and Status Register:

Register Name:	pci_csr				
Block Base Address:	0xC00000	Offset Address	0x1c	Reset Value	0x0000000x
Register Description:	Control and status for the PCI Controller.			Access:	(See below.)

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
(Reserved)															PRST	IC	reserved						ASE	(Rsv'd)		DBT	ABE	PBS	ABS	ARBEN	HOST

Register	pci_csr				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:17	reserved	reserved – read as 0	0x0000	RO	RO
16	PRST	PCI Reset. When set to a 1, resets the PCI controller and floats the outputs (even in the middle of a PCI transaction). Setting PRST does not park the bus.	0x0	RO	RW
15	IC	Initialization Complete. When at a logic 0 state, forces the PCI Controller Target Interface to retry PCI cycles. When set to a 1, PCI cycles are accepted.	0	RO	RW
14:9	reserved	reserved – read as 0	0x00	RO	RO
8	ASE	Assert System Error. When set to a 1, the PCI_SERR_N output is asserted for 1 PCI clock cycle if the pci_srcr.SER bit is set.	0	RO	RW
7:6	reserved	reserved – read as 0	00	RO	RO
5	DBT	Doorbell Test mode enable. When set to a 1, the doorbell registers pci_ahbdoorbell, pci_pcidoorbell become normal read/write registers from the AHB bus.	0	RO	RW
4	ABE	AHB big-endian addressing. When 0, little-endian addressing is employed on both AHB master and slave interfaces. When 1, big-endian addressing is implemented.	0	RO	RW
3	PBS	PCI byte swap. Controls byte lane data routing between PCI and AHB busses during PCI Target accesses of the AHB bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
2	ABS	AHB byte swap. Controls byte lane data routing between PCI and AHB busses during AHB Slave accesses of the PCI bus. When 1, byte lane swapping is performed. When 0, no swapping is done.	0	RO	RW
1	ARBEN	Arbiter enable status. Indicates the state of the exp_pciarb input at the deassertion of reset_n. ***Note:***	0 or 1	RO	RO
0	HOST	Host status.Indicates the state of the exp_host input at the deassertion of reset_n. ***Note:***	0 or 1	RO	RO

9.4.3.9 PCI Controller Interrupt Status Register

The tables below describe the PCI Controller Interrupt Status Register:

Register Name:	pci_isr				
Block Base Address:	0xC00000	Offset Address	0x20	Reset Value	0x00000000
Register Description:	Indicates the PCI controller interrupt source(s).			Access:	(See below.)

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
(Reserved)																								PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE

Register	pci_isr				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:8	reserved	reserved – read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt. Asserted high when any one of the bits in the pci_pcidoorbell register is set.	0	RO	RO
6	ADB	AHB Doorbell interrupt. Asserted high when any one of the bits in the pci_ahbdoorbell register is set.	0	RO	RO
5	PADC	PCI-to-AHB DMA Complete. Asserted high when a PCI-to-AHB DMA transfer is complete.	0	RO	RO
4	APDC	AHB-to-PCI DMA Complete. Asserted high when a AHB-to-PCI DMA transfer is complete.	0	RO	RO
3	AHBE	AHB Error indication. Set to a 1 when the AHB Master Interface receives an ERROR response.	0	RO	RW1C
2	PPE	PCI Parity Error. Set to a 1 when a parity error occurs on the PCI bus: Parity error detected during Master Interface read transaction. PCI_PERR_N asserted by an external target during a Master write transaction. Parity error detected during a Target write transaction.	0	RO	RW1C
1	PFE	PCI Fatal Error. Set to a 1 when one of the following errors occurs on the PCI bus: • Master abort (target did not respond) • Target abort • TRDY timeout (external target asserts PCI_DEVSEL_N but never asserts PCI_TRDY_N) • Retry timeout (external target issued more retries than specified by the RetryTO field in the pci_rtotto register)	0	RO	RW1C
0	PSE	PCI System Error. Set to a 1 when the PCI Controller detects that the PCI PCI_SERR_N signal has been asserted.	0	RO	RW1C

9.4.3.10 PCI Controller Interrupt Enable Register

The tables below describe the PCI Controller Interrupt Enable Register:

Register Name:	pci_inten				
Block Base Address:	0xC00000	Offset Address	0x24	Reset Value	0x00000000
Register Description:	Interrupt enables for the interrupt status bits in the pci_isr register. Set to a 1 to enable the particular interrupt.			Access:	(See below.)

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
(Reserved)																								PDB	ADB	PADC	APDC	AHBE	PPE	PFE	PSE

Register	pci_inten				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:8	reserved	reserved – read as 0	0x000000	RO	RO
7	PDB	PCI Doorbell interrupt enable.	0	RO	RW
6	ADB	AHB Doorbell interrupt enable.	0	RO	RW
5	PADC	PCI-to-AHB DMA Complete interrupt enable.	0	RO	RW
4	APDC	AHB-to-PCI DMA Complete interrupt enable.	0	RO	RW
3	AHBE	AHB Error indication interrupt enable.	0	RO	RW
2	PPE	PCI Parity Error interrupt enable.	0	RO	RW
1	PFE	PCI Fatal Error interrupt enable.	0	RO	RW
0	PSE	PCI System Error interrupt enable.	0	RO	RW

9.4.3.11 DMA Control Register

The tables below describe the DMA Control Register:

Register Name:	pci_dmactrl				
Block Base Address:	0xC00000	Offset Address	0x28	Reset Value	0x00000000
Register Description:	Control and status for the DMA Controller channels.			Access:	(See below.)

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
(Reserved)																PADE1	PADC1	PADE0	PADC0	(Reserved)			PADCEN	APDE1		APDC1	APDE0		APDC0	(Rsv'd)	

Register		pci_dmactrl			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:16	reserved	reserved – read as 0	0x0000	RO	RO
15	PADE1	PCI-to-AHB DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC1 bit.	0	RO	RO
14	PADC1	PCI-to-AHB DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_ptadma1_xxx registers is complete or terminated due to an error.	0	RO	RW1C
13	PADE0	PCI-to-AHB DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the PADC0 bit.	0	RO	RO
12	PADC0	PCI-to-AHB DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_ptadma0_xxx registers is complete or terminated due to an error.	0	RO	RW1C
11:9	reserved	reserved – read as 0	000	RO	RO
8	PADCEN	PCI-to-AHB DMA Complete interrupt enable.	0	RO	RW
7	APDE1	AHB-to-PCI DMA error for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC1 bit.	0	RO	RO
6	APDC1	AHB-to-PCI DMA complete for buffer 1. Set to a 1 when the DMA transfer specified by the pci_atpdma1_xxx registers is complete or terminated due to an error.	0	RO	RW1C
5	APDE0	AHB-to-PCI DMA error for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers terminates due to an error. Read-only, cleared when a 1 is written to the APDC0 bit.	0	RO	RO
4	APDC0	AHB-to-PCI DMA complete for buffer 0. Set to a 1 when the DMA transfer specified by the pci_atpdma0_xxx registers is complete or terminated due to an error.	0	RO	RW1C
3:1	reserved	reserved – read as 0	000	RO	RO
0	APDCEN	AHB-to-PCI DMA Complete interrupt enable.	0	RO	RW

9.4.3.12 AHB Memory Base Address Register

The tables below describe the AHB Memory Base Address Register:

Register Name:								pci_ahbmembase																							
Block Base Address:								0xC00000					Offset Address					0x2c					Reset Value					0xc0000000			
Register Description:								Provides upper 8 AHB address bits for PCI accesses of AHB bus. Lower 24 bits of AHB address provided directly from PCI bus. Four AHBbase fields correspond to accesses from the PCI bus that target addresses in PCI configuration base address registers pci_bar0/1/2/3.													Access:								(See below.)			
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
AHBbase0								AHBbase1								AHBbase2								AHBbase3							

Register	pci_ahbmembase			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	AHBbase0	Upper 8 AHB address bits for PCI accesses that target pci_bar0. By default this register maps to an upper region of the AHB memory map.	0xc0	RO	RW
23:16	AHBbase1	Upper 8 AHB address bits for PCI accesses that target pci_bar1.	0x00	RO	RW
15:8	AHBbase2	Upper 8 AHB address bits for PCI accesses that target pci_bar2.	0x00	RO	RW
7:0	AHBbase3	Upper 8 AHB address bits for PCI accesses that target pci_bar3.	0x00	RO	RW

9.4.3.13 AHB I/O Base Address Register

The tables below describe the AHB I/O Base Address Register:

Register Name:	pci_ahbiobase				
Block Base Address:	0xC00000	Offset Address	0x30	Reset Value	0x00000000
Register Description:	Provides upper 24 AHB address bits for PCI accesses of AHB I/O space. Lower 8 bits of AHB address provided directly from PCI bus.			Access:	(See below.)

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
(Reserved)								IObase																							

Register	pci_ahbiobase			
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:24	reserved	Reserved – read as 0	0x00	RO	RO
23:0	IObase	Upper 24 AHB address bits for PCI accesses that target pci_bar5.	0x000000	RO	RW

9.4.3.14 PCI Memory Base Address Register

The tables below describe the PCI Memory Base Address Register:

Register Name:	pci_pcimembase				
Block Base Address:	0xC00000	Offset Address	0x34	Reset Value	0x00000000
Register Description:	Provides upper 8 PCI address bits for AHB accesses of PCI memory space. Lower 24 bits of PCI address provided directly from the AHB bus. Four Membase fields correspond to accesses from the AHB bus that target specific AHB address ranges.			Access:	(See below.)

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
Membase0								Membase1								Membase2									Membase3						

Register	pci_pcimembase				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:24	PCIbase0	Upper 8 PCI address bits for AHB accesses that target the first 16MB PCI memory partition.	0x00	RO	RW
23:16	PCIbase1	Upper 8 PCI address bits for AHB accesses that target the second 16MB PCI memory partition.	0x00	RO	RW
15:8	PCIbase2	Upper 8 PCI address bits for AHB accesses that target the third 16MB PCI memory partition.	0x00	RO	RW
7:0	PCIbase3	Upper 8 PCI address bits for AHB accesses that target the fourth 16MB PCI memory partition.	0x00	RO	RW

9.4.3.15 AHB Doorbell Register

The tables below describe the AHB Doorbell Register:

Register Name:	pci_ahbdoorbell				
Block Base Address:	**0xC00000**	**Offset Address**	**0x38**	**Reset Value**	**0x00000000**
Register Description:	This register is write-1-to-set from PCI and write-1-to-clear from AHB. The PCI device writes a 1 to a bit or pattern of bits to generate the interrupt. The AHB agent reads the register and writes 1(s) to clear the bit(s) and deassert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.			**Access:**	(See below.)

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
ADB																															

Register	pci_ahbdoorbell				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	ADB	PCI generated doorbell interrupt to an AHB agent. Normally read/write-1-to-set from PCI and read/write-1-to-clear from AHB. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x00000000	RW1S	RW1C (RW if pci_csr.DBT=1)

9.4.3.16 PCI Doorbell Register

The tables below describe the PCI Doorbell Register:

Register Name:	pci_pcidoorbell				
Block Base Address:	**0xC00000**	**Offset Address**	**0x3c**	**Reset Value**	**0x00000000**
Register Description:	The Intel XScale processor writes this register to generate an interrupt to an external PCI device on PCI_INTA_N. Any bit set to a 1 generates the PCI interrupt if the PCI doorbell interrupt is enabled (pci_inten.PDBEN = 1). This register is write-1-to-set from AHB and write-1-to-clear from PCI. The Intel XScale processor writes a 1 to a bit or pattern of bits to generate the interrupt. The external PCI device reads the register and writes 1(s) to clear the bit(s) and deassert the interrupt. If the DBT (Doorbell Test) bit is set in the pci_csr register, all bits become read/write from the AHB bus.			**Access:**	(See below.)

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
PDB																															

Register	pci_pcidoorbell				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:0	PDB	AHB generated doorbell interrupt to PCI. Normally read/write-1-to-set from AHB and read/write-1-to-clear from PCI. Read/write from the AHB side if Doorbell Test mode is enabled by setting pci_csr.DBT to a 1.	0x0000 0000	RW1C	RW1S (RW if pci_csr. DBT=1)

9.4.3.17 AHB-to-PCI DMA AHB Address Register 0

The tables below describe the AHB-to-PCI DMA AHB Address Register 0:

Register Name:	pci_atpdma0_ahbaddr				
Block Base Address:	0xC00000	Offset Address	0x40	Reset Value	0x00000000
Register Description:	Source address on the AHB bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_ahbaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_atpdma0_ahbaddr				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	AHB word address	0x0000 0000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

9.4.3.18 AHB-to-PCI DMA PCI Address Register 0

The tables below describe the AHB-to-PCI DMA PCI Address Register 0:

Register Name:	pci_atpdma0_pciaddr				
Block Base Address:	0xC00000	Offset Address	0x44	Reset Value	0x00000000
Register Description:	Destination address on the PCI bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_pciaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_atpdma0_pciaddr				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	PCI word address	0x0000 0000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

9.4.3.19 AHB-to-PCI DMA Length Register 0

The tables below describe the AHB-to-PCI DMA Length Register 0:

Register Name:	pci_atpdma0_length				
Block Base Address:	0xC00000	Offset Address	0x48	Reset Value	0x00000000
Register Description:	Provides word count and control for AHB-to-PCI DMA transfers. Paired with pci_atpdma1_length to allow buffering of DMA transfer requests.			Access:	(See below.)

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
EN	(Rsvd)		DS	(Reserved)												Wordcount															

Register	pci_atpdma0_length				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29	reserved	Reserved. Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

9.4.3.20 AHB-to-PCI DMA AHB Address Register 1

The tables below describe the AHB-to-PCI DMA AHB Address Register 1:

Register Name:	pci_atpdma1_ahbaddr				
Block Base Address:	0xC00000	Offset Address	0x4c	Reset Value	0x00000000
Register Description:	Source address on the AHB bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_ahbaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_atpdma1_ahbaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	AHB word address	0x0000 0000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

9.4.3.21 AHB-to-PCI DMA PCI Address Register 1

The tables below describe the AHB-to-PCI DMA PCI Address Register 1:

Register Name:	pci_atpdma1_pciaddr				
Block Base Address:	0xC00000	Offset Address	0x50	Reset Value	0x00000000
Register Description:	Destination address on the PCI bus for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_pciaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_atpdma1_pciaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	PCI word address	0x0000 0000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

9.4.3.22 AHB-to-PCI DMA Length Register 1

The tables below describe the AHB-to-PCI DMA Length Register 1:

Register Name:	pci_atpdma1_length				
Block Base Address:	0xC00000	Offset Address	0x54	Reset Value	0x00000000
Register Description:	Provides word count and control for AHB-to-PCI DMA transfers. Paired with pci_atpdma0_length to allow buffering of DMA transfer requests.			Access:	(See below.)

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
EN	(Rsvd)		DS	(Reserved)												wordcount															

Register	pci_atpdma1_length				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29	reserved	Reserved. Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the AHB bus is byte swapped before being sent to the PCI bus. When 0, no swapping is done.	0	RO	RW
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

9.4.3.23 PCI-to-AHB DMA AHB Address Register 0

The tables below describe the PCI-to-AHB DMA AHB Address Register 0:

Register Name:	pci_ptadma0_ahbaddr				
Block Base Address:	0xC00000	Offset Address	0x58	Reset Value	0x00000000
Register Description:	Destination address on the AHB bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_ahbaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_ptadma0_ahbaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	AHB word address	0x00000000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

9.4.3.24 PCI-to-AHB DMA PCI Address Register 0

The tables below describe the PCI-to-AHB DMA PCI Address Register 0:

Register Name:	pci_ptadma0_pciaddr				
Block Base Address:	0xC00000	Offset Address	0x5c	Reset Value	0x00000000
Register Description:	Source address on the PCI bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_pciaddr to allow buffering of DMA transfer requests.			Access:	(See below.)

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
address																														0	0

Register	pci_ptadma0_pciaddr				
Bits	Name	Description	Reset Value	PCI Access	AHB Access
31:2	address	PCI word address	0x0000 0000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

9.4.3.25 PCI-to-AHB DMA Length Register 0

The tables below describe the PCI-to-AHB DMA Length Register 0:

Register Name:	pci_ptadma0_length				
Block Base Address:	0xC00000	Offset Address	0x60	Reset Value	0x00000000
Register Description:	Provides word count and control for PCI-to-AHB DMA transfers. Paired with pci_ptadma1_length to allow buffering of DMA transfer requests.			Access:	(See below.)

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
EN	(Rsvd)		DS	(Reserved)												wordcount															

Register		pci_ptadma0_length			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29	reserved	Reserved. Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.	0	RO	RW
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

9.4.3.26 PCI-to-AHB DMA AHB Address Register 1

The tables below describe the PCI-to-AHB DMA AHB Address Register 1:

Register Name:																														
Block Base Address:						**0xC00000**						**Offset Address**						**0x64**						**Reset Value**						**0x00000000**
Register Description:						Destination address on the AHB bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_ahbaddr to allow buffering of DMA transfer requests.																		**Access:**						(See below.)		
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**
address																														0	0

Register		pci_ptadma1_ahbaddr			
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	AHB word address	0x0000 0000	RO	RW
1:0		Lower AHB address bits hard-wired to zero.	00	RO	RO

9.4.3.27 PCI-to-AHB DMA PCI Address Register 1

The tables below describe the PCI-to-AHB DMA PCI Address Register 1:

Register Name:																														
Block Base Address:						**0xC00000**						**Offset Address**						**0x68**						**Reset Value**						**0x00000000**
Register Description:						Source address on the PCI bus for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_pciaddr to allow buffering of DMA transfer requests.																		**Access:**						(See below.)		
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**
address																														0	0

Register	pci_ptadma1_pciaddr				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31:2	address	PCI word address	0x0000 0000	RO	RW
1:0		Lower PCI address bits hard-wired to zero.	00	RO	RO

9.4.3.28 PCI-to-AHB DMA Length Register 1

The tables below describe the PCI-to-AHB DMA Length Register 1:

Register Name:	pci_ptadma1_length				
Block Base Address:	0xC00000	Offset Address	0x6c	Reset Value	0x00000000
Register Description:	Provides word count and control for PCI-to-AHB DMA transfers. Paired with pci_ptadma0_length to allow buffering of DMA transfer requests.			Access:	(See below.)

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
EN	(Rsvd)		BE	(Reserved)												wordcount															

Register	pci_ptadma1_length				
Bits	**Name**	**Description**	**Reset Value**	**PCI Access**	**AHB Access**
31	EN	Channel enable. When set to a 1, executes a DMA transfer if wordcount is nonzero. When 0, the channel is disabled. Hardware clears this bit when the DMA transfer is complete.	0	RO	RW
30:29	reserved	Reserved. Read as 0.	00	RO	RO
28	DS	Data Swap indicator. When set to a 1, data from the PCI bus is byte swapped before being sent to the AHB bus. When 0, no swapping is done.	0	RO	RW
27:16	reserved	Reserved. Read as 0.	0x000	RO	RO
15:0	wordcount	Number of words to transfer.	0x0000	RO	RW

9.5 Error / Abnormal Conditions

This section describes the behavior of the PCI Controller under various error conditions that can occur on the PCI and AHB busses.

9.5.1 Error Handling as a PCI Target

This section describes behavior of the PCI Controller during error handling as a PCI Target:

9.5.1.0.1 A PCI Target Read Received an Error Response During the AHB Read Operation and the PCI Transfer is Not Complete

This case occurs when the target interface is delivering data to the PCI Initiator and an abort indication is received from the AHB Master Interface due to an ERROR response received during the AHB read transfer.

1. The AHB Master terminates the read transfer and stops delivering data to the PCI Core.
2. The PCI Target delivers all remaining data in the Target Read FIFO to the PCI Initiator, then terminates the cycle with a Target Abort response. With the ECC/ parity implementation of the DDR and Expansion bus controller on the IXP43X network processors, if an ECC or parity error exists on any word within an 8-word line (even if the PCI Initiator does not ask for this word), the PCI responds with Target Abort on the first word of the read transfer.
3. The PCI Configuration Register pci_srcr.STA bit is set to indicate that this device signalled a Target Abort.
4. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.

9.5.1.0.2 A PCI Target Read Received an Error Response During the AHB Read Operation After the PCI Transfer is Complete

This scenario is possible since the AHB Master Interface does not know ahead of time how much data the PCI Initiator needs and thus keeps generating AHB read cycles until the target interface detects the end of the PCI read cycle. Due to latencies involved in notifying the AHB Master Interface, additional AHB reads are generated after the read cycle completes on PCI. If an error response is received during one of these read-ahead AHB cycles:

1. The PCI cycle completes normally with all requested data delivered to the Initiator and no Target Abort response generated.
2. Any additional data left in the Target Read FIFO is discarded.
3. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.

9.5.1.0.3 A PCI Target Write Received an Error Response During the AHB Write Operation

1. The AHB Master Interface terminates the write transfer.
2. Any additional data in the Target Write FIFO is read out and discarded until the last-data marker is encountered indicating that the PCI cycle is complete.
3. Since writes are posted by the PCI Target Interface, the cycle could complete on PCI well before the AHB write error happens. For this reason, a Target Abort response is not generated on the PCI bus. All writes complete normally on PCI regardless of the status of the transfer on the AHB bus.
4. The pci_isr.AHBE CSR bit is set to a 1 to indicate that an AHB error has occurred.

9.5.1.0.4 The PCI Target Interface Detected an Address Phase Parity Error During a Target Write

1. If the transfer is longer than 1 data phase, the PCI Target Interface terminates the cycle with a Target Abort and sets the PCI configuration Register bit pci_srcr.STA
2. If the transfer consists of just one data phase, the PCI Target Interface accepts the transfer without signalling Target Abort but the data is discarded and no AHB write operation is performed.
3. The PCI Configuration Register bit pci_srcr.DPE is set indicating that a parity error has been detected.
4. If the PCI Configuration Register bit pci_srcr.SER is set, the PCI_SERR_N signal is asserted on the bus and the pci_srcr.SSE bit is set indicating that a system error has been signalled by this device.

9.5.1.0.5 The PCI Target Interface Detected a Data Phase Parity Error During a Target Write

1. The PCI Target Interface tags the errored data but otherwise processes the transfer normally, passing the data to the AHB Master.
2. The PCI Configuration Register bit pci_srcr.DPE is set to a 1. If the pci_srcr.PER bit is set, the PCI_PERR_N signal is asserted on the PCI bus.
3. The AHB Master Interface ignores the error and processes the write transfer normally.

9.5.1.0.6 The Initiator of a PCI Target Read Asserts PCI_PERR_N During the Transfer

1. The error indication is ignored. Handling of the error is left to the initiator of the transfer.

9.5.2 Error Handling as a PCI Initiator During PCI Direct Access from the AHB Bus

This section describes error handling procedures when the PCI Initiator Interface encounters a fatal error condition during a PCI transfer request received from the AHB Target Interface.

9.5.2.0.1 An AHB Target Read Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Read Operation

Note: A PCI_TRDY_N timeout occurs if the intended PCI target of the transfer asserts PCI_DEVSEL_N but fails to assert PCI_TRDY_N or abort the transfer with a Target Abort within the number of PCI clocks specified in pci_rtotto.TRDYTO. A RETRY Timeout occurs if the intended target exceeds the number of RETRY responses indicated in pci_rtotto.RetryTO.

1. If the AHB read operation is not complete, the AHB Target Interface asserts an ERROR response to terminate the AHB operation. If the AHB read is complete because the PCI error occurred during a read-ahead transfer, no action is taken. In either case, the AHB Target Interface blocks any further PCI requests until the error condition has cleared.
2. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
3. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

9.5.2.0.2 An AHB Target Write Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Write Operation

1. The Initiator Request FIFO is flushed and all queued requests are lost. Any data in the Initiator Transmit FIFO is flushed as well.
2. If the write is still in progress on the AHB bus, an ERROR response is not issued. The remaining data in the burst is discarded and the AHB transfer completes normally.
3. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
4. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

Caution: Note that in this error condition, the entire contents of the Initiator Request FIFO is discarded. Any queued write operations are lost since they have already been posted by the AHB Target Interface. A queued read, though flushed from the queue, completes since the AHB Master that originated the read must try the transfer again after getting

a RETRY response. After the error condition has cleared, the AHB Target Interface requests the read again when the master retries the operation.

9.5.2.0.3 The PCI Initiator Interface Detected a Parity Error During a PCI Read Operation

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.
3. The PCI Configuration Register bit pci_srcr.DPE is set.

9.5.2.0.4 The Target of a PCI Write Operation Asserted PCI_PERR_N During the Transfer

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.

9.5.3 Error Handling as a PCI Initiator During Non-Prefetch Operations

This section describes error handling procedures when the PCI Initiator Interface encounters a fatal error condition during a PCI transfer request initiated by a non-prefetch operation.

9.5.3.0.1 The Non-Prefetch Operation Encountered a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the PCI Operation

1. The non-prefetch operation terminates and if the operation is a read, no data is written to the pci_np_rdata register. If the operation is a write, the Initiator Request FIFO and Initiator Transmit FIFO are flushed but since requests cannot be queued behind a non-prefetch operation, no requests can occur.
2. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.
3. The PCI Configuration Register bits pci_srcr.RMA or pci_srcr.RTA are set if a Master Abort or Target Abort occurred, respectively.

9.5.3.0.2 The PCI Initiator Interface Detected a Parity Error During a PCI Read Operation

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.
3. The PCI Configuration Register bit pci_srcr.DPE is set.

9.5.3.0.3 The Target of a PCI Write Operation Asserted PCI_PERR_N During the Transfer

1. The PCI Initiator Interface ignores the error and continues with the transfer.
2. The pci_isr.PPE bit is set to indicate a PCI parity error has occurred.

9.5.4 Error Handling During PCI-to-AHB DMA Channel Operations

This section describes error handling during PCI-to-AHB DMA channel operations:

9.5.4.0.1 A PCI Read Received a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.PADCx and pci_dmactrl.PADEx (x = 0 or 1 depending on the DMA buffer set that encountered

the error) bits are set indicating that the DMA transfer is complete and an error was detected.

2. The pci_dmactrl.PADCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and commences normally.
4. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.

9.5.4.0.2 An AHB Write Received an Error Response During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.PADCx and pci_dmactrl.PADEx (x = 0 or 1 depending on the DMA buffer set that encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.PADCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and commences normally.
4. The pci_isr.AHBE bit is set to indicate a fatal PCI error has occurred.

9.5.5 Error Handling During AHB-to-PCI DMA Channel Operations

This section describes error handling during AHB-to-PCI DMA channel operations:

9.5.5.0.1 A PCI Write Received a Master Abort, Target Abort, PCI_TRDY_N Timeout, or RETRY Timeout During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.APDCx and pci_dmactrl.APDEx (x = 0 or 1 depending on the DMA buffer set that encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.APDCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and commences normally.
4. The pci_isr.PFE bit is set to indicate a fatal PCI error has occurred.

9.5.5.0.2 An AHB Read Received an Error Response During the DMA Transfer

1. The current DMA transfer is aborted and the pci_dmactrl.APDCx and pci_dmactrl.APDEx (x = 0 or 1 depending on the DMA buffer set that encountered the error) bits are set indicating that the DMA transfer is complete and an error was detected.
2. The pci_dmactrl.APDCEN bit is set.
3. If the other DMA buffer set for this DMA channel is enabled, that transfer is not affected and commences normally.
4. The pci_isr.AHBE bit is set to indicate a fatal PCI error has occurred.

§ §

10.0 USB 2.0 Host Controller

10.1 Overview

This unit describes the USB Host Controller functionality for the Intel® IXP43X Product Line of Network Processors. The functions being performed is defined by the USB 2.0 specification, further information can be found at http://www.usb.org/.

Not all features defined by the USB 2.0 specification are supported by the IXP43X network processors. The supported features are:

- EHCI register interface
- Host function
- High-speed interface
- Low-speed interface
- Full-speed interface
- UTMI+ Level 2 Compliant

The following is a partial list of the USB 2.0 features that are not supported by the IXP43X network processors:

- Device function
- OTG function

The USB Host core is a standards-based Serial Interface Engine for implementing a USB interface in compliance with the USB 2.0 specification. The register and data structure interfaces to the core are based on the Enhanced Host Controller Interface (EHCI) specification from Intel. By leveraging industry standards throughout the core, the compatibility and quality of the final product are insured.

The USB Host core is designed to make efficient use of the system resources in this design. The 32-bit system bus interface contains a chaining DMA (Direct Memory Access) engine. This DMA engine reduces the interrupt load on the application processor and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface.

10.2 USB

The Universal Serial Bus (USB) is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached as the host and other peripherals are in operation.

USB software provides a uniform view of the system for all application software, hiding implementation details making application software more portable. It manages the dynamic attach and detach of peripherals.

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller. The Host Controller can be implemented in a combination of hardware, firmware, or software.

There can be multiple USB devices in any system such as hubs, joysticks, speakers, printers, and so on. USB devices present a standard USB interface in terms of comprehension, response, and standard capability.

The host initiates transactions to specific peripherals, as the device responds to control transactions. The device sends and receives data to and from the host using a standard USB data format. The early versions of the USB specification (USB 1.0 and 1.1) operated at 12 Mbps or 1.5 Mbps.

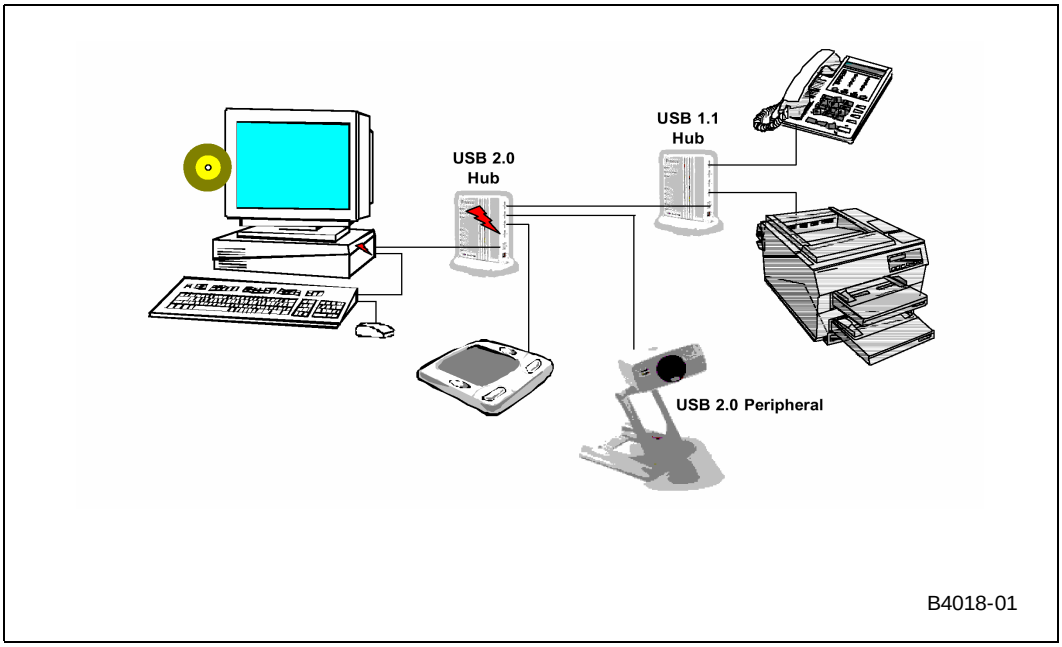
10.3 USB 2.0

The USB 2.0 specification supersedes the earlier versions of the USB specification. USB 2.0 offers a high-speed protocol that offers the user a larger bandwidth and increases data throughput by a factor of 40. All the peripherals used with the previous versions of USB work perfectly with USB 2.0 and also offering a larger choice of higher performance peripherals, such as video-conferencing cameras, next-generation scanners and printers, a fast storage device, and so on.

In addition to the 1.5Mbps and 12Mbps data rates of USB 1.1, the evolution from USB 1.1 to USB 2.0 adds an additional data rate of 480 Mbps. Existing USB 1.1 connectors and full-speed cables support the higher speeds of the USB 2.0 without any changes. USB 2.0 specifies a microframe, that is 1/8th of a 1ms frame allowing small buffers even at the high data rate.

USB 2.0 system configurations look identical to the end user to configurations based on earlier revisions of the USB specification. The end user must distinguish between USB 2.0 hubs and USB 1.1 hubs to optimize the placement of USB 2.0 high-speed devices. A USB 2.0 hub accepts high-speed transactions at a faster rate but also matches the data rate of the peripherals, as the USB 1.1 hub only supports the lower 12Mbps and 1.5Mbps data rates.

Figure 82. Example USB 2.0 System Configuration



The roles of the components of the USB 2.0 system have minor changes from the roles in a USB 1.1 system. But, current peripheral products work with the USB 2.0 system without any changes. Additional performance is not required for many of the human interface devices such as mice, keyboards, and game pads. Both USB 1.1 and USB 2.0 devices operates in a USB 2.0 system and opening up the possibilities of exciting new higher bandwidth peripherals.

For additional information, refer to the USB 2.0 specification.

Note: There is a performance degradation impact on USB2.0 high speed mode for the IXP43X network processors while utilizing WinCE* and potentially other operating systems when Intel XScale® Processor is configured in little-endian data coherent mode of operation

10.4 Feature List

- Includes a UTM and a single-ported physical transceiver (PHY) to support USB High speed interface at 480Mbps, full speed interface at 12Mbps and low speed interface at 1.5Mbps.
- Intel EHCI host controller. The USB host controller registers and data structures are compliant to Intel EHCI specification. Device controller registers and data structures are implemented as extensions to the EHCI programmers interface. Further information can be found on Intel's web site at: http://www.intel.com/technology/usb/ehcispec.htm
- Directly connected USB legacy (USB 1.1) Full and Low speed devices without a companion USB 1.1 host controller or host controller driver software using EHCI standard data structures.
- Configurable dual port RAM buffers isolate memory latency on the system bus from the timing requirements of the USB.
- UTMI+ Level 2 Compliant. The USB host controller is able to connect directly to both USB 2.0 high speed devices, or legacy USB 1.1 full and low speed devices. Additionally, the USB host controller can connect indirectly to high/full/low speed devices via a high speed USB hub, or to full speed devices via a full-speed USB hub. But, the USB host controller cannot connect to a low speed device via a full-speed USB hub. This is because UTMI+ Level 2 does not support FS PREamble PID that is required to signal to all USB devices that the next transaction is in LS protocol. The table below shows the supported connectivities:

Table 121. UTMI+ Level 2 Connectivity

Hub	Device	Connectivity
No Hub / Direct	High speed device	OK
	Full speed device	OK
	Low speed device	OK
High Speed Hub	High speed device	OK
	Full speed device	OK
	Low speed device	OK
Full Speed Hub	Full speed device	OK
	Low speed device	Not allowed

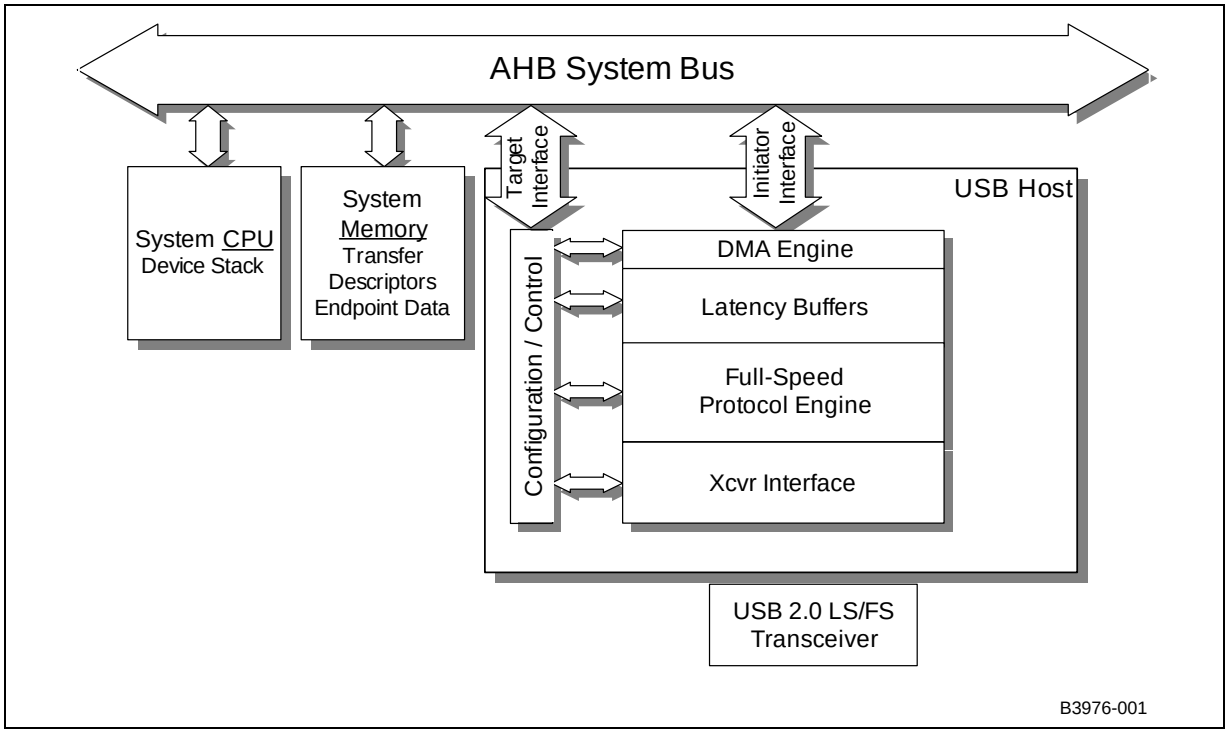
Note: The Intel® IXP43X Product Line of Network Processors does not support high bandwidth interrupt transfer.

10.5 Block Diagram

Figure 83 shows the top-level block diagram for the module. In this case, the system CPU is the Intel XScale processor and the system memory is the DDRII/DDRI memory controller the USB block has its DMA engine pointed to.

Note: In this case, the USB 2.0 compliance is restricted to a legacy full-speed protocol and that the system memory is restricted to Memory Controller Unit accesses only.

Figure 83. Top-Level Block Diagram



10.6 Theory of Operation

10.6.1 Software Model

The Host Stack provides a layered software architecture to control all aspects of a USB bus system. The Host Controller Device (HCD) interface controls the functions of an embedded EHCI host controller. The USB driver layer provides all the USB driver functions to enumerate, manage and schedule a USB bus system, as the upper layers of the stack support standard USB device class interfaces to the device drives running on your embedded system. Detailed information can be obtained by referring to the USB 2.0 specification located at www.usb.org.

10.6.2 Host Data Structure

The host data structures are used to communicate control, status, and data between software and the Host Controller. The Periodic Frame List is an array of pointers for the periodic schedule. A sliding window on the Periodic Frame List is used. The Asynchronous Transfer List is where all the control and bulk transfers are managed. The Host API incorporates and abstracts for the application developer all of the information contained in the host operational model.

Figure 84. Periodic Schedule Organization

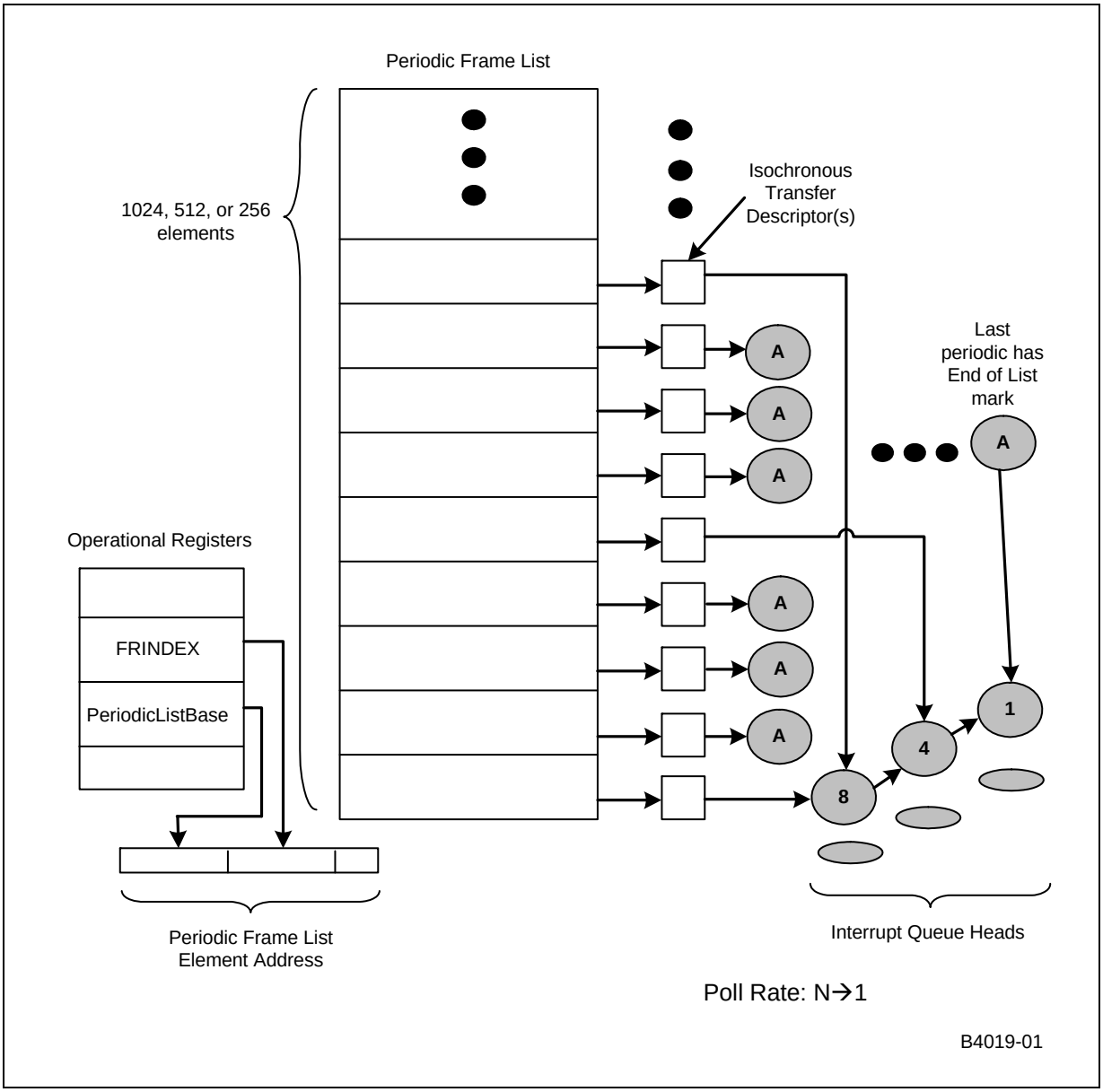
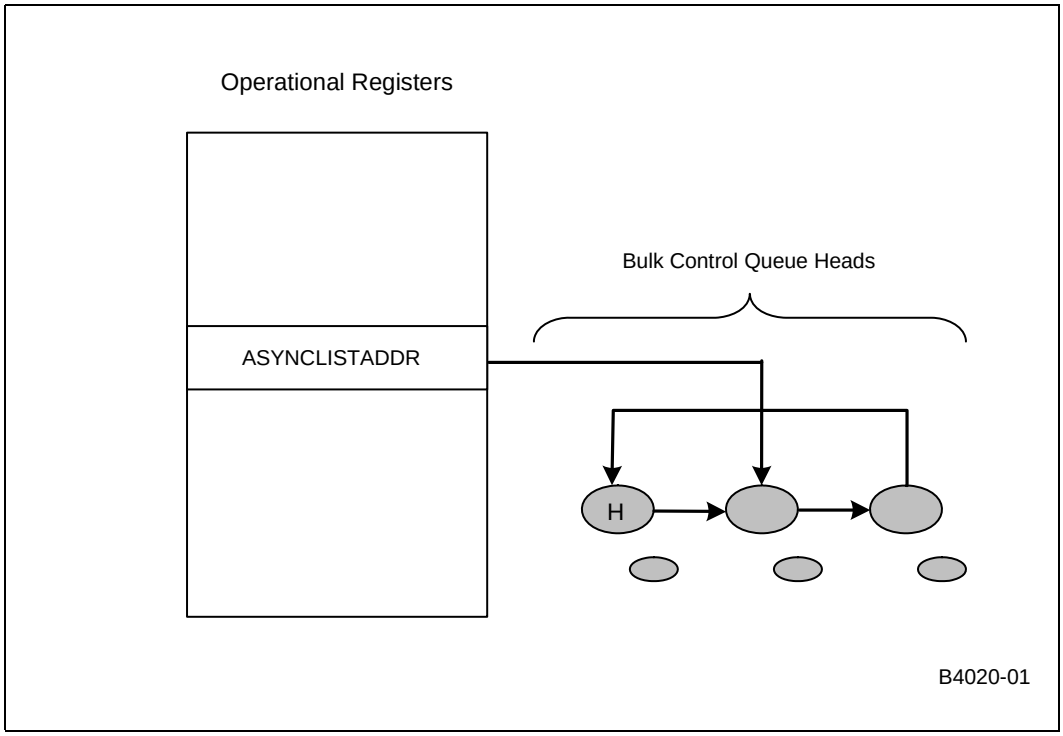


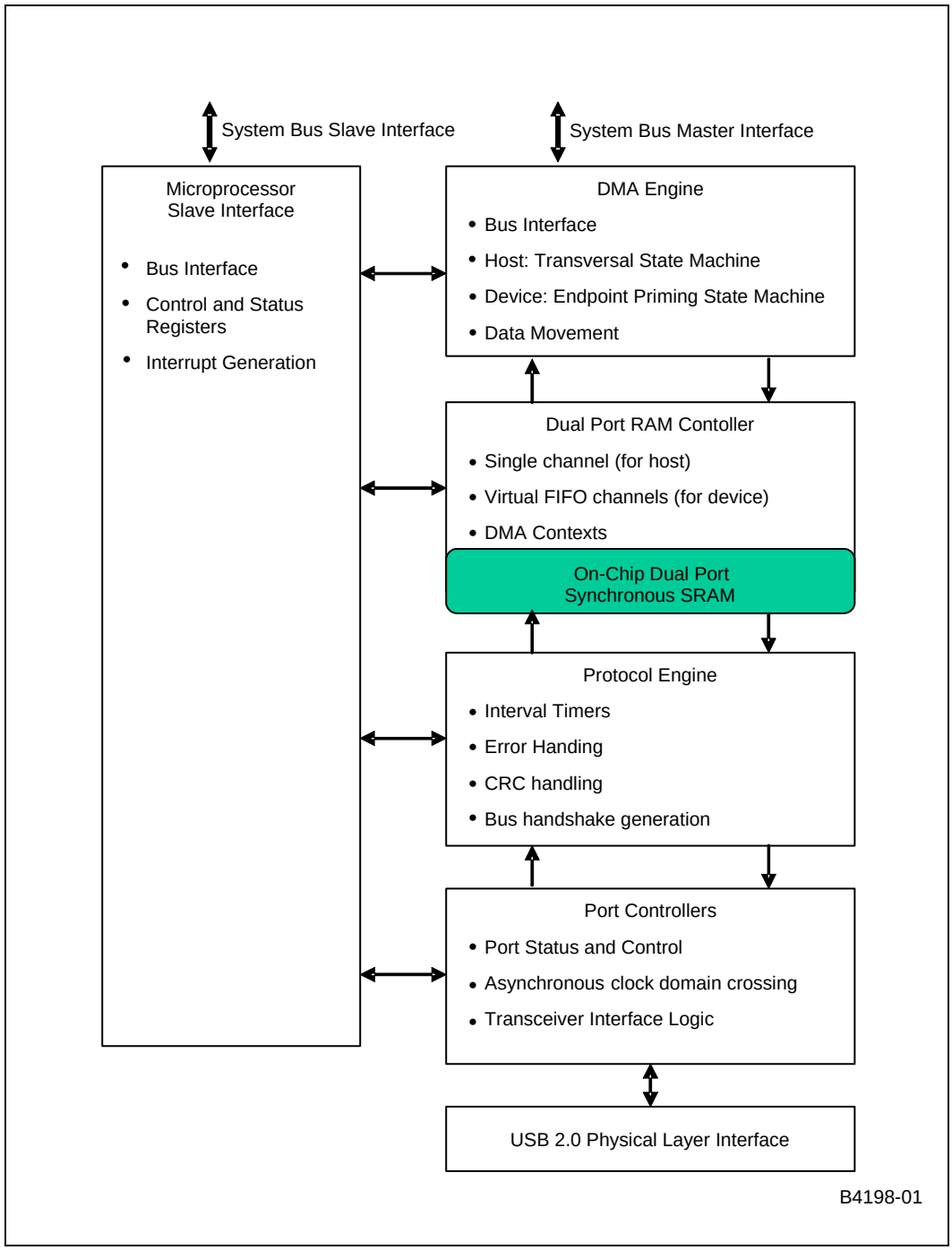
Figure 85. Asynchronous Schedule Organization



10.6.3 Hardware Model

10.6.3.1 Block Diagram

Figure 86. Block Diagram



10.6.3.2 Microprocessor Interface

The microprocessor interface block provides an AMBA target (slave) interface on the AHB. A simple synchronous bus interface signaling is used for the microprocessor bus allowing the core to be easily connected to the system bus.

The Microprocessor interface block contains all the control and status registers that allow a processor to interface to the USB Host core. These registers allow a microprocessor to control the configuration of the core, ascertain the capabilities of the core and control the core in operation for host mode.

Two groups of registers exist in the interface. The USB host controller registers are compatible with the USB host controller registers defined in the Intel EHCI specification.

10.6.3.3 DMA Engine

The DMA Engine Block presents a bus initiator (master) interface to the system memory bus. It is responsible for moving all of the data to be transferred over the USB between the USB Host core and buffers in system memory. Like the microprocessor interface block the DMA block uses the AHB protocol.

The DMA controller must access both control information and packet data from system memory. The control information is contained in link list based queue structures. The DMA controller has state machines that are able to parse all of the data structures defined in this controller specification. In host mode, the data structures are from the EHCI specification and represent queues of transfers to be performed by the host controller, including the split transaction requests that allow an EHCI controller to direct packets to Low and Full speed devices.

10.6.3.4 Dual Port RAM Controller

The Dual Port RAM Controller is used for context information and to build configurable FIFOs between the Protocol Engine block and the DMA controller. These FIFOs decouple the system processor memory bus request from the extremely tight timing required by the USB itself.

10.6.3.5 Protocol Engine

The Protocol Engine parses all the USB tokens and generates the response packets. It is responsible for all error checking, check field generation, formatting all the handshake, ping and data response packets on the bus, and for any signals that must be generated based on a USB based time frame. In host mode, the Protocol engine also generates all of the token packets required by the USB protocol. The Protocol engine contains several sub-functions:

- The token state machines track all of the tokens on the bus and filter the traffic based on the address and endpoint information in the token. In host mode, these state machines also generate the tokens required for data transfer and bus control.
- The CRC5 and CRC16 CRC generator/checker circuits check and generate the CRC check fields for the token and data packets.
- The data and handshake state machines generate any responses required on the USB and move the packet data through the dual port memory FIFOs to the DMA controller block.
- The Interval timers provide timing strobes that identify important bus timing events: the bus timeout interval, the microframe interval, the start of frame interval, and the bus reset, resume, and suspend intervals.
- Reports all transfer status to the DMA engine.

10.6.3.6 Transaction Translator

There is no separate Transaction Translator block in the system. The transaction translator function normally associated with a USB 2.0 high speed hub has been implemented within the DMA and Protocol engine blocks to support connection to full and low speed devices. The internal transaction translator takes split transaction requests addressed to the Hub and issues the corresponding full and low speed transactions on directly connected full or low speed ports.

10.6.3.7 Port Controller

The Port Controller block interfaces to the full-speed transceiver or any UTMI compatible transceiver macrocell core. The primary function of the Port Controller block is to isolate the rest of the core from the transceiver and to move all of the transceiver signaling into the primary clock domain of the USB core. This allows the USB core to run synchronously with the system processor and its associated resources.

The Port Controller block interfaces to the UTM (Universal Transceiver Macrocell) via the UTMI interface. The UTM connects to the Analog Front End, that then provides the HS/FS/LS differential signaling to external devices. The primary function of the Port Controller block is to isolate the rest of the core from the transceiver and to move all of the transceiver signaling into the primary clock domain of the USB core.

This allows the USB core to run synchronously with the system processor and it's associated resources. For outgoing data, the UTM performs the parallel to serial conversion for 8-bit data coming from the Port Controller, bit stuff the serial stream, encode it into NRZI (non-return to zero invert) format, and send it to the Analog Front End. For incoming data, the UTM first decode the NRZI bit stream, performs bit destuffing and convert the serial stream to parallel 8-bit data before sending it to the Port Controller.

The Port Controller and the parallel interface of the UTM runs at 60MHz, as the UTM serial interface and the AFE runs at 480MHz.

10.6.3.8 System Bus Interface

The USB Host core interfaces to the system memory and processing resources using an AMBA AHB. This provides the features required for high performance/high clock frequency systems including:

- Synchronous Single clock edge operation
- Non-tristate on-chip implementation
- Initiator (master) and Target (slave) interfaces
- Burst transfers

10.7 System Level Issues and Core Configuration

The core is designed to be part of a larger system containing a processor, a bus architecture, and a memory system. This section is intended to provide information about the core that is required by a system designer to construct a complete system.

10.7.1 Configuration Constants

The following table summarizes the configuration controls.

Table 122. Configuration Controls (Sheet 1 of 2)

Constant	Description	Supported Values
VUSB_HS_RESET_TYPE	Controls the reset type.	0 = Synchronous 1 = Asynchronous.
VUSB_HS_RESET_OPTIONAL	Adds resets to register files and other groups of registers where a reset is not functionally required. This is only used in the MPH design and has no effect on DEV, SPH, or OTG. The MPH product has this option because a large number of flops are affected and adding resets to those flops has considerable increase on the gate count. All registers have reset in DEV, SPH, and OTG.	0 = No extra resets 1 = Reset all flops
VUSB_HS_CLOCK_ CONFIGURATION	Controls the clocking architecture.	[0,1,2,3] See following section on System Clocking.
VUSB_HS_NUM_PORT	Configures the number of ports for the multi-port host product. All other products must leave this value at 1.	[1-8] Number of downstream host ports.
VUSB_HS_PHY16_8	Controls the data interface to the UTMI transceiver. Should be set to match the transceiver connected to the core.	0= 8-bit data bus [60MHz] 1= 16-bit data bus [30MHz] 2= Software Controlled reset to 8-bit data bus [60MHz] 3= Software Controlled reset to 16-bit data bus [30MHz]
VUSB_HS_PHY_MODE	Controls the USB transceiver interface that is being used.	0 = UTMI Only 1 = Philips (w/Serial FS) 2 = Serial FS Only 3 = Software controlled reset to UTMI 4 = Software controlled reset to Philips 5 = Software controlled reset to Serial FS
VUSB_HS_SERIAL_MODE	Controls the usage of the xcvr_ser_se0_vmo output. This constant is only used when the serial engine is used	0 = xcvr_ser_se0_vmo is se0 1 = xcvr_ser_se0_vmo is vmo
VUSB_HS_BANDWIDTH_TESTING	Controls the testing of the core. This constant should always be zero.	0 = Normal Bandwidth Testing 1 = Bandwidth Starved Testing (subset of Normal Bandwidth Testing) Used for system clock less than 30 MHz.
VUSB_HS_DEV_EP	Controls the maximum number of endpoints supported by the core.	Integer values between 2 and 16. Set this to 1 for single and multi-port host-only products.
VUSB_HS_NUM_PORT	Controls the number of downstream ports in a host implementation.	Integer values between 1 and 8. Set this to 1 for non-multi-port products.
VUSB_HS_TT_PERIODIC_CONTEXTS	USB 2.0 specification requires a hub Transaction Translator to have 16 periodic contexts. But, for some host applications 4 can be adequate and a gate count savings can be realized.	4 or 16.
VUSB_HS_RX_DEPTH	Controls the size of the receive latency buffer.	Powers of 2 from 8 to 2048.
VUSB_HS_RX_BURST	Controls the Bus burst size for Rx DMA data transfers.	Integer values from 4 to 128.
VUSB_HS_TX_CHAN	Controls the size of each of the transmit latency buffers.	Powers of 2 from 16 to 128.
VUSB_HS_TX_BURST	Controls the Bus burst size for Tx DMA data transfers.	Integer values from 4 to 128.

Table 122. Configuration Controls (Sheet 2 of 2)

Constant	Description	Supported Values
VUSB_HS_TX_LOCAL_ CONTEXT_REGISTERS	Determines if device transmit context registers are implemented as a register file or stored in the TX-FIFO.	1 = Implement device transmit contexts in registers. (Removes need for read port A on TX-FIFO) 0 = Store device transmit contexts in TX FIFO. (Smaller total gate count but uses 4 words of VUSB_HS_TX_CHAN per endpoint)
VUSB_HS_HCIVERSION	Software readable host silicon version.	Unsigned 16-bit integer.
VUSB_HS_DCIVERSION	Software readable device silicon version.	Unsigned 16-bit integer.

10.8 Detailed Register Descriptions

The MMR registers for the USB Host are (for the most part) EHCI specification compliant, with exceptions as noted. The USB Host supports byte access to the MMR space and the default access is little-endian after power-up or hardware reset. The endian support for the MMR space is programmable and enables big endian access. This is important for byte access support and it is expected that one of the very first operations being performed on the USB host is to correctly set the required endian mode. Refer to the USBMODE register description for details.

Table 123. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible

Slave accesses from the controlling processor enables access to the configuration, control, and status registers. One function of the system address map is the registers base address, and must begin on a DWord (32-bit) boundary.

Configuration, control and status registers are divided into three categories:

- Identification registers — Identification Registers are used to declare the slave interface presence along with the complete set of the hardware configuration parameters. These are internal registers to the design and are not exposed to the user of the product.
- Capability registers — Static, read only capability registers define the software limits, restrictions, and capabilities of the host controller.
- Operational registers — Comprises dynamic control or status registers that can be read only, read/write, or read/write to clear. The following sections define the use of these registers.

EHCI registers are listed alongside device registers to show the complementary nature of host and device control.

Table 124. Interface Register Sets

Offset	Register Set	Explanation
000h to 0fch	Identification Registers	Identification registers are used to declare the slave interface presence and include a table of the hardware configuration parameters.
100h to 124h	Capability Registers	Capability registers specify the limits, restrictions, and capabilities of a host controller implementation. These values are used as parameters to the host driver.
140h to 1FCh	Operational Registers	Operational registers are used by the system software to control and monitor the operational state of the host controller.

10.9 Configuration, Control and Status Register Set

Offset addresses of registers are used in the following discussion. Figure 125 shows the base addresses of the USB 2.0 Host Controllers:

Table 125. Base Address of the USB 2.0 Host Controller

Device	Base Address
USB 2.0 Host Controller port 0	CD00_0000
USB 2.0 Host Controller port 1	CE00_0000

Table 126. Host Capability Registers (Sheet 1 of 2)

Offset	Size (Bytes)	Mnemonic	Register Name			Single Port Host (SPH)	
000h	4	ID	Identification Register			÷	
004h	4	HWGENERAL	General Hardware Parameters			÷	
008h	4	HWHOST	Host Hardware Parameters			÷	
010h	4	HWTXBUF	TX Buffer Hardware Parameters			÷	
014h	4	HWRXBUF	RX Buffer Hardware Parameters			÷	
020h-0FCh	232	(Reserved)	N/A				
100h	1	CAPLENGTH	Capability Register Length			÷	
101h	1	(Reserved)	N/A				
102h	2	HCIVERSION	Host Interface Version Number			÷	
104h	4	HCSPARAMS	Host Ctrl. Structural Parameters			÷	
108h	4	HCCPARAMS	Host Ctrl. Capability Parameters			÷	
10Ch – 11Fh	20	(Reserved)	N/A				
120h	2	(Reserved)					
122h	2	(Reserved)	N/A				

Table 126. Host Capability Registers (Sheet 2 of 2)

Offset	Size (Bytes)	Mnemonic	Register Name			Single Port Host (SPH)	
124h	4	DCCPARAMS	Device Ctrl. Capability Parameters				
128h – 13Ch	24	(Reserved)	N/A				
140h	4	USBCMD	USB Command			÷	
144h	4	USBSTS	USB Status			÷	
148h	4	USBINTR	USB Interrupt Enable			÷	
14Ch	4	FRINDEX	USB Frame Index			÷	
150h	4	(Reserved)	4G Segment Selector				
154h	4	PERIODICLISTBASE	Frame List Base Address			÷	
		Device Addr	USB Device Address				
158h	4	ASYNCLISTADDR	Next Asynchronous List Address			÷	
		Endpointlist Addr	Address at Endpoint list in memory				
15Ch	4	TTCTRL	TT status and control			÷	
160h	4	BURSTSIZE	Programmable Burst Size			÷	
164h	4	TXFILLTUNING	Host Transmit Pre-Buffer Packet Tuning			÷	
16Ch	4	N/A	(Reserved)				
170-17Ch	16	N/A	(Reserved)				
184h	4	PORTSC1	Port Status/Control 1			÷	
1A8h	4	USBMODE	USB Device Mode			÷	

10.10 Identification Registers

Identification registers are used to declare the slave interface presence and include a table of the hardware configuration parameters.

10.10.1 ID

Address: Base + 000h

Default Value: 0x0042FA05

Attribute: Read Only

Size: 32 bits

The Identification register (ID) provides a simple way to determine if the USB 2.0 core is provided in the system. The ID register identifies the USB 2.0 core and its revision.

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
(Reserved)								REVISION								1	1	NID						0	0	ID					

Table 127. Identification Register Fields

Field	Description
ID[5:0]	Configuration number. This number is set to 0x05 and indicates that the peripheral is the USB 2.0 core.
NID[5:0]	Ones complement version of ID[5:0].
REVISION[7:0]	Revision number of the core
(Reserved)	These bits are reserved and should be zero.

10.10.2 HWGENERAL

Address: Base + 004h

Default Value: 0x00000003

Attribute: Read Only

Size: 32 bits

General hardware parameters as defined in Section 10.7, "System Level Issues and Core Configuration".

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
(Reserved)																						SM	PHYM			PHYW		BWT	CLKC		RT

Table 128. HWGENERAL – General Hardware Parameters: Fields

Field	Description
(Reserved)	These bits are reserved and should be zero.
SM	VUSB_HS_SERIAL_MODE
PHYM	VUSB_HS_PHY_MODE
PHYW	VUSB_HS_PHY16_8
BWT	VUSB_HS_BANDWIDTH_TESTING
CLKC	VUSB_HS_CLOCK_CONFIGURATION
RT	VUSB_HS_RESET_TYPE

10.10.3 HWHOST

Address: Base + 008h

Default Value: 0x10020001

Attribute: Read Only

Size: 32 bits

Host hardware parameters as defined in Section 10.7, "System Level Issues and Core Configuration".

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
TTPER								TTASY								(Reserved)												NPORT			HC

Table 129. HWHOST – Host Hardware Parameters

Field	Description
TTPER	VUSB_HS_TT_PERIODIC_CONTEXTS
TTASY	VUSB_HS_TT_ASYNC_CONTEXTS
(Reserved)	These bits are reserved and should be zero.
NPORT	VUSB_HS_NUM_PORT-1
HC	VUSB_HS_HOST

10.10.4 HWTXBUF

Address: Base + 010h

Default Value: 0x80050608

Attribute: Read Only

Size: 32 bits

TX buffer hardware parameters as defined in Section 10.7, "System Level Issues and Core Configuration".

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
TXLCR	(Reserved)							TXCHANADD								TXADD								TCBURST							

Table 130. HWTXBUF – TX Buffer Hardware Parameters

Field	Description
TXLC	VUSB_HS_TX_LOCAL_CONTEXT_REGISTERS
(Reserved)	These bits are reserved and should be zero.
TXCHANADD	VUSB_HS_TX_CHAN_ADD
TXADD	VUSB_HS_TX_ADD
TCBURST	VUSB_HS_TX_BURST

10.10.5 HWRXBUF

Address: Base + 014h

Default Value: 0x00000608

Attribute: Read Only

Size: 32 bits

RX buffer hardware parameters as defined in Section 10.7, "System Level Issues and Core Configuration".

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
(Reserved)																RXADD								RXBURST							

Table 131. HWRXBUF – RX Buffer Hardware Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
RXADD	VUSB_HS_RX_ADD
RXBURST	VUSB_HS_RX_BURST

10.11 Host Capability Registers

Host Capability registers specify the software limits, restrictions, and capabilities of the host controller implementation.

10.11.1 CAPLENGTH – EHCI Compliant

Address: Base + 100h

Default Value: 40h

Attribute: Read Only

Size: 8 bits

This register is used to indicate the offset to add to the register base address at the beginning of the Operational Register.

7	6	5	4	3	2	1	0
CAPLENGTH[7:0]							

10.11.2 HCIVERSION – EHCI Compliant

Address: Base + 102h

Default Value: 0100h

Attribute: Read Only

Size: 16 bits

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this host controller. The most significant byte of this register represents a major revision and the least significant byte is the minor revision.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HCIVERSION[15:0]															

10.11.3 HCSPARAMS – EHCI Compliant with Extensions

Address: Base + 104h

Default Value: 0x0001_0011

Attribute: Read Only

Size: 32 bits

Port steering logic capabilities are described in this register.

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
(Reserved)				N_TT				N_PTT				(Rsvd)			PI	N_CC				N_PCC				(Rsvd)			PPC	N_PORTS			

Table 132. HCSPARAMS – Host Control Structural Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
N_TT[3:0]	Number of Transaction Translators (N_TT). This field indicates the number of embedded transaction translators associated with the USB 2.0 host controller. For this implementation, N_TT = “0000.” This in a non-EHCI field to support embedded TT.
N_PTT[3:0]	Number of Ports per Transaction Translator (N_PTT). This field indicates the number of ports assigned to each transaction translator within the USB 2.0 host controller. For this implementation, N_PTT = “0000.” This in a non-EHCI field to support embedded TT.
PI	Port Indicators (P INDICATOR). This bit indicates whether the ports support port indicator control. When set to one, the port status and control registers include a read/writable field for controlling the state of the port indicator. This field is always “1.”
N_CC[3:0]	Number of Companion Controller (N_CC). This field indicates the number of companion controllers associated with this USB 2.0 host controller. A zero in this field indicates there are no internal Companion Controllers. Port-ownership hand-off is not supported. A value larger than zero in this field indicates there are companion USB1.1 host controller(s). Port-ownership hand-offs are supported. High, Full- and Low-speed devices are supported on the host controller root ports. In this implementation this field is always “0.”
N_PCC[3:0]	Number of Ports per Companion Controller. This field indicates the number of ports supported per internal Companion Controller. It is used to indicate the port routing configuration to the system software. For example, if N_PORTS has a value of 6 and N_CC has a value of 2 then N_PCC could have a value of 3. The convention is that the first N_PCC ports are assumed to be routed to companion controller 1, the next N_PCC ports to companion controller 2, and so on. In the previous example, the N_PCC could have been 4, where the first 4 are routed to companion controller 1 and the last two are routed to companion controller 2. The number in this field must be consistent with N_PORTS and N_CC. In this implementation this field is always “0”.
PPC	Port Power Control. This field indicates whether the host controller implementation includes port power control. A one indicates the ports have port power switches. A zero indicates the ports do not have port power switches. The value of this field affects the functionality of the Port Power field in each port status and control register.
N_PORTS[3:0]	Number of downstream ports. This field specifies the number of physical downstream ports implemented on this host controller. The value of this field determines how many port registers are addressable in the Operational Register. Valid values are in the range of 1h to Fh. A zero in this field is undefined. The number of ports for a host implementation can be a parameter from 1 to 8.

10.11.4 HCCPARAMS – EHCI Compliant

Address: Base + 108h

Default Value: 0006h

Attribute: Read Only

Size: 32 bits

This register identifies multiple mode control (time-base bit functionality) addressing capability.

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
(Reserved)																EECP[7:0]								IST[7:4]				(Rsvd)	ASP	PFL	ADC

Table 133. HCCPARAMS – Host Control Capability Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
EECP[7:0]	EHCI Extended Capabilities Pointer. Default = 0. This optional field indicates the existence of a capabilities list. A value of 00h indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 40h or greater if implemented to maintain the consistency of the PCI header defined for this class of device. For this implementation this field is always “0”.
IST[7:4]	Isochronous Scheduling Threshold. This field indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit [7] is zero, the value of the least significant 3 bits indicates the number of micro-frames a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit [7] is a one, then host software assumes the host controller may cache an isochronous data structure for an entire frame. This field is always “0”.
(Rsvd)	These bits are reserved and should be zero.
ASP	Asynchronous Schedule Park Capability. Default = 1. If this bit is set to a one, then the host controller supports the park feature for high-speed queue heads in the Asynchronous Schedule. The feature can be disabled or enabled and set to a specific level by using the Asynchronous Schedule Park Mode Enable and Asynchronous Schedule Park Mode Count fields in the USBCMD register. This field is always “1”
PFL	Programmable Frame List Flag. If this bit is set to zero, then the system software must use a frame list length of 1024 elements with this host controller. The USBCMD register Frame List Size field is a read-only register and must be set to zero. If set to a one, then the system software can specify and use a smaller frame list and configure the host controller via the USBCMD register Frame List Size field. The frame list must always be aligned on a 4K-page boundary. This requirement ensures that the frame list is always physically contiguous. This field is always “1”.
ADC	64-bit Addressing Capability. This field is always “0”. No 64-bit addressing capability is supported.

10.11.5 Reserved Register #1

Address: Base + 120h

Default Value: 0001h

Attribute: Read Only - Reserved for Future Use

Size: 16 bits

10.11.6 DCCPARAMS (Non-EHCI)

Address: Base + 124h

Default Value: 0x00000182

Attribute: Read Only

Size: 32 bits

These fields describe the overall host capability of the controller.

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
(Reserved)																							HC	RES	(Rsvd)	(Rsvd)	RES				

Table 134. DCCPARAMS - Device Control Capability Parameters

Field	Description
(Reserved)	These bits are reserved and should be zero.
HC	Host Capable. When this bit is 1, this controller is capable of operating as an EHCI compatible USB 2.0 host controller.
(Reserved)	This value when read returns as a 1
(Reserved)	These bits are reserved and should be zero.
(Reserved)	This value when read returns as a 4

10.12 Host Operational Registers

Operational registers are comprised of dynamic control or status registers that can be read only, read/write, or read/write to clear. The following sections define the use of these registers.

10.12.1 USBCMD

Address: Base + 140h

Default Value: 0x00080B00 (host mode)

Attribute: Read Only, Read/Write, Write Only (field dependent)

Size: 32 bits

The serial bus host controller executes the command indicated in this register.

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
(Reserved)								ITC[7:0]								FS2	R	SUTW	ATDTW	ASPE	R	ASP1	ASP0	LR	IAA	ASE	PSE	FS1	FS0	RST	RS

Table 135. USBCMD – USB Command Register (Sheet 1 of 2)

Field	Description
R, (Reserved)	These bits are reserved and should be zero.
ITC[7:0]	Interrupt Threshold Control —Read/Write. Default 08h. The system software uses this field to set the maximum rate at which the host controller issues interrupts. ITC contains the maximum interrupt interval measured in micro-frames. Valid values are shown below. Value Maximum Interrupt Interval 00h Immediate (no threshold) 01h 1 micro-frame 02h 2 micro-frames 04h 4 micro-frames 08h 8 micro-frames 10h 16 micro-frames 20h 32 micro-frames 40h 64 micro-frames
SUTW	Setup TripWire – Read/Write. This bit is used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off, there exists a hazard when new setup data arrives and the DCD copying setup from the QH for a previous setup packet. This bit is set and cleared by software and is cleared by hardware when a hazard exists.
ATDTW	Add dTD TripWire – Read/Write. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit must also be cleared by hardware when its state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized.
ASPE	Asynchronous Schedule Park Mode Enable (OPTIONAL) æ Read/Write. Software uses this bit to enable or disable Park mode. When this bit is one, Park mode is enabled. When this bit is a zero, Park mode is disabled. This field is set to “1” in this implementation.
ASP[1:0]	Asynchronous Schedule Park Mode Count (OPTIONAL) æ Read/Write. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. See Asynchronous Schedule Park Mode for full operational details. Valid values are 1h to 3h. Software must not write a zero to this bit when Park Mode Enable is a one as this results in undefined behavior. This field is set to 3h in this implementation.
LR	Light Host Controller Reset (OPTIONAL) — Read Only. Not Implemented. This field is always “0”.
IAA	Interrupt on Async Advance Doorbell — Read/Write. This bit is used as a doorbell by software to tell the host controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the host controller has evicted all appropriate cached schedule states, it sets the Interrupt on Async Advance status bit in the USBSTS register. If the Interrupt on Sync Advance Enable bit in the USBINTR register is one, then the host controller asserts an interrupt at the next interrupt threshold. The host controller sets this bit to zero after it has set the Interrupt on Sync Advance status bit in the USBSTS register to one. Software should not write a one to this bit when the asynchronous schedule is inactive. Doing so yields undefined results. This bit is only used in host mode.
ASE	Asynchronous Schedule Enable — Read/Write. Default 0b. This bit controls whether the host controller skips processing the Asynchronous Schedule. 0 = Do not process the Asynchronous Schedule. 1 = Use the ASYNCLISTADDR register to access the Asynchronous Schedule. Only the host controller uses this bit.

Table 135. USBCMD – USB Command Register (Sheet 2 of 2)

Field	Description
PSE	Periodic Schedule Enable— Read/Write. Default Ob. This bit controls whether the host controller skips processing the Periodic Schedule. 0 = Do not process the Periodic Schedule. 1 = Use the PERIODICLISTBASE register to access the Periodic Schedule. Only the host controller uses this bit.
FS[2:0]	Frame List Size — (Read/Write or Read Only). Default 000b. This field is Read/Write only if Programmable Frame List Flag in the HCCPARAMS registers is set to one. This field specifies the size of the frame list that controls the bits in the Frame Index Register that should be used for the Frame List Current index. ***Note:*** This field is made up from USBCMD bits 15, 3 and 2. 000 = 1024 elements (4096 bytes) Default value 001 = 512 elements (2048 bytes) 010 = 256 elements (1024 bytes) 011 = 128 elements (512 bytes) 100 = 64 elements (256 bytes) 101 = 32 elements (128 bytes) 110 = 16 elements (64 bytes) 111 = 8 elements (32 bytes) Only the host controller uses this field.
RST	Controller Reset (RESET) — Read/Write. Software uses this bit to reset the controller. This bit is set to zero by the Host Controller when the reset process is complete. Software cannot terminate the reset process early by writing a zero to this register. Host Controller: When software writes a one to this bit, the Host Controller resets its internal pipelines, timers, counters, state machines and so on. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit to a one when the HCHalted bit in the USBSTS register is a zero. Attempting to reset an actively running host controller results in undefined behavior.
RS	**Run/Stop (RS) – Read/Write.** Default 0b. 0 = Stop. 1 = Run. Host Controller: When set to a 1, the Host Controller proceeds with the execution of the schedule. The Host Controller continues execution as long as this bit is set to a one. When this bit is set to 0, the Host Controller completes the current transaction on the USB and then halts. The HC Halted bit in the status register indicates when the Host Controller has finished the transaction and has entered the stopped state. Software should not write a one to this field unless the host controller is in the Halted state (that is, HCHalted in the USBSTS register is a one).

10.12.2 USBSTS

Address: Base + 144h

Default Value: 0x00001000 (host mode)

Attribute: Read Only, Read/Write, Read/Write-Clear (field dependent)

Size: 32 bits

This register indicates various states of the Host Controller and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them.

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
(Reserved)																AS	PS	RCL	HCH	(Rsvd)	(Rsvd)	(Rsvd)	(Rsvd)	SRI	(Rsvd)	AAI	SEI	FRI	PCI	UEI	UI

Table 136. USBSTS – USB Status (Sheet 1 of 2)

Field	Description
(Reserved)	These bits are reserved and should be zero.
AS	Asynchronous Schedule Status — Read Only. 0=Default. This bit reports the current real status of the Asynchronous Schedule. When set to zero the asynchronous schedule status is disabled and if set to one the status is enabled. The Host Controller is not required to immediately disable or enable the Asynchronous Schedule when software transitions the Asynchronous Schedule Enable bit in the USBCMD register. When this bit and the Asynchronous Schedule Enable bit are the same value, the Asynchronous Schedule is enabled (1) or disabled (0). Only used by the host controller.
PS	Periodic Schedule Status — Read Only. 0=Default. This bit reports the current real status of the Periodic Schedule. When set to zero the periodic schedule is disabled, and if set to one the status is enabled. The Host Controller is not required to immediately disable or enable the Periodic Schedule when software transitions the Periodic Schedule Enable bit in the USBCMD register. When this bit and the Periodic Schedule Enable bit are the same value, the Periodic Schedule is enabled (1) or disabled (0). Only used by the host controller.
RCL	Reclamation — Read Only. 0=Default. This is a read-only status bit used to detect an empty asynchronous schedule. Only used by the host controller.
HCH	HCHalted — Read Only. 1=Default. This bit is a zero whenever the Run/Stop bit is a one. The Host Controller sets this bit to one after it has stopped executing because of the Run/Stop bit being set to 0, by software or by the Host Controller hardware (for example, internal error). Only used by the host controller.
R	(Reserved). These bits are reserved and should be zero.
(Reserved)	(Reserved) - These bits are reserved and should be zero
SRI	SOF Received – R/WC. 0=Default. In host mode, this bit is set every 125us and can be used by host controller driver as a time base. Software writes a 1 to this bit to clear it. This is a non-EHCI status bit.
(Reserved)	(Reserved) - These bits are reserved and should be zero
AAI	Interrupt on Async Advance — R/WC. 0=Default. System software can force the host controller to issue an interrupt the next time the host controller advances the asynchronous schedule by writing a one to the Interrupt on Async Advance Doorbell bit in the USBCMD register. This status bit indicates the assertion of that interrupt source. Only used by the host controller.
SEI	System Error— R/WC. This bit is not used in this implementation and is always set to “0”.
FRI	Frame List Rollover — R/WC. The Host Controller sets this bit to a one when the Frame List Index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the Host Controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host controller.

Table 136. USBSTS – USB Status (Sheet 2 of 2)

Field	Description
PCI	Port Change Detect — R/WC. The Host Controller sets this bit to a one when on any port a Connect Status occurs, a Port Enable/Disable Change occurs, or the Force Port Resume bit is set as the result of a J-K transition on the suspended port. This bit is not EHCI compatible.
UEI	USB Error Interrupt (USBERRINT) — R/WC. When completion of a USB transaction results in an error condition, this bit is set by the Host Controller. This bit is set along with the USBINT bit, if the TD where the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section (Reference Host Operation Model: Transfer/Transaction Based Interrupt – that is, 4.15.1 in EHCI) for a complete list of host error interrupt conditions.
UI	USB Interrupt (USBINT) — R/WC. This bit is set by the Host Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the Host Controller when a short packet is detected. A short packet is when the actual number of bytes received is less than the expected number of bytes.

10.12.3 USBINTR

Address: Base + 148h

Default Value: 0x00000000

Attribute: Read/Write

Size: 32 bits

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB Status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

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
(Reserved)																							(Rsvd)	(Rsvd)	(Rsvd)	AAE	SEE	FRE	PCE	UEE	UE

Table 137. USBINTR – USB Interrupt Enable

Field	Interrupt Source	Description
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
(Reserved)	(Reserved)	These bits are reserved and should be zero.
AAE	Interrupt on Async Advance Enable	When this bit is a one, and the Interrupt on Async Advance bit in the USBSTS register is a one, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the Interrupt on Async Advance bit. Only used by the host controller.
SEE	System Error Enable	When this bit is a one, and the System Error bit in the USBSTS register is a one, the host controller issues an interrupt. The interrupt is acknowledged by software clearing the System Error bit.

Table 137. USBINTR – USB Interrupt Enable

Field	Interrupt Source	Description
FRE	Frame List Rollover Enable	When this bit is a one, and the Frame List Rollover bit in the USBSTS register is a one, the host controller issues an interrupt. The interrupt is acknowledged by software clearing the Frame List Rollover bit. Only used by the host controller.
PCE	Port Change Detect Enable	When this bit is a one, and the Port Change Detect bit in the USBSTS register is a one, the host controller issues an interrupt. The interrupt is acknowledged by software clearing the Port Change Detect bit.
UEE	USB Error Interrupt Enable	When this bit is a one, and the USBERRINT bit in the USBSTS register is a one, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBERRINT bit in the USBSTS register.
UE	USB Interrupt Enable	When this bit is a one, and the USBINT bit in the USBSTS register is a one, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBINT bit.

10.12.4 FRINDEX

Address: Base + 14Ch

Default Value: Undefined (free running counter)

Attribute: Read/Write in host mode

Size: 32 bits

This register is used by the host controller to index the periodic frame list. The register updates every 125 ms (once each micro-frame). Bits [N: 3] are used to select a particular entry in the Periodic Frame List during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the Frame List Size field in the USBCMD register.

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the Host Controller is in the 'Halted' state as indicated by the HCHalted bit. A write to this register as the Run/Stop hit is set to a one produces undefined results. Writes to this register also affect the SOF value.

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
(Reserved)																		FRINDEX[13:0]													

Table 138. FRINDEX – USB Frame Index

Field	Description
(Reserved)	These bits are reserved and should be zero.
FRINDEX	Frame Index. The value, in this register, increments at the end of each time frame (for example, micro-frame). Bits [N: 3] are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or micro-frames) before moving to the next index. The following illustrates values of N based on the value of the Frame List Size field in the USBCMD register, when used in host mode. USBCMD (Frame List Size) Number Elements N 000b (1024) 12 001b (512) 11 010b (256) 10 011b (128) 9 100b (64) 8 101b (32) 7 110b (16) 6 111b (8) 5 bits 2:0 indicate the current microframe.

10.12.5 CTRLDSSEGMENT

Address: Base + 150h

Default Value: 0x00000000

Attribute: Read Only

Size: 32 bits

This register is not used in this implementation.

10.12.6 PERIODICLISTBASE

Address: Base + 154h

Default Value: 0x00000000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

10.12.6.1 Host Controller (PERIODICLISTBASE)

This 32-bit register contains the beginning address of the Periodic Frame List in the system memory. HCD loads this register prior to starting the schedule execution by the Host Controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the Frame Index Register (FRINDEX) to enable the Host Controller to step through the Periodic Frame List in sequence.

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
PERBASE[31:12]																				(Reserved)											

Table 139. PERIODICLISTBASE - Host Controller Frame List Base Address

Field	Description
BASEADR	Base Address (Low). These bits correspond to memory address signals [31:12], respectively. Only used by the host controller.
(Reserved)	Must be written as zeros. During runtime, the values of these bits are undefined.

10.12.7 ASYNCLISTADDR; ENDPOINTLISTADDR

Address: Base + 158h

Default Value: 0x00000000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

10.12.7.1 Host Controller (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits [4:0] of this register cannot be modified by the system software and always returns a zero when read.

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
ASYBASE[31:5]																											(Reserved)			

Table 140. ASYNCLISTADDR - Host Controller Next Asynchronous Address

Field	Description
ASYBASE[31:5]	Link Pointer Low (LPL). These bits correspond to memory address signals [31:5], respectively. This field may only reference a Queue Head (OH). Only used by the host controller.
(Reserved)	These bits are reserved and their value has no effect on operation.

10.12.8 TTCTRL

Address: Base + 15Ch

Default Value: 0x00000000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

This register contains parameters needed for internal TT operations.

This register is not used in the device controller operation.

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
(Rsvd)	TTHA							(Reserved)																							

Table 141. TTCTRL Register

Field	Description
(Reserved)	These bits are reserved and their value has no effect on operation.
TTHA	Internal TT Hub Address Representation. (Read/Write) [Default = 0]. This field is used to match against the Hub Address field in QH and siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the Hub Address in the QH or siTD does not match this address then the packet is broadcast on the High Speed ports designated for a downstream High Speed hub with the address in the QH/siTD. Only used in the host controller operation.

10.12.9 BURSTSIZE

Address: Base + 160h

Default Value: 0x0000_0808

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

This register is used to control dynamically change the burst size used during data movement on the initiator (master) interface.

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
(Reserved)															TXPBURST									RXPBURST							

Table 142. BURSTSIZE - Host Controller Embedded TT Async. Buffer Status

Field	Description
(Reserved)	These bits are reserved and their value has no effect on operation.
TXPBURST	Programmable TX Burst Length. (Read/Write) Default is the constant VUSB_HS_TX_BURST. This register represents the maximum length of a the burst in 32-bit words when moving data from system memory to the USB bus.
RXPBURST	Programmable RX Burst Length. (Read/Write) Default is the constant VUSB_HS_RX_BURST. This register represents the maximum length of a the burst in 32-bit words when moving data from the USB bus to system memory.

10.12.10 TXFILLTUNING

Address: Base + 164h

Default Value: 0x00020000

Attribute: Read/Write (Writes must be DWord Writes)

Size: 32 bits

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
(Reserved)										TXFIFOTHRES						(Rsvd)			TXSCHEALTH					TXSCHOH							

The fields in this register control performance tuning associated with how the host controller posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

- T0 = Standard packet overhead
- T1 = Time to send data payload
- Tff = Time to fetch packet into TX FIFO up to specified level.
- Ts = Total Packet Flight Time (send-only) packet
- Ts = T0 + T1
- Tp = Total Packet Time (fetch and send) packet
- Tp = Tff + T0 + T1

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure Tp remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at anytime during the pre-fill operation the time remaining the [micro]frame is < Ts then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the host controller eventually recovers, a mark is made the scheduler health counter to note the occurrence of a back-off event.

When a back-off event is detected, the partial packet fetched should be discarded from the latency buffer to make room for periodic traffic that begins after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH (Tff) described below.

Table 143. TXFILLTUNING - Performance Tuning Control Register

Field	Description
(Reserved)	These bits are reserved and their value has no effect on operation.
TXFIFOTHRES	FIFO Burst Threshold. (Read/Write) [Default = 2] This register controls the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be as low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if the Stream Disable bit in USBMODE register is set.
TXSCHHEALTH	Scheduler Health Counter. (Read/Write To Clear) [Default = 0] This register increments when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter becomes maximum at 31.
TXSCHOH	Scheduler Overhead. (Read/Write) [Default = 0] This register adds an additional fixed offset to the schedule time estimator described above as *Tff*. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is always 1.267 the MPH product.

10.12.11 PORTSCx

Address: Base + 184h + (4*(Port Number –1))

where Port Number =1,2,3,…8

Default Value: IIII_0000_0000_0000_0000_XX00_0000_0000b (host mode)

I = Implementation dependent, this implementation: 0x0

X = Unknown

Attribute: R0, Read/Write, R/WC (field dependent)

Size: 32 bits

10.12.11.1 Host Controller

A host controller must implement one to eight port registers. The number of port registers implemented by a particular instantiation of a host controller is documented in the HCSPARAMs register. Software uses this information as an input parameter to determine how many ports need service.

This register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

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
PTS		STS	PTW	PSPD		(Rsvd	PFSC	PHCD	WKOC	WKDS	WKCN	PTC[3:0]				PIC		PO	PP	LS		HSP	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS

Table 144. PORTSCx - Port Status Control[1:8] (Sheet 1 of 5)

Field	Description
PTS	Parallel Transceiver Select – Read/Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY_MODE to control the parallel transceiver interface that is selected. If VUSB_HS_PHY_MODE is set for 0,1 or 2 then this bit is read only. If VUSB_HS_PHY_MODE is 3,4 or 5 then this bit is read/write. 0 = selects the UTMI parallel Interface. 1 = selects the Philips parallel Interface. This bit is not defined in the EHCI specification.
STS	Serial Transceiver Select – Read/Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY_MODE to control whether the parallel or serial transceiver interface is selected. If VUSB_HS_PHY_MODE is set for 0,1 or 2 then this bit is read only. If VUSB_HS_PHY_MODE is 3,4 or 5 then this bit is read/write. 0 = selects the parallel transceiver Interface. 1 = selects the Serial (Full Speed) transceiver interface. For the UTMI interface, this bit can be used to select between a Parallel UTMI PHY and a standard 1.1 PHY. For the Philips interface, this bit is used to prevent use of the Parallel PHY and generate signalling exclusively through the serial interface. (For Philips this also implies no chirping.) This bit is not defined in the EHCI specification.

Table 144. PORTSCx - Port Status Control[1:8] (Sheet 2 of 5)

Field	Description
PTW	Parallel Transceiver Width – Read/Write. This register bit is used in conjunction with the configuration constant VUSB_HS_PHY8_16 to control whether the data bus width of the UTMI transceiver interface. If VUSB_HS_PHY8_16 is set for 0 or 1 then this bit is read only. If VUSB_HS_PHY8_16 is 2 or 3 then this bit is read/ write. 0 = selects the 8-bit [60MHz] UTMI interface. 1 = selects the 16-bit [30MHz] UTMI interface. This bit is not defined in the EHCI specification.
PSPD	Port Speed – Read Only. This register field indicates the speed that the port is operating. For HS mode operation in the host controller the port routing steers data to the Protocol engine. For FS and LS mode operation in the host controller, the port routing steers data to the Protocol Engine w/ Embedded Transaction Translator. 00 – Full Speed 01 – Low Speed 10 – High Speed This bit is not defined in the EHCI specification.
PFSC	Port Force Full Speed Connect – Read/Write. Default = 0b. Writing this bit to a 1b forces the port to only connect at Full Speed. It disables the chirp sequence that allows the port to identify itself as High Speed. This is useful for testing FS configurations with a HS host, hub or device. This bit is not defined in the EHCI specification. This bit is for debugging purposes.
PHCD	PHY Low Power Suspend - Clock Disable (PLPSCD) – Read/Write. Default = 0b. Writing this bit to a 1b disables the PHY clock. Writing a 0b enables it. Reading this bit indicates the status of the PHY clock. ***Note:*** The PHY clock cannot be disabled if it is being used as the system clock. The PHY can be put into Low Power Suspend – Clock Disable when the downstream device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of software. This bit is not defined in the EHCI specification.
WKOC	Wake on Over-current Enable (WKOC_E) — Read/Write. Default = 0b. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events. This field is zero if Port Power (PP) is zero. This bit is output from the controller as signal pwrctl_wake_ovrcurr_en (host core only) for use by an external power control circuit.
WKDC	Wake on Disconnect Enable (WKDSCNNT_E) — Read/Write. Default=0b. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events. This field is zero if Port Power (PP) is zero. This bit is output from the controller as signal pwrctl_wake_dscnnt_en (host core only) for use by an external power control circuit.
WKCN	Wake on Connect Enable (WKCNNT_E) — Read/Write. Default=0b. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events. This field is zero if Port Power (PP) is zero. This bit is output from the controller as signal pwrctl_wake_dscnnt_en (host core only) for use by an external power control circuit.

Table 144. PORTSCx - Port Status Control[1:8] (Sheet 3 of 5)

Field	Description
PTC[3:0]	Port Test Control — Read/Write. Default = 0000b. Any other value than zero indicates that the port is operating in test mode. Value Specific Test 0000b TEST_MODE_DISABLE 0001b J_ STATE 0010b K_STATE 0011b SEQ_NAK 0100b Packet 0101b FORCE_ENABLE 0110b to 1111b Reserved Refer to Chapter 7 of the USB Specification Revision 2.0 for details on each test mode.
PIC[1:0]	Port Indicator Control — Read/Write. Default = 0b. Bit Value Meaning 00b Port indicators are off 01b Amber 10b Green 11b Undefined Refer to the USB Specification Revision 2.0 for a description on how these bits are to be used. This field is output from the controller as signals port_ind_ctl_1 & port_ind_ctl_0 for use by an external led driving circuit.
PO	Port Owner—Read Only. Default = 0. System software uses this field to release ownership of the port to a selected host controller (in the event that the attached device is not a high-speed device). Software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port. Port owner handoff is not implemented in this design, therefore this bit is always 0.
PP	Port Power (PP)—Read/Write The function of this bit depends on the value of the Port Power Switching (PPC) field in the HCSPARAMS register. The behavior is as follows: PPC PP Operation 1b 1b/0b – RW. Host controller requires port power control switches. This bit represents the current setting of the switch (0=off, 1=on). When power is not available on a port (that is, PP equals a 0), the port is non-functional and does not report attaches, detaches, and so on. When an over-current condition is detected on a powered port and PPC is a one, the PP bit in each affected port can be transitioned by the host controller driver from a one to a zero (removing power from the port). This feature is implemented in the host controller (PPC = 1).
LS[1:0]	Line Status—Read Only. These bits reflect the current logical levels of the D+ (bit 11) and D- (bit 10) signal lines. The encoding of the bits are: Bits [11:10] Meaning 00b SE0 10b J-state 01b K-state 11b Undefined The use of line state by the host controller driver is not necessary (unlike EHCI), because the port controller state machine and the port routing manage the connection of LS and FS.
HSP	High-Speed Port — Read Only. Default = 0b. When the bit is one, the host connected to the port is in high-speed mode and if set to zero, the host connected to the port is not in a high-speed mode. ***Note:*** HSP is redundant with PSPD(27:26) but remains in the design for compatibility. This bit is not defined in the EHCI specification.

Table 144. PORTSCx - Port Status Control[1:8] (Sheet 4 of 5)

Field	Description
PR	Port Reset - Read/Write. This field is zero if Port Power (PP) is zero. 0 = Port is not in Reset. 1 = Port is in Reset. Default 0. When software writes a one to this bit the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. This bit automatically changes to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver.
SUSP	Suspend In Host Mode: Read/Write. 0 = Port not in suspend state. 1 = Port in suspend state. Default=0. Port Enabled Bit and Suspend bit of this register define the port states as follows: Bits [Port Enabled, Suspend]Port State 0x Disable 10 Enable 11 Suspend When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction is in progress when this bit is written to 1. In the suspend state, the port is sensitive to resume detection. ***Note:*** The bit status does not change until the port is suspended and that there can be a delay in suspending a port if there is a transaction currently in progress on the USB. The host controller unconditionally sets this bit to zero when software sets the Force Port Resume bit to zero. A write of zero to this bit is ignored by the host controller. If host software sets this bit to a one when the port is not enabled (that is, Port enabled bit is a zero) the results are undefined. This field is zero if Port Power (PP) is zero in host mode.
FPR	Force Port Resume —Read/Write. 0 = No resume (K-state) detected/driven on port. 1 = Resume detected/driven on port. Default = 0. Software sets this bit to one to drive resume signaling. The Host Controller sets this bit to one if a J-to-K transition is detected as the port is in the Suspend state. When this bit transitions to a one because a J-to-K transition is detected, the Port Change Detect bit in the USBSTS register is also set to one. This bit automatically changes to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver. ***Note:*** When the Host controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a one. This bit remains a one until the port has switched to the high-speed idle. Writing a zero has no affect because the port controller times the resume operation clear the bit the port control state switches to HS or FS idle. This field is zero if Port Power (PP) is zero in host mode. This bit is not-EHCI compatible.
OCC	Over-current Change—R/WC. Default=0. This bit gets set to one when there is a change to Over-current Active. Software clears this bit by writing a one to this bit position. For host implementations the user can provide over-current detection to the vbus_pwr_fault input for this condition.
OCA	Over-current Active—Read Only. Default 0. 0 = This port does not have an over-current condition. 1 = This port currently has an over-current condition. This bit automatically transitions from one to zero when the over current condition is removed. For host implementations the user can provide over-current detection to the vbus_pwr_fault input for this condition.

Table 144. PORTSCx - Port Status Control[1:8] (Sheet 5 of 5)

Field	Description
PEC	Port Enable/Disable Change—R/WC. 0 = No change. 1 = Port enabled/disabled status has changed. Default = 0. For the root hub, this bit gets set to a one only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a one to it. This field is zero if Port Power (PP) is zero.
PE	Port Enabled/Disabled—Read/Write. 0 = Disable. 1 = Enable. Default 0. Ports can only be enabled by the host controller as a part of the reset and enable. Software cannot enable a port by writing a one to this field. Ports can be disabled by a fault condition (disconnect event or other fault condition) or by the host software. ***Note:*** The bit status does not change until the port state actually changes. There can be a delay in disabling or enabling a port due to other host controller and bus events. When the port is disabled, (0b) downstream propagation of data is blocked except for reset. This field is zero if Port Power (PP) is zero in host mode.
CSC	Connect Status Change—R/WC. 0 = No change. 1 = Change in Current Connect Status. Default 0. Indicates a change has occurred in the port's Current Connect Status. The host controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware is 'setting' an already-set bit (that is, the bit remains set). Software clears this bit by writing a one to it. This field is zero if Port Power (PP) is zero in host mode.
CCS	Current Connect Status—Read Only. 0 = No device is present. 1 = Device is present on port. Default = 0. This value reflects the current state of the port, and may not correspond directly to the event that caused the Connect Status Change bit (Bit 1) to be set. This field is zero if Port Power (PP) is zero in host mode.

10.12.12 USBMODE

Address: Base + 1A8h

Default Value: 00000003h (host mode)

Attribute: R/WO, Read Only

Size: 32 bits

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
(Reserved)																											SDIS	SLOM	ES	CM		

Table 145. USBMODE - USB Device Mode

Field	Description
(Reserved)	These bits are reserved and should be zero.
SDIS	Stream Disable Mode. 0 = Inactive [Default] 1 = Active Setting to a '1' ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring that the TX latency is filled to capacity before the packet is launched onto the USB. ***Notes:*** 1. Time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature. 2. The use of this feature substantially limits of the overall USB performance that can be achieved.
ES	Endian Select – Read/Write. This bit can change the byte alignment of the transfer buffers to match the host microprocessor. The bit fields in the microprocessor interface and the data structures are unaffected by the value of this bit because they are based upon the 32-bit word. Bit Meaning 0 little-endian [default] 1 big-endian
CM[1:0]	Controller Mode – R/WO. Controller mode is defaulted to the proper mode for host only implementations. For those designs that contain host capability, the controller defaults to an idle state and must be initialized to the desired operating mode after reset. For host controllers, this register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to the RESET bit in the USBCMD register before reprogramming this register. Bit Meaning 00 Idle [Default for combination host] 01 Reserved 10 Reserved 11 Host Controller [Default for host only controller]

10.13 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the Enhanced Host Controller (hardware). The data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of a Periodic Schedule, Periodic Frame List, Asynchronous Schedule, Isochronous Transaction Descriptors, Split-transaction Isochronous Transfer Descriptors, Queue Heads, and Queue Element Transfer Descriptors.

The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using Isochronous Transaction Descriptors. Isochronous split-transaction data streams are managed with Split-transaction Isochronous Transfer Descriptors. All Interrupt, Control, and Bulk data streams are managed via queue heads and Queue Element Transfer Descriptors[1]. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note: Software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

1. Split transaction Interrupt, Bulk and Control are also managed using queue heads and queue element transfer descriptors.

The data structures defined in this section are a mix of read-only and read/writable fields from the host controller's perspective. The host controller must preserve the read-only fields on all data structure writes.

10.13.1 Periodic Frame List

This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the PERIODICLISTBASE address register and the FRINDEX register. The periodic schedule is based on an array of pointers called the Periodic Frame List. The PERIODICLISTBASE address register is combined with the FRINDEX register to produce a memory pointer into the frame list. The Periodic Frame List implements a sliding window of work over time as shown in Figure 87. (This figure is identical to Figure 84; it is duplicated here for convenience.)

The periodic frame list is a 4K page aligned array of Frame List Link pointers. The length of the frame list is programmable. The programmability of the periodic frame list is exported to system software via the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, the length can be selected by system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into Frame List Size field in the USBCMD register.

Frame List Link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the Frame List.

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	
Fame List Link Pointer																											0		Typ			03-00H

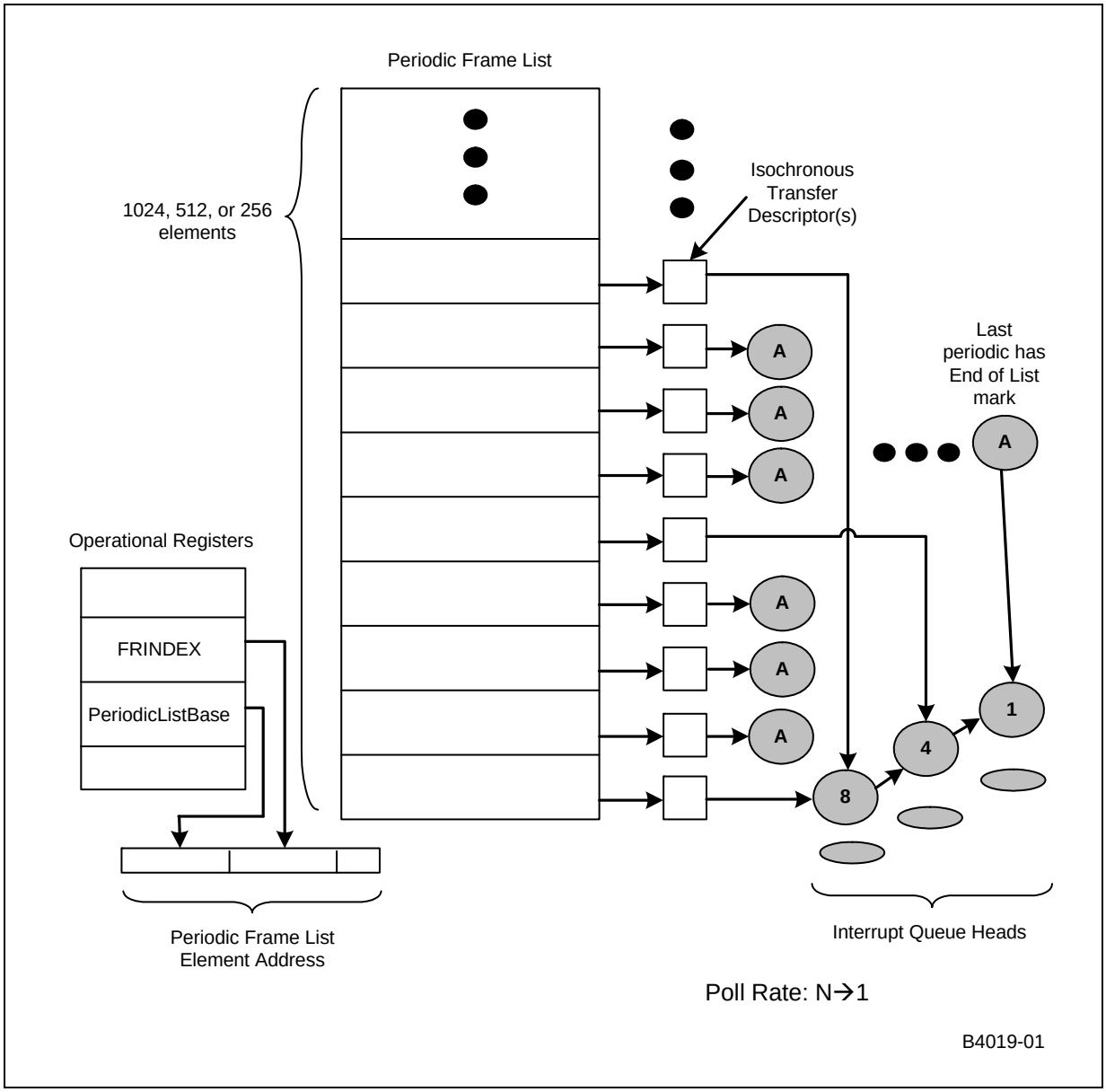
Frame List Link pointers always reference memory objects that are 32-byte aligned. The referenced object can be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least significant bits in a frame list pointer are used to key the host controller as to the type of object the pointer is referencing.

The least significant bit is the T-Bit (bit 0). When this bit is set to a one, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field is used to indicate the exact type of data structure being referenced by this pointer. The value encodings are:

Table 146. Typ Field Value Definitions

Value	Meaning
00b	Isochronous Transfer Descriptor
01b	Queue Head
10b	Split Transaction Isochronous Transfer Descriptor.
11b	Frame Span Traversal Node.

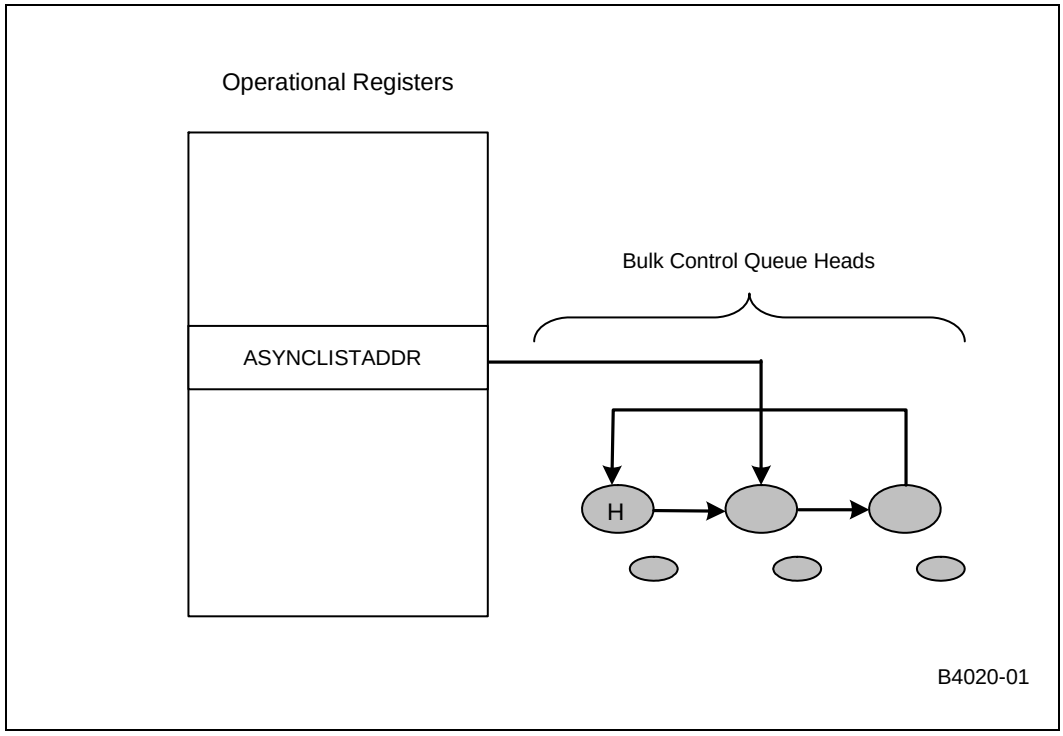
Figure 87. Periodic Schedule Organization



10.13.2 Asynchronous List Queue Head Pointer

The Asynchronous Transfer List (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers use this list only when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.

Figure 88. Asynchronous Schedule Organization

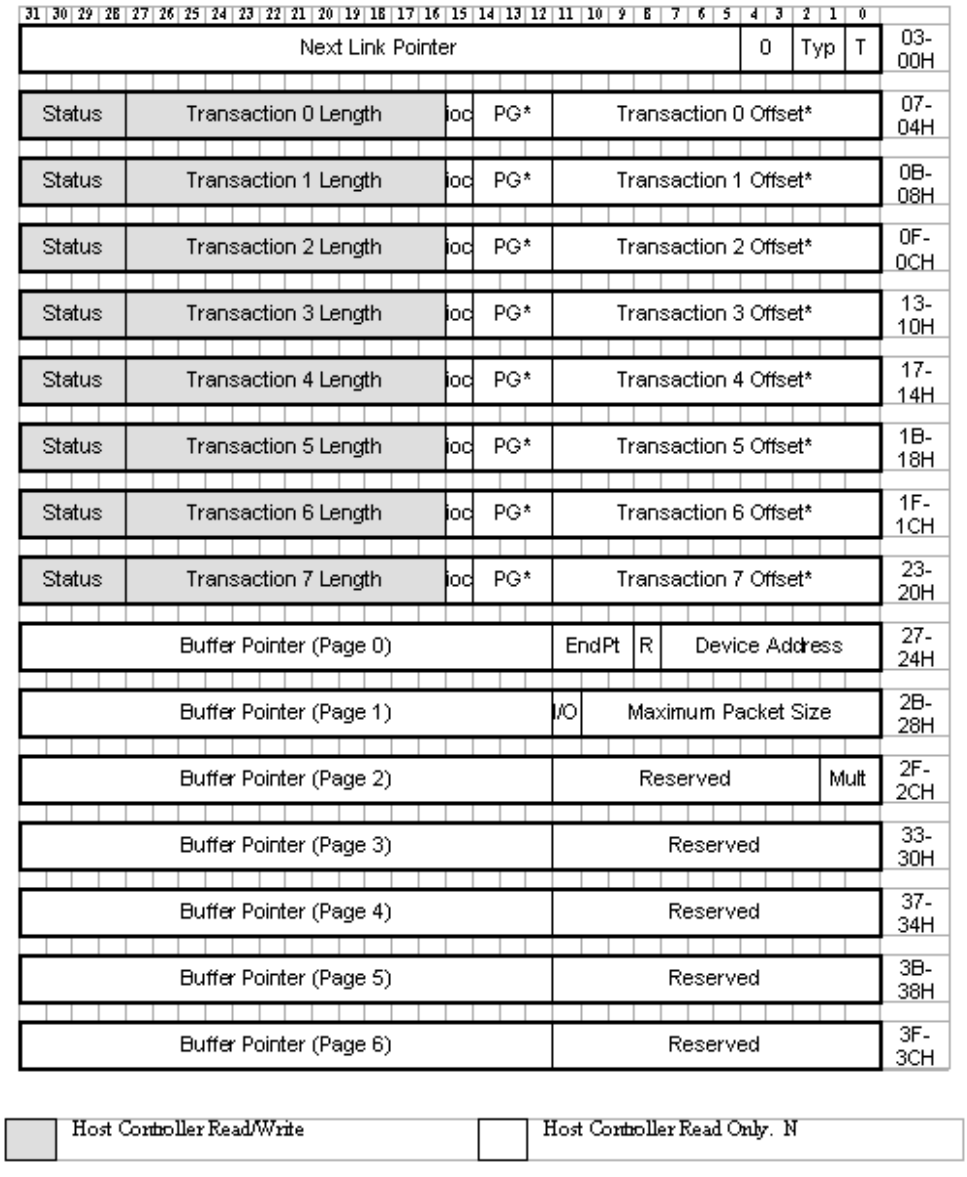


The Asynchronous list is a simple circular list of queue heads as shown in Figure 88. (This figure is identical to Figure 85; it is duplicated here for convenience.) The ASYNCLISTADDR register is a pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

10.13.3 Isochronous (High-Speed) Transfer Descriptor (iTD)

The format of an isochronous transfer descriptor is illustrated in Figure 89. This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.

Figure 89. Isochronous Transaction Descriptor (iTD)



Note: These fields can be modified by the host controller if the I/O field indicates an OUT.

10.13.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

Table 147. Next Schedule Element Pointer

Bit	Description
31:5	Link Pointer (LP). These bits correspond to memory address signals [31:5], respectively. This field points to another Isochronous Transaction Descriptor (iTD/siTD) or Queue Head (QH).
4:3	(Reserved). These bits are reserved and their value has no effect on operation. Software should initialize this field to zero.
2:1	QH/(s)iTD Select (Typ). This field indicates to the Host Controller whether the item referenced is an iTD, siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor) 11b FSTN (frame span traversal node)
0	Terminate (T). 0 = Link Pointer field is valid. 1 = Link Pointer field is not valid.

10.13.3.2 iTD Transaction Status and Control List

DWords 1 through 8 are eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction X Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description plus the endpoint information contained in the first three DWords of the Buffer Page Pointer list, to execute a transaction on the USB.

Table 148. iTD Transaction Status and Control (Sheet 1 of 2)

Bit	Description
31:28	Status. This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:
31	Active. Set to one by software to enable the execution of an isochronous transaction by the Host Controller. When the transaction associated with this descriptor is completed, the Host Controller sets this bit to zero indicating that a transaction for this element should not be executed when it is next encountered in the schedule.
30	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, no action is necessary.
29	Babble Detected. Set to one by the Host Controller during status update when” babble” is detected during the transaction generated by this descriptor.

Table 148. iTD Transaction Status and Control (Sheet 2 of 2)

Bit	Description
28	Transaction Error (XactErr). Set to one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, and so on.). This bit may only be set for isochronous IN transactions.
27:16	Transaction X Length. For an OUT, this field is the number of data bytes the host controller sends during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (for example, 0Æzero length data, 1Æone byte, 2Ætwo bytes, and so on.). The maximum value this field may contain is 0xC00 (3072).
15	Interrupt On Complete (IOC). If this bit is set to one, it specifies that when this transaction completes, the Host Controller should issue an interrupt at the next interrupt threshold.
14:12	Page Select (PG). These bits are set by software to indicate the buffer page pointers that the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11:0	Transaction X Offset. This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

10.13.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9-15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow for 3 (transactions) * 1,024 (maximum packet size) * 8 (transaction records) (24,576 bytes) to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4K-aligned page pointer, the least significant 12 bits in several of the page pointers are used for other purposes.

Table 149. iTD Buffer Pointer Page 0 (Plus)

Bit	Description
31:12	Buffer Pointer (Page 0). This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	(Reserved). Bit reserved for future use and should be initialized by software to zero.
6:0	Device Address. This field selects the specific device serving as the data source or sink.

Table 150. iTD Buffer Pointer Page 1 (Plus)

Bit	Description
31:12	Buffer Pointer (Page 1). This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11	Direction (I/O). 0 = OUT 1 = IN. This field encodes whether the high-speed transaction should use an IN or OUT PID.
10:0	Maximum Packet Size. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (for example, per micro-frame). This field is used with the Multi field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (400h). Any value larger yields undefined results.

Table 151. iTD Buffer Pointer Page 2 (Plus)

Bit	Description
31:12	Buffer Pointer. This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:2	(Reserved). This bit reserved for future use and should be zero.
1:0	Multi. This field is used to indicate to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame). The valid values are: Value Meaning 00b (Reserved). A zero in this field yields undefined results. 01b One transaction to be issued for this endpoint per micro-frame 10b Two transactions to be issued for this endpoint per micro-frame 11b Three transactions to be issued for this endpoint per micro- frame

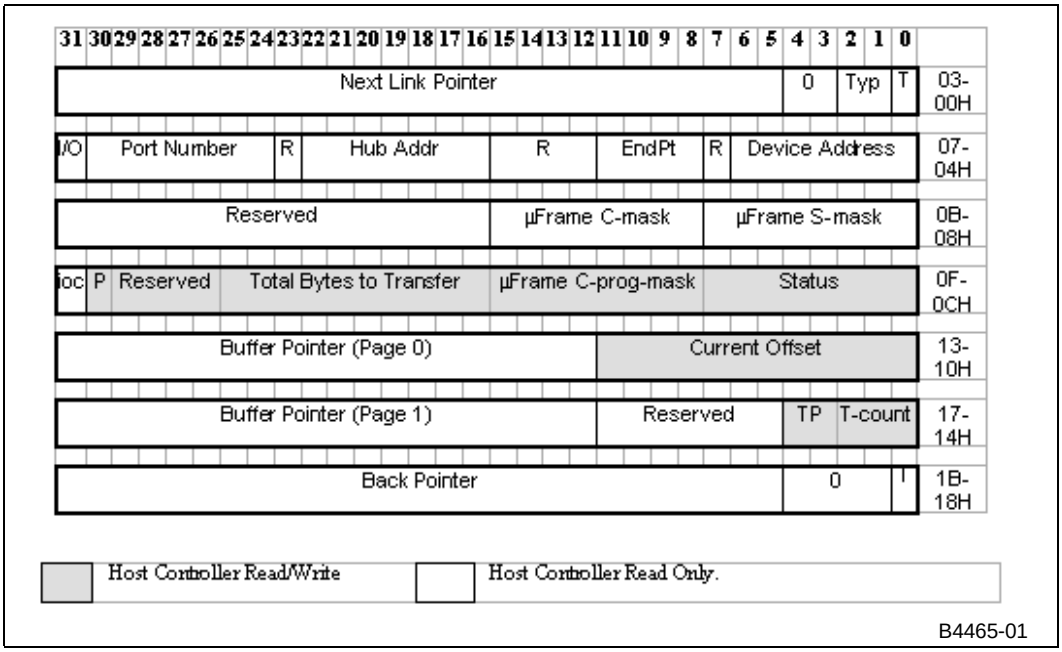
Table 152. iTD Buffer Pointer Page 3-6

Bit	Description
31:12	Buffer Pointer. This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:0	(Reserved). These bits reserved for future use and should be zero.

10.13.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

Figure 90. Split-transaction Isochronous Transaction Descriptor (siTD)



10.13.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.

Table 153. Next Link Pointer

Bit	Description
31:5	Next Link Pointer (LP). This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as zeros.
2:1	QH/(s)iTD Select (Typ). This field indicates to the Host Controller whether the item referenced is an iTD/siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor 11b FSTN (frame span traversal node)
0	Terminate (T). 0 = Link Pointer field is valid. 1 = Link Pointer field is not valid.

10.13.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

Table 154. Endpoint and Transaction Translator Characteristics

Bit	Description
31	Direction (I/O). 0 = OUT 1 = IN. This field encodes whether the full-speed transaction should be an IN or OUT.
30:24	Port Number. This field is the port number of the recipient Transaction Translator.
23	(Reserved). Bit reserved and should be zero.
22:16	Hub Address. This field holds the device address of the Companion Controllers' hub.
15:12	(Reserved). Field reserved and should be zero.
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	(Reserved). Bit is reserved for future use. It should be zero.
6:0	Device Address. This field selects the specific device serving as the data source or sink.

Table 155. Micro-Frame Schedule Control

Bit	Description
31:16	(Reserved). This field reserved for future use. It should be zero.
15:8	Split Completion Mask (mFrame C-Mask). This field (along with the Active and SplitX- state fields in the Status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the mFrame C-Mask field is a one, then this siTD is a candidate for transaction execution. There can be more than one bit in this mask set.
7:0	Split Start Mask (mFrame S-mask). This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the mFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

10.13.4.3 siTD Transfer State

DWords 3-6 are used to manage the state of the transfer.

Table 156. siTD Transfer Status and Control (Sheet 1 of 2)

Bit	Description
31	**Interrupt On Complete (ioc).** 0 = Do not interrupt when transaction is complete. 1 = Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed it asserts a hardware interrupt at the next interrupt threshold.
30	Page Select (P). Used to indicate the data page pointer that should be concatenated with the CurrentOffset field to construct a data buffer pointer (0 selects Page 0 pointer and 1 selects Page 1). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).
29:26	(Reserved). This field reserved for future use and should be zero.
25:16	Total Bytes To Transfer. This field is initialized by software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)

Table 156. siTD Transfer Status and Control (Sheet 2 of 2)

Bit	Description
15:8	mFrame Complete-split Progress Mask (C-prog-Mask). This field is used by the host controller to record the split-completes that has been executed.
7:0	Status. This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding
7	Active. Set to one by software to enable the execution of an isochronous split transaction by the Host Controller.
6	ERR. Set to a one by the Host Controller when an ERR response is received from the Companion Controller.
5	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the Host Controller transmits an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.
4	Babble Detected. Set to a one by the Host Controller during status update when" babble" is detected during the transaction generated by this descriptor.
3	Transaction Error (XactErr). Set to a one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, and so on.). This bit is only set for IN transactions.
2	Missed Micro-Frame. The host controller detected that a host-induced hold- off caused the host controller to miss a required complete-split transaction.
1	Split Transaction State (SplitXstate). The bit encodings are: Value Meaning 00b Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask. 01b Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask.
0	(Reserved). Bit reserved for future use and should be zero.

10.13.4.4 siTD Buffer Pointer List (Plus)

DWords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most significant 20 bits of each DWord in this section are the 4K (page) aligned buffer pointers. The least significant 12 bits of each DWord are used as additional transfer state.

Table 157. Buffer Page Pointer List (Plus) (Sheet 1 of 2)

Bit	Description
31:12	Buffer Pointer List. Bits [31:12] of DWords 4 and 5 are 4K paged aligned, physical memory addresses. These bits correspond to physical address bits [31:12] respectively. The lower 12 bits in each pointer are defined and used as specified below. The field P specifies the current active pointer

Table 157. Buffer Page Pointer List (Plus) (Sheet 2 of 2)

Bit	Description	
11:0	Page 0: Current Offset. The 12 least significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero). The least significant bits of Page 1 pointer is split into three sub-fields Page 1:	
	11:5	(Reserved).
	4:3	Transaction position (TP). This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: Value Meaning 00b All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01b Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10b Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 11b End. This is the last payload for a full-speed OUT transaction that is larger than 188 bytes.
	2:0	Transaction count (T-Count). Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

10.13.4.5 siTD Back Link Pointer

DWord 6 of a siTD is another schedule link pointer. This pointer is always zero, or references a siTD. This pointer cannot reference any other schedule data structure.

Table 158. siTD Back Link Pointer

Bit	Description
31:5	siTD Back Pointer. This field is a physical memory pointer to a siTD.
4:1	(Reserved). This field is reserved for future use. It should be zero.
0	Terminate (T). 0 = siTD Back Pointer field is valid. 1 = siTD Back Pointer field is not valid.

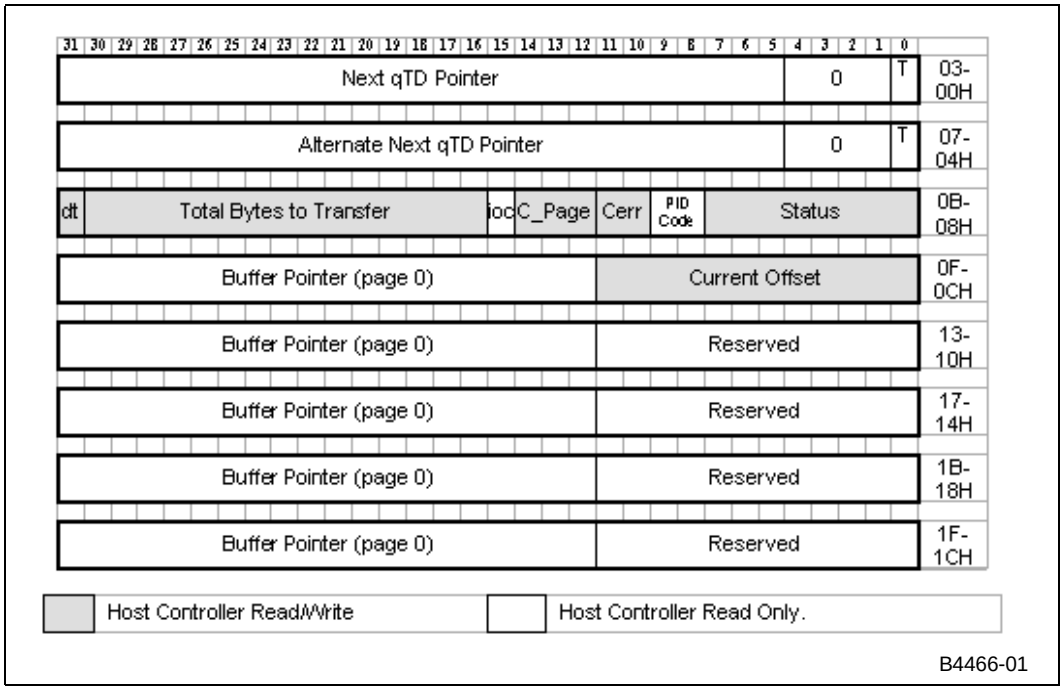
10.13.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5*4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

Figure 91. Queue Element Transfer Descriptor Block Diagram



Queue Element Transfer Descriptors must be aligned on 32-byte boundaries.

10.13.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 159. qTD Next Element Transfer Pointer (DWord 0)

Bit	Description
31:5	Next Transfer Element Pointer. This field contains the physical memory address of the next qTD to be processed. The field corresponds to memory address signals[31:5], respectively.
4:1	(Reserved). These bits are reserved and their value has no effect on operation.
0	Terminate (T). 0 = Pointer is valid (points to a valid Transfer Element Descriptor). 1 = pointer is invalid. This bit indicates to the Host Controller that there are no more valid entries in the queue.

10.13.5.2 Alternate Next qTD Pointer

The second DWord of a queue element transfer descriptor is used to support hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit the host controller always uses this pointer when the current qTD is retired due to short packet.

Table 160. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bit	Description
31:5	Alternate Next Transfer Element Pointer. This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively.
4:1	(Reserved). These bits are reserved and their value has no effect on operation.
0	Terminate (T). 0 = Pointer is valid (points to a valid Transfer Element Descriptor). 1 = pointer is invalid. This bit indicates to the Host Controller that there are no more valid entries in the queue.

10.13.5.3 qTD Token

The third DWord of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head).

Note: The field descriptions forward reference fields defined in the queue head. These forward references are preceded with a QH notation as required.

Table 161. qTD Token (DWord 2) (Sheet 1 of 3)

Bit	Description
31	Data Toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the Data Toggle Control bit in the queue head.
30:16	Total Bytes to Transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value software may store in this field is 5 * 4K (5000H). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that Total Bytes To Transfer be an even multiple of QHD.Maximum Packet Length. If software builds such a transfer descriptor for an OUT transfer, the last transaction is always less than QHD.Maximum Packet Length. Although it is possible to create a transfer up to 20K this assumes the 1st offset into the first page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K**. Therefore, the maximum recommended transfer is 16K(4000H).
15	Interrupt On Complete (IOC). If this bit is set to a one, it specifies that when this qTD is completed, the Host Controller should issue an interrupt at the next interrupt threshold.
14:12	Current Page (C_Page). This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0H to 4H. The host controller is not required to write this field back when the qTD is retired.

Table 161. qTD Token (DWord 2) (Sheet 2 of 3)

Bit	Description
11:10	Error Counter (CERR). This field is a 2-bit down counter that keeps track of the number of consecutive Errors detected when executing this qTD. If this field is programmed with a non-zero value during set-up, the Host Controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the Host Controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused CERR to decrement to zero. An interrupt is generated if the USB Error Interrupt Enable bit in the USBINTR register is set to a one. If HCD programs this field to zero during set-up, the Host Controller does not count errors for this qTD and there is no limit on the retries of this qTD. Note: Write-backs of intermediate execution state are to the queue head overlay area, not the qTD. Error — Decrement Counter Transaction Error — Yes Data Buffer Error — No3 Stalled — No1 Babble Detected — No1 No Error — No2
	1 — Detection of Babble or Stall automatically halts the queue head. Thus, count is not decremented
	2 — If the EPS field indicates a HS device or the queue head is in the Asynchronous Schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset CERR to extend the total number of errors for this transaction. For example, CERR should be reset with maximum value (3) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 00b. See "Split Transaction Execution State Machine for Interrupt" for CERR adjustment rules when the EPS field indicates a FS or LS device and the queue head is in the Periodic Schedule. See "Asynchronous — Do Complete Split" for CERR adjustment rules when the EPS field indicates a FS or LS device, the queue head is in the Asynchronous schedule and the PIDCode indicates a SETUP.
	3 — Data buffer errors are host problems. They don't count against the device's retries.
	Note: Software must not program CERR to a value of zero when the EPS field is programmed with a value indicating a Full- or Low-speed device. This combination could result in undefined behavior.
9:8	PID Code. This field is an encoding of the token, that should be used for transactions associated with this transfer descriptor. Encodings are: 00b OUT Token generates token (E1H) 01b IN Token generates token (69H) 10b SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt the queue head is non-zero.) transfer type, for example, mFrame S-mask field in 11b (Reserved)

Table 161. qTD Token (DWord 2) (Sheet 3 of 3)

Bit	Description	
7:0	Status. This field is used by the Host Controller to communicate individual command execution states back to HCD. This field contains the status of the last transaction performed on this qTD. The bit encodings are:	
	7	Active. Set to one by software to enable the execution of transactions by the Host Controller.
	6	Halted. Set to a one by the Host Controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set to a one, the Active bit is also set to zero.
	5	Data Buffer Error. Set to a one by the Host Controller during status update to indicate that the Host Controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the Host Controller forces a timeout condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	4	Babble Detected. Set to a one by the Host Controller during status update when" babble" is detected during the transaction. In addition to setting this bit, the Host Controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.
	3	Transaction Error (XactErr). Set to a one by the Host Controller during status update in the case where the host did not receive a valid response from the device (Timeout, CRC, Bad PID, and so on.). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	2	Missed Micro-Frame. This bit is ignored unless the QH.EPS field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
	1	Split Transaction State (SplitXstate). This bit is ignored by the host controller unless the QH.EPS field indicates a full- or low-speed endpoint. When a Full- or Low-speed device, the host controller uses this bit to track the state of the split- transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are: Value Meaning 0b Do Start Split. This value directs the host controller to issue a Start split transaction to the endpoint. 1b Do Complete Split. This value directs the host controller to issue a Complete split transaction to the endpoint.
	0	Ping State (P)/ERR. If the QH.EPS field indicates a High-speed device and the PID_Code indicates an OUT endpoint, then this is the state bit for the Ping protocol. The bit encodings are: Value Meaning 0b Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1b Do Ping. This value directs the host controller to issue a PING PID to the endpoint. If the QH.EPS field does not indicate a High-speed device, then this field is used as an error indicator bit. It is set to a one by the host controller whenever a periodic split-transaction receives an ERR handshake.

10.13.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor is an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

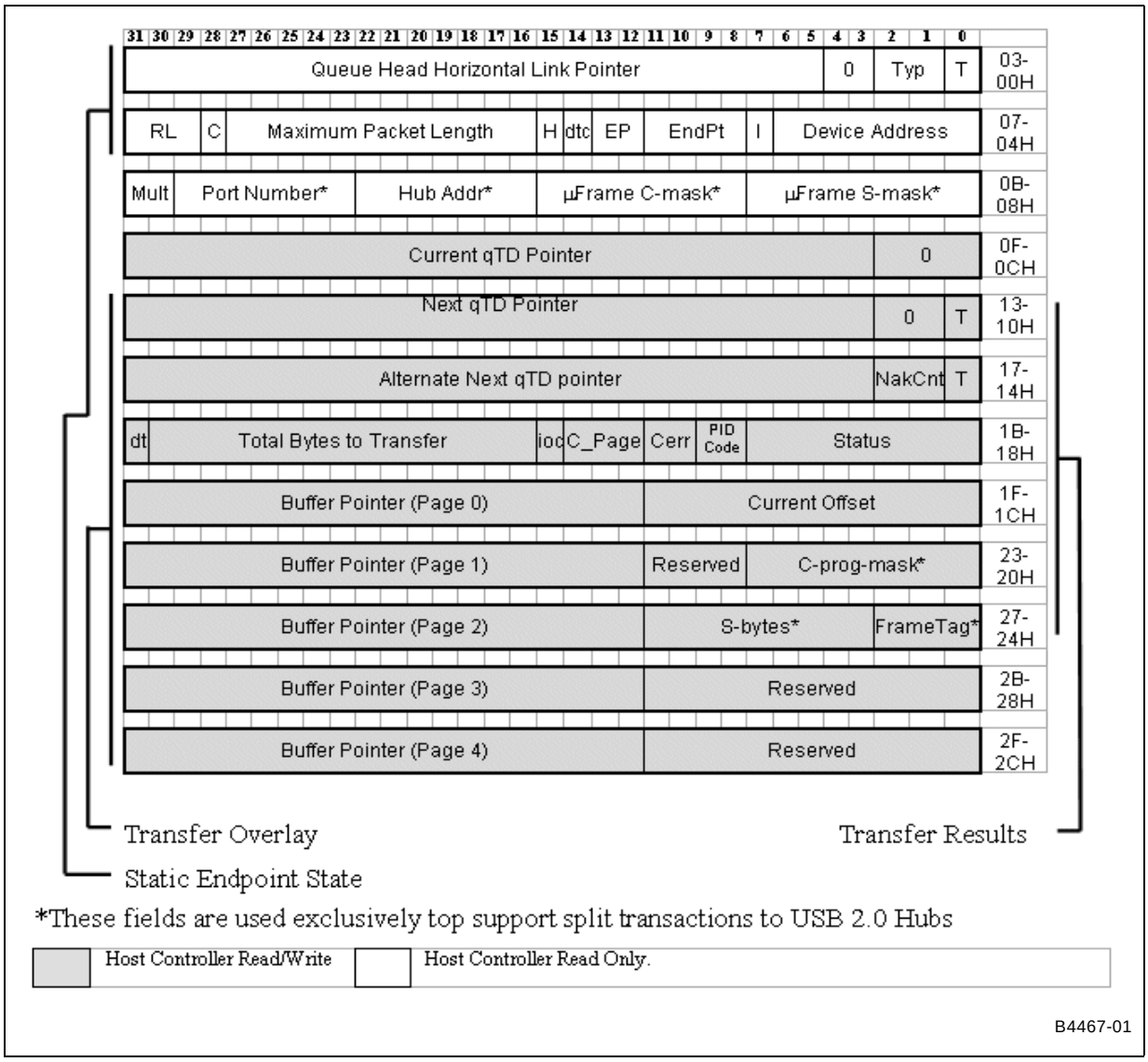
System software initializes Current Offset field to the starting offset into the current page, where current page is selected via the value in the C_Page field.

Table 162. qTD Buffer Pointer(s) (DWords 3-7)

Bit	Description
31:12	Buffer Pointer List. Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11:0	Current Offset (Reserved). This field is reserved in all pointers except the first one (for example, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. Software should ensure the Reserved fields are initialized to zeros.

10.13.6 Queue Head

Figure 92. Queue Head Structure Layout



10.13.6.1 Queue Head Horizontal Link Pointer

The first DWord of a Queue Head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, and the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

Table 163. Queue Head DWord 0

Bit	Description
31:5	Queue Head Horizontal Link Pointer (QHLP). This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as zeros.
2:1	QH/(s)iTD Select (Typ). This field indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor) 11b FSTN (frame span traversal node)
0	Terminate (T). 0 = Pointer is valid. 1 = Last QH (pointer is invalid). If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the Asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.

10.13.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specifies static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint Characteristics. These are the USB endpoint characteristics including addressing, maximum packet size, and endpoint speed.
- Endpoint Capabilities. These are adjustable parameters of the endpoint. They effect how the endpoint data stream is managed by the host controller.
- Split Transaction Characteristics. This data structure is used to manage full- and low-speed data streams for bulk, control, and interrupt via split transactions to USB 2.0 Hub Transaction Translator. There are additional fields used for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 164. Endpoint Characteristics: Queue Head DWord 1 (Sheet 1 of 2)

Bit	Description
31:28	Nak Count Reload (RL). This field contains a value, that is used by the host controller to reload the Nak Counter field.
27	Control Endpoint Flag (C). If the QH.EPS field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26:16	Maximum Packet Length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15	Head of Reclamation List Flag (H). This bit is set by System Software to mark a queue head as being the head of the reclamation list.
14	Data Toggle Control (DTC). This bit specifies where the host controller should get the initial data toggle on an overlay transition. 0b Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1b Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.

Table 164. Endpoint Characteristics: Queue Head DWord 1 (Sheet 2 of 2)

Bit	Description
13:12	Endpoint Speed (EPS). This is the speed of the associated endpoint. Bit combinations are: Value Meaning 00b Full-Speed (12Mbps) 01b Low-Speed (1.5Mbps) 10b High-Speed (480 Mbps) 11b (Reserved) This field must not be modified by the host controller.
11:8	Endpoint Number (Endpt). This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	Inactivate on Next Transaction (I). This bit is used by system software to request that the host controller set the Active bit to zero. See "Rebalancing the Periodic Schedule" for full operational details. This field is only valid when the queue head is in the Periodic Schedule and the EPS field indicates a Full or Low-speed endpoint. Setting this bit to a one when the queue head is in the Asynchronous Schedule or the EPS field indicates a high-speed device yields undefined results.
6:0	Device Address. This field selects the specific device serving as the data source or sink.

Table 165. Endpoint Capabilities: Queue Head DWord 2

Bit	Description
31:30	High-Bandwidth Pipe Multiplier (Mult). This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). The valid values are: Value Meaning 00b (Reserved). A zero in this field yields undefined results. 01b One transaction to be issued for this endpoint per micro-frame 10b Two transactions to be issued for this endpoint per micro-frame 11b Three transactions to be issued for this endpoint per micro-frame
29:23	Port Number. This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.
22:16	Hub Addr. This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.
15:8	Split Completion Mask (mFrame C-Mask). This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the mFrame C- Mask field is a one, then this queue head is a candidate for transaction execution. There can be more than one bit in this mask set.
7:0	Interrupt Schedule Mask (mFrame S-mask). This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the mFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

10.13.6.3 Transfer Overlay

The nine DWords in this area represent a transaction working space for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the Queue Head Horizontal Link Pointer to the next queue head. The host controller never follows the Next Transfer Queue Element or Alternate Queue Element pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The DWord3 of a Queue Head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 166. Current qTD Link Pointer

Bit	Description
31:5	Current Element Transaction Descriptor Link Pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively.
4:0	(Reserved) (R). These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The DWords 4-11 of a queue head are the transaction overlay area. This area has the same base structure as a Queue Element Transfer Descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

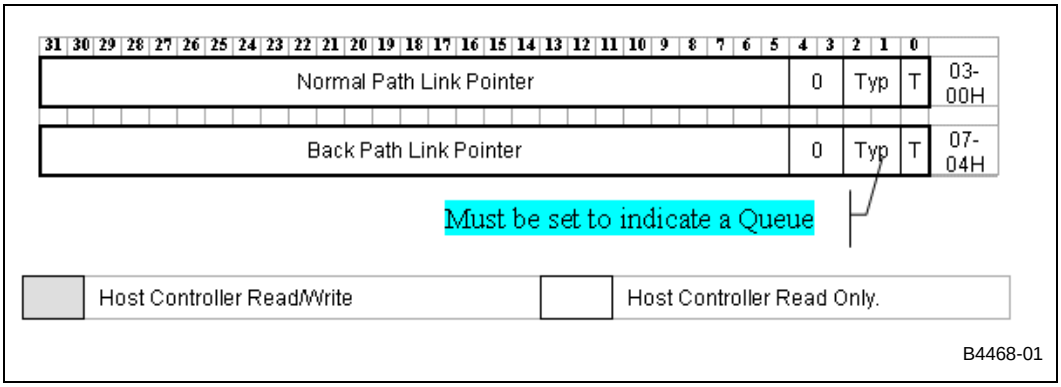
Table 167. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8 and 9)

DWord	Bit	Description
5	4:1	Nak Counter (NakCnt)mRW. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay.
6	31	Data Toggle. The Data Toggle Control controls whether the host controller preserves this bit when an overlay operation is performed.
6	15	Interrupt On Complete (IOC). The IOC control bit is always inherited from the source qTD when the overlay operation is performed.
6	11:10	Error Counter (C_ERR). This two-bit field is copied from the qTD during the overlay and written back during queue advancement.
6	0	Ping State (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	7:0	Split-transaction Complete-split Progress (C-prog-mask). This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	4:0	Split-transaction Frame Tag (Frame Tag). This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	11:5	S-bytes. Software must ensure that the S-bytes field in a qTD is zero before activating the qTD. This field is used to keep track of the number of bytes sent or received during an IN or OUT split transaction.

10.13.7 Periodic Frame Span Traversal Node (FSTN)

This data structure is to be used only for managing Full- and Low-speed transactions that span a Host-frame boundary. See "Host Controller Operational Model for FSTNs" for full operational details. Software must not use an FSTN in the Asynchronous Schedule. An FSTN in the Asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation below 0096h. FSTNs are not defined for implementations before 0.96 and their use yields undefined results.

Figure 93. Frame Span Traversal Node Structure Layout



10.13.7.1 FSTN Normal Path Pointer

The first DWord of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

Table 168. FSTN Normal Path Pointer Signals

Bit	Description
31:5	Normal Path Link Pointer (NPLP). This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as 0s.
2:1	QH/(s)iTD/FSTN Select (Typ). This field indicates to the Host Controller whether the item referenced is an iTD/siTD, a QH or a FSTN. This allows the Host Controller to perform the proper type of processing on the item after it is fetched. Value encodings are: Value Meaning 00b iTD (isochronous transfer descriptor) 01b QH (queue head) 10b siTD (split transaction isochronous transfer descriptor) 11b FSTN (frame span traversal node)
0	Terminate (T). 0 = Link Pointer is valid. 1 = Link Pointer field is not valid.

10.13.7.2 FSTN Back Path Link Pointer

The second DWord of an FTSN node contains a link pointer to a queue head. If the T-bit in this pointer is a zero, then this FSTN is a Save-Place indicator. Its Typ field must be set by software to indicate the target data structure is a queue head. If the T-bit in this pointer is set to a one, then this FSTN is the Restore indicator. When the T-bit is a one, the host controller ignores the Typ field.

Table 169. FSTN Back Path Link Pointer Signals

Bit	Description
31:5	Back Path Link Pointer (BPLP). This field contains the address of a Queue Head. This field corresponds to memory address signals [31:5], respectively.
4:3	(Reserved). These bits must be written as 0s.
2:1	Typ. Software must ensure this field is set to indicate the target data structure is a Queue Head. Any other value in this field yields undefined results.
0	Terminate (T). 0 = Link Pointer is valid that is, the host controller may use bits [31:5] (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address. This value also indicates that this FSTN is a Save-Place indicator. 1 = Link Pointer field is not valid that is, the host controller must not use bits [31:5] (in combination with the CTRLDSSEGMENT register if applicable as a valid memory address). This value also indicates that this FSTN is a Restore indicator.

10.14 Host Operational Model

The general operational model is for the enhanced interface host controller hardware and enhanced interface host controller driver (generally referred to as system software). Each significant operational feature of the EHCI host controller is discussed in a separate section. Each section presents the operational model requirements for the host controller hardware. Where appropriate, recommended system software operational models for features are also presented.

10.14.1 Host Controller Initialization

After initial power-on or HCReset (hardware or via HCReset bit in the USBCMD register), all of the operational registers is at their default values, as illustrated in Table 170. After a hardware reset, only the operational registers not contained in the Auxiliary power well is at their default values.

Table 170. Default Values of Operational Register Space

Operational Register	Default Value (after Reset)
USBCMD	0x00080000 (0x00080B00 if Asynchronous Schedule Park Capability is a one)
USBSTS	0x00001000
USBINTR	0x00000000
FRINDEX	0x00000000
CTRLDSSEGMENT	0x00000000
PERIODICLISTBASE	Undefined
ASYNCLISTADDR	Undefined
PORTSC	0x00002000 (w/PPC set to one); 0x00003000 (w/PPC set to a zero)

To initialize the host controller, software should perform the following steps:

- Program the CTRLDSSEGMENT register with 4-Gigabyte segment where all of the interface data structures are allocated.
- Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
- Write the base address of the Periodic Frame List to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the Periodic Frame List should have their T-Bits set to a one.

- Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable) and turn the host controller ON via setting the Run/Stop bit.

At this point, the host controller is up and running and the port registers begins reporting device connects, and so on. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled High-speed ports, but the schedules have not yet been enabled. The EHCI Host controller does not transmit SOFs to enabled Full- or Low-speed ports.

To communicate with devices via the asynchronous schedule, system software must write the ASYNDLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to the Asynchronous Schedule Enable bit in the USBCMD register. To communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to the Periodic Schedule Enable bit in the USBCMD register.

Note: The schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

10.14.2 Port Power

The Port Power Control (PPC) bit in the HCSPARAMS register indicates whether the USB 2.0 host controller has port power control (See Section 10.11.3, "HCSPARAMS – EHCI Compliant with Extensions"). When this bit is a zero, then the host controller does not support software control of port power switches. When in this configuration, the port power is always available and the companion host controllers must implement functionality consistent with port power always on. When the PPC bit is a one, then the host controller implementation includes port power switches. Each available switch has an output enable, that is referred to in this discussion as PortPowerOutputEnable (PPE). PPE is controlled based on the state of the combination bits PPC bit and individual Port Power (PP) bits. Table 171 illustrates the summary behavioral model.

Table 171. Port Power Enable Control Rules (Sheet 1 of 2)

CHC[2] (PP)	EHC3 (PP)	Owner	PPE[1]	Description
0	X	CHC	0	When the EHCI controller has not been configured, the port is owned by the companion host controller. When the companion HC's port power select is off, then the port power is off.
1	X	CHC	1	Similar to previous entry. When the companion HC's port power select is on, then the port power is on.
0	0	CHC	0	Port owner has port power turned off, the power to port is off.
0	0	EHC	0	Port owner has port power turned off, the power to port is off.
0	1	EHC	1	Port owner has port power on, so power to port is on.
0	1	CHC	1	If either HC has port power turned on, the power to the port is on.
1	0	EHC	1	If either HC has port power turned on, the power to the port is on.

Notes:
1. PPE (Port Power Enable). This bit actually turns on the port power switch (if one exists).
2. CHC (Companion Host Controller).
3. EHC (EHCI Host Controller).

Table 171. Port Power Enable Control Rules (Sheet 2 of 2)

CHC[2] (PP)	EHC3 (PP)	Owner	PPE[1]	Description
1	0	CHC	1	Port owner has port power on, so power to port is on.
1	1	CHC	1	Port owner has port power on, so power to port is on.
1	1	EHC	1	Port owner has port power on, so power to port is on.

Notes:
1. PPE (Port Power Enable). This bit actually turns on the port power switch (if one exists).
2. CHC (Companion Host Controller).
3. EHC (EHCI Host Controller).

10.14.2.1 Port Reporting Over-Current

Host controllers are by definition power providers on USB. Whether the ports are considered high or low-powered is a platform implementation issue. Each EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

The over current detection and limiting logic usually resides outside the host controller logic. This logic can be associated with one or more ports. When this logic detects an over-current condition it is made available to both the companion and EHCI ports. The effect of an over-current status on a companion host controller port is beyond the scope of this document. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- Over-current Active bits are set to a one. When the over-current condition goes away, the Over-current Active bit transitions from a one to a zero.
- Over-current Change bits are set to a one. On every transition of the Over-current Active bit the host controller sets the Over-current Change bit to a one. Software sets the Over-current Change bit to a zero by writing a one to this bit.
- Port Enabled/Disabled bit is set to a zero. When this change bit gets set to a one, then the Port Change Detect bit in the USBSTS register is set to a one.
- Port Power (PP) bits may optionally be set to a zero. There is no requirement in USB that a power provider shut off power in an over current condition. It is sufficient to limit the current and leave power applied. When the Over-current Change bit transitions from a zero to a one, the host controller also sets the Port Change Detect bit in the USBSTS register to a one. In addition, if the Port Change Interrupt Enable bit in the USBINTR register is a one, then the host controller issues an interrupt to the system. Refer to Table 172 for summary behavior for over-current detection when the host controller is halted (suspended from a device component point of view).

10.14.3 Suspend/Resume

The EHCI host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely via software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software initiated resumes are called Resume Events/Actions. Bus-initiated resume events are called wake-up events. The classes of wake-up events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, EHCI controllers must always respond to explicit device resume signaling and wake up the system (if necessary).

- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off by using the per-port control bits in the PORTSC registers.

Selective suspend is a feature supported by every PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the entire bus, it should selectively suspend all enabled ports, then shut off the host controller by setting the Run/Stop bit in the USBCMD register to a zero. The EHCI module can then be placed into a lower device state.

When a wake event occurs the system resumes operation and system software eventually sets the Run/Stop bit to a one and resume the suspended ports. Software must not set the Run/Stop bit to a one until it is confirmed that the clock to the host controller is stable. This is usually confirmed in a system implementation in that all of the clocks in the system are stable before the CPU is restarted. So, by definition, if software is running, clocks in the system are stable and the Run/Stop bit in the USBCMD register can be set to a one.

10.14.3.1 Port Suspend/Resume

System software places individual ports into suspend mode by writing a one into the appropriate PORTSC Suspend bit. Software must only set the Suspend bit when the port is in the enabled state (Port Enabled bit is a one) and the EHCI is the port owner (Port Owner bit is a zero).

The host controller may evaluate the Suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there can be several micro-frames of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

System software can initiate a resume on a selectively suspended port by writing a one to the Force Port Resume bit. Software should not attempt to resume a port unless the port reports that it is in the suspended state (see Section 10.12.11, “PORTSCx”). If system software sets Force Port Resume bit to a one when the port is not in the suspended state, the resulting behavior is undefined. To assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates that it is suspended (Suspend bit is a one) before initiating a port resume via the Force Port Resume bit. When Force Port Resume bit is a one, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then sets the Force Port Resume bit to a zero. When the host controller receives the write to transition Force Port Resume to zero, it completes the resume sequence as defined in the USB specification, and sets both the Force Port Resume and Suspend bits to zero. Software-initiated port resumes do not affect the Port Change Detect bit in the USBSTS register nor do they cause an interrupt if the Port Change Interrupt Enable bit in the USBINTR register is a one. An external USB event may also initiate a resume. The wake events are defined above. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 msec. The port's Force Port Resume bit is set to a one and the Port Change Detect bit in the USBSTS register is set to a one. If the Port Change Interrupt Enable bit in the USBINTR register is a one the host controller issues an hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 ms), then terminates the resume sequence by writing zero to the Force Port Resume bit in the port. The host controller receives the write of zero to Force Port Resume, terminates the resume sequence and sets Force Port Resume and Suspend port bits to zero. Software can determine that the port is enabled (not suspended) by

sampling the PORTSC register and observing that the Suspend and Force Port Resume bits are zero. Software must ensure that the host controller is running (that is, HCHalted bit in the USBSTS register is a zero), before terminating a resume by writing a zero to a port's Force Port Resume bit. If HCHalted is a one when Force Port Resume is set to a zero, then SOFs does not occur down the enabled port and the device returns to suspend mode in a maximum of 10 milliseconds.

Table 172 summarizes the wake-up events. Whenever a resume event is detected, the Port Change Detect bit in the USBSTS register is set to a one. If the Port Change Interrupt Enable bit is a one in the USBINTR register, the host controller also generates an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the Port Change Detect status bit in the USBSTS register.

Table 172. Behavior During Wake-Up Events

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	Not D0
Port disabled, resume K-State received	No Effect	N/A	N/A
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. Force Port Resume status bit in PORTSC register is set to a one. Port Change Detect bit in USBSTS register set to a one.	1, 2	2
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is a one. A disconnect is detected.	Depending in the initial port state, the PORTSC Connect and Enable status bits are set to zero, and the Connect Change status bit is set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 2	2
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is a zero. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect and Enable status bits are set to zero, and the Connect Change status bit is set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3
Port is not connected and the port's WKCNNT_E bit is a one. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 2	2
Port is not connected and the port's WKCNNT_E bit is a zero. A connect is detected.	PORTSC Connect Status and Connect Status Change bits are set to a one. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set to a one. If Port Enable/Disable bit is a one, it is set to a zero. Port Change Detect bit in the USBSTS register is set to a one	1, 2	2
Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs.	PORTSC Over-current Active, Over-current Change bits are set to a one. If Port Enable/Disable bit is a one, it is set to a zero. Port Change Detect bit in the USBSTS register is set to a one.	1, 3	3

Notes:
1. Hardware interrupt issued if Port Change Interrupt Enable bit in the USBINTR register is a one.
2. PME# asserted if enabled (Note: PME Status must always be set to a one).
3. PME# not asserted.

10.14.4 Schedule Traversal Rules

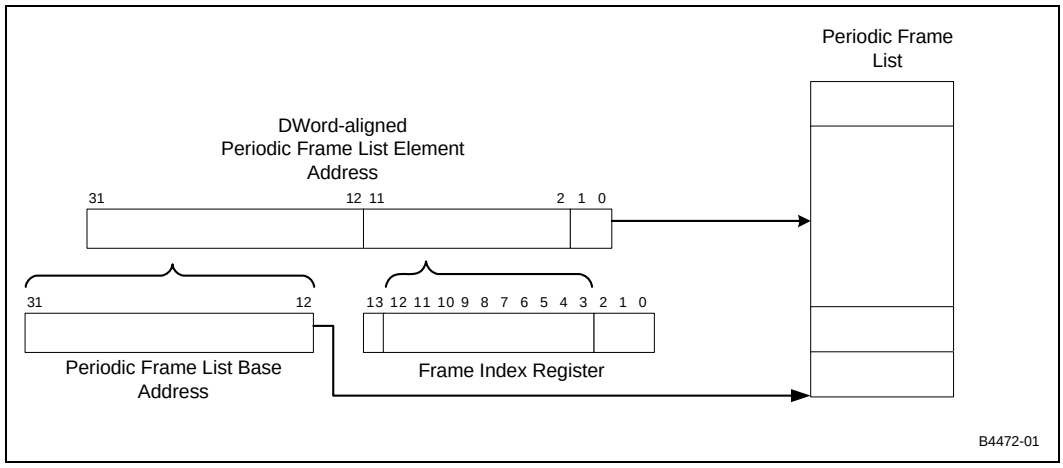
The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by USB, minimize memory traffic and hardware / software complexity.

System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the PERIODICLISTBASE register (see Section 10.12.6, “PERIODICLISTBASE”). The PERIODICLISTBASE register is the physical memory base address of the periodic frame

list. The periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in Section 10.6.2, “Host Data Structure”. In each micro-frame, if the periodic schedule is enabled (see Section 10.14.6, “Periodic Schedule”) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It only executes from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 94). It fetches the element and begins traversing the graph of linked schedule data structures.

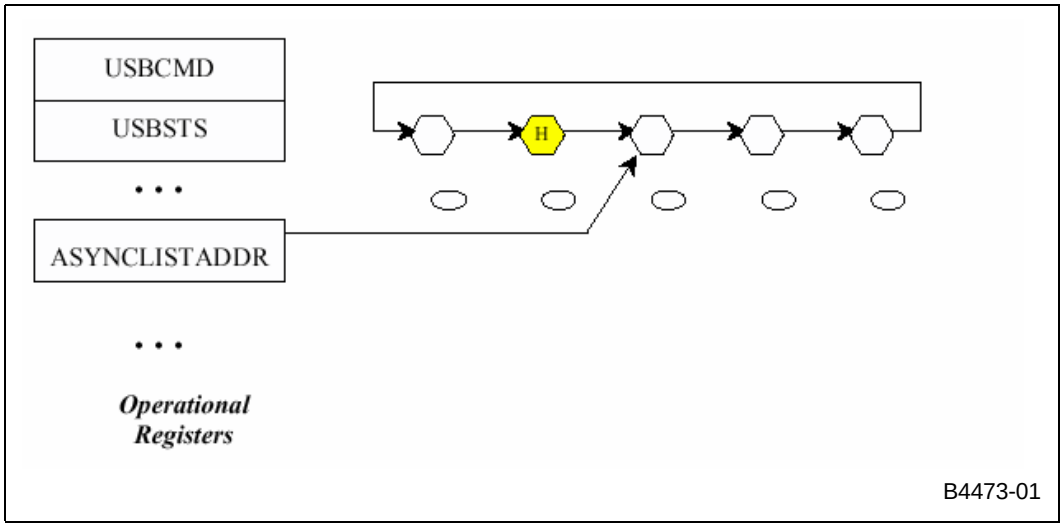
The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set to a one. When the host controller encounters a T-Bit set to a one during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

Figure 94. Derivation of Pointer into Frame List Array



When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, see Figure 95.

Figure 95. General Format of Asynchronous Schedule List



The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule. See Section 10.14.8, "Asynchronous Schedule" for complete operational details.

10.14.4.1 Example: Preserving Micro-Frame Integrity

One of the requirements of a USB host controller is to maintain Frame Integrity. This means that the HC must preserve the micro-frame boundaries. For example: SOF packets must be generated on time (within the specified allowable jitter), and High-speed EOF1, 2 thresholds must be enforced. The end of micro-frame timing points EOF1 and EOF2 are clearly defined in the USB Specification Revision 2.0. One implication of this responsibility is that the HC must ensure that it does not start transactions that is not completed before the end of the micro-frame. Exactly, no transactions should be started by the host controller, and cannot be completed in their entirety before the EOF1 point. To enforce this rule, the host controller must check each transaction before it starts to ensure that it completes before the end of the micro-frame.The transaction data payload, plus bit stuffing, plus transaction overhead must be involved in this check. It is possible to be extremely accurate on how much time the next transaction takes.

Take OUTs for an example. The host controller must fetch all of the OUT data from memory to send it onto the USB bus. A host controller implementation could pre-fetch all of the OUT data, and pre-compute the actual number of bits in the token and data packets. In addition, the system knows the depth of the target endpoint, so it could closely estimate turnaround time for handshake. In addition, the host controller knows the size of a handshake packet. Pre-computing effects of bit stuffing and summing up the other overhead numbers could allow the host controller to know exactly whether there is enough bus time, before EOF1 to complete the OUT transaction. To accomplish this particular approach takes an inordinate amount of time and hardware complexity.

The alternative is to make a reasonable guess whether the next transaction can be started. An example approximation algorithm is described below. This example algorithm relies on the EHCI policy that periodic transactions are scheduled first in the micro-frame. It is a reasonable assumption that software never over-commits the micro-frame to periodic transactions greater than the specification allowable 80%. In

the available remaining 20% bandwidth, the host controller has some ability (in this example) to decide whether to execute a transaction. The result of this algorithm is that sometimes, under some circumstances a transaction is not executed that could have been executed. But, under all circumstances, a transaction is never started unless there is enough time in the frame to complete the transaction.

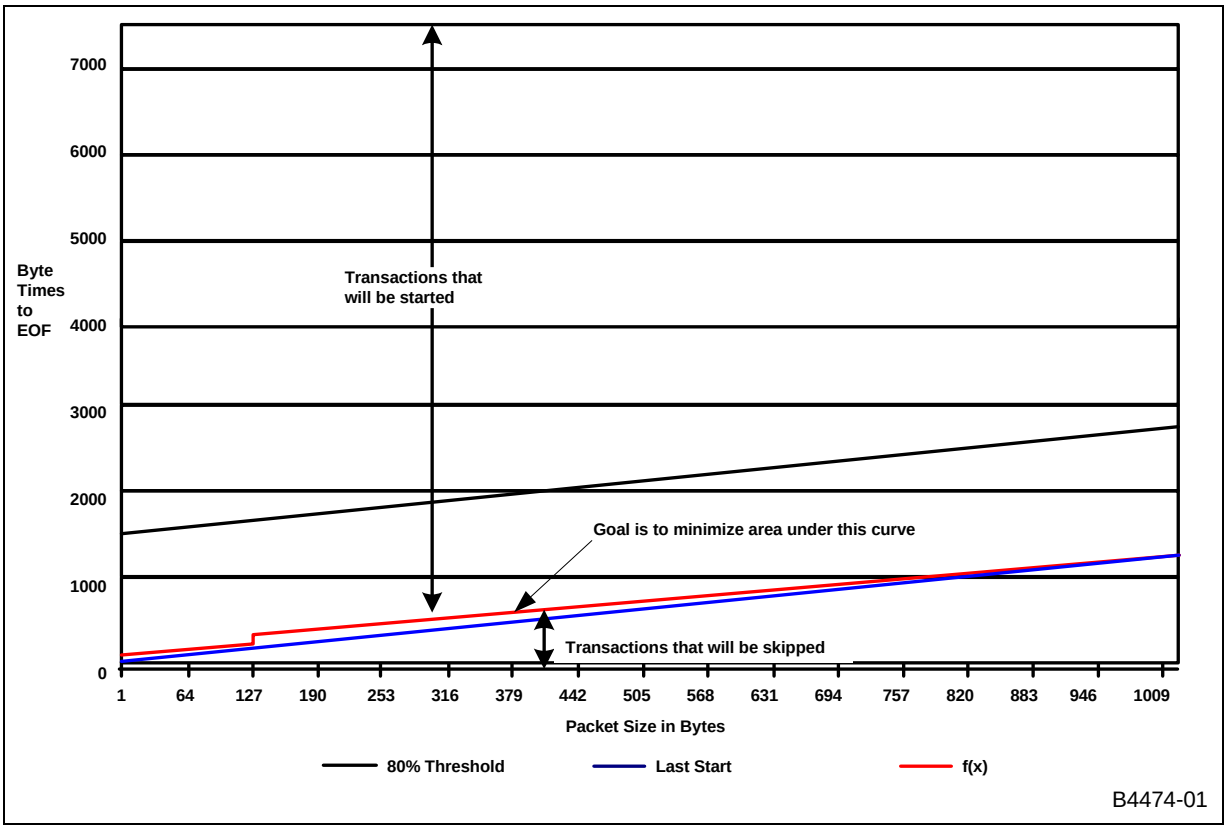
10.14.4.1.1 Transaction Fit: A Best-Fit Approximation Algorithm

A curve is calculated and represents the latest start time for every packet size, when software schedules the start of a periodic transaction. This curve is the 80% bandwidth curve. Another curve is calculated that is the absolute, latest permitted start time for every packet size. This curve represents the absolute latest time, that a transaction of each packet size can be started and completed, in the micro-frame. A plot of these two curves is illustrated in Figure 96. The plot Y-axis represents the number of byte-times left in a frame.

The space between the 80% and the Last Start plots is bandwidth reclamation area. In this algorithm the host controller may skip transactions during this time if it is prudent.

The Best-Fit Approximation method plots a function (f(x)) between the 80% and Last Start curves. The function f(x) adds a constant to every transaction's maximum packet size and the result compared with the number of bytes left in the frame. The constant represents an approximation of the effects of bit stuffing and protocol overhead. The host controller starts transactions whose results land above the function curve. The host controller does not start transactions whose results land below the function curve.

Figure 96. Best Fit Approximation



The LastStart line is calculated in this example to assume the absolute worst-case bus overhead per transaction. The particular transaction used was a start-split, zero-length OUT transaction with a handshake. Summaries of the component parts are listed in Table 173. The component times were derived from the protocol timings defined in the USB Specification Revision 2.0.

Table 173. Example Worst-Case Transaction Timing Components

Component	Bit Time	Byte Time	Explanation
Split Token	76	9.5	Split token as defined in USB core specification. Includes synchronization, token, eop, and so on.
Host 2 Host IPG	88	11	Number of bit times required between consecutive host packets.
Token	67	8.375	Token as defined in USB core specification. Includes synchronization, token, eop, and so on.
Host 2 Host IPG	88	11	Same as above.
Data Packet (0 data bytes)	66.7	8.34	Zero-length data packet. Includes synchronization, PID, crc16, eop, and so on.
Turnaround time	721	90.125	Time for packet initiator (Host) to see the beginning of a response to a transmitted packet.
Handshake packet	48	6	Handshake packet as defined in USB core specification. Includes synchronization, PID, eop, and so on.
		144	**Total**

The exact details of the function (f(x)) are up to the particular implementation. But, it should be obvious that the goal is to minimize the area under the curve between the approximation function and the Last Start curve, without dipping below the LastStart line, and at the same time keeping the check as simple as possible for hardware implementation. The f(x) in Figure 96 is constructed using the following pseudo-code test on each transaction size data point. This algorithm assumes that the host controller keeps track of the remaining bits in the frame.

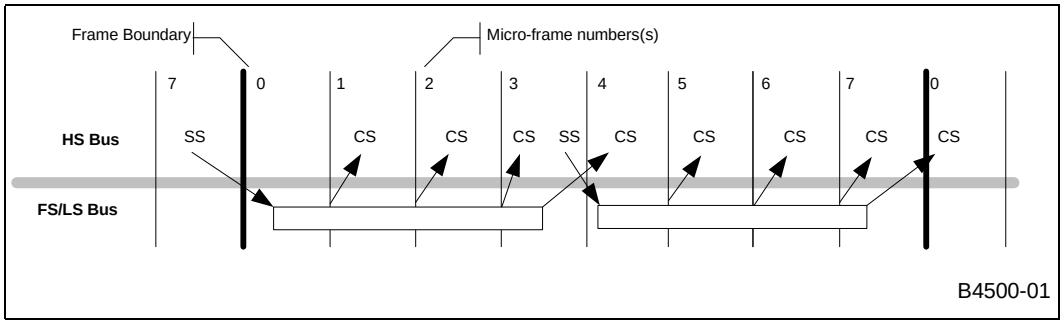
```
Algorithm CheckTransactionWillFit (MaximumPacketSize, HC_BytesLeftInFrame)

Begin

                    Local Temp = MaximumPacketSize + 192

                    Local rvalue = TRUE


                    If MaximumPacketSize >= 128 then

                        Temp += 128

                    End If


                    If Temp > HC_BytesLeftInFrame then

                        Rvalue = FALSE

                    End If

                    Return rvalue

End
```

This algorithm takes two inputs, the current maximum packet size of the transaction and a hardware counter of the number of bytes left in the current micro-frame. It unconditionally adds a simple constant of 192 to the maximum packet size to account for a first-order effect of transaction overhead and bit stuffing. If the transaction size is greater than or equal to 128 bytes, then an additional constant of 128 is added to the running sum to account for the additional worst-case bit stuffing of payloads larger than 128. An inflection point is inserted at 128 because the f(x) plot is getting close to the LastStart line.

10.14.5 Periodic Schedule Frame Boundaries versus Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(s) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 97). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

Figure 97. Frame Boundary Relationship between HS bus and FS/LS Bus

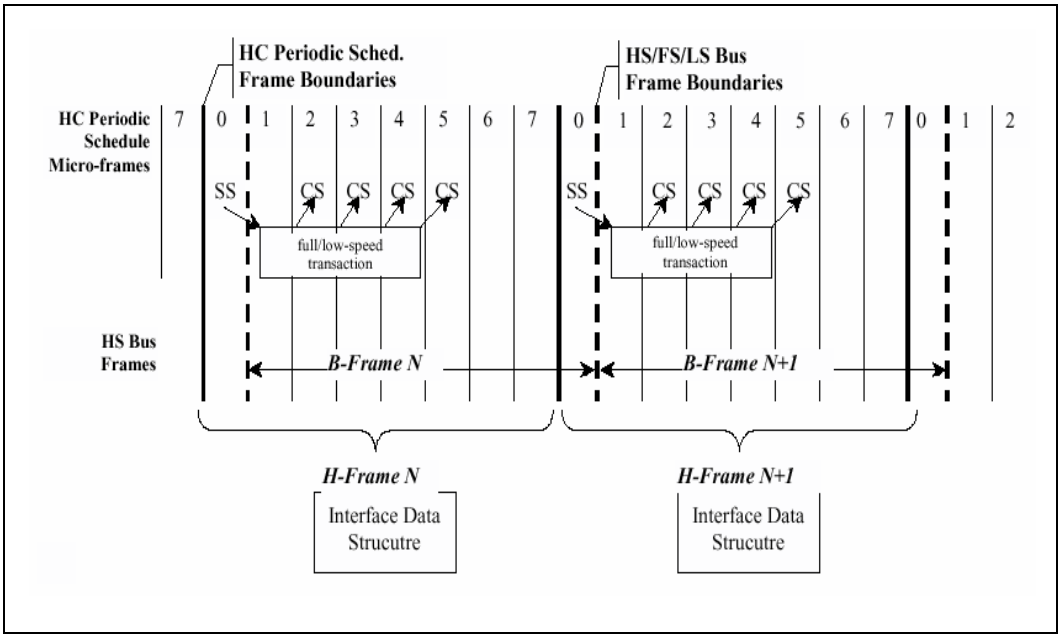


The simple projection, as Figure 97 illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. To reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX) documented in Section 10.12.4, “FRINDEX” and initially illustrated in Section 10.14.4, “Schedule Traversal Rules”. Bits FRINDEX[2:0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13:3]. Both FRINDEX[13:3] and the SOF value are incremented based on FRINDEX[2:0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 98. This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface. The reasons for selecting this phase-shift are beyond the scope of this specification.

Figure 98 illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the 1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-ms boundaries is called B-Frames.

Figure 98. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries



H-Frame boundaries for the host controller correspond to increments of FRINDEX[13:3]. Micro-frame numbers for the H-Frame are tracked by FRINDEX[2:0]. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus sees eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 98. The host controller's periodic schedule is naturally aligned to H-Frames.

Software schedules transactions for full and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in Section 10.12.4, "FRINDEX", the SOF Value can be implemented as a shadow register (in this example, called SOFV), that lags the FRINDEX register bits [13:3] by one micro-frame count. Table 174 illustrates the required relationship between the value of FRINDEX and the value of SOFV. This lag behavior can be accomplished by incrementing FRINDEX[13:3] based on carry-out on the 7 to 0 increment of FRINDEX[2:0] and incrementing SOFV based on the transition of 0 to 1 of FRINDEX[2:0].

Software is allowed to write to FRINDEX. Section 10.12.4, "FRINDEX" provides the requirements that software should adhere when writing a new value in FRINDEX.

Table 174. Operation of FRINDEX and SOFV (SOF Value Register) (Sheet 1 of 2)

	Current			Next	
FRINDEX[F]	SOFV	FRINDEX[µF]	FRINDEX[F]	SOFV	FRINDEX[µF]
N	N	111b	N+1	N	000b
N+1	N	000b	N+1	N+1	001b
N+1	N+1	001b	N+1	N+1	010b
N+1	N+1	010b	N+1	N+1	011b
Note: Where [F] = [13:3]; [mF] = [2:0]					

Table 174. Operation of FRINDEX and SOFV (SOF Value Register) (Sheet 2 of 2)

	Current			Next	
N+1	N+1	011b	N+1	N+1	100b
N+1	N+1	100b	N+1	N+1	101b
N+1	N+1	101b	N+1	N+1	110b
N+1	N+1	110b	N+1	N+1	111b
Note: Where [F] = [13:3]; [mF] = [2:0]					

10.14.6 Periodic Schedule

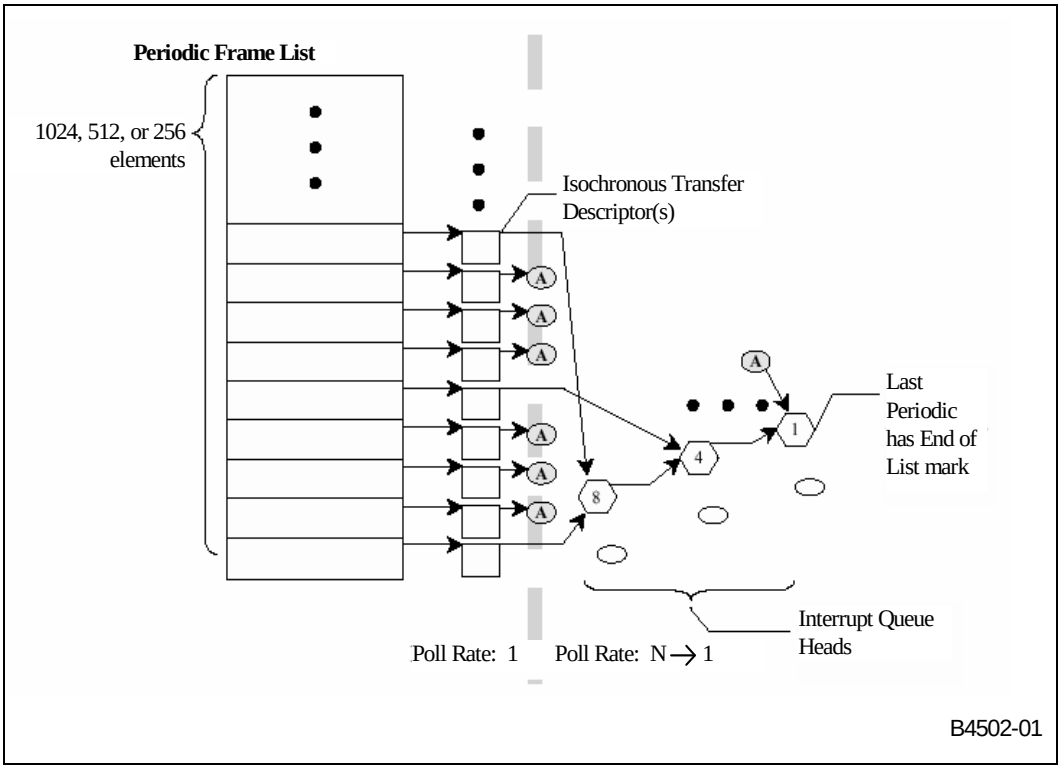
The periodic schedule traversal is enabled or disabled via the Periodic Schedule Enable bit in the USBCMD register. If the Periodic Schedule Enable bit is set to a zero, then the host controller does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when the Periodic Schedule Enable bit is a one, then the host controller does use the PERIODICLISTBASE register to traverse the periodic schedule. The host controller does not react to modifications to the Periodic Schedule Enable immediately.

To eliminate conflicts with split transactions, the host controller evaluates the Periodic Schedule Enable bit only when FRINDEX[2:0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 000b micro-frame. These work items must be removed from the schedule before the Periodic Schedule Enable bit is written to a zero.

The Periodic Schedule Status bit in the USBSTS register indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by writing a one (or zero) to the Periodic Schedule Enable bit in the USBCMD register. Software then can poll the Periodic Schedule Status bit to determine when the periodic schedule has made the desired transition. Software must not modify the Periodic Schedule Enable bit unless the value of the Periodic Schedule Enable bit equals that of the Periodic Schedule Status bit.

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. Figure 99 illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

Figure 99. Example Periodic Schedule



10.14.7 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Section 10.13.3, "Isochronous (High-Speed) Transfer Descriptor (iTD)". There are four distinct sections to an iTD:

- The first field is the Next Link Pointer. This field is for schedule linkage purposes only;
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

10.14.7.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits [12:3] to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, that map directly to FRINDEX register bits [2:0]. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits [2:0] to index into the transaction description array.

If the active bit in the Status field of the indexed transaction description is set to zero, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum packet size, and so on.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller stores Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 00B) field with the transaction offset field.

As the transaction moves data, the host controller must detect when an increment of the current buffer pointer crosses a page boundary. When this occurs the host controller replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes via the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current micro-frame. In other words, the Mult field represents a transaction count for the endpoint in the current micro-frame. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction X Length field represents the total bytes to be sent during the micro-frame. The Mult field must be set by software to be consistent with Transaction X Length and Maximum Packet Sixe. The host controller sends the bytes in Maximum Packet Size'd portions. After each transaction, the host controller decrements it's local copy of Transaction X Length by Maximum Packet Size. The number of bytes the host controller sends is always Maximum Packet Size or Transaction X Length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3 x 1024 bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction X Length field. After all transactions for the endpoint have completed for the micro-frame, Transaction X Length contains the total bytes received. If the final value of Transaction X Length is less than the value of Maximum Packet Size, then less data than was allowed for was received from the associated endpoint. This short packet condition does not set the USBINT bit in the USBSTS register to a one. The host controller does not detect this condition. If the device sends more than Transaction X Length or Maximum Packet Size bytes (whichever is less), then the host controller sets the Babble Detected bit to a one and set the Active bit to a zero.

Note: The host controller is not required to update the iTD field Transaction X Length in this error scenario.

If the Mult field is greater than one, then the host controller automatically executes the value of Mult transactions. The host controller does not execute all Mult transactions if:

- The endpoint is an OUT and Transaction X Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame. Refer to Appendix D for a table summary of the host controller required behavior for all the high-bandwidth transaction cases.

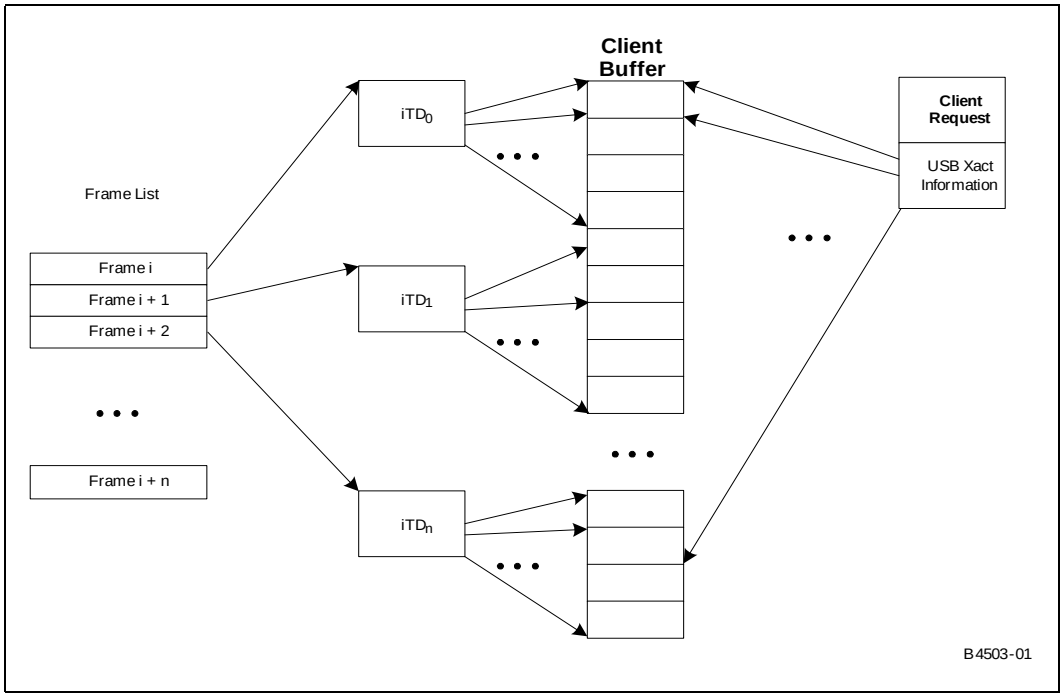
10.14.7.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 100 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (that is, the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify the portions of the buffer that should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request.

Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller's frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

Figure 100. Example Association of iTDs to Client Request Buffer



As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate the buffer sections that are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2:0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field.

The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer wraps a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to 'alias' the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

10.14.7.2.1 Periodic Scheduling Threshold

The Isochronous Scheduling Threshold field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by system software when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.

The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty-factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but always dumps any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

10.14.8 Asynchronous Schedule

The Asynchronous schedule traversal is enabled or disabled via the Asynchronous Schedule Enable bit in the USBCMD register. If the Asynchronous Schedule Enable bit is set to a zero, then the host controller does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, when the Asynchronous Schedule Enable bit is a one, then the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the Asynchronous Schedule Enable bit are not necessarily immediate. The new value of the bit alone is taken into consideration the next time the host controller uses the value of the ASYNCLISTADDR register to get the next queue head.

The Asynchronous Schedule Status bit in the USBSTS register indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to the Asynchronous Schedule Enable bit in the USBCMD register. Software then can poll the Asynchronous Schedule Status bit to

determine when the asynchronous schedule has made the desired transition. Software must not modify the Asynchronous Schedule Enable bit unless the value of the Asynchronous Schedule Enable bit equals that of the Asynchronous Schedule Status bit.

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when the Asynchronous Schedule Enable bit is a zero.

Software may only write this register with defined results when the schedule is disabled, for example, Asynchronous Schedule Enable bit in the USBCMD and the Asynchronous Schedule Status bit in the USBSTS register are zero. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting the Asynchronous Schedule Enable bit is set to one. The asynchronous schedule is actually enabled when the Asynchronous Schedule Status bit is a one.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition (that is, see Section 10.14.8.3, “Empty Asynchronous Schedule Detection”)
- The schedule has been disabled via the Asynchronous Schedule Enable bit in the USBCMD register.

The queue heads in the asynchronous list are linked into a simple circular list as shown in Figure 95 on page 428. Queue head data structures are the only valid data structures that can be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

10.14.8.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. System software writes the physical memory address of a queue head into the ASYNCLISTADDR register, then enables the list by setting the Asynchronous Schedule Enable bit in the USBCMD register to a one.

When inserting a queue head into an active list, software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have T-Bits set to a one or reference valid qTDs and the Horizontal Pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)

                    --

                    -- Requirement: all inputs must be properly initialized.

                    --

                    -- pQHeadCurrent is a pointer to a queue head that is

                    -- already in the active list

                    -- pQHeadNew is a pointer to the queue head to be added

                    --

                    -- This algorithm links a new queue head into a existing

                    -- list

                    --

                    pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer


                    pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)


End InsertQueueHead
```

10.14.8.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. Software deactivates the asynchronous schedule by setting the Asynchronous Schedule Enable bit in the USBCMD register to a zero. Software can determine when the list is idle when the Asynchronous Schedule Status bit in the USBSTS register is a zero.

The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list via the following algorithm. As illustrated, the unlinking is quite easy. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```
UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)

                    --

                    -- Requirement: all inputs must be properly initialized.

                    --

                    -- pQHeadPrevious is a pointer to a queue head that

                    -- references the queue head to remove

                    -- pQHeadToUnlink is a pointer to the queue head to be

                    -- removed

                    -- pQheadNext is a pointer to a queue head still in the

                    -- schedule. Software provides this pointer with the

                    -- following strict rules:

                    --  if the host software is one queue head, then

                    --  pQHeadNext must be the same as

                    --  QueueheadToUnlink.HorizontalPointer. If the host

                    --  software is unlinking a consecutive series of

                    --  queue heads, QHeadNext must be set by software to

                    --  the queue head remaining in the schedule.

                    --

                    -- This algorithm unlinks a queue head from a circular list

                    --

                    pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer

pQueueHeadToUnlink.HorizontalPointer = pQHeadNext

End UnlinkQueueHead
```

If software removes the queue head with the H-bit set to a one, it must select another queue head still linked into the schedule and set its H-bit to a one. This should be completed before removing the queue head. The requirement is that the software should keep one queue head in the asynchronous schedule with its H-bit set to a one. At the point where the software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and can be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers, and so on.). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

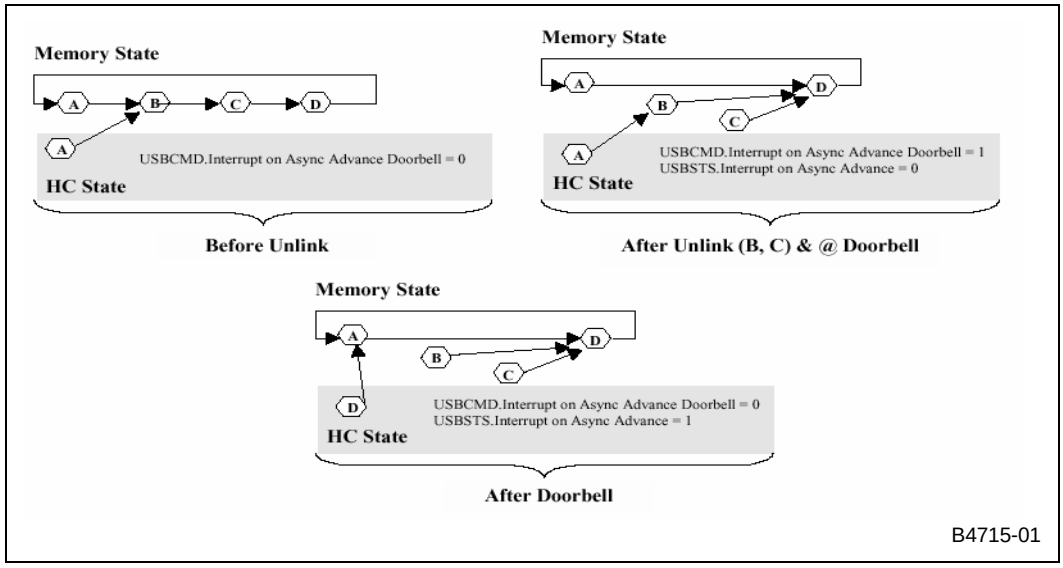
The handshake is implemented with three bits in the host controller. The first bit is a command bit (Interrupt on Async Advance Doorbell bit in the USBCMD register) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (Interrupt on Async Advance bit in the USBSTS register) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit to a one, it also sets the command bit to a zero. The third bit is an interrupt enable (Interrupt on Async Advance bit in the USBINTR register) that is matched with the status bit. If the status bit is a one and the interrupt enable bit is a one, then the host controller asserts a hardware interrupt.

Figure 101 illustrates a general example. In this example, consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that remains in the asynchronous schedule.

When the host controller observes that doorbell bit being set to a one, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (that is, traversed beyond queue head (B) in this example).

Figure 101. Generic Queue Head Unlink Scenario



Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue [twice]) before setting the Advance on Async status bit to a one.

Software may re-use the memory associated with the removed queue heads after it observes the Interrupt on Async Advance status bit is set to a one, following assertion of the doorbell. Software should acknowledge the Interrupt on Async Advance status as indicated in the USBSTS register, before using the doorbell handshake again.

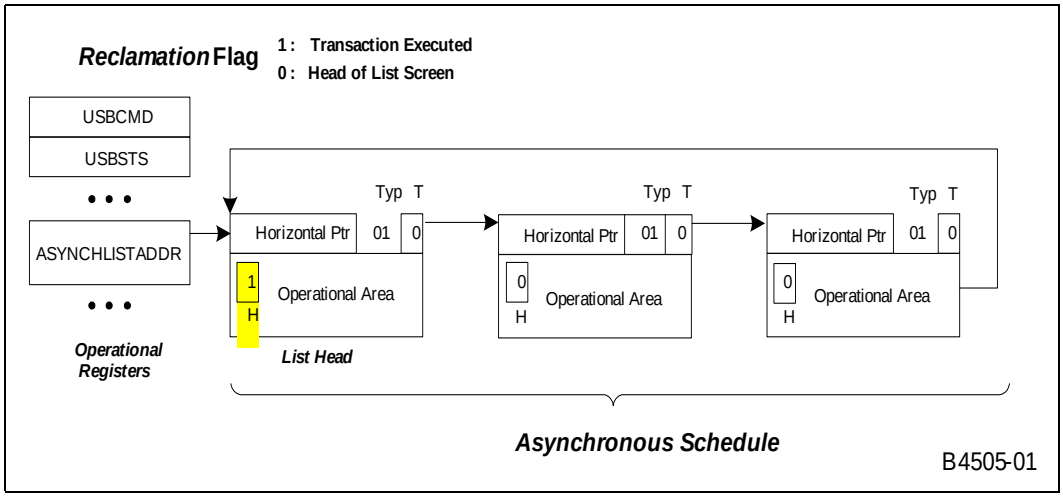
10.14.8.3 Empty Asynchronous Schedule Detection

The Enhanced Host Controller Interface uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see Figure 92 on page 417) defines an H-bit in the queue head, that allows software to mark a queue head as being the head of the reclaim list. The Enhanced Host Controller Interface also keeps a 1-bit flag in the USBSTS register (Reclamation) that is set to a zero when the Enhanced Interface Host Controller observes a queue head with the H-bit set to a one. The reclamation flag in the status register is set to one when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see Section 10.14.8.5, “Asynchronous Schedule Traversal: Start Event”).

If the Enhanced Host Controller Interface ever encounters an H-bit of one and a Reclamation bit of zero, the EHCI controller stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is illustrated in Figure 102.

Figure 102. Asynchronous Schedule List with Annotation to Mark Head of List



Software must ensure there is at most one queue head with the H-bit set to a one, and that it is always coherent with respect to the schedule.

10.14.8.4 Restarting Asynchronous Schedule Before EOF

There are many situations where the host controller detects an empty list long before the end of the micro-frame. It is important to remember that under many circumstances the schedule traversal has stopped due to Nak/Nyet responses from all endpoints.

An example of particular interest is when a start-split for a bulk endpoint occurs early in the micro-frame. Given the EHCI simple traversal rules, the complete-split for that transaction may Nak/Nyet out very quickly. If it is the only item in the schedule, then the host controller ceases traversal of the Asynchronous schedule very early in the micro-frame. To provide reasonable service to this endpoint, the host controller should issue the complete-split before the end of the current micro-frame, instead of waiting until the next micro-frame. When the reason for host controller idling asynchronous

schedule traversal is because of empty list detection, it is mandatory the host controller implement a 'waking' method to resume traversal of the asynchronous schedule. An example method is described below.

10.14.8.4.1 Example Method for Restarting Asynchronous Schedule Traversal

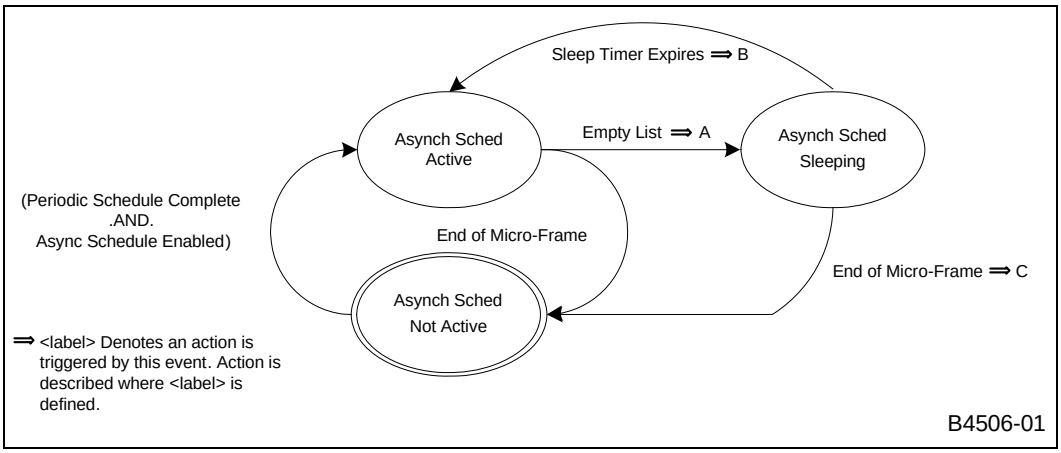
The reason for idling the host controller when the list is empty is to keep the host controller from unnecessarily occupying too much memory bandwidth. The question is: how long should the host controller stay idle before restarting?

The answer in this example is based on deriving a manifest constant, that is, the amount of time the host controller stays idle before restarting traversal. In this example, the manifest constant is called AsyncSchedSleepTime, and has a value of 10msec. The value is derived based on the analysis in "Example Derivation for AsyncSchedSleepTime" The traversal algorithm is simple:

- Traverse the Asynchronous schedule until an End-Of-micro-Frame event occurs, or an empty list is detected. If the event is an End-of-micro-Frame, go attempt to traverse the Periodic schedule. If the event is an empty list, then set a sleep timer and go to a schedule sleep state.
- When the sleep timer expires, set working context to the Asynchronous Schedule start condition and go to schedule active state. The start context allows the HC to reload Nakcnt fields, and so on. so the HC has a chance to run for more than one iteration through the schedule.

This process repeats itself each micro-frame. Figure 103 illustrates a sample state machine to manage the active and sleep states of the Asynchronous Schedule traversal policy. There are three states: Actively traversing the Asynchronous schedule, Sleeping, and Not Active. The last two are similar in terms of interaction with the Asynchronous schedule, but the Not Active state means that the host controller is busy with the Periodic schedule or the Asynchronous schedule is not enabled. The Sleeping state is specifically a special state where the host controller is just waiting for a period of time before resuming execution of the Asynchronous schedule.

Figure 103. Example State Machine for Managing Asynchronous Schedule Traversal



The actions referred to in Figure 103 are defined in Table 175.

Table 175. Asynchronous Schedule State Machine Transition Actions

Action	Action Description Label
A	On detection of the empty list, the host controller sets the AsynchronousTraversalSleepTimer to AsyncSchedSleepTime.
B	When the AsynchronousTraversalSleepTimer expires, the host controller sets the Reclamation bit in the USBSTS register to a one and moves the Nak Counter reload state machine to WaitForListHead.
C	The host controller cancels the sleep timer (AsynchronousTraversalSleepTimer).

Async Sched Not Active

This is the initial state of the traversal state machine after a host controller reset. The traversal state machine does not leave this state when the Asynchronous Schedule Enable bit in the USBCMD register is a zero.

This state is entered from Async Sched Active or Async Sched Sleeping states when the end-of-micro-frame event is detected.

Async Sched Active

This state is entered from the Async Sched Not Active state when the periodic schedule is not active. It is also entered from the Async Sched Sleeping states when the AsyncrhonousTraversalSleepTimer expires. On every transition into this state, the host controller sets the Reclamation bit in the USBSTS register to a one.

When in this state, the host controller continually traverses the asynchronous schedule until the end of micro-frame or an empty list condition is detected.

Async Sched Sleeping

The state is entered from the Async Sched Active state when a schedule empty condition is detected. On entry to this state, the host controller sets the AsynchronousTraversalSleepTimer to AsyncSchedSleepTime.

10.14.8.4.2 Example Derivation for AsyncSchedSleepTime

The derivation is based on analysis of what work the host controller could be doing next. It assumes the host controller does not keep any state about what work is possibly pending in the asynchronous schedule. The schedule can contain any mix of the possible combinations of high, full or low speed control and bulk requests. Table 176 summarizes some of the typical next transactions' that could be in the schedule, and the amount of time (for example, footprint, or wall clock) the transaction takes to complete.

Table 176. Typical Low- /Full-Speed Transaction Times (Sheet 1 of 2)

Transaction Attributes		Footprint (Time)	Description
Speed	HS	11.9 µs 9.45 µs	Maximum foot print for a worst-case, full-sized bulk data transaction. Maximum footprint for an approximate best-case, full-sized bulk data transaction.
Size	512		
Type	Bulk		
Speed	FS	~50 µs	Approximate typical for full-sized bulk data. An 8-byte low-speed is about 2x, or between 90 and 100 µs.
Size	64		
Type	Bulk		

Table 176. Typical Low- /Full-Speed Transaction Times (Sheet 2 of 2)

Transaction Attributes		Footprint (Time)	Description
Speed	FS	~12 µs	Approximate typical for 8-byte bulk/control (that is, setup)
Size	8		
Type	Cntrl		

An AsyncSchedSleepTime value of 10 ms provides a reasonable relaxation of the system memory load and still provides a good level of service for the various transfer types and payload sizes. For example, say we detect an empty list after issuing a start-split for a 64-byte full-speed bulk request. Assuming this is the only thing in the list, the host controller gets the results of the full-speed transaction from the hub during the fifth complete-split request. If the full-speed transaction is an IN and it nak'd, the 10ms sleep period would allow the host controller to get the NAK results on the first complete-split.

10.14.8.5 Asynchronous Schedule Traversal: Start Event

Once the HC has idled itself via the empty schedule detection (Section 10.14.8.3, "Empty Asynchronous Schedule Detection"), it naturally activates and begin processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the HC must occasionally re-activate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. The requirements and method for this restart are described in Section 10.14.8.4, "Restarting Asynchronous Schedule Before EOF". Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state (see Section 10.14.8.4, "Restarting Asynchronous Schedule Before EOF").

10.14.8.6 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature (Section 10.14.8.3, "Empty Asynchronous Schedule Detection") depends on the proper management of the Reclamation bit in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head when traversing the asynchronous schedule (See Section 10.14.10.1, "Fetch Queue Head").

It is required that the host controller sets the Reclamation bit to a one whenever an asynchronous schedule traversal Start Event, as documented in Section 10.14.8.5, "Asynchronous Schedule Traversal: Start Event", occurs. The Reclamation bit is also set to a one whenever the host controller executes a transaction when traversing the asynchronous schedule (see Section 10.14.10.3, "Execute Transaction"). The host controller sets the Reclamation bit to a zero whenever it finds a queue head with its H-bit set to a one. Software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If software sets the H-bit in an interrupt queue head to a one, the resulting behavior is undefined. The host controller may set the Reclamation bit to a zero when executing from the periodic schedule.

10.14.9 Operational Model for Nak Counter

This section describes the operational model for the NakCnt field defined in a queue head (see Section 10.13.6, "Queue Head"). Software should not use this feature for interrupt queue heads. This rule is not required to be enforced by the host controller.

USB protocol has built-in flow control via the Nak response by a device. There are several scenarios, beyond the Ping feature, where an endpoint may naturally Nak or Nyet the majority of the time. An example is the host controller management of the split transaction protocol for control and bulk endpoints. All bulk endpoints (High- or Full-speed) are serviced via the same asynchronous schedule. The time between the Start-split transaction and the first Complete-split transaction could be very short (that is, like when the endpoint is the only one in the asynchronous schedule). The hub NYETs (effectively Naks) the Complete-split transaction until the classic transaction is complete. This could result in the host controller thrashing memory, repeatedly fetching the queue head and executing the transaction to the hub, that does not complete until after the transaction on the classic bus completes.

There are two component fields in a queue head to support the throttling feature: a counter field (NakCnt), and a counter reload field (RL). NakCnt is used by the host controller as one of the criteria to determine whether to execute a transaction to the endpoint. There are two operational modes associated with this counter:

- Not Used. This mode is set when the RL field is zero. The host controller ignores the NakCnt field for any execution of transactions through a queue head with an RL field of zero. Software must use this selection for interrupt endpoints.
- Nak Throttle Mode. This mode is selected when the RL field is non-zero. In this mode, the value in the NakCnt field represents the maximum number of Nak or Nyet responses the host controller tolerates on each endpoint. In this mode, the HC decrements the NakCnt field based on the token/handshake criteria listed in Table 177. The host controller must reload NakCnt when the endpoint successfully moves data (for example, policy to reward device for moving data).

Table 177. NakCnt Field Adjustment Rules

Token	Handshake	
	Handshake NAK	NYET
IN/PING	decrement NakCnt	N/A (protocol error)
OUT	decrement NakCnt	No Action[1] Start
Split	decrement NakCnt	N/A (protocol error)
Complete Split	No Action	Decrement NakCnt

Notes:
1. Recommended behavior on this response is to reload NakCnt.

In summary, system software enables the counter by setting the reload field (RL) to a non-zero value. The host controller may execute a transaction if NakCnt is non-zero. The host controller does not execute a transaction if NakCnt is zero. The reload mechanism is described in detail in Section 10.14.9.1, "Nak Count Reload Control".

Note: When all queue heads in the Asynchronous Schedule exhausts all transfers or all NakCnt's go to zero, then the host controller detects an empty asynchronous schedule and idle schedule traversal (see Section 10.14.8.3, "Empty Asynchronous Schedule Detection").

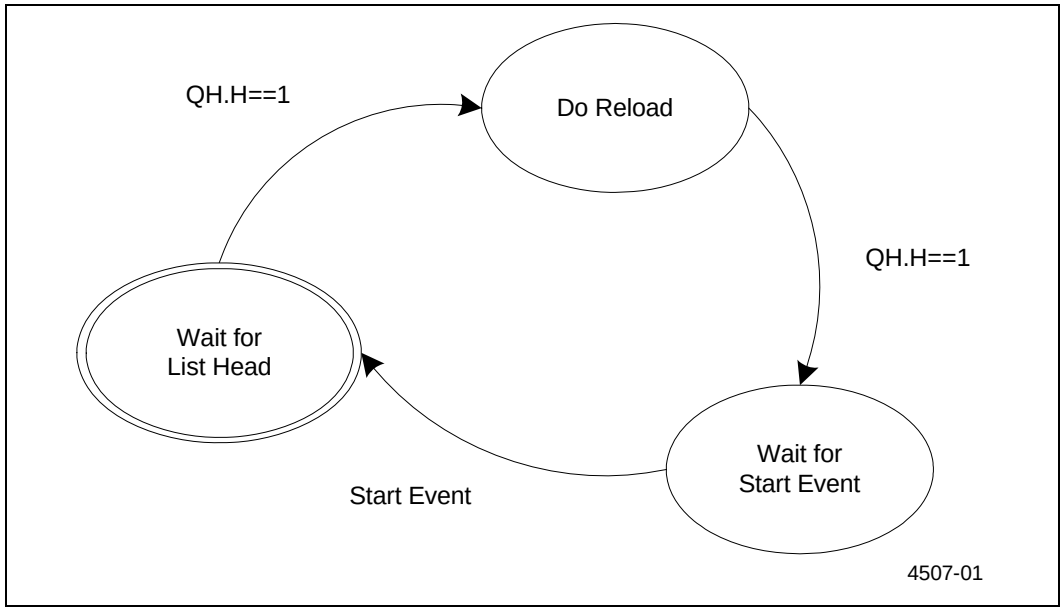
Any time the host controller begins a new traversal of the Asynchronous Schedule, a Start Event is assumed, see Section 10.14.8.5, "Asynchronous Schedule Traversal: Start Event". Every time a Start-Event occurs, the Nak Count reload procedure is enabled.

10.14.9.1 Nak Count Reload Control

When the host controller reaches the Execute Transaction state for a queue head (meaning that it has an active operational state), it checks to determine whether the NakCnt field should be reloaded from RL (see Section 10.14.10.3, "Execute Transaction"). If the answer is yes, then RL is copied into NakCnt. After the reload or if the reload is not active, the host controller evaluates whether to execute the transaction.

The host controller must reload nak counters (NakCnt see Figure 92) in queue heads during the first pass through the reclamation list after an asynchronous schedule Start Event (see Section 10.14.8.5, "Asynchronous Schedule Traversal: Start Event" for the definition of the Start Event). The Asynchronous Schedule should have at most one queue head marked as the head (see Figure 102). Figure 104 illustrates an example state machine that satisfies the operational requirements of the host controller detecting the first pass through the Asynchronous Schedule. This state machine is maintained internal to the host controller and is only used to gate reloading of the nak counter during the queue head traversal state: Execute Transaction (Figure 105). The host controller does not perform the nak counter reload operation if the RL field (see Figure 92) is set to zero.

Figure 104. Example HC State Machine for Controlling Nak Counter Reloads



10.14.9.1.1 Wait for List Head

This is the initial state. The state machine enters this state from Wait for Start Event when a start event as defined in Section 10.14.8.5, "Asynchronous Schedule Traversal: Start Event" occurs. The purpose of this state is to wait for the first observation of the head of the Asynchronous Schedule. This occurs when the host controller fetches a queue head whose H-bit is set to a one.

10.14.9.1.2 Do Reload

This state is entered from the Wait for List Head state when the host controller fetches a queue head with the H-bit set to a one. When in this state, the host controller performs nak counter reloads for every queue head visited that has a non-zero nak reload value (RL) field.

10.14.9.1.3 Wait for Start Event

This state is entered from the Do Reload state when a queue head with the H-bit set to a one is fetched. when in this state, the host controller does not perform nak counter reloads.

10.14.10 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the Queue Element Transfer Descriptor (qTD) structure defined in Section 10.13.5, “Queue Element Transfer Descriptor (qTD)”. One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed (see Overlay area defined in Figure 92). Each qTD represents one or more bus transactions, that is defined in the context of this specification as a transfer.

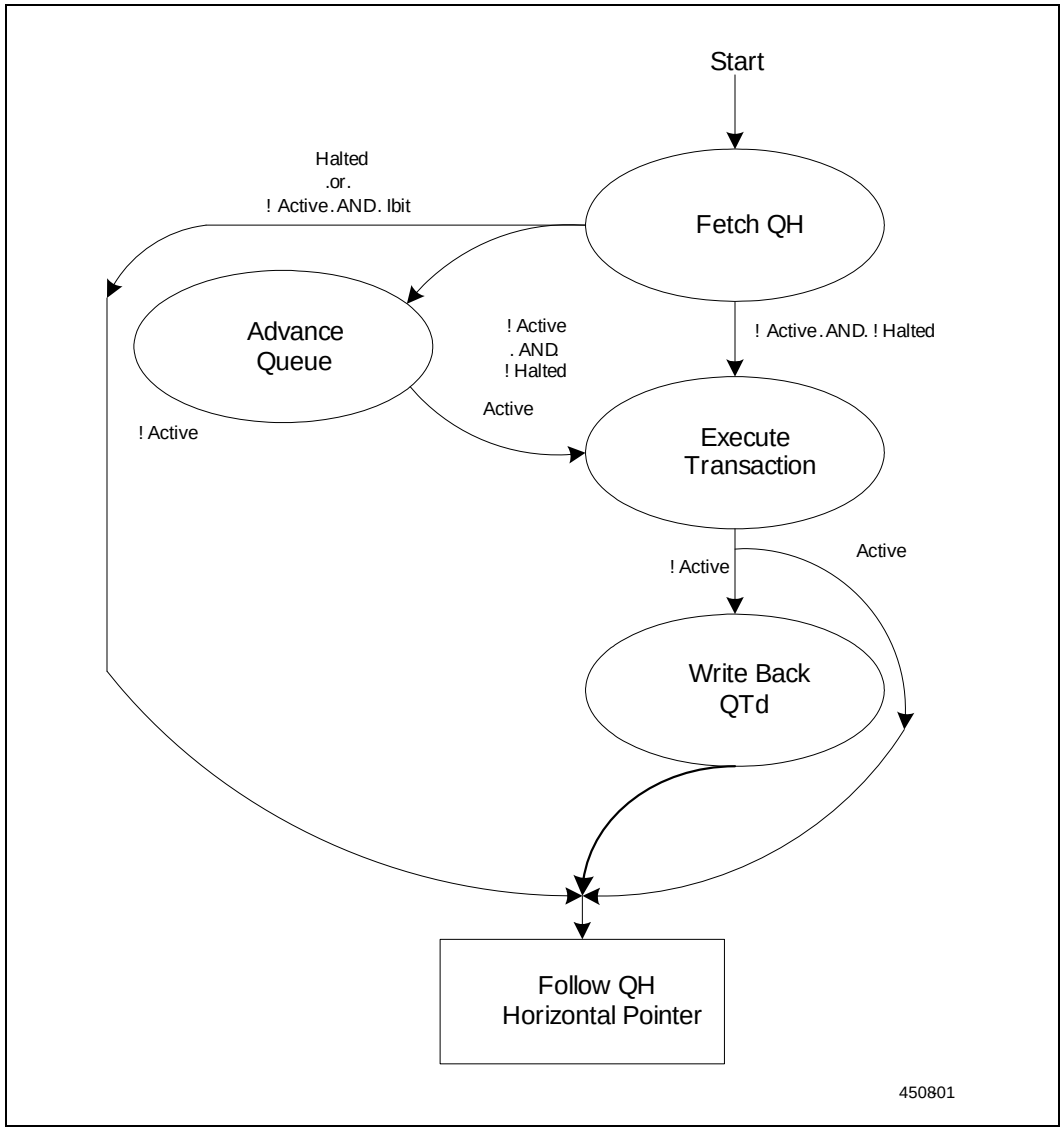
The general processing model for the host controller's use of a queue head is simple:

- read a queue head,
- execute a transaction from the overlay area,
- write back the results of the transaction to the overlay area
- move to the next queue head.

If the host controller encounters errors during a transaction, the host controller sets one (or more) of the error reporting bits in the queue head's Status field. The Status field accumulates all errors encountered during the execution of a qTD (for example, the error bits in the queue head Status field are 'sticky' until the transfer (qTD) has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (for example, buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions occurs for the endpoint and the host controller does not advance the queue.

An example host controller operational state machine of a queue head traversal is illustrated in Figure 105. This state machine is a model for how a host controller should traverse a queue head. The host controller must be able to advance the queue from the Fetch QH state to avoid all hardware/software race conditions. This simple mechanism allows software to link qTDs to the queue head and activate them, then the host controller always finds them if/when they are reachable.

Figure 105. Host Controller Queue Head Traversal State Machine



This traversal state machine applies to all queue heads, regardless of transfer type or whether split transactions are required. The following sections describe each state. Each state description describes the entry criteria. The Execute Transaction state (Section 10.14.10.3, “Execute Transaction”) describes the basic requirements for all endpoints. Section 10.14.12.1, “Split Transactions for Asynchronous Transfers” and Section 10.14.12.2, “Split Transaction Interrupt” describe details of the required extensions to the Execute Transaction state for endpoints requiring split transactions.

Note: Prior to software placing a queue head into the periodic or asynchronous list, software must ensure the queue head is properly initialized. Minimally, the queue head should be initialized to the following:

- Valid static endpoint state
- For the very first use of a queue head, software may zero-out the queue head transfer overlay, then set the Next qTD Pointer field value to reference a valid qTD.

10.14.10.1 Fetch Queue Head

A queue head can be referenced from the physical address stored in the ASYNCLISTADDR Register (Section 10.12.7, “ASYNCLISTADDR; ENDPOINTLISTADDR”). Additionally, it can be referenced from the Next Link Pointer field of an iTD, siTD, FSTN or another Queue Head. If the referencing link pointer has the Typ field set to indicate a queue head, it is assumed to reference a queue head structure as defined in Figure 92.

When in this state, the host controller performs operations to implement empty schedule detection (Section 10.14.8.3, “Empty Asynchronous Schedule Detection”) and Nak Counter reloads (Section 10.14.9, “Operational Model for Nak Counter”). After the queue head has been fetched, the host controller conducts the following queries for empty schedule detection:

- If queue head is not an interrupt queue head (that is, S-mask is a zero), and
- The H-bit is a one, and
- The Reclamation bit in the USBSTS register is a zero.

When these criteria are met, the host controller stops traversing the asynchronous list (as described in Section 10.14.8.3, “Empty Asynchronous Schedule Detection”). When the criteria are not met, the host controller continues schedule traversal. If the queue head is not an interrupt and the H-bit is a one and the Reclamation bit is a one, then the host controller sets the Reclamation bit in the USBSTS register to a zero before completing this state. The operations for reloading of the Nak Counter are described in detail in Section 10.14.9, “Operational Model for Nak Counter”.

This state is complete when the queue head has been read on-chip.

10.14.10.2 Advance Queue

To advance the queue, the host controller must find the next qTD, adjust pointers, perform the overlay and write back the results to the queue head.

This state is entered from the FetchQHD state if the overlay Active and Halt bits are set to zero. On entry to this state, the host controller determines the next pointer that has to be used to fetch a qTD, fetches a qTD and determines whether to perform an overlay.

Note: If the I-bit is a one and the Active bit is a zero, the host controller immediately skips processing of this queue head, exits this state and uses the horizontal pointer to the next schedule data structure.

If the field Bytes to Transfer is not zero and the T-bit in the Alternate Next qTD Pointer is set to zero, then the host controller uses the Alternate Next qTD Pointer. Otherwise, the host controller uses the Next qTD Pointer. If Next qTD Pointer’s T-bit is set to a one, then the host controller exits this state and uses the horizontal pointer to the next schedule data structure.

Using the selected pointer the host controller fetches the referenced qTD. If the fetched qTD has its Active bit set to a one, the host controller moves the pointer value used to reach the qTD (Next or Alternate Next) to the Current qTD Pointer field, then performs the overlay. If the fetched qTD has its Active bit set to a zero, the host controller aborts the queue advance and follows the queue head's horizontal pointer to the next schedule data structure. The host controller performs the overlay based on the following rules:

- The value of the data toggle (dt) field in the overlay area depends on the value of the data toggle control (dtc) bit (see Table 164).

- If the EPS field indicates the endpoint is a high-speed endpoint, the Ping state field is preserved by the host controller. The value of this field is not changed as a result of the overlay.
- C-prog-mask field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- Frame Tag field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- NakCnt field in the overlay area is loaded from the RL field in the queue head's Static Endpoint State.
- All other areas of the overlay are set by the incoming qTD.

The host controller exits this state when it has committed the write to the queue head.

10.14.10.3 Execute Transaction

The host controller enters this state from the Fetch Queue Head state only if the Active bit in Status field of the queue head is set to a one.

On entry to this state, the host controller executes a few pre-operations, then checks some pre-condition criteria before committing to executing a transaction for the queue head.

The pre-operations performed and the pre-condition criteria depend on whether the queue head is an interrupt endpoint. The host controller can determine that a queue head is an interrupt queue head when the queue head's S-mask field contains a non-zero value. It is the responsibility of software to ensure the S-mask field is appropriately initialized based on the transfer type. There are other criteria that must be met if the EPS field indicates that the endpoint is a low- or full-speed endpoint, see Section 10.14.12.1, "Split Transactions for Asynchronous Transfers" and Section 10.14.12.2, "Split Transaction Interrupt".

- Interrupt Transfer Pre-condition Criteria

  If the queue head is for an interrupt endpoint (for example, non-zero S-mask field), then the FRINDEX[2:0] field must identify a bit in the S-mask field that has a one in it. For example, an S-mask value of 00100000b would evaluate to true only when FRINDEX[2:0] is equal to 101b. If this condition is met then the host controller considers this queue head for a transaction.

- Asynchronous Transfer Pre-operations and Pre-condition Criteria

  If the queue head is not for an interrupt endpoint (for example, a zero S-mask field), then the host controller performs one pre-operation and then evaluates one pre-condition criteria: The pre-operation is:

  — Checks the Nak counter reload state (Section 10.14.9, "Operational Model for Nak Counter"). It is necessary for the host controller to reload the Nak Counter field. The reload is performed at this time.

  The pre-condition evaluated is:

  — Whether the NakCnt field has been reloaded, the host controller checks the value of the NakCnt field in the queue head. If NakCnt is non-zero, or if the Reload Nak Counter field is zero, then the host controller considers this queue head for a transaction.

- Transfer Type Independent Pre-operations

  Regardless of the transfer type, the host controller always performs at least one pre-operation and evaluates one pre-condition. The pre-operation is:

  — A host controller internal transaction (down) counter qHTransactionCounter is loaded from the queue head's Mult field. A host controller implementation is allowed to ignore this for queue heads on the asynchronous list. It is

mandatory for interrupt queue heads. Software should ensure that the Mult field is set appropriately for the transfer type.

The pre-conditions evaluated are:

— The host controller determines whether there is enough time in the micro-frame to complete this transaction (see Section 10.14.4.1, "Example: Preserving Micro-Frame Integrity" for an example evaluation method). If there is not enough time to complete the transaction, the host controller exits this state.

— If the value of qHTransactionCounter for an interrupt endpoint is zero, then the host controller exits this state.

When the pre-operations are complete and pre-conditions are met, the host controller sets the Reclamation bit in the USBSTS register to a one and then begins executing one or more transactions using the endpoint information in the queue head. The host controller iterates qHTransactionCounter times in this state executing transactions. After each transaction is executed, qHTransactionCounter is decremented by one. The host controller exits this state when one of the following events occurs:

- The qHTransactionCounter decrements to zero, or
- The endpoint responds to the transaction with any handshake other than an ACK,[1] or
- The transaction experiences a transaction error, or
- The Active bit in the queue head goes to a zero, or
- There is not enough time in the micro-frame left to execute the next transaction (see "Transaction Fit: A Best-Fit Approximation Algorithm" for example method for implementing the frame boundary test).

The results of each transaction is recorded in the on-chip overlay area. If data is successfully moved during the transaction, the transfer state in the overlay area is advanced. To advance queue head's transfer state, the Total Bytes to Transfer field is decremented by the number of bytes moved in the transaction, the data toggle bit (dt) is toggled, the current page offset is advanced to the next appropriate value (for example, advanced by the number of bytes successfully moved), and the C_Page field is updated to the appropriate value (if necessary). See Section 10.14.10.6, "Buffer Pointer List Use for Data Streaming with qTDs".

Note: The Total Bytes To Transfer field can be zero when all the other criteria for executing a transaction are met. When this occurs, the host controller executes a zero-length transaction to the endpoint.

If the PID_Code field indicates an IN transaction and the device delivers data, the host controller detects a packet babble condition, set the babble and halted bits in the Status field, set the Active bit to a zero, write back the results to the source qTD, then exit this state.

In the event an IN token receives a data PID mismatch response, the host controller must ignore the received data (for example, not advance the transfer state for the bytes received). Additionally, if the endpoint is an interrupt IN, then the host controller must record that the transaction occurred (for example, decrement qHTransactionCounter). It is recommended (but not required) the host controller continue executing transactions for this endpoint if the resultant value of qHTransactionCounter is greater than one.

1. For a high-bandwidth interrupt OUT endpoint, the host controller may optionally immediately retry the transaction if it fails.

If the response to the IN bus transaction is a Nak (or Nyet) and RL is non-zero, NakCnt is decremented by one. If RL is zero, then no write-back by the host controller is required (for a transaction receiving a Nak or Nyet response and the value of CErr did not change). Software should set the RL field to zero if the queue head is an interrupt endpoint. Host controller hardware is not required to enforce this rule or operation.

After the transaction has finished and the host controller has completed the post processing of the results (advancing the transfer state and possibly NakCnt, the host controller writes back the results of the transaction to the queue head's overlay area in main memory.

The number of bytes moved during an IN transaction depends on how much data the device endpoint delivers. The maximum number of bytes a device can send is Maximum Packet Size. The number of bytes moved during an OUT transaction is Maximum Packet Length bytes or Total Bytes to Transfer, whichever is less.

If there was a transaction error during the transaction, the transfer state (as defined above) is not advanced by the host controller. The CErr field is decremented by one and the status field is updated to reflect the type of error observed. Transaction errors are summarized in "Transaction Error".

The following events causes the host controller to clear the Active bit in the queue head's overlay status field. When the Active bit transitions from a one to a zero, the transfer in the overlay is considered complete. The reason for the transfer completion (clearing the Active bit) determines the next state.

- CErr field decrements to zero. When this occurs the Halted bit is set to a one and Active is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.
- The device responds to the transaction with a STALL PID. When this occurs, the Halted bit is set to a one and the Active bit is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.
- The Total Bytes to Transfer field is zero after the transaction completes.

Note: For a zero length transaction, it was zero before the transaction was started. When this condition occurs, the Active bit is set to zero.

- The PID code is an IN, and the number of bytes moved during the transaction is less than the Maximum Packet Length. When this occurs, the Active bit is set to zero and a short packet condition exists. The short-packet condition is detected during the Advance Queue state. Refer to Section 10.14.12, "Split Transactions" for additional rules for managing low- and full-speed transactions.
- The PID Code field indicates an IN and the device sends more than the expected number of bytes (for example, Maximum Packet Length or Total Bytes to Transfer bytes, whichever is less) (for example, a packet babble). This results in the host controller setting the Halted bit to a one.

With the exception of a NAK response (when RL field is zero), the host controller always writes the results of the transaction back to the overlay area in main memory. This includes when the transfer completes. For a high-speed endpoint, the queue head information written back includes minimally the following fields:

- NakCnt, dt, Total Bytes to Transfer, C_Page, Status, CERR, and Current Offset

For a low- or full-speed device the queue head information written back also includes the fields:

- C-prog-mask, FrameTag and S-bytes.

The duration of this state depends on the time it takes to complete the transaction(s) and the status write to the overlay is committed.

10.14.10.3.1 Halting a Queue Head

A halted endpoint is defined only for the transfer types that are managed via queue heads (control, bulk and interrupt). The following events indicate that the endpoint has reached a condition where no more activity can occur without intervention from the driver:

- An endpoint may return a STALL handshake during a transaction,
- A transaction had three consecutive error conditions, or
- A Packet Babble error occurs on the endpoint.

When any of these events occur (for a queue head) the Host Controller halts the queue head and set the USBERRINT status bit in the USBSTS register to a one. To halt the queue head, the Active bit is set to a zero and the Halted bit is set to a one. There can be other error status bits that are set when a queue is halted. The host controller always writes back the overlay area to the source qTD when the transfer is complete, regardless of the reason (normal completion, short packet or halt). The host controller does not advance the transfer state on a transaction that results in a Halt condition (for example, no updates necessary for Total Bytes to Transfer, C_Page, Current Offset, and dt). The host controller must update CErr as appropriate. When a queue head is halted, the USB Error Interrupt bit in the USBSTS register is set to a one. If the USB Error Interrupt Enable bit in the USBINTR register is set to a one, a hardware interrupt is generated at the next interrupt threshold.

10.14.10.3.2 Asynchronous Schedule Park Mode

Asynchronous Schedule Park mode is a special execution mode that can be enabled by system software, where the host controller is permitted to execute more than one bus transaction from a high-speed queue head in the Asynchronous schedule before continuing horizontal traversal of the Asynchronous schedule. This feature has no effect on queue heads or other data structures in the Periodic schedule. This feature is similar in intent as the Mult feature that is used in the Periodic schedule. Where-as the Mult feature is a characteristic that is tunable for each endpoint; park-mode is a policy that is applied to all high-speed queue heads in the asynchronous schedule. It is essentially the specification of an iterator for consecutive bus transactions to the same endpoint. All of the rules for managing bus transactions and the results of those as defined in Section 10.14.10.3, “Execute Transaction” apply. This feature merely specifies how many consecutive times the host controller is permitted to execute from the same queue head before moving to the next queue head in the Asynchronous List. This feature should allow the host controller to attain better bus utilization for those devices that are capable of moving data at maximum rate, and at the same time providing a fair service to all endpoints.

A host controller exports its capability to support this feature to system software by setting the Asynchronous Schedule Park Capability bit in the HCCPARAMs register to a one. This information keys system software that the Asynchronous Schedule Park Mode Enable and Asynchronous Schedule Park Mode Count fields in the USBCMD register are modifiable. System software enables the feature by writing a one to the Asynchronous Schedule Park Mode Enable bit.

When park-mode is not enabled (for example, Asynchronous Schedule Park Mode Enable bit in the USBCMD register is a zero), the host controller must not execute more than one bus transaction per high-speed queue head, per traversal of the asynchronous schedule. When park-mode is enabled, the host controller must not apply the feature to a queue head whose EPS field indicates a Low/Full-speed device (that is, only one bus

transaction is allowed from each Low/Full-speed queue head per traversal of the asynchronous schedule). Park-mode may only be applied to queue heads in the Asynchronous schedule whose EPS field indicates that it is a high-speed device.

The host controller must apply park mode to queue heads whose EPS field indicates a high-speed endpoint. The maximum number of consecutive bus transactions a host controller may execute on a high-speed queue head is determined by the value in the Asynchronous Schedule Park Mode Count field in the USBCMD register. Software must not set Asynchronous Schedule Park Mode Enable bit to a one and also set Asynchronous Schedule Park Mode Count field to a zero. The resulting behavior is not defined.

A sample behavioral example describes the operational requirements for the host controller implementing park-mode. This feature does not affect how the host controller handles the bus transaction as defined in Section 10.14.10.3, “Execute Transaction”. It only effects how many consecutive bus transactions for the current queue head can be executed. All boundary conditions, error detection and reporting applies as usual. This feature is similar in concept to the use of the Mult field for high-bandwidth Interrupt for queue heads in the Periodic Schedule.

The host controller effectively loads an internal down-counter PM-Count from Asynchronous Schedule Park Mode Count when Asynchronous Schedule Park Mode Enable bit is a one, and a high-speed queue head is first fetched and meets all the criteria for executing a bus transaction. After the bus transaction, PM-Count is decremented. The host controller may continue to execute bus transactions from the current queue head until PM-Count goes to zero, an error is detected, the buffer for the current transfer is exhausted or the endpoint responds with a flow-control or STALL handshake. Table 178 summarizes the responses that effect whether the host controller continues with another bus transaction for the current queue head.

Table 178. Actions for Park Mode, Based on Endpoint Response and Residual Transfer State (Sheet 1 of 2)

PID	Endpoint Response	Transfer State after Transaction		Action
		PM-Count	Bytes to Transfer	
IN	DATA[0,1] w/ Maximum Packet sized data	Not zero	Not Zero	Allowed to perform another bus transaction.1, 2
		Not zero	Zero	Retire qTD and move to next QH
		Zero	Don’t care	Move to next QH.
	DATA[0,1] w/ short packet	Don’t care	Don’t care	Retire qTD and move to next QH.
	NAK	Don’t care	Don’t care	Move to next QH.
	STALL, XactErr	Don’t care	Don’t care	Move to next QH.
OUT	ACK	Not zero	Not Zero	Allowed to perform another bus transaction.2
		Not zero	Zero	Retire qTD and move to next QH
		Zero	Don’t’ care	Move to next QH.

Notes:
1. The host controller may continue to execute bus transactions from the current high-speed queue head (if PM-Count is not equal to zero), if a PID mismatch is detected (for example, expected DATA1 and received DATA0, or visa-versa).
2. This specification does not require that the host controller execute another bus transaction when PM-Count is non-zero. Implementations are encouraged to make appropriate complexity and performance trade-offs.

Table 178. Actions for Park Mode, Based on Endpoint Response and Residual Transfer State (Sheet 2 of 2)

PID	Endpoint Response	Transfer State after Transaction		Action
		PM-Count	Bytes to Transfer	
	NYET, NAK	Don't care	Don't care	Move to next QH.
	STALL, XactErr	Don't care	Don't care	Move to next QH
PING	ACK	Not Zero	Not Zero	Allowed to perform another bus transaction.2
	NAK	Don't care	Don't care	Move to next QH
	STALL, XactErr	Don't care	Don't care	Move to next QH

Notes:
1. The host controller may continue to execute bus transactions from the current high-speed queue head (if PM-Count is not equal to zero), if a PID mismatch is detected (for example, expected DATA1 and received DATA0, or visa-versa).
2. This specification does not require that the host controller execute another bus transaction when PM-Count is non-zero. Implementations are encouraged to make appropriate complexity and performance trade-offs.

10.14.10.4 Write Back qTD

This state is entered from the Execute Transaction state when the Active bit is set to a zero. The source data for the write-back is the transfer results area of the queue head overlay area (see Figure 92). The host controller uses the Current qTD Pointer field as the target address for the qTD. The queue head transfer result area is written back to the transfer result area of the target qTD. This state is also referred to as: qTD retirement. The fields that must be written back to the source qTD include Total Bytes to Transfer, Cerr, and Status.

The duration of this state depends on when the qTD write-back has been committed.

10.14.10.5 Follow Queue Head Horizontal Pointer

The host controller must use the horizontal pointer in the queue head to the next schedule data structure when any of the following conditions exist:

- If the Active bit is a one on exit from the Execute Transaction state, or
- When the host controller exits the Write Back qTD state, or
- If the Advance Queue state fails to advance the queue because the target qTD is not active, or
- If the Halted bit is a one on exit from the Fetch QH state.

There is no functional requirement that the host controller wait until the current transaction is complete before using the horizontal pointer to read the next linked data structure. But, it must wait until the current transaction is complete before executing the next data structure.

10.14.10.6 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers, that is used to reference the data buffer for a transfer. This specification requires that the buffer associated with the transfer be virtually contiguous. This means: if the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.

- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, that may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

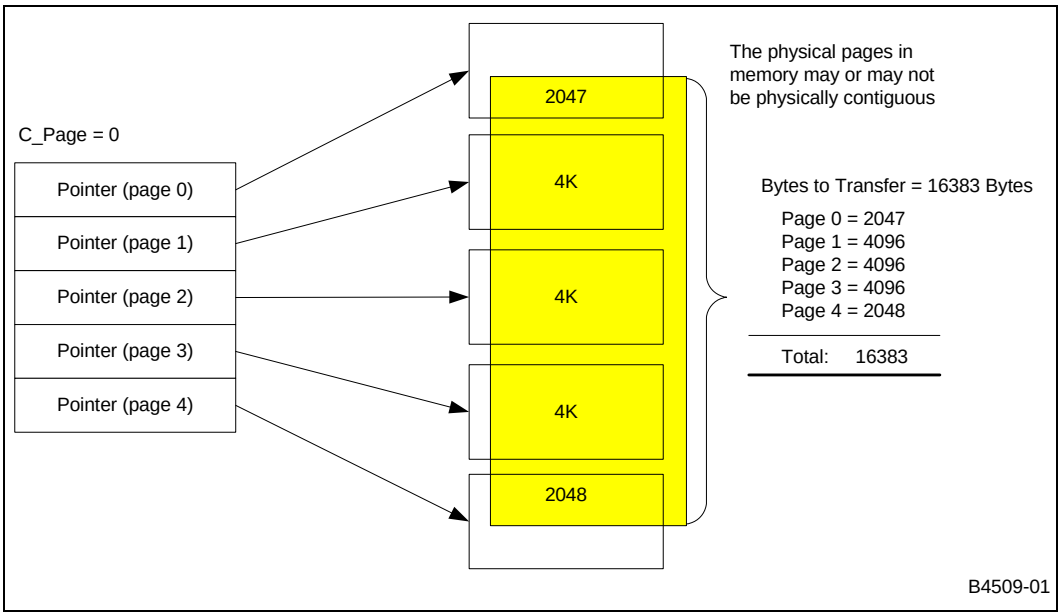
The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16-Kbyte buffer with any starting buffer alignment.

The host controller uses the field C_Page field as an index value to determine the buffer pointer in the list that should be used to start the current transaction. The host controller uses another buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the Bytes to Transfer field.

Figure 106 illustrates a nominal example of how System software would initialize the buffer pointers list and the C_Page field for a transfer size of 16,383 bytes. C_Page is set to zero. The upper 20-bits of Page 0 references the start of the physical page. Current Offset (the lower 12-bits of queue head Dword 7) holds the offset in the page that is, 2,049 (for example, 4,096-2,047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

Figure 106. Example Mapping of qTD Buffer Pointers to Buffer Pages



For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is set to zero) and concatenates the Current Offset field. The 512 bytes are moved during the transaction, the Current Offset and Total Bytes to Transfer are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller increments C_Page (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page

pointer is the page 1 pointer and Current Offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller handles C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer when streaming data to/from the USB.
- The current transaction completes on a page boundary (that is, the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

Note: The only valid adjustment the host controller may make to C_Page is to increment by one.

10.14.10.7 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes frames where a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's S-Mask to indicate the micro-frame with-in a 1-ms period that a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in Table 179.

Table 179. Example Periodic Reference Patterns for Interrupt Transfers with 2-ms Poll Rate

Frame # Reference Sequence	Description
0, 2, 4, 6, 8, and so on S-Mask = 01h	A queue head for the bInterval of 2 ms (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 01h, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8, and so on. S-Mask = 02h	Another example of a queue head with a bInterval of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. But, the S-Mask is set to 02h indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

10.14.10.8 Managing Transfer Complete Interrupts from Queue Heads

The host controller sets an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an Interrupt on Complete (IOC) bit set to a one, or whenever a transfer (qTD) completes with a short packet. If system software needs multiple qTDs to complete a client request (that is, like a control transfer) the intermediate qTDs do not require interrupts. System software may only need a single interrupt to notify it that the complete buffer has been transferred. System software may set IOC's to occur more frequently. A motivation for this is that it wants early notification so that interface data structures can be re-used in a timely manner.

10.14.11 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called Ping. Ping is required for all USB 2.0 High-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The Status field has a Ping State bit, that the host controller uses to determine the next actual PID it uses in the next transaction to the endpoint (see Table 161). The Ping State bit is only managed by the host controller for queue heads that meet the following criteria:

- Queue head is not an interrupt and
- EPS field equals High-Speed and
- PIDCode field equals OUT

Table 180 illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the USB Specification Revision 2.0 for detailed description on the Ping protocol.

Table 180. Ping Control State Transition Table

Current	Event		Next
	Host	Device	
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr1	Do Ping
Do Ping	PING	Stall	N/C2
OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping3
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr1	Do Ping
Do OUT	OUT	Stall	N/C2

Notes:
1. Transaction Error (XactErr) is any time the host misses the handshake.
2. No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (for example, Active set to zero and Halt set to a one). Software intervention is required to restart queue.
3. A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.

The Ping State bit has the following encoding:

Table 181. Ping State Encoding

Value	Meaning
0b	Do OUT The host controller uses an OUT PID during the next bus transaction to this endpoint.
1b	Do Ping The host controller uses a PING PID during the next bus transaction to this endpoint.

The defined ping protocol (see USB 2.0 Specification, Chapter 8) allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when we don't know whether there is space on the device or not). The host controller manages the Ping State bit. System software sets the initial value in the queue head when it

initializes a queue head. The host controller preserves the Ping State bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

10.14.12 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB-1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below USB 2.0 hub, utilizing the split transaction protocol. Refer to USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and Low-speed devices are enumerated identically as high-speed devices, but the transactions to the Full- and Low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the Full- or Low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and Transaction Translator below that the Full or Low-speed device is attached.

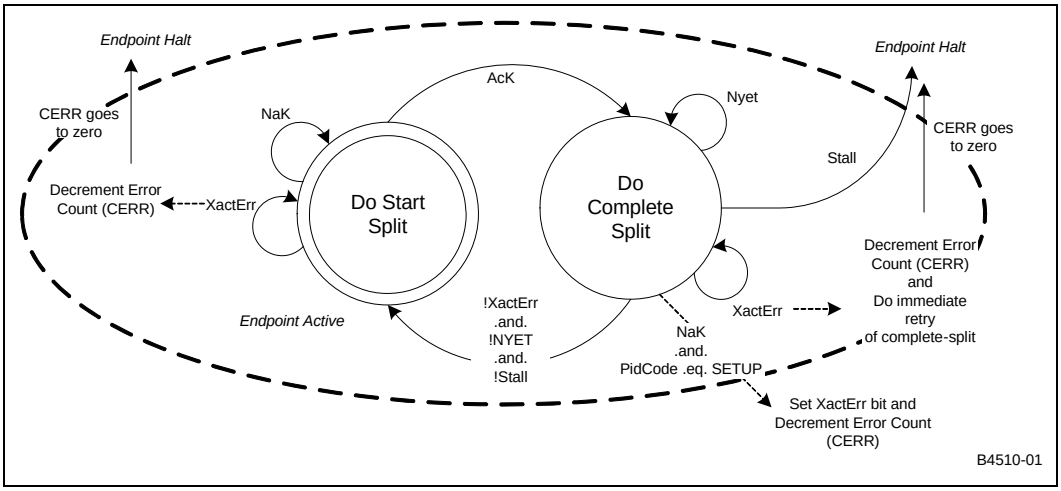
The EHCI interface uses dedicated data structures for managing full-speed isochronous data streams (see Section 10.13.4, “Split Transaction Isochronous Transfer Descriptor (siTD)”). Control, Bulk, and Interrupt are managed using the queuing data structures. The interface data structures must be programmed with the device address and the Transaction Translator number of the USB 2.0 hub operating as the Low-/Full-speed host controller for this link. The following sections describe the details of how the host controller must process and manage the split transaction protocol.

10.14.12.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an EPS field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full-, low-speed control are managed via queue heads in the asynchronous schedule.

Software must initialize the queue head with the appropriate device address and port number for the transaction translator that is serving as the full/low-speed host controller for the links connecting the endpoint. Software must also initialize the split transaction state bit (SplitXState) to Do-Start-Split. Finally, if the endpoint is a control endpoint, then system software must set the Control Transfer Type (C) bit in the queue head to a one. If this is not a control transfer type endpoint, the C bit must be initialized by software to be a zero. This information is used by the host controller to properly set the Endpoint Type (ET) field in the split transaction bus token. When the C bit is a zero, the split transaction token's ET field is set to indicate a bulk endpoint. When the C bit is a one, the split transaction token's ET field is set to indicate a control endpoint. Refer to Chapter 8 of USB Specification Revision 2.0 for details.

Figure 107. Host Controller Asynchronous Schedule Split-Transaction State Machine



10.14.12.1.1 Asynchronous — Do Start Split

This is the state when software must initialize a full- or low-speed asynchronous queue head. This state is entered from the Do Complete Split state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller executes a start-split transaction to the appropriate transaction translator. If the bus transaction completes without an error and PidCode indicates an IN or OUT transaction, then the host controller reloads the error counter (CErr). If it is a successful bus transaction and the PidCode indicates a SETUP, the host controller does not reload the error counter. If the transaction translator responds with a Nak, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response) the host controller decrements Cerr and proceeds to the next queue head in the asynchronous schedule.

10.14.12.1.2 Asynchronous — Do Complete Split

This state is entered from the Do Start Split state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the appropriate transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PidCode indicates an IN or OUT, the host controller reloads the error counter (CErr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PidCode indicates a SETUP, the host controller must not adjust the value of CErr.

Independent of PIDCode, the following responses have the effects:

- Transaction Error (XactErr). Timeout or data CRC failure, and so on. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller MUST ensure that the next time the host controller begins executing from the Asynchronous schedule, it must begin

executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (that were never delivered to the host). This is why the core specification states the retries must be immediate. A method to accomplish this behavior is to not advance the asynchronous schedule. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule is the retry for this endpoint. If Cerr went to zero, the host controller must halt the queue.

- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PidCode is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set to a one and the CErr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PidCode indicates an IN, then any of following responses are expected:

- DATA0/1. On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer, for example, move the data pointer by the number of bytes received, decrement BytesToTransfer field by the number of bytes received, and toggle the dt bit. The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited. If the PidCode indicates an OUT/SETUP, then any of following responses are expected:

- ACK. The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue (see Section 10.14.10, “Managing Control/Bulk/Interrupt Transfers via Queue Heads”).

10.14.12.2 Split Transaction Interrupt

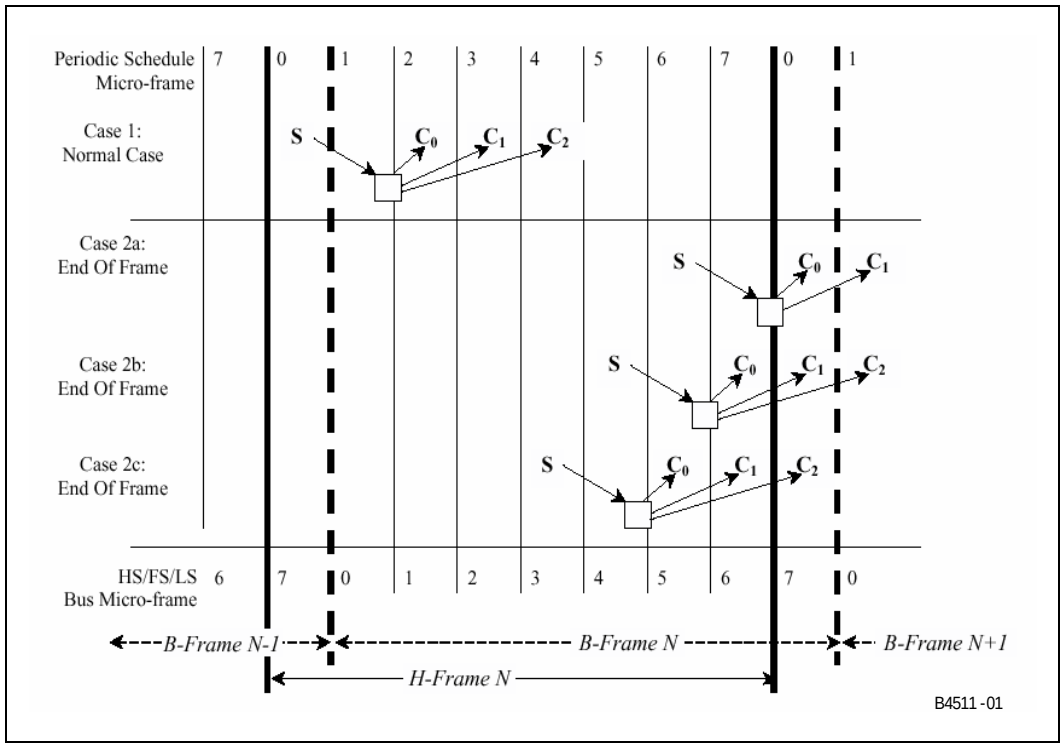
Split-transaction Interrupt-IN/OUT endpoints are managed via the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, that is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, for a high-speed endpoint, the host controller visits a queue head, execute a high-speed transaction (if criteria are met) and advance the transfer state (or not) depending on the results of the entire transaction. For low- and full-speed endpoints, the details of the execution phase are different (that is, takes more than one bus transaction to complete), but the remainder of the operational framework is intact. This means that the transfer advancement, and so forth occurs as defined in Section 10.14.10, “Managing Control/Bulk/Interrupt Transfers via Queue Heads”, but only occurs on the completion of a split transaction.

10.14.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly when micro-frames the start-splits and complete-splits for each endpoint occurs. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 108 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The S and $\mathbf{C}_X$ labels indicate micro-frames where software can schedule start-splits and complete splits, respectively.

Figure 108. Split Transaction, Interrupt Scheduling Boundary Conditions

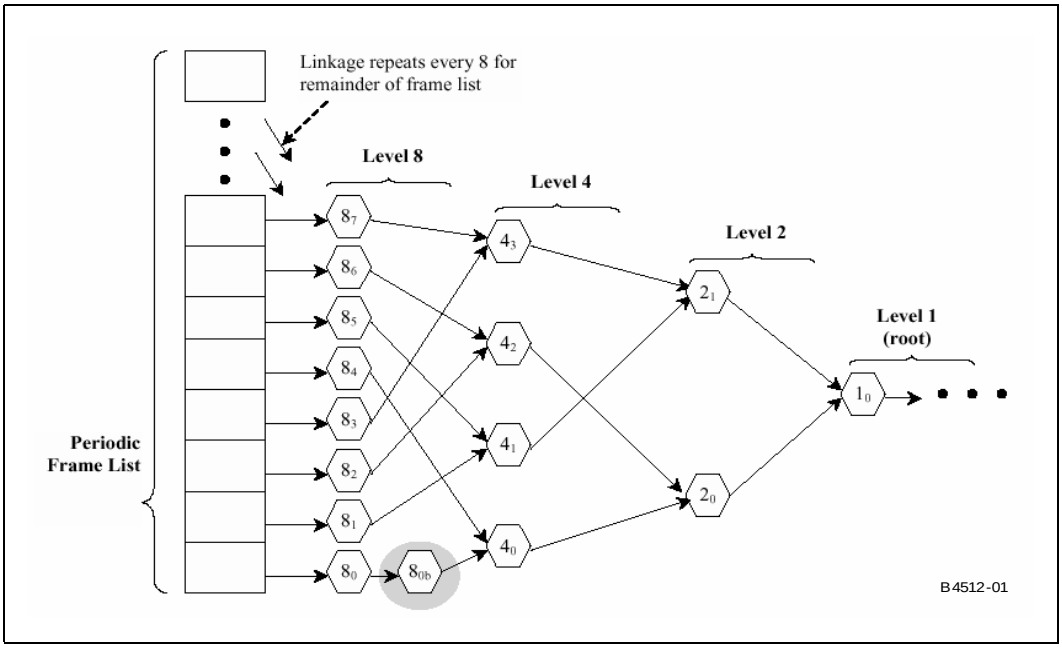


The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient

schedule that satisfies this requirement unless it uses FSTNs. Figure 109 illustrates the general layout of the periodic schedule.

Figure 109. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading



The periodic frame list is effectively the leaf level a binary tree, and is always traversed leaf to root. Each level in the tree corresponds to a 2N poll rate. Software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate eight queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 80b where such an endpoint. Without additional support on the interface, to get 80b reachable at the correct time, software would have to link 81 to 80b. It would then have to move 41 and everything linked after into the same path as 40. This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. Section "Host Controller Operational Model for FSTNs" defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by system software to instruct the host controller when to execute portions of the split-transaction protocol.

- SplitXState. This is a single bit residing in the Status field of a queue head (see Table 161). This bit is used to track the current state of the split transaction.
- Frame S-mask. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 108, case one, the S-mask would have a value of 00000001b indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Start, and the

current micro-frame as indicated by FRINDEX[2:0] is 0, then execute a start-split transaction.

- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 108, case one, the C-mask would have a value of 00011100b indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask

10.14.12.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure is used to manage Low/Full-speed interrupt queue heads that must be reached from consecutive frame list locations (that is, boundary cases 2a through 2c). An FSTN is essentially a back pointer, similar in intent to the back pointer field in the siTD data structure (see Section 10.13.4.5, “siTD Back Link Pointer”).

This feature provides software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in Section 10.13.7, “Periodic Frame Span Traversal Node (FSTN)”.
- A Save Place indicator. This is always an FSTN with its Back Path Link Pointer. T-bit set to zero.
- A Restore indicator. This is always an FSTN with its Back Path Link Pointer.T-bit set to a one.
- Host controller FSTN traversal rules.

Host Controller Operational Model for FSTNs

When the host controller encounters an FSTN during micro-frames 2 through 7 it follows the node’s Normal Path Link Pointer to access the next schedule data structure.

Note: The FSTN’s Normal Path Link Pointer.T-bit may set to a one, that the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in micro-frames 0 or 1, it saves the value of the Normal Path Link Pointer and set an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller’s rules for how it traverses the schedule and limits the data structures that is considered for execution of bus transactions. The host controller continues executing in Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the micro-frame (see details in the list below).

The rules for schedule traversal and limited execution as in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, synchronization executing in Recovery Path mode, it must never follow an FSTN’s Back Path Link Pointer.
- Do not process an siTD or, iTD data structure, follow its Next Link Pointer.

- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device. Follow its Horizontal Link Pointer.
- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete

Note: This applies whether the PID Code indicates an IN or an OUT.

- See Section 10.14.10.3, "Execute Transaction" and section "Tracking Split Transaction Progress for Interrupt Transfers" for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction.

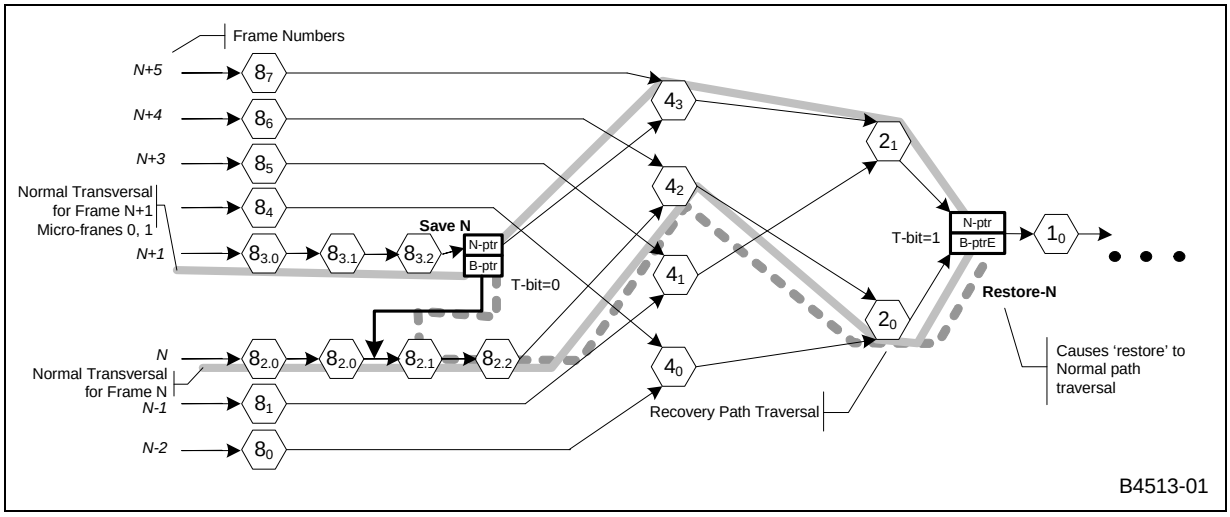
Note: The host controller must not execute a Start-split transaction when executing in Recovery Path mode. See section "Periodic Interrupt - Do Complete Split" for special handling when in Recovery Path mode.

- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in Figure 110.

Figure 110. Example Host Controller Traversal of Recovery Path via FSTNs



In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N.Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is

because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: {82.0, 82.1, 82.2, 82.3, 42, 20, Restore-N, 10 …}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters Save-Path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N.Normal Path Link Pointer and follows Save-N.Back Path Link Pointer. At the same time, it sets an internal flag indicating that it is now in Recovery Path mode (the recovery path is annotated in Figure 110 with a large dashed line).

The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set to a one (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {83.0, 83.1, 83.2, Save-A, **82.2, 82.3, 42, 20, Restore-N**, 43, 21, Restore-N, 10 …}. The nodes on the recovery-path are bold faced.

In frame N+1 (micro-frames 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {83.0, 83.1, 83.2, Save-A, 43, 21, Restore-N, 10 …}.

Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each Save-Place indicator requires a matching Restore indicator.

  The Save-Place indicator is an FSTN with a valid Back Path Link Pointer and T-bit equal to zero.

Note: Back Path Link Pointer.Typ field must be set to indicate the referenced data structure is a queue head. The Restore indicator is an FSTN with its Back Path Link Pointer.T-bit set to a one.

A Restore FSTN can be matched to one or more Save-Place FSTNs. For example, if the schedule includes a poll-rate 1 level, then system software must only place a Restore FSTN at the beginning of this list to match all possible Save-Place FSTNs.

- If the schedule does not have elements linked at a poll-rate level of one, and one or more Save-Place FSTNs are used, then System Software must ensure the Restore FSTN's Normal Path Link Pointer's T-bit is set to a one, as this is used to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one Save-Place FSTN reachable in any single frame.

Note: There are times when two (or more, depending on the implementation) could exist as full/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the Save-Place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

10.14.12.2.3 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue must be halted, thus signaling system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- C-prog-mask. This is an eight-bit, bit-vector where the host controller keeps track of the complete-splits that have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits must be executed in order. The host controller must detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number where the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- FrameTag. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- S-bytes. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

10.14.12.2.4 Split Transaction Execution State Machine for Interrupt

In the following presentation, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- cMicroFrameBit. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX

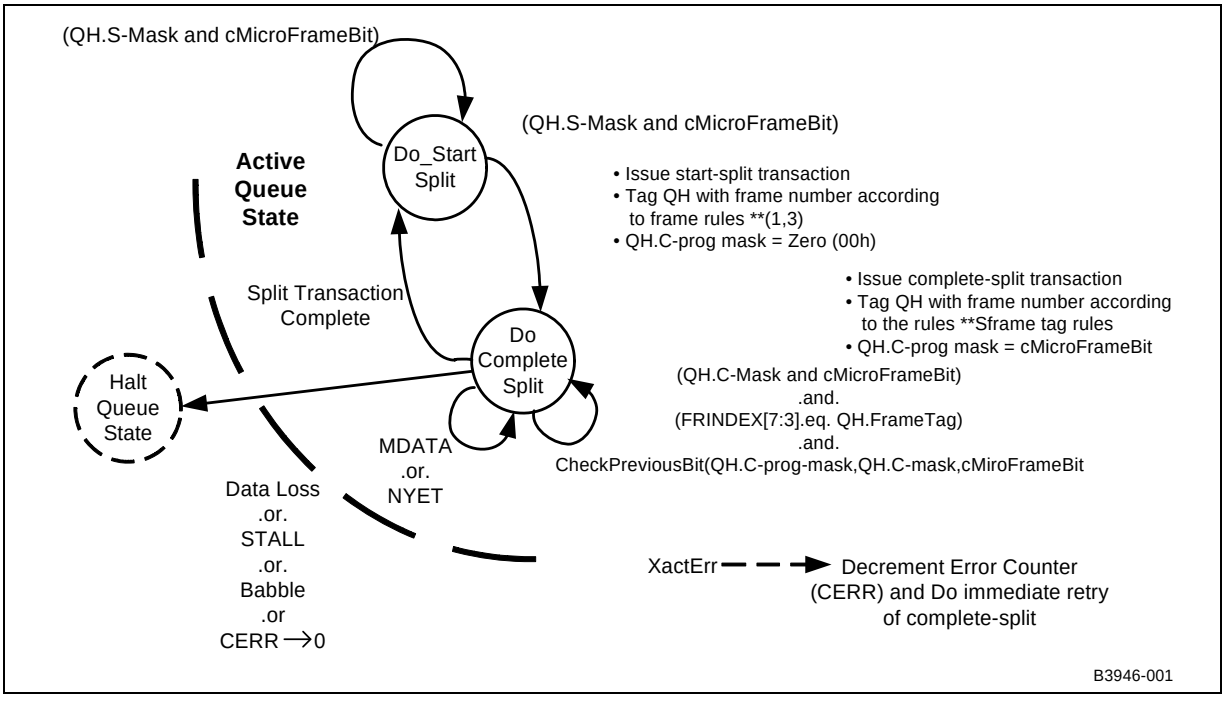
register (that is, cMicroFrameBit = (1 shifted-left(FRINDEX[2:0]))). The cMicroFrameBit has at most one bit asserted, and that always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit equals 00000001b.

The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 111 illustrates the state machine for managing a complete interrupt split transaction. There are two phases to each split transaction. The first is a single start-split transaction, that occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH.S-mask. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there are more than one complete-split transaction scheduled by software for the Do_Complete state. This translates to the fact that there are multiple bits set to a one in the QH.C-mask field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halt the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).

Figure 111. Split Transaction State Machine for Interrupt



See section "Managing QH.FrameTag Field" for the frame tag management rules.

Periodic Interrupt - Do Start Split

This is the state software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete. This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, and so on.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see section "Periodic Interrupt - Do Complete Split" for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it performs the following test to determine whether to execute a start-split.

- QH.S-mask is bit-wise ANDed with cMicroFrameBit (that is, the and of the two bits).

If the result is non-zero, then the host controller issues a start-split transaction. If the PIDCode field indicates an IN transaction, the host controller must zero-out the QH.S-bytes field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH.FrameTag field (see section "Managing QH.FrameTag Field"), set C-prog-mask to zero (00h), and exits this state.

Note: The host controller must not adjust the value of CErr as a result of completion of a start-split transaction.

Periodic Interrupt - Do Complete Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH.C-mask field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH.FrameTag is compared with the current contents of FRINDEX[7:3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```
Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
                If not(previousBit bitAND QH.C-prog-mask) then
                rvalue = FALSE;
                End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It probably gets caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
 rvalue = FALSE;
End if
return (rvalue)
End Algorithm
```

- Test D. Check to see if a start-split should be executed in this micro-frame.

Note: This is the same test performed in the Do Start Split state (see section "Periodic Interrupt - Do Complete Split"). Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A and B and C and not (D)), the host controller executes a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH.C-prog-mask by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH.FrameTag to the expected H-Frame number (see section "Managing QH.FrameTag Field"). The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects:

Note: Any responses that result in decrementing of the CErr results in the queue head being halted by the host controller if the result of the decrement is zero

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.

The test for whether this is the Last complete split can be performed by XOR QH.C-mask with QH.C-prog-mask. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set to a one and the CErr field is decremented.

- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust CErr on this response.
- Transaction Error (XactErr). Timeout, data CRC failure, and so on. The CErr field is decremented and the XactErr bit in the Status field is set to a one. The complete split transaction is immediately retried (if Cerr is non-zero).If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or CErr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and CErr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section full- and low-speed Interrupts) in the USB Specification Revision 2.0 for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller then exits this state for this queue head. The host controller must reload CErr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue (see Section 10.14.10, "Managing Control/Bulk/Interrupt Transfers via Queue Heads").

- MDATA. This response only occurs for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH.S-bytes. The host controller must not adjust CErr on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH.S-bytes. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue (see Section 10.14.10, "Managing Control/ Bulk/Interrupt Transfers via Queue Heads").

If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.

- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload CErr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set to a one, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is set to zero and the Halted bit is set to a one and the qTD is retired. Responses that are not enumerated in the list or that are received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. Table 182 lists the possible combinations and the appropriate action.

Table 182. Interrupt IN/OUT Do Complete Split State Execution Criteria (Sheet 1 of 2)

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current micro-frame.Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PIDCode is an IN, then the Queue head must be halted. If PIDCode is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect CERR. In either case, set the Missed Micro-frame bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PIDCode is an IN, then the Queue head must be halted. If PIDCode is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect CERR. In either case, set the Missed Micro-frame bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.

Table 182. Interrupt IN/OUT Do Complete Split State Execution Criteria (Sheet 2 of 2)

Condition	Action	Description
D	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PIDCode is an IN, then the Queue head must be halted. If PIDCode is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect CERR. In either case, set the Missed Micro-frame bit in the status field to a one. Note: When executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode.

Managing QH.FrameTag Field

The QH.FrameTag field in a queue head is completely managed by the host controller. The rules for setting QH.FrameTag are simple:

- Rule 1: If you are transitioning from Do Start Split to Do Complete Split and the current value of FRINDEX[2:0] is 6 QH.FrameTag is set to FRINDEX[7:3] + 1. This accommodates split transactions whose start-split and complete-splits are in different H-Frames (case 2a, see Figure 108).
- Rule 2: If the current value of FRINDEX[2:0] is 7, QH.FrameTag is set to FRINDEX[7:3] + 1. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c (Figure 108).
- Rule 3: If you are transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2:0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2:0]) is not 7, FrameTag is set to FRINDEX[7:3]. This accommodates all other cases (Figure 108).

10.14.12.2.5 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (that is, new S-mask and C-mask values).

It is imperative that System software must not update these masks to new values in the midst of a split transaction. To avoid any race conditions with the update, the EHCI host controller provides a simple assist to system software. System software sets the Inactivate-on-next-Transaction (I) bit to a one to signal the host controller that it intends to update the S-mask and C-mask on this queue head. System software then waits for the host controller to observe the I-bit is a one and transition the Active bit to a zero. The rules for how and when the host controller sets the Active bit to zero are enumerated below:

- If the Active bit is a zero, no action is taken. The host controller does not attempt to advance the queue when the I-bit is a one.
- If the Active bit is a one and the SplitXState is DoStart (regardless of the value of S-mask), the host controller sets Active bit to a zero. The host controller is not required to write the transfer state back to the current qTD.

Note: If the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction. It must set the Active bit to zero.

System software must save transfer state before setting the I-bit to a one. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set to a one and the host controller executed its final bus-transaction and set Active to a zero.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, system software must use the following algorithm to coherently re-activate a queue head that has been stopped via the I-bit.

5. Set the Halted bit to a one, then
6. Set the I-bit to a zero, then
7. Set the Active bit to a one and the Halted bit to a zero in the same write.

Setting the Halted bit to a one inhibits the host controller from attempting to advance the queue between the time the I-bit goes to a zero and the Active bit goes to a one.

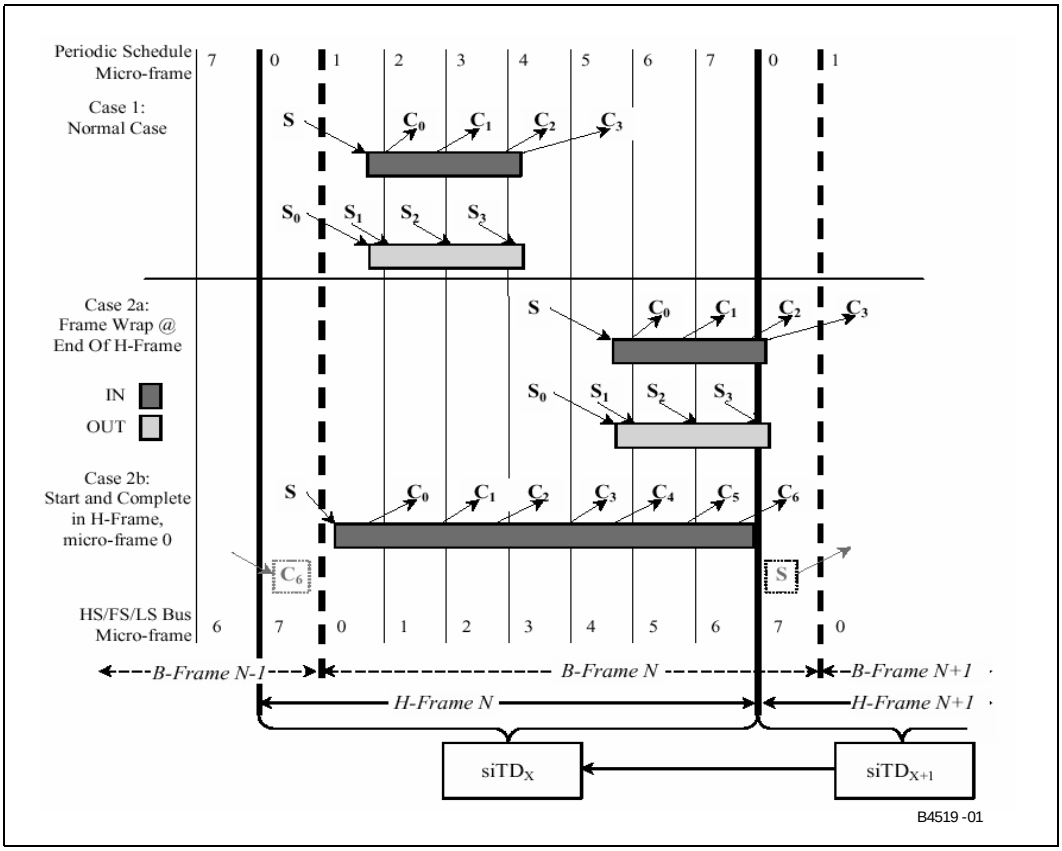
10.14.12.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The EHCI controller utilizes siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (iTD, Section 10.13.3, “Isochronous (High-Speed) Transfer Descriptor (iTD)”) (see Section 10.14.7, “Managing Isochronous Transfers Using iTDs” for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

10.14.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly when micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in section “Split Transaction Scheduling Mechanisms for Interrupt” apply. Figure 112 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The S_X and C_X labels indicate micro-frames where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of the figure illustrates the relationship of an siTD to the H-Frame.

Figure 112. Split Transaction, Isochronous Scheduling Boundary Conditions



When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example: the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list. (for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

  Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer, the use of it is described below.

  Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.
- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579-byte

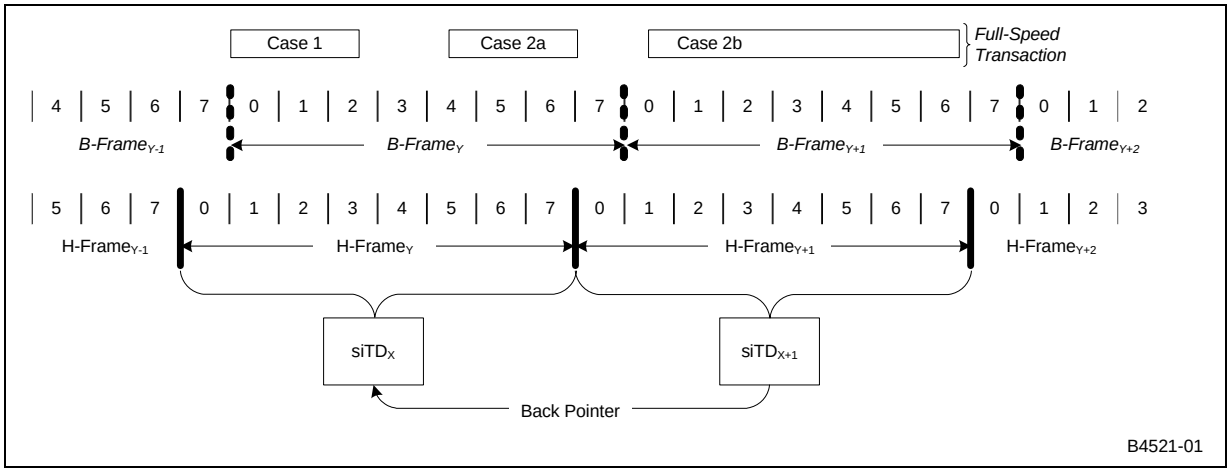
maximum packet size. A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol.

- SplitXState. This is a single bit residing in the Status field of an siTD (see Table 156). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in section "Split Transaction Execution State Machine for Isochronous".
- Frame S-mask. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 112, case one, the S-mask would have a value of 00000001b indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2:0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 112, case one, the C-mask would have a value of 00111100b indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN an OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. Figure 113 illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

Figure 113. siTD Scheduling Boundary Examples



Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. But, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTDX is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-FrameY+1, or micro-frame 0 of H-FrameY+2. The complete splits are scheduled using siTDX+2 (not shown). The complete-splits to extract this data must use the buffer pointer from siTDX+1. The only way for the host controller to reach siTDX+1 from H-FrameY+2 is to use siTDX+2's back pointer. The host controller rules for when to use the back pointer are described in section "Periodic Isochronous - Do Complete Split".

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so that it completes before the end of the B-Frame.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of H-Frame N and the last complete-split should occur in micro-frame 1 of H-Frame N+1. But, it is impossible to discriminate between cases 2a and case 2b, that has significant impact on the complexity of the host controller.

10.14.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where device to host data is lost. Isochronous endpoints do not employ the concept of a halt on error, but the host is required to identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs, for their transfers, and the data structures are only reachable via the schedule in the exact micro-frame where they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. But, it must ensure that the results of split transaction N are consumed and the siTD reinitialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate the portions of the split transaction data that have arrived. Control over the low-level protocol is exposed in an siTD via the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there are more than one start-split transaction, each of that require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host is not complete, that is detected and handled by the transaction translator. See section “Periodic Isochronous - Do Start Split” for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD.T-Count and siTD.TP are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- C-prog-mask. This is an eight-bit, bit-vector where the host controller keeps track of the complete-splits that have been executed. Due to the nature of the Transaction Translator periodic pipeline, the complete-splits must be executed in order. The host controller must detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2:0]) number where the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. Refer to section “Periodic Isochronous - Do Complete Split” for a description on how the state of the transfer is advanced. It is important to note that an IN siTD is retired based solely on the responses from the Transaction Translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID.

The number of bytes in the MDATA's data payload could cause the siTD field Total Bytes to Transfer to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example,

high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in setting of the Active bit to zero. But, in this case, the result has not been delivered by the Transaction Translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a Transaction Translator (see Chapter 11 of the Universal Serial Bus Revision 2.0).

In summary the periodic pipeline rules require that on a micro-frame boundary, the Transaction Translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and give the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the Transaction Translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the Transaction Translator responds with an MDATA and send all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement it's Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction to extract the results of the full-speed transaction from the Transaction Translator (for example, the Transaction Translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and react to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. But, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

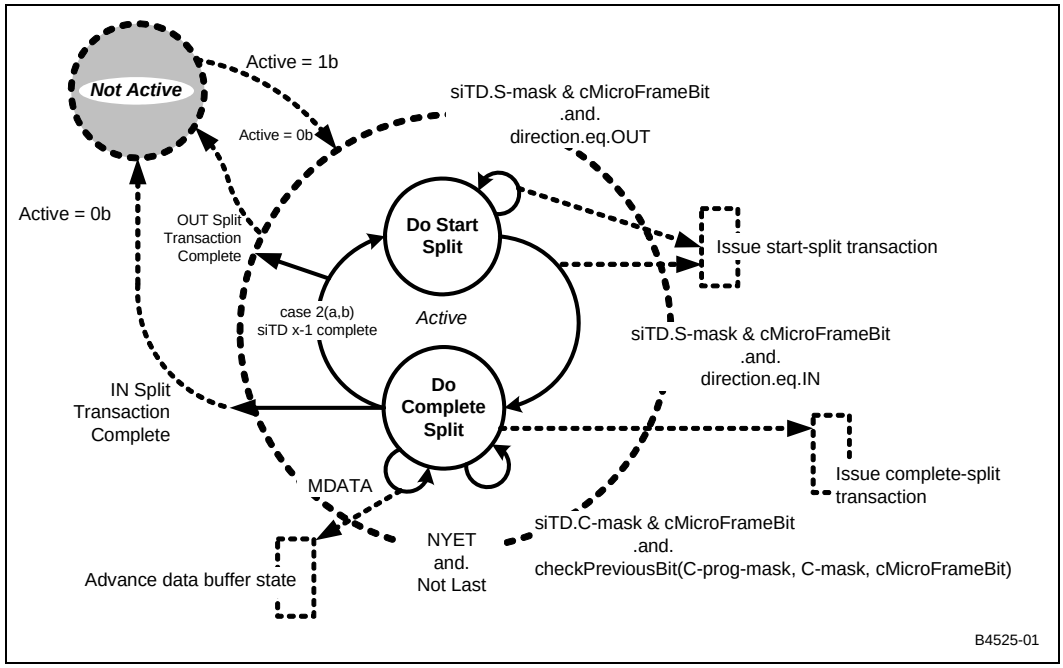
10.14.12.3.3 Split Transaction Execution State Machine for Isochronous

In the following presentation, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continue traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in section "Tracking Split Transaction Progress for Isochronous Transfers", plus the variable cMicroFrameBit defined in section "Split Transaction Execution State Machine for Interrupt" to track the progress of an isochronous split transaction.

Figure 114 illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place as a result of a transition or from being in a state.

Figure 114. Split Transaction State Machine for Isochronous



Periodic Isochronous - Do Start Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is initialized to this state, or the siTD transitions to this state from Do Complete Split when a case 2a (IN) or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the siTD.S-mask against cMicroFrameBit. If there is a one in the appropriate position, the siTD executes a start-split transaction. By definition, the host controller cannot reach an siTD at the wrong time. If the I/O field indicates an IN, then the start-split transaction includes only the extended token plus the full-speed token. Software must initialize the siTD.Total Bytes To Transfer field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (that is, the I/O field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating siTD.Current Offset with the page pointer indicated by the page selector field (siTD.P). A zero in this field selects Page 0 and a 1 selects Page 1.

During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the siTD.P-bit from a zero to a one, and begin using the siTD.Page 1 with siTD.Current Offset as the memory address pointer. The field siTD.TP is used to annotate each start-split transaction with the indication of the part of the split-transaction data that the current payload represents (ALL, BEGIN, MID, END). In all cases the host controller uses the value in siTD.TP to mark the start-split with the correct transaction position code.

T-Count is always initialized to the number of start-splits for the current frame. TP is always initialized to the first required transaction position identifier. The scheduling boundary case (see Figure 113) is used to determine the initial value of TP. The initial cases are summarized in Table 183.

Table 183. Initial Conditions for OUT siTD's TP and T-count Fields

Case	T-count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates T-Count and TP appropriately so that the next start-split is correctly annotated. Table 184 illustrates all of the TP and T-count transitions, that must be accomplished by the host controller.

Table 184. Transaction Position (TP) / Transaction Count (T-Count) Transition Table

TP	T-count Next	TP Next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when T-count starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when T-count starts at greater than 2.
MID	!=1	MID	TP stays at MID as T-count is not equal to 1 (that is, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The siTD.Total Bytes To Transfer and the siTD.Current Offset fields are adjusted to reflect the number of bytes transferred.
- The siTD.P (page selector) bit is updated appropriately.
- The siTD.TP and siTD.T-count fields are updated appropriately as defined in Table 184.

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must set the Active bit to a zero when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the initial condition is Total Bytes to Transfer equal to zero and T-count is equal to a one, then the host controller issues a single start-split, with a zero-length data payload. Software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination yields undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer does not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in section "Split Transaction for Isochronous — Processing Examples".

A host controller implementation can optionally track the progress of an OUT split transaction by setting appropriate bits in the siTD.C-prog-mask as it executes each scheduled start-split. The checkPreviousBit() algorithm defined in section "Periodic Isochronous - Do Complete Split" can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then set the siTD's Active bit to zero and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

Periodic Isochronous - Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on the micro-frame that the host controller is currently executing, and means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with siTD.C-mask field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.
- Test B. The siTD.C-prog-mask bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is below (this is different than the algorithm used in section "Periodic Interrupt - Do Complete Split"). The sequence where this test is applied depends on the current value of FRINDEX[2:0]. If FRINDEX[2:0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

```
Algorithm Boolean CheckPreviousBit (siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)

Begin

    Boolean rvalue = TRUE;

    previousBit = cMicroFrameBit rotate-right(1)

    -- Bit-wise anding previousBit with C-mask indicates whether
there was an intent

    -- to send a complete split in the previous micro-frame. So, if      the
'previous bit' is set in C-mask, check C-prog-mask to make   sure it happened.

    if previousBit bitAND siTD.C-mask then

        if not (previousBit bitAND siTD.C-prog-mask) then

        rvalue = FALSE

        End if

    End if

    Return rvalue

End Algorithm
```

If Test A is true and FRINDEX[2:0] is zero or one, then this is a case 2a or 2b scheduling boundary (see Figure 112). See section "Periodic Isochronous - Do Complete Split" for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH.C-prog-mask by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD.Total Bytes To Transfer,
- Adjust siTD.Current Offset by the number of bytes received,
- Adjust siTD.P (page selector) field if the transfer caused the host controller to use the next page pointer, and
- Set any appropriate bits in the siTD.Status field, depending on the results of the transaction.

Note: If the host controller encounters a condition where siTD.Total Bytes To Transfer is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD.Status.Active bit must be set to zero and the siTD.Status.Babble Detected bit must be set to a one. The fields siTD.Total Bytes To Transfer, siTD.Current Offset, and siTD.P (page selector) are not required to be updated as a result of this transaction attempt.

The host controller must accept (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD.Total Bytes To Transfer) MDATA and DATA0/1 data payloads up to and including 192 bytes. A host controller implementation may optionally set siTD.Status Active to a zero and siTD.Status.Babble Detected to a one when it receives and MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD.Status field and sets the Active bit to a zero.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, and so on. The siTD.Status field XactErr field is set to a one and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller does not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the Active bit is set to zero.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is set to a zero. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USBINT status bit in the USBSTS register to a one. The host controller does not detect this condition.
- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in section "Periodic Interrupt - Do Complete Split". If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is set to a zero. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last). See above description for testing for Last. This can only occur when there is an error condition. There has been a babble condition on the full-speed link, that delayed the completion of the full-speed transaction, or software set up the S-mask and/or C-masks incorrectly. The host controller must set XactErr bit to a one and the Active bit is set to a zero.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and sets the Active bit to a zero.

Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see Figure 112) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. Table 185 enumerates the transaction state fields.

Table 185. Summary siTD Split Transaction State

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) T-count (transaction count)	All bits in the status field	C-prog-mask

Note: TP and T-count are used only for Host to Device (OUT) endpoints.

If software has budgeted the schedule of this data stream with a frame wrap case, then it must initialize the siTD.Back Pointer field to reference a valid siTD and has the siTD.Back Pointer.T-bit in the siTD.Back Pointer

field set to a zero. Otherwise, software must set the siTD.Back Pointer.T-bit in the siTD.Back Pointer field to a one. The host controller's rules for interpreting when to use the siTD.Back Pointer field are listed below. These rules apply only when the siTD's Active bit is a one and the SplitXState is Do Complete Split.

- When cMicroFrameBit is a 1h and the siTDX.Back Pointer.T-bit is a zero, or
- If cMicroFrameBit is a 2h and siTDX.S-mask[0] is a zero

When either of these conditions apply, then the host controller must use the transaction state from siTDX-1.

To access siTDX-1, the host controller reads on-chip the siTD referenced from siTDX.Back Pointer.

The host controller must save the entire state from siTDX when processing siTDX-1. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD.Back Pointers.

If siTDX-1 is active (Active bit is a one and SplitXStat is Do Complete Split), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see Table 185) of siTDX-1 is appropriately advanced based on the results and written back to memory. If the resultant state of siTDX-1's Active bit is a one, then the host controller returns to the context of siTDX, and follows its next pointer to the next schedule item. No updates to siTDX are necessary.

If siTDX-1 is active (Active bit is a one and SplitXStat is Do Start Split), then the host controller must set Active bit to a zero and Missed Micro-Frame status bit to a one and the resultant status written back to memory.

If siTDX-1's Active bit is a zero, (because it was zero when the host controller first visited siTDX-1 via siTDX's back pointer, it is transitioned to zero as a result of a detected error, or the results of siTDX-1's complete-split transaction transitioned it to zero), then the host controller returns to the context of siTDX and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is a 1b and siTDX.S-mask[0] is a 1b). If this criterion is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of siTDX, then follows siTDX.Next Pointer to the next schedule item. If the criterion is not met, the host controller follows siTDX.Next Pointer to the next schedule item.

Note: In the case of a 2b boundary case, the split-transaction of siTDX-1 has its Active bit set to zero when the host controller returns to the context of siTDX. Also, note that

software should not initialize an siTD with C-mask bits 0 and 1 set to a one and an S-mask with bit zero set to a one. This scheduling combination is not supported and the behavior of the host controller is undefined.

10.14.12.3.4 Split Transaction for Isochronous — Processing Examples

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced via the Execute Transaction queue head traversal state machine (see Figure 105).

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that should occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, Table 186 illustrates a couple of frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 186. Example Case 2a - Software Scheduling siTDs for an IN Endpoint

siTDX		Micro-Frames								Initial SplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 10h (a one bit in micro-frame 4) and C-mask value of C3h (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the Back Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are set to zero).

The initial SplitXState of the first siTD is Do Start Split. The host controller visits the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has it's SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it visits the second siTD eight times. During micro-frames 0 and 1 it detects that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that siTDX+1's Back Pointer.T-bit is a zero, saves the state of siTDX+1 and fetches siTDX. It executes the complete split transaction using the transaction state of siTDX. If the siTDX split transaction is complete, siTD's Active bit is set to zero and results written back to siTDX.

The host controller retains the fact that siTDX is retired and transitions the SplitXState in the siTDX+1 to Do Start Split. At this point, the host controller is prepared to execute the start-split for siTDX+1 when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in section "Periodic Isochronous - Do Complete Split"), that is, before all the scheduled complete-splits have been executed, the host controller transitions siTDX.SplitXState to Do Start Split early and naturally skip the remaining scheduled complete-split transactions. For this example, siTDX+1 does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame X+2, micro-frame 0, the host controller detects that siTDX+2's Back Pointer.T-bit is a zero, saves the state of siTDX+2 and fetches siTDX+1. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the Active bit. The host controller returns to the context of siTDX+2, and traverses it's next pointer without any state change updates to siTDX+2. S

During H-Frame X+2, micro-frame 1, the host controller detects siTDX+2's S-mask[0] is a zero, saves the state of siTDX+2 and fetches siTDX+1. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and sets the Active bit to a zero. It returns to the state of siTDX+2 and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for siTDX+2 when it reaches micro-frame 4.

10.14.13 Host Controller Pause

When the host controller's HCHalted bit in the USBSTS register is a zero, the host controller is sending SOF (Start OF Frame) packets down all enabled ports. When the schedules are enabled, the EHCI host controller accesses the schedules in main memory each micro-frame. This constant pinging of main memory is known to create CPU power management problems for mobile systems. Specifically, mobile systems aggressively manage the state of the CPU, based on recent history usage. In the more aggressive power saving modes, the CPU can disable its caches. Current PC architectures assume that bus-master accesses to main memory must be cache-coherent. So, when bus masters are busy touching memory, the CPU power management software can detect this activity over time and inhibit the transition of the CPU into its lowest power savings mode. USB controllers are bus-masters and the frequency at which they access their memory-based schedules keeps the CPU power management software from placing the CPU into its lowest power savings state.

USB Host controllers don't access main memory when they are suspended. But, there are a various reasons why placing the USB controllers into suspend won't work, but they are beyond the scope of this document. The base requirement is that the USB controller must be kept out of main memory, at the same time, the USB bus is kept from going into suspend.

EHCI controllers provide a large-grained mechanism that can be manipulated by system software to change the memory access pattern of the host controller. System software can manipulate the schedule enable bits in the USBCMD register to turn on/off the scheduling traversal. A software heuristic can be applied to implement an on/off duty cycle that allows the USB to make reasonable progress and allow the CPU power management to get the CPU into its lowest power state. This method is not intended to be applied at all times to throttle USB, but should only be applied in very specific configurations and usage loads. For example, when only a keyboard or mouse is

attached to the USB, the heuristic could detect times when the USB is attempting to move data only very infrequently and can adjust the duty cycle to allow the CPU to reach it's low power state for longer periods of time. Similarly, it could detect increases in the USB load and adjust the duty cycle appropriately, even to the point where the schedules are never disabled. The assumption here is that the USB is moving data and the CPU is required to process the data streams.

It is suggested that to provide a complete solution for the system, the companion host controllers should also provide a similar method to allow system software to inhibit the companion host controller from accessing it's shared memory based data structures (schedule lists or otherwise).

10.14.14 Port Test Modes

EHCI host controllers must implement the port test modes Test J_State, Test K_State, Test_Packet, Test Force_Enable, and Test SE0_NAK as described in the USB Specification Revision 2.0. The system is only allowed to test ports that are owned by the EHCI controller. System software is allowed to have at most one port in test mode at a time. Placing more than one port in test mode yields undefined results. The required, per port test sequence is:

- Disable the periodic and asynchronous schedules by setting the Asynchronous Schedule Enable and Periodic Schedule Enable bits in the USBCMD register to a zero.
- Place all enabled root ports into the suspended state by setting the Suspend bit in each appropriate PORTSC register to a one.
- Set the Run/Stop bit in the USBCMD register to a zero and wait for the HCHalted bit in the USBSTS register, to transition to a one.

Note: An EHCI host controller implementation may optionally allow port testing with the Run/Stop bit set to a one. But, all host controllers must support port testing with Run/Stop set to a zero and HCHalted set to a one.

- Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then the Run/Stop bit in the USBCMD register must then be transitioned back to one, to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (HCHalted bit is a one) then it terminates and exits test mode by setting HCReset to a one.

10.14.15 Interrupts

The EHCI Host Controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, and so on.), and
- Host Controller error events

All transaction-based sources are maskable through the Host Controller's Interrupt Enable register (USBINTR, see Section 10.12.3, "USBINTR"). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts can be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the Interrupt Threshold Control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer does not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller does not generate interrupts any more frequently than once every eight micro-frames.

Section "Host System Error" details effects of a host system error.

If an interrupt has been scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to host memory. This may sometimes result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The exact mechanism to accomplish the transfer is OS specific. For this discussion it is assumed that control is received.

When the interrupt handler receives control, its first action is to reads the USBSTS (USB Status Register). It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism), schedules a deferred procedure call (DPC) that executes later. The DPC routine processes the results of the schedule execution. The exact mechanisms used are beyond the scope of this document.

Note: The host controller is not required to de-assert a currently active interrupt condition when software sets the interrupt enables (in the USBINR register, see Section 10.12.3, "USBINTR") to a zero. The only reliable method the software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register (Section 10.12.2, "USBSTS") from a one to a zero.

10.14.15.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

10.14.15.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. Table 187 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the CErr field is decremented. When the PIDCode indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set to a one and the CErr field being decremented.

- EPS field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device and it returns a Nyet handshake to a SETUP.

- EPS field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

Table 187. Summary of Transaction Errors

Event / Result	Queue Head / qTD / iTD / siTD Side-effects		USB Status Register (USBSTS)
	Cerr	Status Field	USBERRINT
CRC	-1	XactErr set to a one.	1[1]
Timeout	-1	XactErr set to a one.	1[1]
Bad PID[2]	-1	XactErr set to a one.	1[1]
Babble	N/A	Section "Serial Bus Babble"	1
Buffer Error	N/A	Section "Data Buffer Error"	

Notes:
1. If occurs in a queue head, then USBERRINT is asserted only when CErr counts down from a one to a zero. In addition the queue is halted, see section "Halting a Queue Head".
2. The host controller received a response from the device, but it could not recognize the PID as a valid PID.

Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a Packet Babble. When a device sends more data than the Maximum Length number of bytes, the host controller sets the Babble Detected bit to a one and halts the endpoint if it is using a queue head (see Section 10.14.10.3.1, "Halting a Queue Head"). Maximum Length is defined as the minimum of Total Bytes to Transfer and Maximum Packet Size. The CErr field is not decremented for a packet babble condition (only applies to queue heads). A babble condition also exists if IN transaction is in progress at High-speed EOF2 point. This is called a frame babble. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port that the frame babble is detected.

The USBERRINT bit in the USBSTS register is set to a one and if the USB Error Interrupt Enable bit in the USBINTR register is a one, then a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

Note: When a host controller detects a data PID mismatch, it must disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (that is, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, to advance the transmitter's data sequence.

The EHCI interface allows System software to provide buffers for a Control, Bulk or Interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. To properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

Data Buffer Error

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors, and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the Data Buffer Error bit in the queue head, iTD or siTD.

If the data buffer error occurs on a non-isochronous IN, the host controller does not issue a handshake to the endpoint. This forces the endpoint to re-send the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Truncating the packet is not considered acceptable. An acceptable implementation option is to 1's complement the CRC bytes and send them. There are other options suggested in the Transaction Translator section of the USB Specification Revision 2.0.

10.14.15.1.2 USB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (iTDs, siTDs, and queue heads (qTDs)) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes the USB Interrupt (USBINT) bit in the USBSTS register to be set to a one. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes USBINT to be set to a one. If the USB Interrupt Enable bit in the USBINTR register is set to a one, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, the USBERRINT bit in the USBSTS register is also set to a one.

10.14.15.1.3 Short Packet

Reception of a data packet that is less than the endpoint's Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer. Whenever a short packet completion occurs during a queue head execution, the USBINT bit in the USBSTS register is set to a one. If the USB Interrupt Enable bit is set in the USBINTR register, a hardware interrupt is signaled to the system at the next interrupt threshold.

10.14.15.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance, see section "Interrupt on Async Advance").

10.14.15.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set to a one, the host controller sets the Port Change Detect bit in the USBSTS register to a one. If the Port Change Interrupt Enable bit in the USBINTR register is a one, then the host controller issues a hardware interrupt. The port status change bits include:

- Connect Status Change
- Port Enable/Disable Change
- Over-current Change
- Force Port Resume

10.14.15.2.2 Frame List Rollover

This event indicates that the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. If the frame list size is 1024, then the interrupt occurs every 1,024 ms, if it is 512, then it occurs every 512 ms, and so on. When a frame list rollover is detected, the host controller sets the Frame List Rollover bit in the USBSTS register to a one. If the Frame List Rollover Enable bit in the USBINTR register is set to a one, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

10.14.15.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. Whenever the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of the Interrupt on Async Advance Doorbell bit in the USBCMD register. If it is a one, it sets the Interrupt on Async Advance bit in the USBSTS register to a one. If the Interrupt on Async Advance Enable bit in the USBINTR register is a one, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in Section 10.14.8.2, “Removing Queue Heads from Asynchronous Schedule”.

10.14.15.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error can be catastrophic to the host controller (such as a Master Abort) making it impossible for the host controller to continue in a coherent fashion. In the presence of non-catastrophic host errors, such as parity errors, the host controller could potentially continue operation. The recommended behavior for these types of errors is to escalate it to a catastrophic error and halt the host controller. Host-based error must result in the following actions:

- The Run/Stop bit in the USBCMD register is set to a zero.
- The following bits in the USBSTS register are set:
  - — Host System Error bit is to a one.
  - — HCHalted bit is set to a one.
- If the Host System Error Enable bit in the USBINTR register is a one, then the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold. Table 188 summarizes the required actions taken on the various host errors.

Table 188. Summary Behavior of EHCI Host Controller on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal †
siTD fetch (read)	Fatal	Fatal	Fatal †
siTD status write-back (write)	Fatal †	Fatal †	Fatal †
iTD fetch (read)	Fatal	Fatal	Fatal †
iTD status write-back (write)	Fatal †	Fatal †	Fatal †
qTD fetch (read)	Fatal	Fatal	Fatal †
qHD status write-back (write)	Fatal †	Fatal †	Fatal†
Data write	Fatal†	Fatal †	Fatal †
Data read	Fatal	Fatal	Fatal †
† Potentially, a host controller implementation could continue operation without a halt. But, the recommended behavior is to halt the host controller.			

Note: After a Host System Error, Software must reset the host controller via HCReset in the USBCMD register before re-initializing and restarting the host controller.

10.15 EHCI Deviation

For the purposes a dual-role Host/Device controller with support for On-The-Go applications, it is necessary to deviate from the EHCI specification. Device operation and On-The-Go operation is not specified in the EHCI and thus the implementation supported in this core is proprietary. The host mode operation of the core is near EHCI compatible with few minor differences documented in this section.

The particulars of the deviations occur in the areas summarized here:

- Embedded Transaction Translator — Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation — In host mode the device operational registers are generally disabled and thus device mode is mostly transparent when in host mode. But, there are a couple of exceptions documented in the following sections.
- Embedded design interface — This core does not support a PCI interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.
- On-The-Go Operation — This design includes an On-The-Go controller for Port #1.

10.15.1 Embedded Transaction Translator Function

The OTG controller supports directly connected full and low speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high speed hub transaction translator. Although there is no separate Transaction Translator block in the system, the transaction translator function normally associated with a high speed hub has been implemented within the DMA and Protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models that exist in the EHCI specification to support full and low speed devices.

10.15.1.1 Capability Registers

The following additions have been added to the capability registers to support the embedded Transaction Translator Function:

- N_TT added to Table 132.
- N_PTT added to Table 132.

See Section 10.11.3, “HCSPARAMS – EHCI Compliant with Extensions” with extensions for usage information.

10.15.1.2 Operational Registers

The following additions have been added to the operational registers to support the embedded TT:

- TTCTRL is a new register.
- Addition of two-bit Port Speed (PSPD) to the PORTSCx register.

10.15.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a Full speed (FS) or Low speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable is only set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (that is, chirp completes successfully).

Since this controller has an embedded Transaction Translator, the port enable is always set after the port reset operation regardless of the result of the host device chirp result and the resulting port speed is indicated by the PSPD field in PORTSCx.

Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs.

The change is a fundamental one in that is summarized in the next table.

Table 189. Standard EHCI vs. EHCI with Embedded Transaction Translator

Standard EHCI	EHCI with Embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSCx.
FS and LS devices are assumed to be downstream from a HS hub thus, all port-level control is performed through the Hub Class to the nearest hub.	FS and LS device can be downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done using the Hub Class through the nearest hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSCx.
FS and LS devices are assumed to be downstream from a HS hub with HubAddr=X. [where HubAddr > 0 and HubAddr is the address of the hub where the bus transitions from HS to FS/LS (that is, Split target hub)]	FS and LS device can be downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = TTHA (TTHA is programmable and default to 0) and HubAddr is the address of the Root hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)]

10.15.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the Root hub with sm embedded Transaction Translator. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

- QH (for direct attach FS/LS) – Async. (Bulk/Control Endpoints) Periodic (Interrupt)
  - — Hub Address = TTHA (default TTHA = 0)
  - — Transactions to direct attached device/hub.
- QH.EPS = Port Speed
  - — Transactions to a device downstream from direct attached FS hub.
- QH.EPS = Downstream Device Speed

Note: When QH.EPS = 01 (LS) and PORTSCx.PSPD = 00 (FS), a LS-pre-pid is sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

- siTD (for direct attach FS) – Periodic (ISO Endpoint)
  - — All FS ISO transactions:
  - • Hub Address = (default TTHA = 0)
  - • siTD.EPS = 00 (full speed)

Maximum Packet Size must less than or equal to 1,023 or undefined behavior may result.

10.15.1.5 Operational Model

The operational models are well defined for the behavior of the Transaction Translator (see USB 2.0 specification) and for the EHCI controller moving packets between system memory and a USB-HS hub. Since the embedded Transaction Translator exists within the host controller there is no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections discusses the operational model for how the EHCI and Transaction Translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 Transaction Translator operational models.

10.15.1.5.1 Micro-Frame Pipeline

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the Host (H) and the Bus (B). The embedded Transaction Translator uses the same pipeline algorithms specified in the USB 2.0 specification for a hub-based Transaction Translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the micro-frame pipeline implemented in the embedded Transaction Translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 is ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded Transaction Translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream hub-based Transaction Translators.

Once periodic transfers are exhausted, any stored asynchronous transfer is moved. Asynchronous transfers are opportunistic in that they execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer cannot babble through the SOF (start of B-frame 0.)

10.15.1.5.2 Split-State Machines

The start and complete split operational model differs from EHCI slightly because there is no bus medium between the EHCI controller and the embedded Transaction Translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete split operation is simple an internal operation to the embedded Transaction Translator. Table 190 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 190. Condition vs. Emulate TT Response

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full.	NAK
Start-Split: All periodic buffers full.	ERR
Start-Split: Success for start of Async. Transaction.	ACK
Start-Split: Start Periodic Transaction.	No Handshake (Ok)
Complete-Split: Failed to find transaction in queue.	Bus Time Out
Complete-Split: Transaction in Queue is Busy.	NYET
Complete-Split: Transaction in Queue is Complete.	[Actual Handshake from LS/FS device]

The unshaded cells represent Start-Splits and the shaded cells represent Complete-Splits.

10.15.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

USB 2.0 – 11.17.3

- Sequencing is provided & a packet length estimator ensures no full-speed/low-speed packet babbles into SOF time.

USB 2.0 – 11.17.4

- Transaction tracking for 2 data pipes.

USB 2.0 – 11.17.5

- Clear_TT_Buffer capability provided though the use of the TTCTRL register.

10.15.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

USB 2.0 – 11.18.6.[1-2]

- Abort of pending start-splits
  - EOF (and not started in micro-frames 6)
  - Idle for more than 4 micro-frames
- Abort of pending complete-splits
  - EOF
  - Idle for more than 4 micro-frames

USB 2.0 – 11.18.[7-8]

- Transaction tracking for up to 16 data pipes.
  - Some applications may not require transaction tracking up to a maximum of 16 periodic data pipes. The option to limit the tracking to only 4 periodic data pipes exists in the by changing the configuration constant VUSB_HS_TT_PERIODIC_CONTEXTS to 4. The result is a significant gate count savings to the core given the limitations implied.

Warning: Limiting the number of tracking pipes in the EMBedded –TT to four imposes the restriction that no more than four periodic transactions (INTERRUPT/ISOCHRONOUS) can be scheduled through the embedded-tt per frame. The number 16 was chosen in the USB specification because it is sufficient to ensure that the high-speed to full-speed periodic pipeline can remain full. Keeping the pipeline full puts no constraint on the number of periodic transactions that can be scheduled in a frame and the only limit becomes the flight time of the packets on the bus.

- Complete-split transaction searching.

Note: There is no data schedule mechanism for these transactions other than the micro-frame pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 ms) or else undefined behavior may result.

10.15.1.5.5 Multiple Transaction Translators

The maximum number of embedded Transaction Translators that is currently supported is one as indicated by the N_TT field in Table 132.

10.15.2 Device Operation

The co-existence of a device operational controller within the host controller has little effect on EHCI compatibility for host operation except as noted in this section.

10.15.2.1 USBMODE Register

Given that the dual-role controller is initialized not in host nor device mode, the USBMODE register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

10.15.2.2 EHCI Reserved Fields

Some of the reserved fields and reserved addresses in the capability registers and operational register are used in device mode, therefore you must adhere to the following:

- Write operations to all EHCI reserved fields (some of them are device fields) with the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the host controller must properly mask EHCI reserved fields (some of them are device fields) because fields that are used exclusively for the device are undefined in host mode.

10.15.2.3 SOF Interrupt

This SOF Interrupt used for device mode is shared as a free-running, 125-us interrupt for host mode. EHCI does not specify this interrupt but it has been added for convenience and as a potential software time base. See Section 10.12.2, “USBSTS” and Section 10.12.3, “USBINTR”.

10.15.3 Embedded Design Interface

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

10.15.3.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like is provided by the Frame Adjust register in the PCI configuration registers. Starts of micro-frames are timed exactly to 125 us using the transceiver clock as a reference clock. For example, using a 60-MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or using a 30-MHz transceiver clock for 16-bit physical interfaces.

10.15.4 Miscellaneous Variations from EHCI

10.15.4.1 Programmable Physical Interface Behavior

This design supports multiple Physical interfaces that can operate in differing modes when the core is configured with software programmable Physical Interface Modes. See Configuration Constants. Software programmability allows the selection of the Physical

interface part during the board design phase instead of during the chip design phase. The control bits for selecting the Physical Interface operating mode have been added to the PORTSCx register providing a capability that is not defined by EHCI.

10.15.4.2 Discovery

10.15.4.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the PORTSCx register for a duration of 10ms. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10ms reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software writes a '1' to the reset the device.
- Software writes a '0' to the reset the device after 10 ms.
  - This step, that is necessary in a standard EHCI design can be omitted with this implementation. Should the EHCI host controller driver attempt to write a '0' to the reset bit as a reset is in progress the write is ignored and the reset continues until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

10.15.4.2.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation that re-assigns the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non High-Speed devices. Therefore, the following differences are important regarding port speed detection:

- Port Owner is read-only and always reads 0.
- A 2-bit Port Speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit High Speed indicator has been added to PORTSC to signify that the port is in High-Speed vs. Full/Low Speed.This information is redundant with the 2-bit Port Speed indicator above.

10.15.4.3 Port Test Mode

Port Test Control mode behaves fully as described in EHCI since the release of revision 3.2.1. In earlier product revisions, the test packet mode was not EHCI compatible. An alternate host controller driver procedure is no longer necessary or supported.

§ §

11.0 Memory Controller

This chapter describes the integrated Memory Controller Unit (DDR MCU) of the Intel® IXP43X Product Line of Network Processors. The operating modes, initialization, external interfaces, and implementation are detailed in this chapter.

11.1 Overview

The IXP43X network processors integrate a high performance, multi-ported Memory Controller to provide direct interface between the processor and its local memory subsystem. The memory controller supports:

- 128/256/512-Mbit, 1-Gbit DDR-I SDRAM and 256/512-Mbit DDR-II SDRAM technology support.
- Un-buffered DRAM support only, no registered DRAM support
- Dedicated port for Intel XScale® Processor to DDR-I/II SDRAM (Supports critical word first reads).
- Between 32 Mbytes and 1 Gbytes of 32-bit DDR-I SDRAM
- Between 64 Mbytes and 512 Mbytes of 32-bit DDR-II SDRAM
- 16 Mbytes of 16-bit DDR-I SDRAM (support 128Mbit technology only)
- 32 / 64 Mbytes of 16-bit DDR-II SDRAM (support 256Mbit / 512Mbit technology)
- Two AHB ports for access from units other than Intel XScale processor (no critical word first support)
- All MMR accesses must go through the South AHB port.
- Single-bit error correction, multi-bit detection support (ECC).
- 32-, 40-bit wide Memory Interfaces (non-ECC and ECC support), and 16-bit wide Memory Interfaces (non-ECC support).

The DDR-I/II SDRAM interface provides a direct connection to a reliable, high bandwidth memory subsystem. The DDR-I/II SDRAM interface consists of a 16-bit/32-bit wide data path to support up to 1.6 GBytes/sec. throughput. An 8-bit Error Correction Code (ECC) across each 32-bit word improves system reliability. It does not support ECC for 16-bit wide interface option. The ECC is stored into the DDR-I/II SDRAM array along with the data and is checked when the data is read. If the code is incorrect, the MCU corrects the data (if possible) before reaching the initiator of the read. User-defined fault correction software is responsible for scrubbing the memory array.

The MCU supports two physical banks of DDR-I/II SDRAM in the form of discrete chips only. The MCU does not support DIMMs.

- The MCU only supports Unbuffered DDR-I/II memory types.
- The MCU supports a 16-bit/32-bit SDRAM data interface.
- The MCU contains transaction queues for each port enabling pipelining of transactions to the DDR-I/II SDRAM for maximum performance.

- The MCU provides two chip enables to the memory subsystem. These two chip enables service the DDR-I/II SDRAM subsystem (one per bank).
- Performance monitors for page hit / miss and read latency per port.

11.2 Theory of Operation

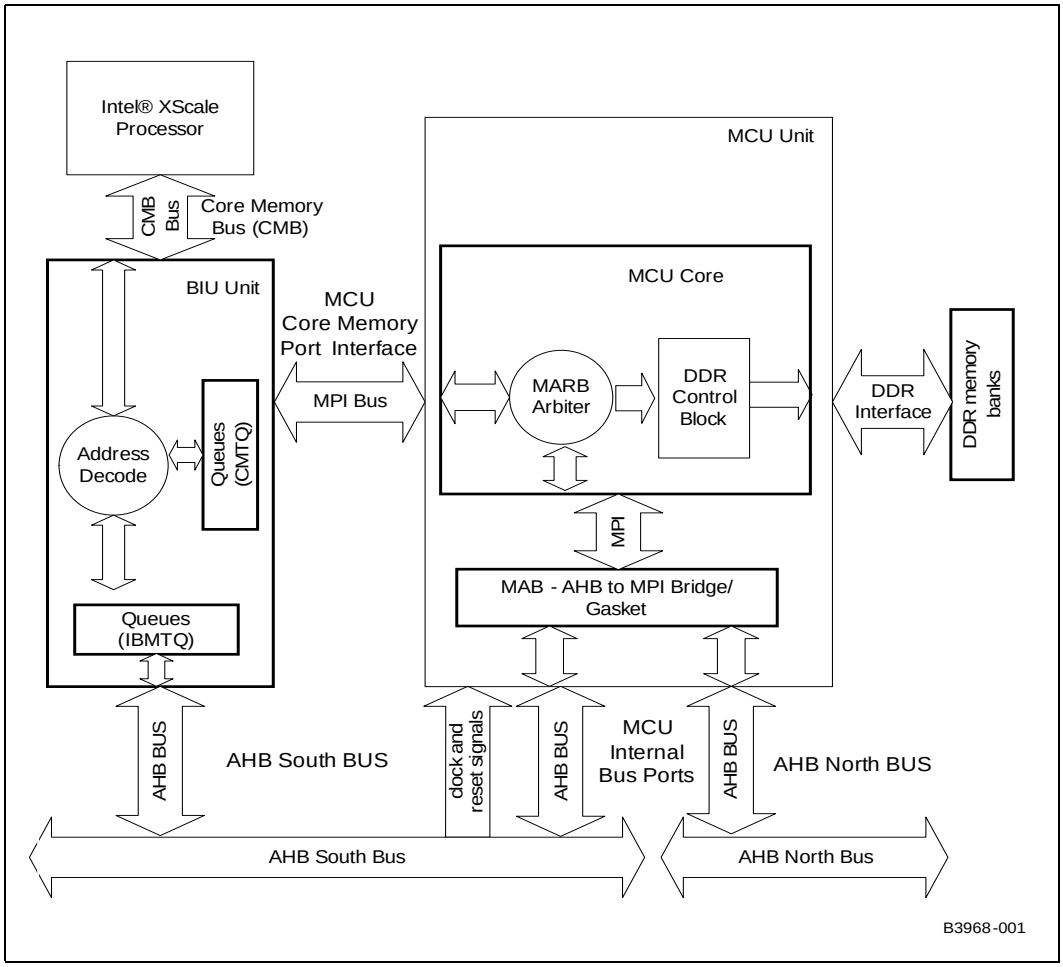
The memory controller of the IXP43X network processors translates the internal bus and Core Processor transactions into the protocol supported by the DDR-I/II SDRAM memory subsystem.

11.2.1 Functional Blocks

The following must be considered while creating low level software for the Intel® IXP43X Product Line.

The Memory Controller Unit (MCU) is made up of three parts (refer Figure 115). The MCU core connects via MPI (Memory Port Interfaces) to the MAB (MPI to AHB bridge) and the Bus Interface Unit (BIU). In turn, the MAB connects the MCU to the North and South AHB, while the BIU connects the Intel XScale processor to the MCU MPI port and the South AHB.

Figure 115. Memory Controller Block Diagram



The MAB supports posting of 1 write from the North AHB and 1 write from the South AHB. This means that once a master on the North and South AHB performs a write to the MAB, the MAB will accept the write, and free the AHB bus allowing the AHB master to continue. Once the MAB accepts a write transaction, the MAB will send the request to the MCU core, which in turn will perform a write to DDRII/I SDRAM.

The BIU also has write posting capabilities. Because the BIU and MAB have write posting capabilities, there can exist a race condition for write transactions posted in the BIU and MAB targeted to the same address.

There is no enforcement of write ordering between the North AHB and South AHB in the MAB for write requests that arrive at the same time and that are destined for the same DDRII/I SDRAM memory location. Therefore if an NPE and a South AHB master (PCI, USB or Intel XScale processor with MPI port disabled) write to the exact DDRII/I SDRAM memory location at the same time there is no guarantee which write will reach memory last.

In addition, there is no enforcement of write ordering for writes posted to the MAB before the Intel XScale processor (with MPI enabled) writes to the exact DDRII/I SDRAM memory location. To ensure completion of an MAB write before an Intel XScale processor (with MPI enabled) writes to the same DDRII/I SDRAM memory location, the Intel XScale processor must perform a read of that location before performing a write.

As stated above both masters must be performing writes to the same exact DDRII/I SDRAM memory location at nearly the same time. There is no problem with 1 master reading and 1 master writing the same exact DDRII/I SDRAM memory location at the same time.

Example:

Masters:

- Master on the PCI Bus

- Intel XScale processor

Transactions:

1. Master on the PCI bus performs a write (destined for DDRII/I) to the PCI Controller's internal queue.
2. PCI Controller initiates a South AHB write to the MAB, which is accepted in the South AHB posted write queue.
3. The MAB sends the write request after sometime to the MCU core, which performs the write to DDRII/I SDRAM memory location A2 (any DDRII/I memory location).
4. Master on the PCI bus writes a pointer to the just written data (destined for Advanced Queue Manager) to the PCI Controller's internal queue.
5. PCI Controller initiates a South AHB write to the Advanced Queue Manager.
6. Intel XScale processor reads pointer from the Advanced Queue Manager
7. Intel XScale processor writes a timestamp to location A2.

In the above example, the PCI Bus write may complete before or after the Intel XScale processor write. The Advanced Queue Manager is not required to have this race condition in this example. The same condition can be observed using an interrupt to hand over control from an AHB master to the Intel XScale processor.

How to avoid

Any of the following will eliminate the race condition between an AHB master writing to the same DDRII/I SDRAM memory location that the Intel XScale processor is writing.

1. Intel XScale processor must perform a read before it tries to write to the same exact DDRII/I SDRAM memory location. As stated above, the MCU core will ensure any writes in the MAB or BIU will be processed before the read is returned thus ensuring the newest data in the DDRII/I SDRAM.
2. The AHB master performs a read of the location before handing over control to the Intel XScale processor.
3. The AHB master performs a write to a different location before handing over control to the Intel XScale processor.
4. Disable the BIU MPI port.

No consideration is required if two or more masters write to adjacent memory locations because each write is destined to its own memory location and if a read is performed the MCU core will ensure that the new data is returned.

11.2.1.1 Transaction Ports

The MCU provides three transaction ports for DDR-I/II SDRAM access. They consist of two Internal Bus ports (AHB ports for the north and south AHB buses) and a direct port from the BIU (Core Processor Port).

11.2.1.1.1 Core Processor Port (From BIU)

The Core Processor port provides a direct connection between core bus interface of the IXP43X network processors and the memory controller. This Core Processor port allows core transactions targeting the DDR-I/II SDRAM to pass directly to the DDR-I/II SDRAM.

11.2.1.1.2 Internal Bus Ports (North and South AHB)

The two Internal Bus Ports (North AHB and South AHB) provide the connection to the DDR-I/II SDRAM from the internal AHB buses. All peripheral unit transactions targeting the DDR-I/II SDRAM are claimed by these ports. Also all accesses to MMR space is through the south AHB Internal bus port; this includes Intel XScale processor MMR accesses.

11.2.1.2 Address Decode Blocks

Address Decode is performed for transactions from input ports to determine if the MCU should claim the transaction. There are two address ranges the MCU ports can claim transactions: DDR-I/II SDRAM memory space and Memory-Mapped Register (MMR) space.

11.2.1.2.1 DDR-I/II SDRAM Memory Space

The DDR-I/II SDRAM memory space is defined as 0x0000_0000 to 0x3FFF_FFFF (1GB) for the IXP43X network processors.

Note: At boot time, the expansion bus resides at location 0000_0000h instead of the DDR-I/II SDRAM. The DDR controller must be configured first and the expansion bus decode programmed to sit at its higher address before DDR transactions are initiated. See Section 12.4.1.1, “Expansion Bus Address Space” for details.

11.2.1.2.2 Memory-Mapped Register Space

The MCU MMR memory space is CCFF E500H to CCFF E5FFH and CCFF F500H to CCFF F5FFH. The registers are detailed in Section 11.6, “Register Definitions”.

The Address Decode for each port is based on the same registers and are only accessible from the South AHB internal bus.

Read and write accesses to all Configuration/Status registers (CSR) must be single word transfers. Byte, half-word, burst, or coalesced transfers are not supported and result in unpredictable operation. All CSR reserved bits must be written with zero.

Each port decodes inbound transactions for these address ranges. The Address Decode blocks determine how the memory controller responds to inbound transactions. The details of the address decode for each port is described below.

11.2.1.2.3 Core Processor Port Address Decode

The address decode block for the Core Processor transactions resides in the BIU. The Core MCU port therefore does not require any decode, and processes any transaction received from the BIU via the Core MCU Port.

If the Core Processor Memory Transaction Queue is disabled (refer to Section 12.5.6, "Configuration Register 1" bit 31 MPI_EN) all core transactions are directed to the core Internal Bus Memory Transaction Queue of the BIU, and would not be sent to the Core MCU port (refer to Figure 115). Therefore core transactions that address the DDR-I/II SDRAM, is claimed by the IB (Internal Bus) port of the MCU and processed via that port as any other IB transaction. It is recommended that MCU transaction queues are not disabled.

11.2.1.2.4 Internal Bus Port Address Decode

Internal Bus transactions are decoded to determine if they address the DDR-I/II SDRAM memory space or MCU MMR Space. If the transaction addresses either of these two spaces, the transaction is claimed by one of the Internal Bus ports (Only South AHB port can claim MMR accesses). If the transaction addresses the DDR-I/II SDRAM memory space, the transaction is queued in the Internal Bus Port Transaction Queue. If the transaction addresses the MCU MMR Space, the transaction is serviced by accessing the Configuration Register block.

11.2.1.3 Memory Transaction Queues

There are two transaction queues for transactions that address the MCU DDR memory space. One transaction queue is for Core Processor transactions and one transaction queue is for Internal Bus transactions (North and South AHB).

11.2.1.3.1 Core Processor Memory Transaction Queue (CMTQ)

The CMTQ stores memory transactions from the core that have not been processed by the memory controller. The CMTQ also supports 8 Core Processor posted write transactions up to 16-Bytes each. The CMTQ supports eight Core Processor read transactions up to 32 Bytes each, and that equals the maximum number of outstanding transactions the Intel XScale processor Processor Bus Controller can support. The eight read transactions are:

- core DCU: 4 - load requests to unique cachelines
- IFU: 2 - prefetch
- IMM: 1 - tablewalk
- DMM: 1 - tablewalk

11.2.1.3.2 AHB Internal Bus Memory Transaction Queue (IBMTQ)

Each IBMTQ stores memory transactions from the Internal Buses that addresses the DDR-I/II SDRAM memory space. Each IBMTQ can hold 2 outstanding Internal Bus read or write transaction requests. The read transactions are processed one at a time, each with up to 32-Bytes of read data. The IBMTQ also supports 2 posted write transactions up to 32 Bytes each.

Note: The MCU Internal Bus Port does not split or retry Internal Bus Read Transactions. Instead it stays on the bus until the read data is returned from the DRAM. This enables queuing of outstanding read transactions without modifying the existing peripheral units of the IXP43X network processors. Internal Bus Write Transactions are posted to improve bus bandwidth.

11.2.1.4 Configuration Registers

The Configuration Registers block contains all of the memory-mapped registers listed in Section 11.6, “Register Definitions”. These registers define the memory subsystem connected to the IXP43X network processors. The status registers indicate the current MCU status.

11.2.1.5 Refresh Counter

The Refresh Counter block keeps track of when the DDR-I/II SDRAM devices must be refreshed. The refresh interval is programmed in the Section 11.6.13, “Refresh Frequency Register RFR”. Once the 13-bit refresh counter reaches the programmed interval, the DDR-I/II SDRAM state machine issues a refresh command to the DDR-I/II SDRAM devices. If a transaction is currently in progress, the DDR-I/II SDRAM State Machine waits for the completion of the transaction to issue the refresh cycle. See Section 11.2.2.14, “DDR SDRAM Refresh Cycle” for more details.

Note: If the memory interface is busy when the refresh counter expires, it is possible for the MCU to generate more than one refresh cycle when the memory interface becomes available.

Note: If the memory controller completes transactions from the Core Processor Memory Transaction Queue (CMTQ) when the refresh counter expires, the Memory Controller Arbiter (MARB) completes the CMTQ tenure based on the Section 11.6.12, “MCU Port Transaction Count Register MPTCR”, before any refresh cycles are issued. Once refresh cycles are initiated, all queued refresh transactions are to be completed in a single tenure.

11.2.1.6 DDR-I/II SDRAM Control Block

The DDR-I/II SDRAM Control Block contains all functionality to process the DDR-I/II SDRAM data accesses per the transactions issued by the MARB. To process a transaction the DDR-I/II SDRAM Control Block employs several sub-blocks. The sub-blocks include the Page Control Block, DDR-I/II SDRAM State Machine and Pipeline Queues, and Error Correction Logic.

11.2.1.6.1 Page Control Block

The Page Control Block records and maintains the open DDR-I/II SDRAM pages. The MCU can keep a maximum of eight pages open simultaneously (four per bank). This block keeps track of open pages and determines if the transactions hit an open page. For more details about the page hit/miss determination, see Section 11.2.2.7, “Page Hit/Miss Determination”.

11.2.1.6.2 DDR-I/II SDRAM State Machine and Pipeline Queues

Since the MCU generates error correction codes based on the data, the MCU is a pipelined architecture. Pipelining also ensures acceptable AC timings to the memory interfaces. The DDR-I/II SDRAM state machine pipelines DDR-I/II SDRAM memory operations for several clocks.

11.2.1.6.3 Error Correction Logic

The Error Correction Logic generates the ECC code for DDR-I/II SDRAM reads and writes. For reads, this logic compares the ECC codes read with the locally generated ECC code. If the codes mismatch then the Error Correction Logic determines the error type. For a single-bit error, this block determines the bit that is in error and corrects the error. For a single-bit or multi-bit error, the Error Correction Logic logs the error in ELOG0 and ELOG1. See Section 11.2.3, “Error Correction and Detection” for more details.

11.2.1.7 DDR-I/II SDRAM RCOMP Block

The DDR RComp circuitry dynamically compensates for variations in operating conditions due to process, temperature, or voltage. These variations are measured through a resistive mechanism in a special I/O pad and evaluated in the associated compensation circuitry. Adjustments are then made to the drive strength of the buffers ensuring error free operation over the entire range of operating conditions.

The DDR RComp circuitry fine-tunes three separate I/O buffer parameters:

- P-Channel drive strength
- N-Channel drive strength
- Buffer slew rate

11.2.2 DDR-I/II SDRAM Memory Support

The memory controller of the IXP43X network processors supports one or two banks of DDR-I/II SDRAM.

The DDR-I/II SDRAM provides the following:

- Allows zero data-to-data wait-state operation
- Offers an extremely wide range of configuration options emerging from the SDRAMs internal interleaving and bursting capabilities

The MCU supports a 32-bit data bus width memory implementation (with and without ECC), and supports a 16-bit data bus width memory implementation (without ECC). The data bus width is controlled by the DDR-I/II SDRAM Control Register.

The MCU supports DDR-I/II SDRAM burst length of four for 32-bit and 16-bit data bus width options. A burst length of four enables seamless read/write bursting of long data streams as long as the memory transaction does not cross the page boundary. Page boundaries are at naturally aligned boundaries. The MCU ensures that the page boundary is not crossed within a single transaction by initiating a disconnect at next ADB (128-byte address boundary) on the internal bus prior to the page boundary.

11.2.2.1 DDR-I/II SDRAM Interface

The DDR-I/II SDRAM interface signals generated by the memory controller unit are for DDR-I/II SDRAM operation. Refer to the *Intel® IXP43X Product Line of Network Processors Datasheet* for specifics on DDR-I/II SDRAM interface signals.

The MCU SDRAM interface provides a flexible mix of combinations including:

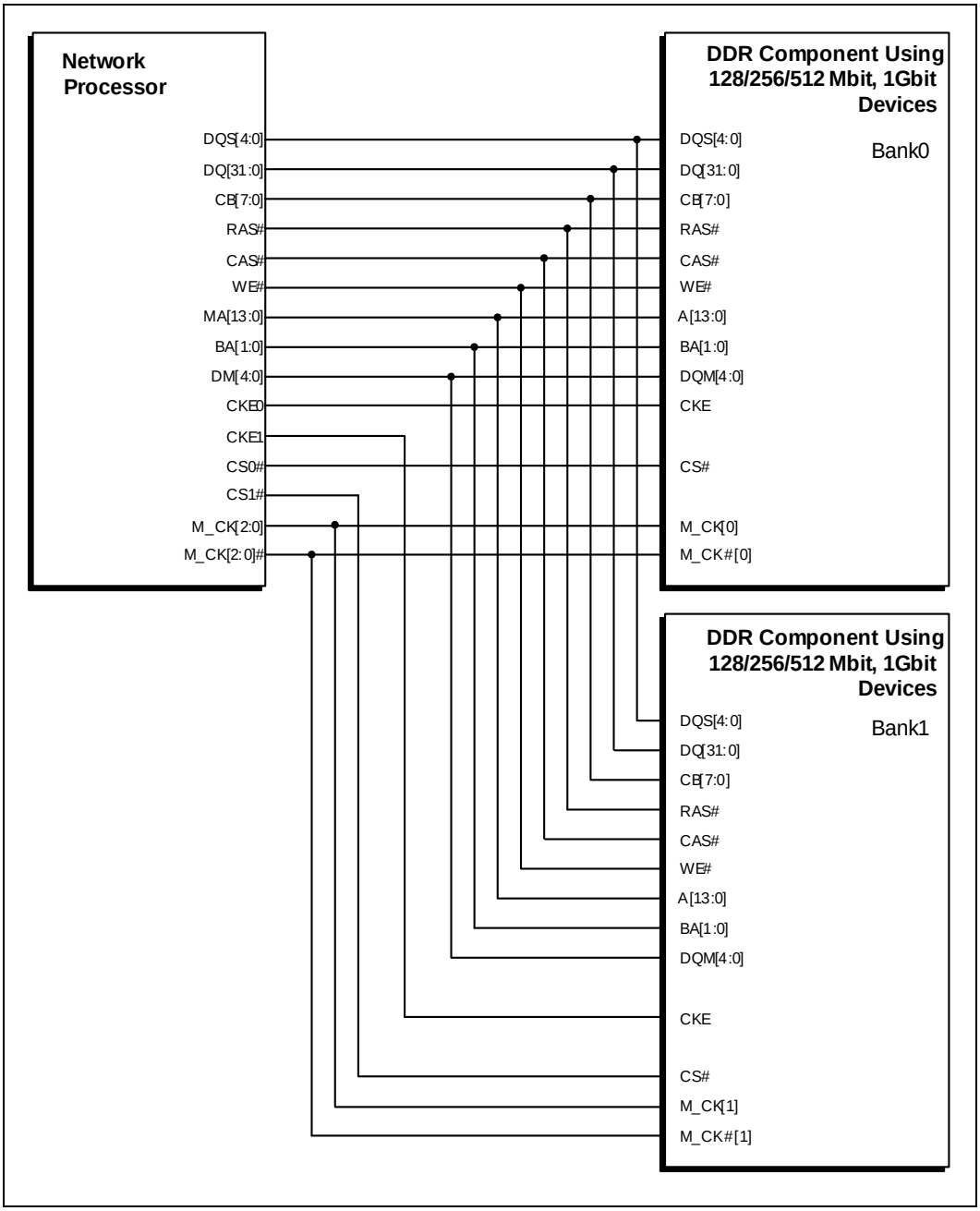
Table 191. DDR-I/II SDRAM Memory Configuration Options

Data Bus Width	ECC Enabled	Maximum Throughput[1]
32 bit	Yes	1600 Mbyte/s
32 bit	No	1600 Mbyte/s
16 bit	No	800 Mbyte/s

Note:
1. Based on DDR-II 400MHz SDRAM

Figure 116 illustrates how two banks of DDR-I/II SDRAM interface with the IXP43X network processors through the MCU.

Figure 116. Dual-Bank DDR SDRAM Memory Subsystem



11.2.2.2 DDR-I/II SDRAM Bank Sizes and Configurations

The MCU supports a memory subsystem ranging from 32 Mbyte to 1 Gbyte for 32-bit memory systems for DDR-I SDRAM, from 64 Mbyte to 512 Mbyte for 32-bit memory systems for DDR-II SDRAM, and supports 16 Mbyte for 16-bit memory systems for DDR-I SDRAM (non-ECC), and 32 to 64 Mbyte for 16-bit memory systems for DDR-II

SDRAM (non-ECC). An ECC or non-ECC system may be implemented using x8, or x16 devices. Table 192, Table 193, Table 194 and Table 195 illustrate the supported DDR-I/II SDRAM configurations.

Note: DDR-I/II SDRAM devices use multiple banks within the device operating in an interleaved mode. The MCU supports 128/256/512 Mbit, 1 Gbit DDR-I/II SDRAM devices containing four internal banks. An internal bank is defined as a leaf (to avoid confusion with a memory bank).

Table 192. Supported DDRI 32-bit SDRAM Configurations

DDR SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size[1]	Page Size[2]
			Row	Column	DDR_BA[1]	DDR_BA[0]		
128 Mbit[3]	16 M x 8	1	12	10	**ADDR[26]**	**ADDR[25]**	64 M	4KB
		2					128 M	4KB
	8 M x 16	1	12	9	**ADDR[25]**	**ADDR[24]**	32 M	2KB
		2					64 M	2KB
256 Mbit	32 M x 8	1	13	10	**ADDR[27]**	**ADDR[26]**	128 M	4KB
		2					256 M	4KB
	16 M x 16	1	13	9	**ADDR[26]**	**ADDR[25]**	64 M	2KB
		2					128 M	2KB
512 Mbit	64 M x 8	1	13	11	**ADDR[28]**	**ADDR[27]**	256 M	8KB
		2					512 M	8KB
	32 M x 16	1	13	10	**ADDR[27]**	**ADDR[26]**	128 M	4KB
		2					256 M	4KB
1 Gbit[3]	128 M x 8	1	14	11	**ADDR[29]**	**ADDR[28]**	512 M	8KB
		2					1 G	8KB
	64 M x 16	1	14	10	**ADDR[28]**	**ADDR[27]**	256 M	4KB
		2					512 M	4KB

1. Table indicates 32-bit wide memory subsystem sizes.
2. Table indicates 32-bit wide memory page sizes.
3. Supported with DDRI SDRAM only

Table 193. Supported DDRII 32-bit SDRAM Configurations

DDR SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size	Page Size
			Row	Column	DDR_BA[1]	DDR_BA[0]		
256 Mbit	32M x 8	1	13	10	ADDR[27]	ADDR[26]	128MB	4KB
		2					256MB	4KB
	16M x16	1	13	9	ADDR[26]	ADDR[25]	64MB	2KB
		2					128MB	2KB
512 Mbit	64M x 8	1	14	10	ADDR[28]	ADDR[27]	256MB	4KB
		2					512MB	4KB
	32M x16	1	13	10	ADDR[27]	ADDR[26]	128MB	4KB
		2					256MB	4KB

Table 194. Supported DDRI 16-bit SDRAM Configurations

DDR SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size[1]	Page Size[2]
			Row	Column	DDR_BA[1]	DDR_BA[0]		
128 Mbit	8M x16	1	12	9	ADDR[23]	ADDR[22]	16MB	1KB

1. Table indicates 16-bit wide memory subsystem sizes.
2. Table indicates 16-bit wide memory page sizes.

Table 195. Supported DDRII 16-bit SDRAM Configurations

DDR SDRAM Technology	DDR SDRAM Arrangement	# Banks	Address Size		Leaf Select		Total Memory Size	Page Size
			Row	Column	DDR_BA[1]	DDR_BA[0]		
256 Mbit	16M x16	1	13	9	ADDR[24]	ADDR[23]	32MB	1KB
512 Mbit	32M x16	1	13	10	ADDR[27]	ADDR[26]	64MB	4KB

128/256/512-Mbit, 1-Gbit DDR-I SDRAM and 256/512-Mbit DDR-II SDRAM devices comprise four internal leaves. The MCU controls the leaf selects within 128/256/512 Mbit, 1-Gbit DDR-I SDRAM and 256/512-Mbit DDR-II SDRAM by toggling DDR_BA[0] and DDR_BA[1].

The two DDR-I/II SDRAM chip enables (DDR_CS_N[1:0]) support an DDR-I/II SDRAM memory subsystem consisting of two banks. The base address for the two contiguous banks are programmed in the DDR-I/II SDRAM Base Register (SDBR) and must be aligned to a 16 Mbyte boundary. The size of each DDR-I/II SDRAM bank is programmed with the DDR-I/II SDRAM boundary registers (SBR0 and SBR1).

Note: Systems must implement memory devices of all the same bus width (x8, x16) due to the single CAS address decode generated for a DDR transaction. For example, systems implementing ECC and using x16 devices for the 32 bit DDR data bus must also use a x16 device for the 8 bit ECC bus, even though 8 bits go unused.The (S32SR) register is not used for the IXP43X network processors since it is always in 16-bit/32-bit mode.

Table 196. DDR-I/II SDRAM Address Register Summary

DDR SDRAM Address Register	Definition
DDR SDRAM Base Register (SDBR)	The lowest address for DDR SDRAM memory space aligned to a 16 Mbyte boundary.
DDR SDRAM Boundary Register 0 (SBR0)	The upper address boundary for bank 0 of DDR SDRAM memory space. SBR0 must be greater than or equal to the value of SDBR[30:24].
DDR SDRAM Boundary Register 1 (SBR1)	The upper address boundary for bank 1 of DDR SDRAM memory space. SBR1 must be greater than or equal to SBR0.

Note: DDR SDRAM memory space must be aligned to a 16 Mbyte boundary and must **never** cross a 1-Gbyte boundary. With 32-bit DDR SDRAM attached to the IXP43X network processors, all DDR SDRAM memory space behaves as 32-bit DDR SDRAM and the value in S32SR is ignored.

The base register defines the upper eight address bits of the DDR-I/II SDRAM memory space. The boundary registers define the address limits for each DDR-I/II SDRAM bank in 16 Mbyte granularity. Table 197 defines the conditions that must be satisfied to activate a DDR-I/II SDRAM memory bank. The base address is default at 0000 0000H for the IXP43X network processors.

Table 197. Address Decoding for DDR-I/II SDRAM Memory Banks

Condition	DDR-I/II SDRAM Bank Selected
ADDR[31] is not equal to the SDBR[31]	None
ADDR[31] is equal to the SDBR[31] AD[30:24] is greater than or equal to the SDBR[30:24] AD[30:24] is less than the value in SBR0	Bank 0
ADDR[31] is equal to the SDBR[31] AD[30:24] is greater than or equal to the value in SBR0 AD[30:24] is less than the value in SBR1	Bank 1
ADDR[31] is equal to the SDBR[31] AD[30:24] is greater than or equal to the value in SBR1	None

Table 198 shows the correct programming values for the DDR-I/II SDRAM Bank Size

Table 198. Programming Codes for the DDR-I/II SDRAM Bank Size

Bank Size	Code	Bank Size	Code
Empty	00H	128 Mbyte	08H
16 Mbyte	01H	256 Mbyte	10H
32 Mbyte	02H	512 Mbyte	20H
64 Mbyte	04H	1 Gbyte	40H

encoding. These Bank Size Codes are used in the calculation of the DDR-I/II SDRAM boundary registers programming values. Equation 1 and Equation 2 show the required equations to calculate the programming values for the DDR-I/II SDRAM boundary registers.

Note: Both banks must be programmed to be the same size for the IXP43X network processors.

Equation 1. Programming Value for DDR-I/II SDRAM Boundary Register 0 (SBR0[7:0])

{SBR0[6:0],SBR0[7]} = Bank 0 Size Code+ SDBR[30:24]

Equation 2. Programming Value for DDR-I/II SDRAM Boundary Register1 (SBR1[7:0])

{SBR1[6:0],SBR1[7]} = Bank 1 Size Code + {SBR0[6:0],SBR0[7]}

Table 199 shows the correct programming values for the 32-bit DDR-I/II SDRAM Size Register.

Table 199. Programming Values for the DDR SDRAM 32-bit Size Register (S32SR[29:20])

32-bit Region Size	S32SR[29:20]	32-bit Region Size	S32SR[29:20]
Empty	000H	32 M	020H
1 M	001H	6 4M	040H
2 M	002H	128 M	080H
4 M	004H	256 M	100H
8 M	008H	512 M	200H
16 M	010H	(reserved)	all other values
Note: For the IXP43X network processors, this is always programmed to 0.			

Example 21. Address Register Programming Example 1 (Default mode for IXP43X network processors)

The user wants to program the DDR-I/II SDRAM memory space to begin at 0000 0000H (IXP43X network processors default value). Bank 0 is 128 Mbyte and Bank 1 is 128 Mbyte yielding a total memory of 256 Mbytes. All the memory is programmed for 32-bit mode, so there is no special 32-bit region using S32SR register. The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	0000 0000H
32-Bit Region Top (end 32-bit region)	N/A
Invalid Region Top (start 64-bit region)	N/A
Bank 0 Top	07FF FFFFH
Bank 1 Top	0FFF FFFFH

The registers would be programmed as follows:

- Bank 0 Size = 128MB, code = 0001000_2
- Bank 1 Size = 128MB, code = 0001000_2
- SDBR =0000 0000H, SDBR[31:24] = 00000000_2
- SBR0[7:0] = 00000100_2 = 04H (size of Bank 0)
- SBR1[7:0] = 00001000_2 = 08H (size of Bank 0 + size of Bank 1)

Example 22. Address Register Programming Example 2 (Default mode for IXP43X network processors)

The user wants to program the DDR-I/II SDRAM memory space to begin at 0000 0000H. Bank 0 is 16 Mbyte and Bank 1 is un-populated yielding a total memory of 16 Mbytes. All the memory is programmed for 16-bit mode, so there is no special 32-bit region using S32SR register. The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	0000 0000H
32-Bit Region Top (end 32-bit region)	N/A
Invalid Region Top (start 64-bit region)	N/A
Bank 0 Top	00FF FFFFH
Bank 1 Top	N/A

The registers would be programmed as follows:

- Bank 0 Size = 16MB, code = 0000001_2
- Bank 1 Size = empty, code = 0000000_2
- SDBR =0000 0000H, SDBR[30:24] = 0000000_2
- SBR0[7:0] = 10000000_2 = 80H (size of Bank 0)
- SBR1[7:0] = 10000000_2 = 80H (size of Bank 0 + size of Bank 1)

Additionally, the following registers should also be programmed before using the DDR:

- DDR SDRAM Control Register 0 SDCR0 - Program according JEDEC specs.
- DDR SDRAM Control Register 1 SDCR1 - Program according JEDEC specs.
- Perform DDR initialization sequence using DDR SDRAM Initialization Register SDIR register.

- Refresh Frequency Register RFR - Program per JEDEC Spec using MCU clock of 133 or 200MHz

Note: All other registers can use their default register values for operation.

11.2.2.3 MPTCR Register setup

To ensure equal bandwidth between the MPI port and the two AHB ports, the MPTCR register must be programmed to 0x00000011H, other wise the default value accepts 12 MPI transactions to every one AHB transaction thereby starving the AHB ports of the memory controller.

11.2.2.4 DDR SDRAM Addressing

Table 200 is used to determine the DDR SDRAM address decode that is used to program **SDRAM Boundary Register 0 - SBR0** and **SDRAM Boundary Register 1 - SBR1**.

Table 200. DDR SDRAM Address Decode Summary

Row Address Size (bits)	Column Address Size (bits)		
	9	10	11
12	#1	#1	N/A
13	#2	#1	#1
14	N/A	#3	#1
#1: See Table 201 #2: See Table 202 #3: See Table 203			

Table 201, Table 202 and Table 203 illustrate the address decode listed in Table 200. These tables show how the internal address is mapped to the DDR_MA[13:0] lines for DDR SDRAM devices.

DDR SDRAM Address Decode #1 is the normal translation used for most memory configurations. When the column and row address size is less than shown in Table 200, the unused address lines are still driven with the internal bus address shown.

Table 201. DDR SDRAM Address Translation #1

DDR_MA [13:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	I_AD[27]	I_AD[25]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	NOTE[1]	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:
1. A10 is used for precharge variations on the read or write command. See Table 204 for more details.
2. For the Leaf Selects, see Table 192

Table 202. DDR SDRAM Address Translation #2

DDR_MA [13:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	I_AD[27]	I_AD[24]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	NOTE[1]	-	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:
1. A10 is used for precharge variations on the read or write command. See Table 204 for more details.
2. For the Leaf Selects, see Table 192.
3. DDR SDRAM Address Translation #2 presents **I_AD[24]** as bit 12 of the row address instead of **I_AD[25]** as in Address Translation #1.

Table 203. DDR SDRAM Address Translation #3

MA[12:0]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row	I_AD[26]	I_AD[25]	I_AD[23]	I_AD[22]	I_AD[21]	I_AD[20]	I_AD[19]	I_AD[18]	I_AD[17]	I_AD[16]	I_AD[15]	I_AD[14]	I_AD[13]	I_AD[12]
Column	-	-	I_AD[26]	NOTE[1]	I_AD[24]	I_AD[11]	I_AD[10]	I_AD[9]	I_AD[8]	I_AD[7]	I_AD[6]	I_AD[5]	I_AD[4]	I_AD[3]

Notes:
1. A10 is used for precharge variations on the read or write command. See Table 204 for more details.
2. For the Leaf Selects, see Table 192.
3. DDR SDRAM Address Translation #3 presents **I_AD[26]** as bit 13 of the row address instead of **I_AD[27]** as in Address Translation #1.

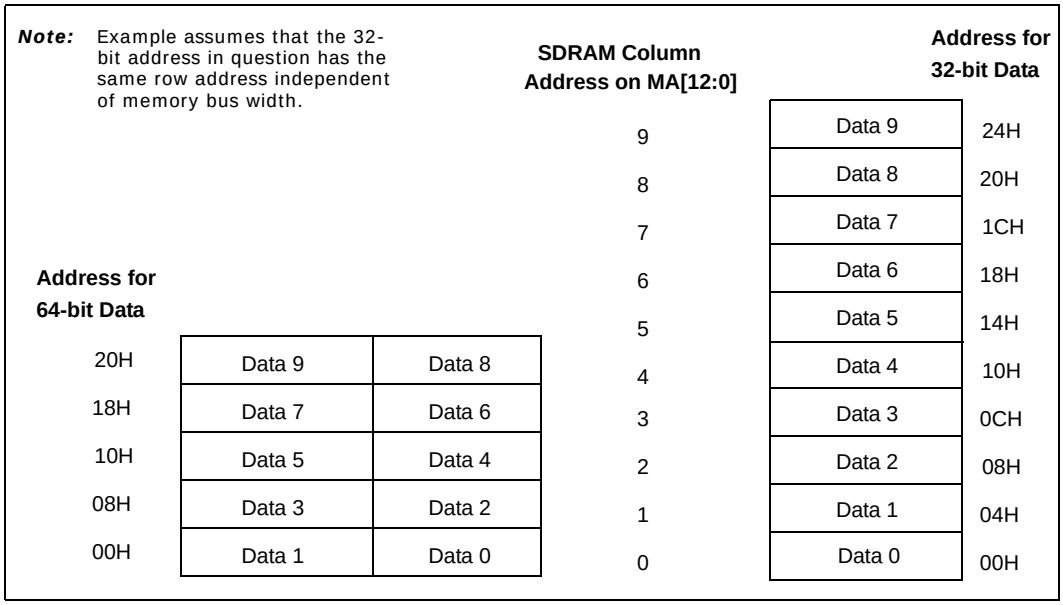
Since the MCU supports DDR SDRAM bursting, the MCU increments the column address based on the burst length of four for each DDR SDRAM read or write burst. The MCU supports a sequential and random burst types. Sequential bursting means that the address issued to the DDR SDRAM is incremented by the DDR SDRAM device in a linear fashion during the burst cycle. Random bursting means that the address issued to the DDR SDRAM is any address in a currently active page.

11.2.2.5 32-bit Data Bus Width

The MCU supports a 32-bit data bus width and a minimum memory size of 32 Mbytes, the maximum bus throughput is 1600 Mbytes/sec.

SDRAM address translation differs between 64- and 32-bit data bus widths. To generate a 32-bit address to the memory subsystem, the internal address ADDR[31:2] is shifted to the left by one bit prior to the address translations illustrated in Table 201 through Table 203. This provides the granularity required for a 32-bit wide memory. See Figure 117 for an example of how shifting the address before generating the SDRAM address on DDR_MA[13:0] results in 32-bit addressing.

Figure 117. 64-bit to 32-bit Addressing



For example, assume an internal bus master issues a write to SDRAM memory space via the Internal Bus Memory port. The address specified on ADDR[31:0] is 0000 0010H.

If 32-bit data bus width is enabled, the MCU shifts ADDR[31:2] by one bit to the left before translating the address to DDR_MA[13:0]. Therefore, the column address becomes four for the first write transaction.

The MCU of the IXP43X network processors supports a 16 or 32-bit data bus, although future products may support a 64-bit data bus width. The data bus width is selected by bit 3 and bit 1 of the SDCR (see "DDR SDRAM Control Register 0 SDCR0" on page 553). The power-on default is a 64-bit bus width but for the IXP43X network processors, the bus width MUST always be programmed to 16-bit or 32-bit bus width.

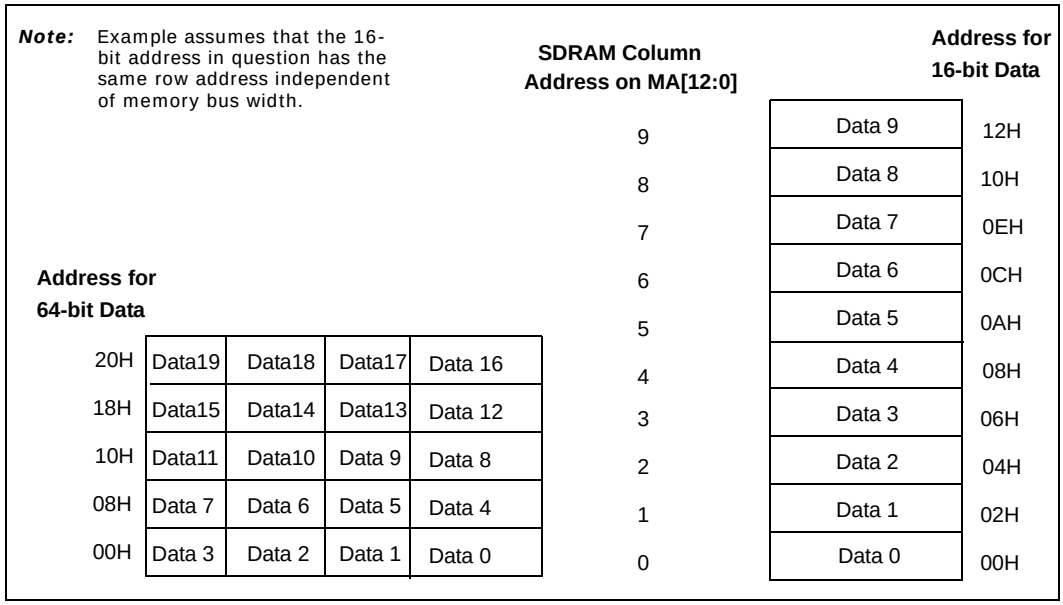
To select 32-bit bus width, bit 3 and bit 1 of the SDCR is set to 0 and 1 respectively.

11.2.2.6 16-bit Data Bus Width

The MCU supports a 16-bit data bus width and a minimum memory size of 16 Mbytes, the maximum bus throughput is 800 Mbytes/sec.

SDRAM address translation differs between 64- and 16-bit data bus widths. To generate a 16-bit address to the memory subsystem, the internal address ADDR[31:1] is shifted to the left by two bits prior to the address translations illustrated in Table 201 through Table 203. This provides the granularity required for a 16-bit wide memory. See Figure 118 for an example of how shifting the address before generating the SDRAM address on DDR_MA[13:0] results in 16-bit addressing.

Figure 118. 64-bit to 16-bit Addressing



For example, assume an internal bus master issues a write to SDRAM memory space via the Internal Bus Memory port. The address specified on ADDR[31:0] is 0000 0010H.

If 16-bit data bus width is enabled, the MCU shifts ADDR[31:1] by two bits to the left before translating the address to DDR_MA[13:0]. Therefore, the column address becomes eight for the first write transaction.

To select 16-bit bus width, bit 3 of the SDCR is set to 1. It is recommended to also set bit 1 of the SDCR to 1.

11.2.2.7 Page Hit/Miss Determination

The MCU keeps up to eight pages open simultaneously; one page each of Bank0/Leaf0, Bank0/Leaf1, Bank0/Leaf2, Bank0/Leaf3, Bank1/Leaf0, Bank1/Leaf1, Bank1/Leaf2, and Bank1/Leaf3. The page size is based on the DDR technology as specified in Table 192.

Figure 119 illustrates the logical flow of page hit determination

The MCU logic determines the hit/miss status for reads and writes. For a new DDR SDRAM transaction, the MCU compares the address of the current transaction with the address stored in the appropriate page address register. Given the supported DDR SDRAM devices and two banks, there are eight pages kept open simultaneously. The DDR SDRAM chip enables **DDR_CS_N[1:0]** and leaf selects **DDR_BA[1:0]** determine the page address that has to be compared.

If the current transaction misses the open page selected then the MCU closes the open page pointed to by **DDR_CS_N[1:0]** and **DDR_BA[1:0]** by issuing a **precharge** command. The MCU opens the current page with a **row-activate** command and the transaction completes with a **read** or **write** command. When the MCU opens the current page, the row address from the corresponding memory transaction queue is stored in the page address register pointed to by **DDR_CS_N[1:0]** and **DDR_BA[1:0]** so it may be compared for future transactions. See Table 201, Table 202 and Table 203 for address mapping to row address.

Once the MARB issues a command to the MCU DDR Control Block, the paging logic makes a hit/miss comparison. The performance is best for page hits and therefore the MCUs behavior is different for the hit and miss scenario.

For a page hit, the MCU need not open the page and avoids the RAS-to-CAS delay achieving greater performance. The waveform for a write including the row activation in the case of a page miss is illustrated in Figure 130. For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.

If the current transaction hits the open page, then the page is already active and the **read** or **write** command may be issued without a **row-activate** command. When the next transaction is the same command type as the current transaction, and also a page hit, the MCU need not issue the command again, but drives column address for an open page.

If the refresh timer expires and the MCU issues an **auto-refresh** command, all pages are closed.

Figure 129 illustrates the performance benefit of a read hit versus a read miss in Figure 132. Figure 130 illustrates the performance benefit of a write hit versus a write miss in Figure 135.

Figure 119. Page Hit / Miss Logic for 128 / 256 / 512 / 1,024-bit Mode

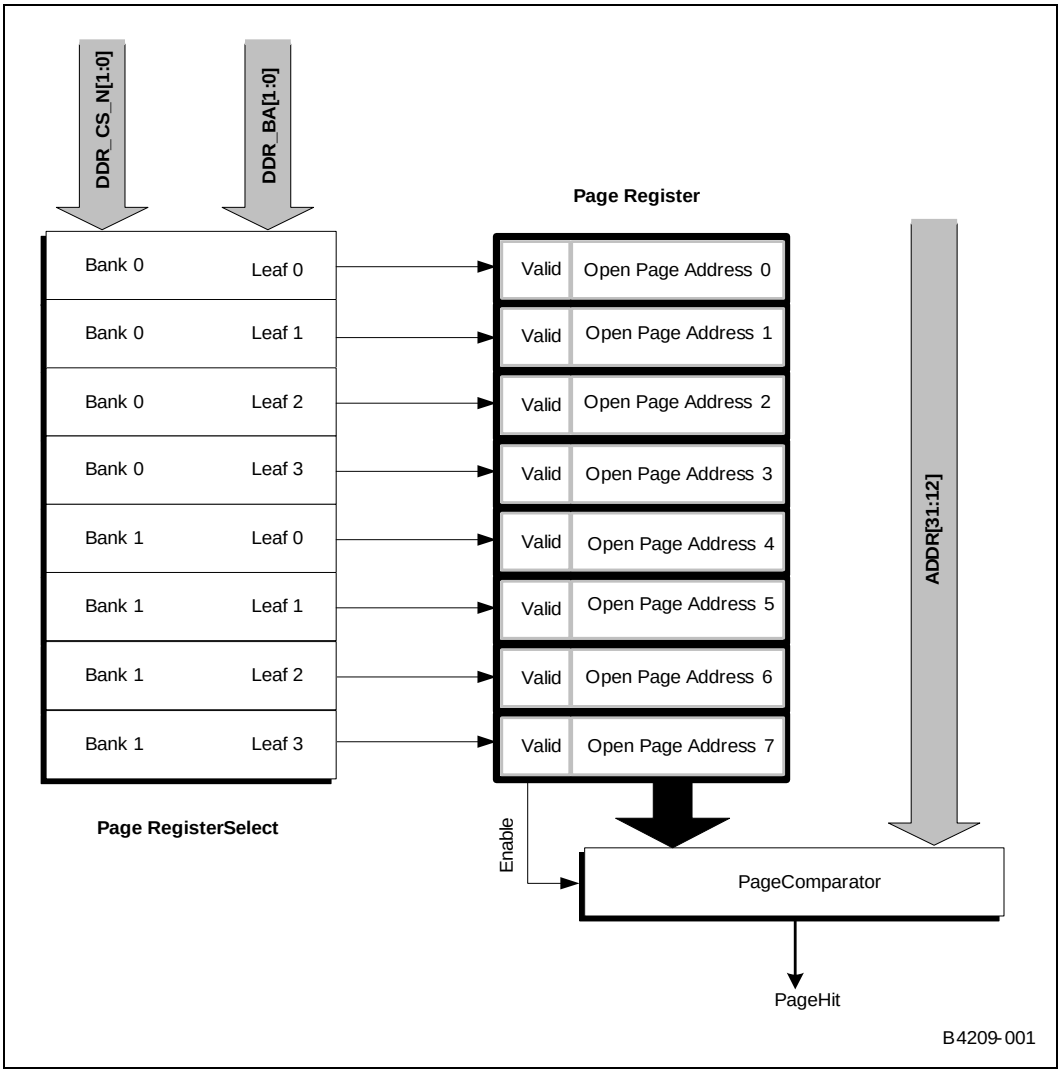


Figure 120. Logical Memory Image of a DDR SDRAM Memory Subsystem

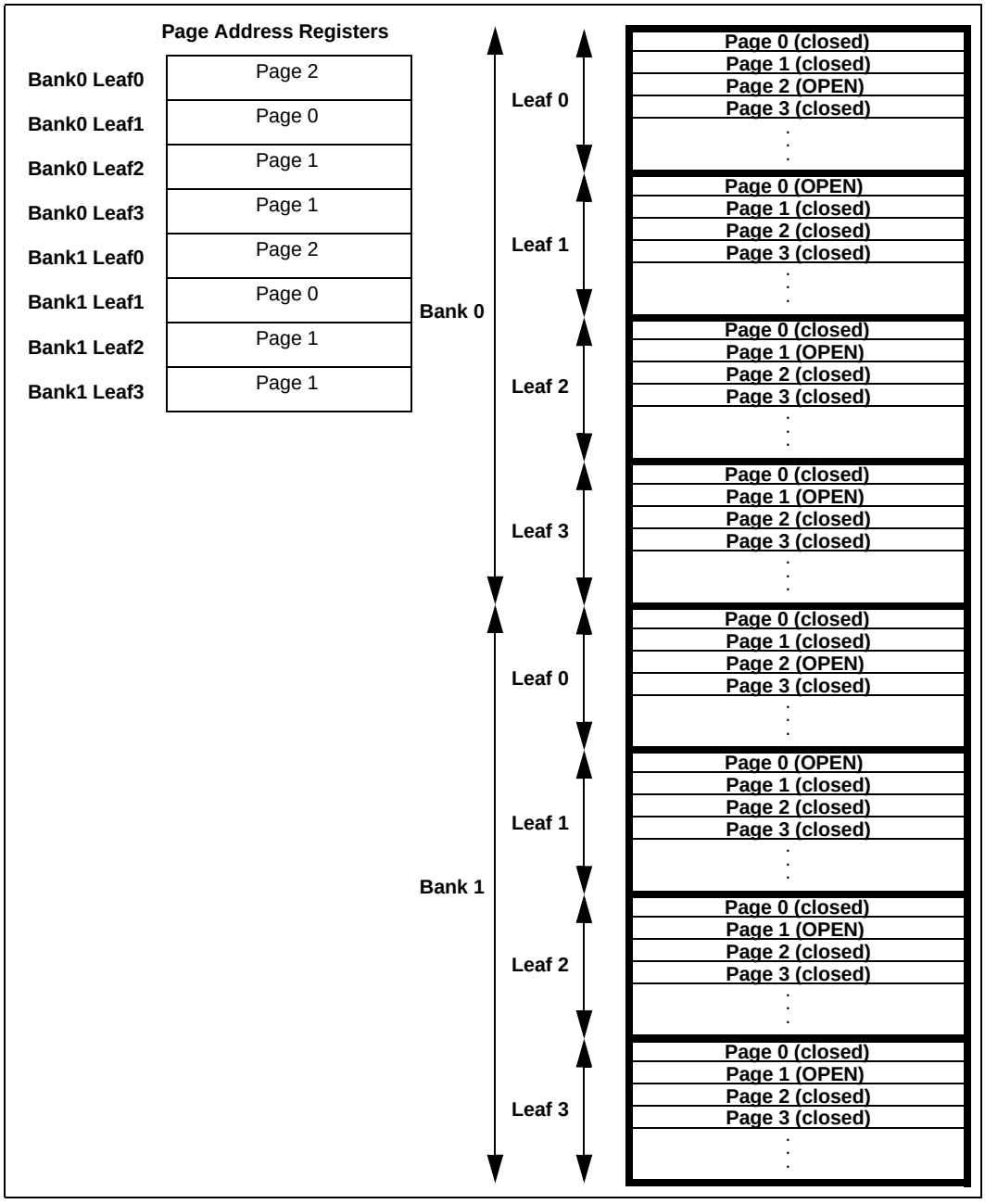


Figure 120 illustrates how the logical memory image is partitioned with respect to open and closed pages. If the above image represents a 128-Mbyte DDR SDRAM memory size, each bank is 64 Mbytes and each leaf is 16 Mbytes.

Only one page may be open within each of the leaf blocks. The page sizes depend on the memory size implemented in the DDR SDRAM memory subsystem as listed in Table 192. The programmer can optimize DDR SDRAM transactions by partitioning code and data across the leaf boundaries to maximize the number of page hits.

11.2.2.8 On DIMM Termination

The memory controller supports On Die Termination (ODT) when controlling DDR-II SDRAM technology. ODT eliminates the need for on-board termination resistors, thereby reducing the implementation cost and area. Control for enabling ODT is performed in Section 11.6.2, "DDR SDRAM Control Register 0 SDCR0".

11.2.2.9 DDR SDRAM Commands

The MCU issues specific commands to the DDR SDRAM devices by encoding them on the DDR_CS_N[1:0], DDR_RAS_N, DDR_CAS_N, and DDR_WE_N inputs. Table 204 lists all of the DDR SDRAM commands understood by DDR SDRAM devices. The MCU supports a subset of these commands.

Table 204. DDR SDRAM Commands

Command[1,2]	Conditions					Comments
	DDR_CS_N[1:0]	DDR_RAS_N	DDR_CAS_N	DDR_WE_N	Other	
NOP	0	1	1	1		No Operation
Mode Register Set	0	0	0	0	**DDR_BA[1:0]** = Sel[3]	Load the (Extended) Mode Register from **DDR_MA[13:0]**
Row Activate	0	0	1	1	**DDR_BA[1:0]** = Leaf	Activate a row specified on **DDR_MA[13:0]**
Read	0	1	0	1	**DDR_BA[1:0]** = Leaf **DDR_MA[10]** = 0	Column burst read Column address on **DDR_MA[13:0]**
Read w/ Auto-Precharge	0	1	0	1	**DDR_BA[1:0]** = Leaf **DDR_MA[10]** = 1	Column burst read with row precharge at the end of the transfer
Write	0	1	0	0	**DDR_BA[1:0]** = Leaf **DDR_MA[10]** = 0	Column burst write Column address on **DDR_MA[13:0]**
Write w/ Auto-Precharge	0	1	0	0	**DDR_BA[1:0]** = Leaf **DDR_MA[10]** = 1	Column burst write with row precharge at the end of the transfer
Precharge	0	0	1	0	**DDR_BA[1:0]** = Leaf **DDR_MA[10]** = 0	Precharge a single leaf
Precharge All	0	0	1	0	**DDR_MA[10]** = 1	Precharge all leaves
Auto-Refresh	0	0	0	1		Refresh both banks from on-chip refresh counter
Self-Refresh	0	0	0	1	**CKE** = 0	Refresh autonomously as **CKE** = 0
Power Down	X	X	X	X	**CKE** = 0	Power down if both banks precharged when **CKE** = 0
Stop	0	1	1	0		Interrupt a read or write burst.

1. This table copied from *New DRAM Technologies* by Steven Przybylski.
2. Shaded boxes indicate commands not supported by the IXP43X network processors. They are included for completeness.
3. During a Mode Register Set command, DDR_BA[1:0] = 00_2 selects the Mode Register, DDR_BA[1:0] = 01_2, 10_2 or 11_2 selects the Extended Mode Registers, all others are reserved.

DDR SDRAM commands are synchronous to the clock so the MCU sets up the above conditions prior to the DDR_CK[2:0] rising edge.

11.2.2.10 DDR SDRAM Initialization

Since DDR SDRAM devices contain a controller within the device, the MCU must initialize them specifically. Upon the deassertion of RESET_N, software initializes the DDR SDRAM devices with the sequence illustrated with Figure 124:

1. The MCU applies the clock (DDR_CK[2:0]) at power up along with system power (clock frequency unknown).
2. The MCU must stabilize DDR_CK[2:0] within 100 µs after power stabilizes.
3. The MCU holds all the control inputs inactive (DDR_RAS_N, DDR_CAS_N, DDR_WE_N, DDR_CS_N[1:0] = 1), places all data outputs and strobes in the High-Z state (DDR_DQS[4:0], DDR_DQ[31:0]), and deasserts DDR_CKE[1:0] for a minimum of 200 µs after supply voltage reaches the desired level.
4. Software disables the refresh counter by setting the RFR to zero.
5. Software issues one **NOP** cycle after the 200 µs device deselect. A **NOP** is accomplished by setting the SDIR to 0011_2. The MCU asserts DDR_CKE[1:0] with the NOP.
6. Software must then wait for 500ns. (Step 6. only necessary for DDR-II devices)
7. Software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0010_2.
8. Software issues an **extended-mode-register2-set** command by writing 0111_2 to the SDIR. (Step 8. only necessary for DDR-II devices)
9. Software issues an **extended-mode-register3-set** command by writing 1000_2 to the SDIR. (Step 9. only necessary for DDR-II devices)
10. Software issues an **extended-mode-register1-set** command to enable the DLL by writing 0100_2 to the SDIR. The MCU supports the following DDRI and DDR-II SDRAM mode parameters:
    a. DLL = Enable/Disable
    b. Off-Chip Driver (OCD) Impedance Adjustment (set DCAL registers) applies to DDR-II SDRAM only.
    c. Additive Latency (AL) is always zero for the IXP43X network processors. The MCU only supports an AL of zero because the MCU does not support back-to-back Active to Read or Write commands that would otherwise be in violation of tRCD (Active to Read/Write command delay): tRCD is obeyed at all times.

Figure 121. Supported DDR SDRAM Extended Mode Register Settings

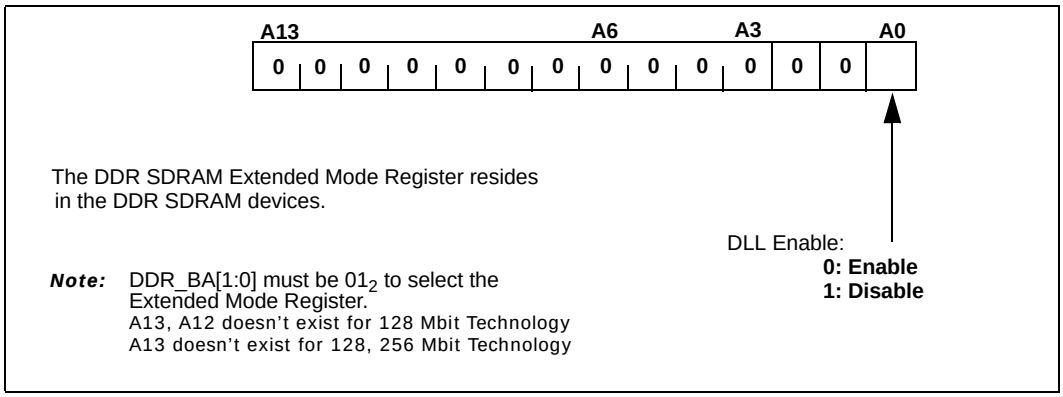
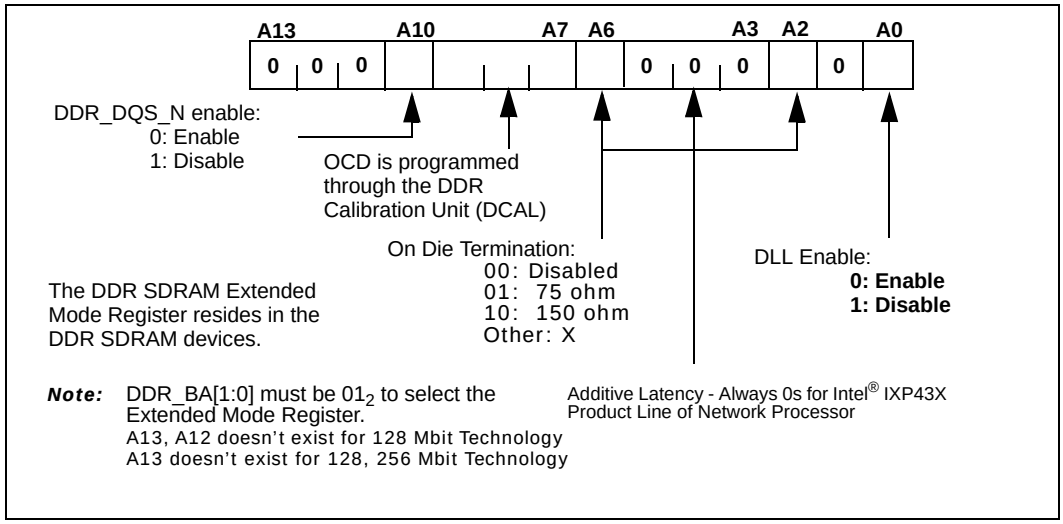
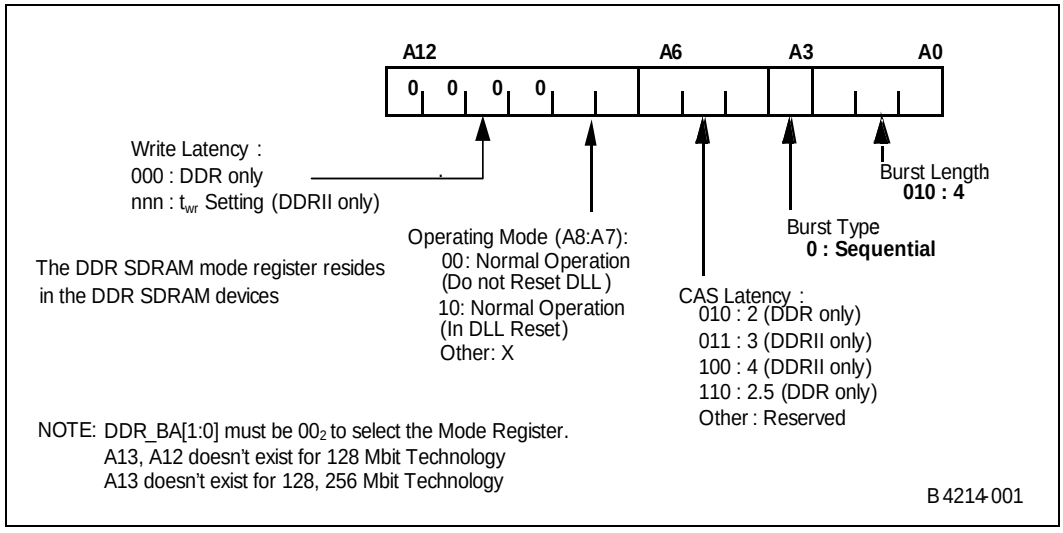


Figure 122. Supported DDR-II SDRAM Extended Mode Register Settings



11. After waiting T_{mrd} cycles, software issues a **mode-register-set** command by writing 0001_2 to the SDIR to program the DDR SDRAM parameters and to reset the DLL. The MCU supports the following DDRI and DDR-II SDRAM mode parameters:
    a. CAS Latency (tCAS) = two or two and one-half for DDRI, or three or four for DDR-II based on the programmed setting in Section 11.6.2, "DDR SDRAM Control Register 0 SDCR0"
    b. Burst Type = Sequential
    c. Burst Length (BL) = four for 16-bit and 32-bit data bus options.

Figure 123. Supported DDR-I/II SDRAM Mode Register Settings



12. After waiting T_{mrd} cycles, software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0010_2.
13. After waiting T_{rp} cycles, software provides two **auto-refresh** cycles. An **auto-refresh** cycle is accomplished by setting the SDIR to 0110_2. Software must ensure at least T_{rfc} cycles between each **auto-refresh** command.

14. Following the second **auto-refresh** cycle, software must wait T_{rfc} cycles. Then, software issues a **mode-register-set** command by writing to the SDIR to program the DDR SDRAM parameters **without** resetting the DLL by writing 0000_2 to the SDIR.
15. The MCU may issue a **row-activate** command T_{mrd} cycles after the **mode-register-set** command.
16. EMRS OCD Default command (A9=A8=A7=1) followed by EMRS OCD Calibration Mode Exit command (A9=A8=A7=0) must be issued. The 'OCD Program' field is mapped to DCALADDR register bit[9:7].
    a. Configure the DCALCSR register (Hex offset: CC00 F500) with bit[2:0]='011'
    b. Configure the DCALADDR register (Hex offset: CC00 F504) with bit[1:0]='01', and bit[9:7]='111' to issue a OCD Default command
    c. Software must wait for at least two cycles. Then configure the DCALADDR register (Hex offset: CC00 F504) with bit[1:0]='01', and bit[9:7]='111'. (To issue a OCD Exit command)

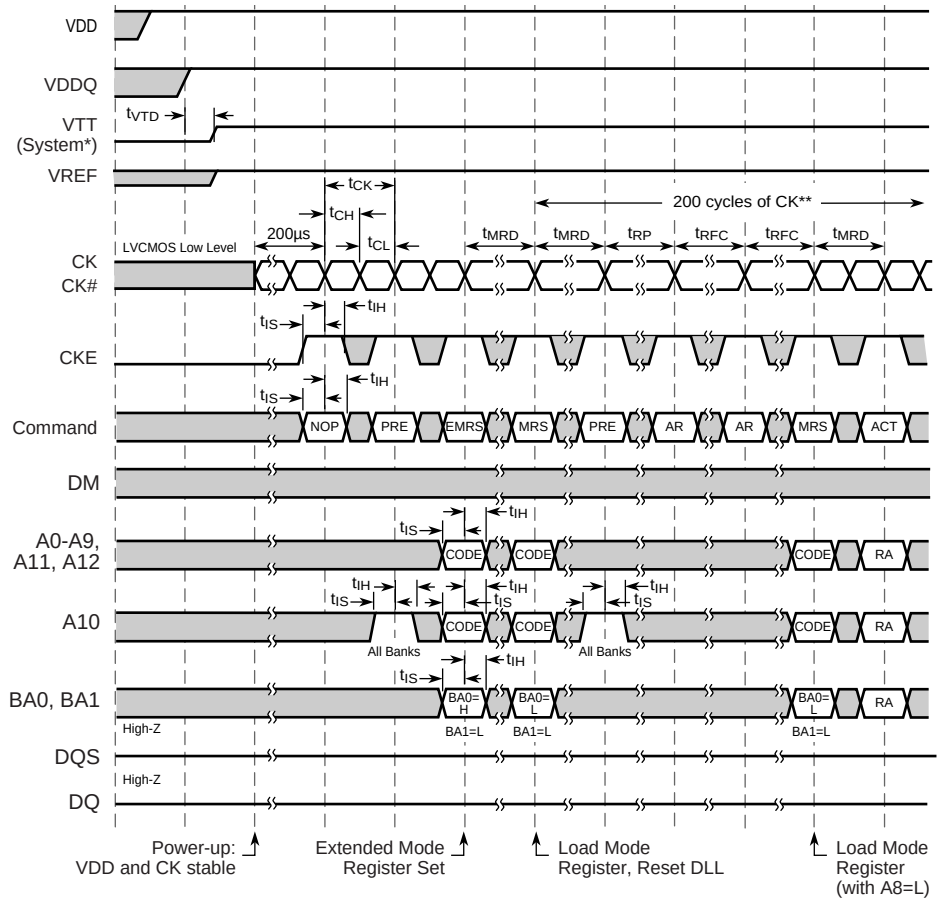
Note: Other fields in the DCALCSR and DCALADDR registers should be left as 'zero' (default value after reset). The 'Qoff', 'RDQS', and 'DQS' fields are mapped to DCALADDR register bit[12:10] respectively.

The DLL enabled is configured through writing '0100' to the SDIR register. The Rtt is configured by writing to SDCR0 register bit[5:4].

17. Software re-enables the refresh counter by setting the RFR to the required value.

The waveform in Figure 124 illustrates the DDR SDRAM initialization sequence.

Figure 124. DDR SDRAM Initialization Sequence (Controlled with Software)



Notes:

* VTT is not applied directly to the device, however t_{VTD} must be greater than or equal to zero to avoid device latchup.

** t_{MRD} is required before any command can be applied and 200 cycles of CK are required before a Read command can be applied.

The two Autorefresh commands may be moved to follow the first MRS, but precede the second Precharge All command.

= Don't Care

B2452-01

If the DDR SDRAM subsystem implements ECC (see Section 11.2.3, "Error Correction and Detection"), then initialization software must initialize the entire memory array with the IXP43X network processors. It is important that every memory location has a valid ECC byte. The Intel XScale processor is used to fill the memory array with a constant, thereby initializing the associated ECC bytes in the process. If the memory array is not initialized, the BIU or PCC may attempt to read memory locations beyond the specified word(s). In this case, the MCU reports an ECC error even though software did not specifically request the un-initialized data.

11.2.2.11 DDR SDRAM Mode Programming

The MCU programs the DDR SDRAM devices through a **mode-register-set** command. During the initialization sequence this command sets the DDR SDRAM mode register (see Section 11.2.2.10, "DDR SDRAM Initialization") by programming the SDIR and SDCR[1:0].

The DDR SDRAM state machine ensures that a **row-activate** command is issued no sooner than T_{mrd} cycles after the **mode-register-set** command.

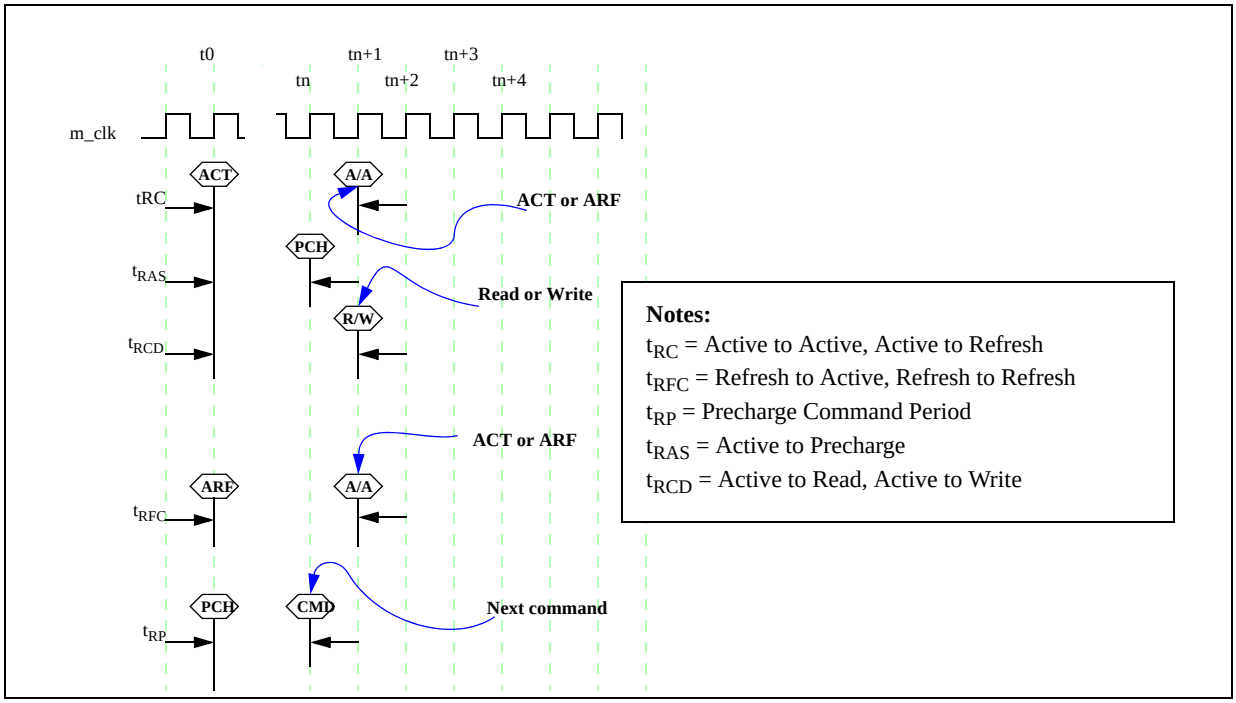
The values to be programmed in the SDCR[1:0] registers are based on the SDRAM devices being interfaced with the IXP43X network processors. Because the parameters that define the time between allowed commands are programmable, this allows flexibility in the type of DDR device that is selected, in addition to de-coupling the hardware to any frequency dependencies.

Note: The MCU_DDRSM does NOT interact properly with the DDR SDRAM until the SDCR[1:0] registers have been programmed.

The duration between valid commands is programmed by the user before the DDRSM can interact properly with the DDR. These parameters are defined by JEDEC and/or within this document in the timing diagrams and equations in Section 11.6.2, "DDR SDRAM Control Register 0 SDCR0" and Section 11.6.3, "DDR SDRAM Control Register 1 SDCR1".

The timing parameters for all non-read and non-write commands are derived directly from the *JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79*, January 2004 and *JEDEC DDR - II SDRAM Specification*, January 2004. These parameters, and their relationship to each other, are outlined in Figure 125. Refer device specification for timing parameters specific to the DDR device that is to be implemented in the system.

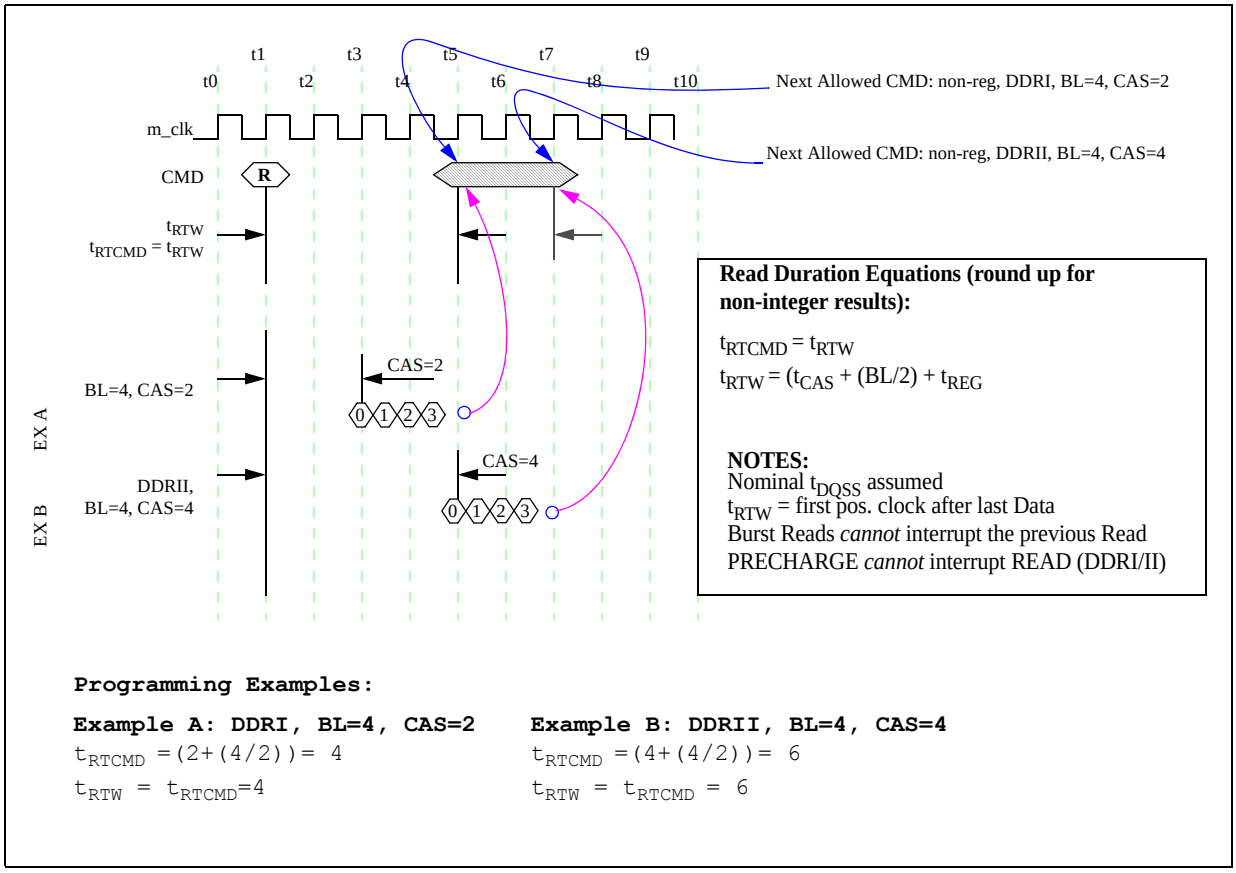
Figure 125. MCU Active, Precharge, Refresh Command Timing Diagram



The timing parameters for DDR reads are defined in Figure 126. Both Read to Write (IXP43X network processors t_{RTW}) and Read to Command (IXP43X network processors t_{RTCMD}) are defined the same. It is important to note that they remain separate for two reasons: 1) For similarity to the DDR Write timing parameters, and 2) to allow for flexibility in programming the MCU that might not have been comprehended at the time of it's design. Both parameters take into account CAS latency (**JEDEC**: t_{CAS}) and Burst Length (**JEDEC**: *BL*).

Note: Burst Length is fixed at four for 16-bit and 32-bit data bus option for the IXP43X network processors. The MCU allows for back-to-back reads, as long as they are to a currently open page.

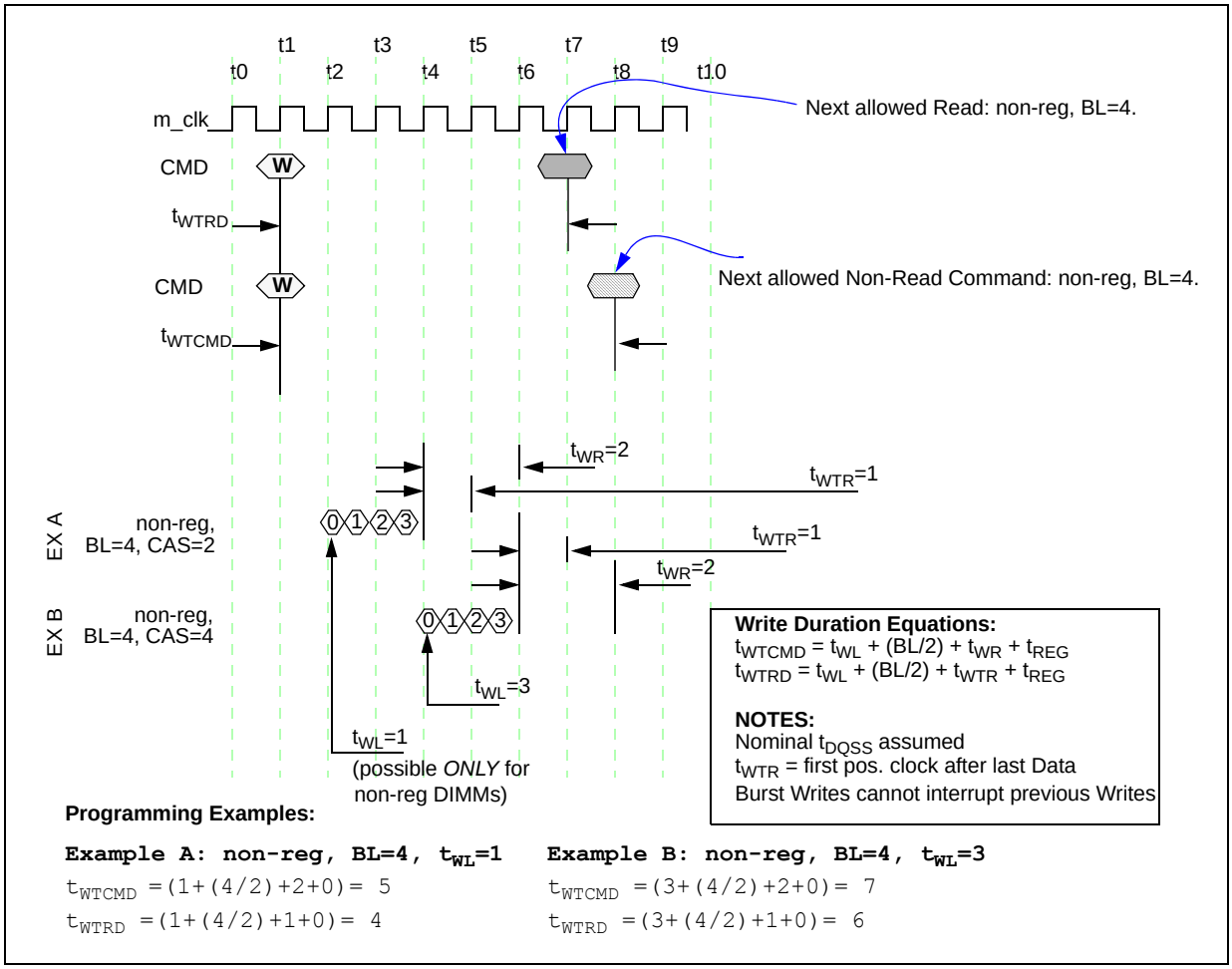
Figure 126. MCU DDR Read Command to Next Command Timing Diagram



The timing parameters for DDR writes are defined in Figure 127. Both Write to Read (IXP43X network processors: t_{WTRD}) and Write to Command (IXP43X network processors: t_{WTCMD}) are defined in much the same manner, both accounting for Write Latency (**JEDEC**: *tWL*) and Burst Length (**JEDEC**: *BL*). The difference lies in the compensation for Write Recovery (**JEDEC**: t_{WR}) and Write to Read (**JEDEC**: t_{WTR}).

Note: Burst Length is fixed at four for 16-bit and 32-bit data bus options for the IXP43X network processors.The MCU allows for back-to-back Writes, as long as they are to an open page.

Figure 127. MCU DDR Write Command to Next Command Timing Diagrams



11.2.2.12 DDR SDRAM Read Cycle

The MCU performance is optimized for page hits and the MCUs behavior is different for the hit and miss scenario.

The waveform for a read including the row activation in the case of a page miss is illustrated in Figure 129. For a page hit, the two cycles required for row activation are saved resulting in lower first word read latency.

The MCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDR SDRAM Control Block to issue the transaction command to the DDR array if the previous transaction is the same type (read or write). In addition, the DDR SDRAM Control Block supports pipelining of transactions that allows the column address of the next transaction to be issued before the current transaction's data transfer is completed by the DDR SDRAM devices. These optimizations are illustrated in Figure 128 for random read memory transactions.

Figure 128. DDR SDRAM Pipelined Reads

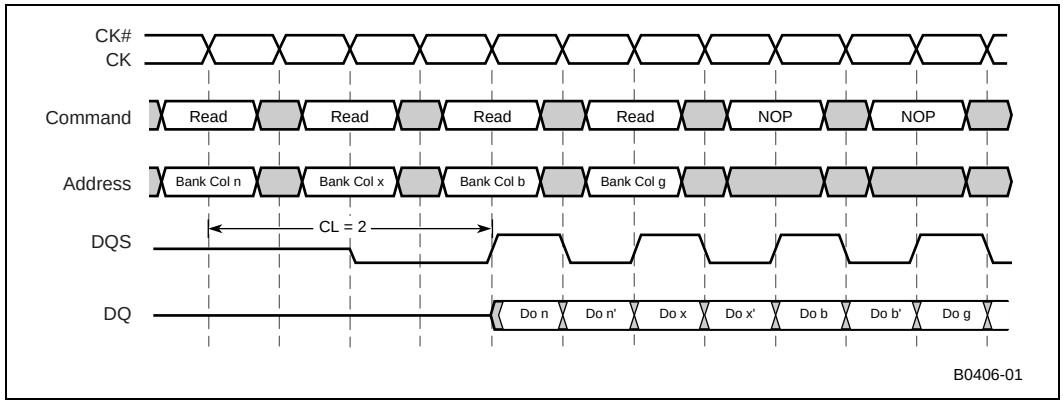
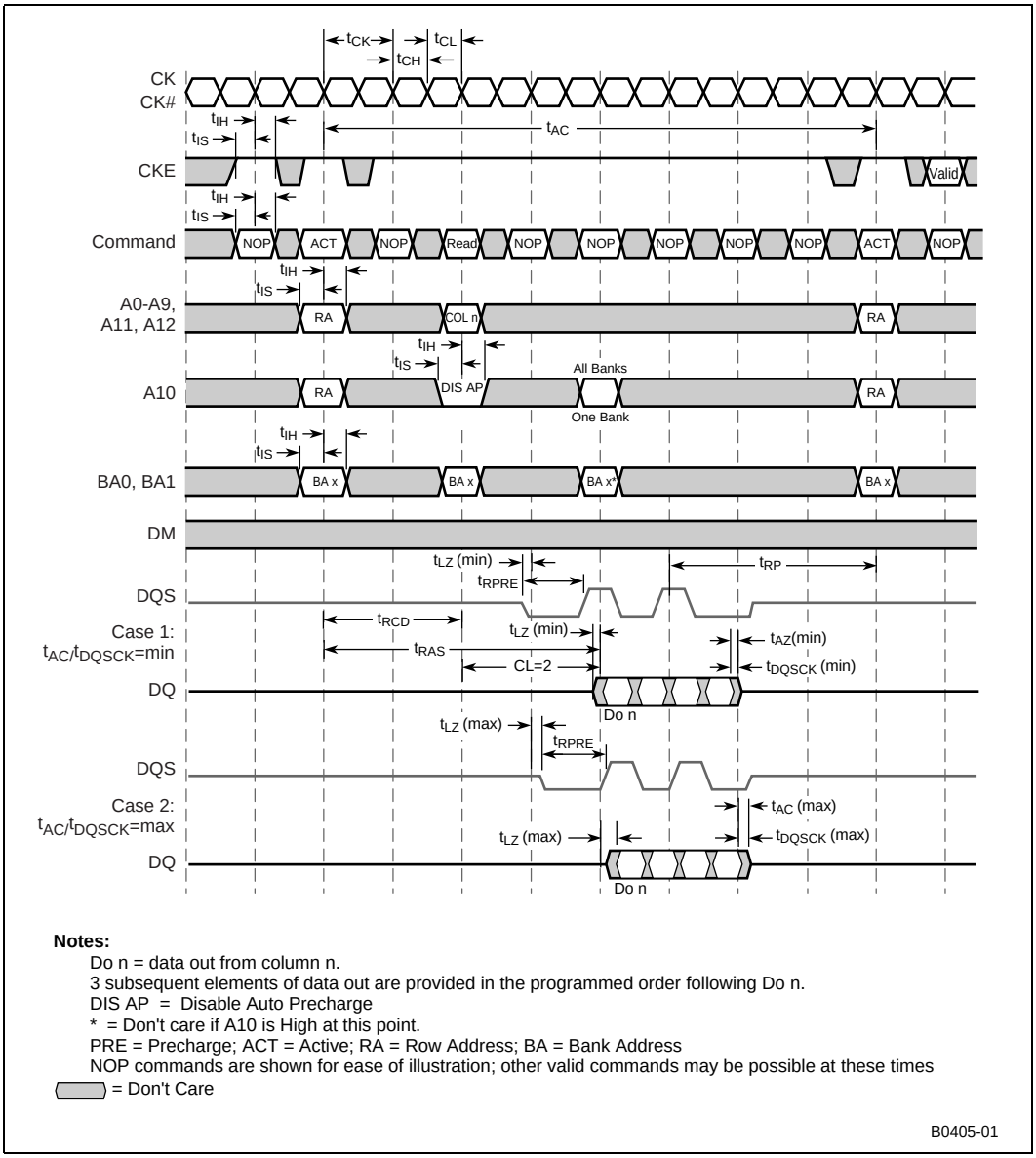


Figure 129. DDR SDRAM Read, 36 Bytes, ECC Enabled, BL=4



Notes:

Do n = data out from column n.
3 subsequent elements of data out are provided in the programmed order following Do n.
DIS AP = Disable Auto Precharge
* = Don't care if A10 is High at this point.
PRE = Precharge; ACT = Active; RA = Row Address; BA = Bank Address
NOP commands are shown for ease of illustration; other valid commands may be possible at these times
= Don't Care

B0405-01

1. Each of the MCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
   — If the address falls in the DDR SDRAM address range indicated by the SDBR, SBR0, and SBR1 the MCU claims the transaction and latches the transaction in the respective memory transaction queue.
2. Once the MARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDR SDRAM Control Block. The DDR SDRAM Control Block decodes the address to determine whether or not any of the open pages are hit.

A read that misses the open pages encounters a miss penalty because the currently open page must be closed before the read is issued to the new page. Refer to Section 11.2.2.7, "Page Hit/Miss Determination" for the paging algorithm details. If a page hit occurs, steps 3-4 are skipped by the MCU.

3. The DDR SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row.
   — The DDR SDRAM Control Block waits T_{rp} cycles after the precharge before issuing the **row-activate** command for the new read transaction.
4. The **row-activate** command enables the appropriate row.
   — The DDR SDRAM Control Block asserts DDR_RAS_N, de-asserts DDR_WE_N, and drives the row address on DDR_MA[13:0].
5. In the following cycle in the case of a page hit or after T_{rcd} cycles in the case of a page miss, the DDR SDRAM Control Block asserts DDR_CAS_N, de-asserts DDR_WE_N, and places the column address on DDR_MA[13:0]. This initiates the burst read cycle.
6. After the CAS latency expires, the DDR SDRAM device drives data to the MCU. A CAS latency of 2 is depicted in Figure 129.
7. Upon receipt of the data, the DDR SDRAM Control Block calculates the ECC code from the data and compares it with the ECC returned by the DDR SDRAM array. Section 11.2.3, "Error Correction and Detection" explains the ECC algorithm in more detail.
8. Assuming the calculated ECC matches the read ECC, the DDR SDRAM Control Block drives the data back to the corresponding memory transaction queue.
   — For each burst read issued, the memory controller increments the column address by four.

The MCU continues to return data to the corresponding memory transaction queue based on the byte count of the transaction.

Note: The burst cycle continues within the SDRAM devices unless the next read/write transaction is ready to be issued. The current SDRAM burst read may be interrupted by issuing the next **read** or **write** command.

The MCU divides transactions if they cross a page boundary. For read transactions, the transactions are completed in multiple completions that end at page boundaries. Write transactions receives posted data to the inbound memory transaction queue up to a full posted write queue (32 Bytes). Multiple write commands is then issued to the DDR SDRAM to complete the write of data to the multiple pages.

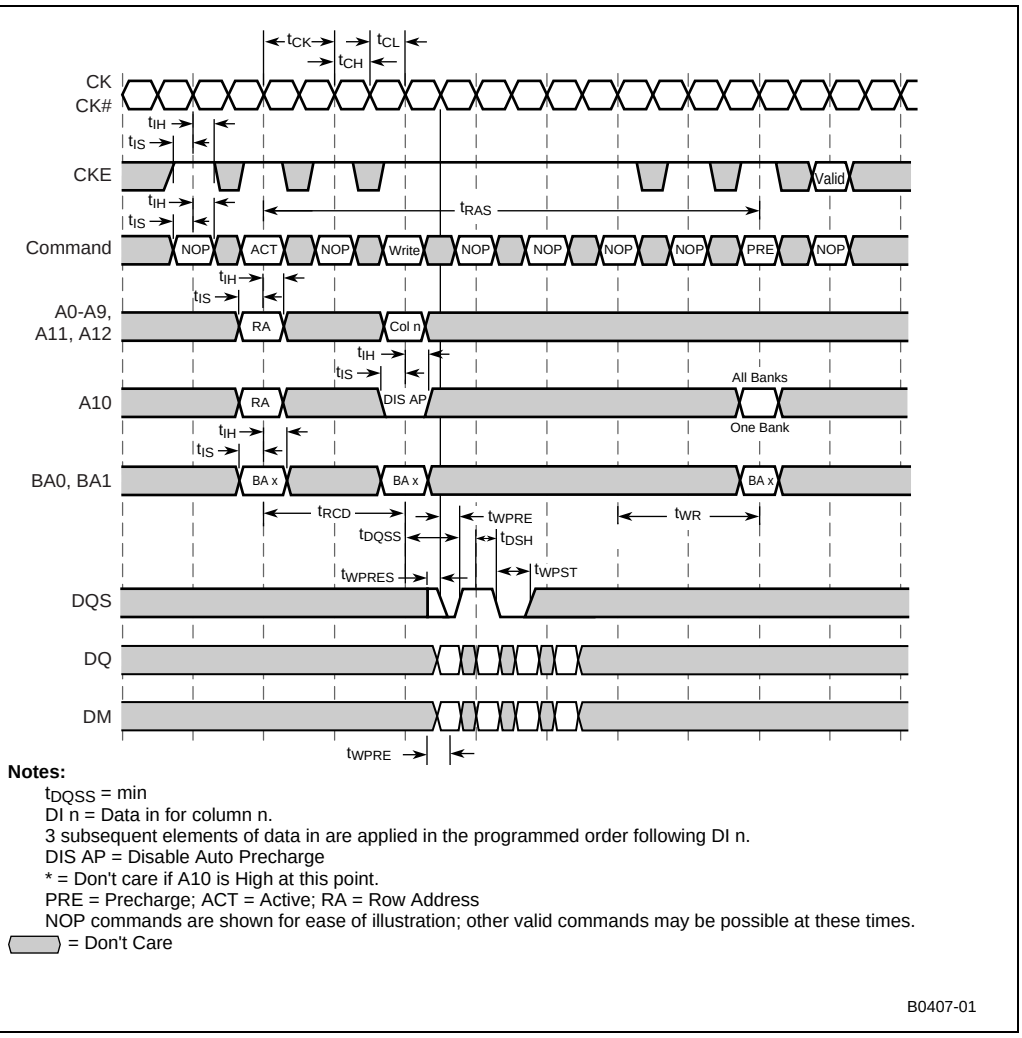
11.2.2.13 DDR SDRAM Write Cycle

All write transactions to the DDR SDRAM are posted to the MCU in the memory transaction queues. This implies that the transaction completes between a given port and corresponding unit prior to data being written to the SDRAM array. Once the MARB issues a **write** command from a memory transaction queue to the MCU DDR Control Block, the paging logic makes a hit/miss comparison (illustrated in Figure 119). The performance is best for page hits and therefore the MCUs behavior is different for the hit and miss scenario.

Write transactions require ECC codes to be generated and stored in the SDRAM array with the data being written. The behavior is different depending on the size of the data being written. Section 11.2.3, "Error Correction and Detection" explains the ECC algorithm in more detail.

For a page hit, the MCU need not open the page (assert **DDR_RAS_N**) and avoids the RAS-to-CAS delay achieving greater performance. The waveform for a write including the row activation in the case of a page miss is illustrated in Figure 130. For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.

Figure 130. DDR SDRAM Write, 36 Bytes, ECC Enabled, BL=4



1. Each of the MCU inbound memory transaction ports decodes the address to determine if the transaction should be claimed.
   — If the address falls in the DDR SDRAM address range indicated by the SDBR, SBR0, and SBR1 the MCU claims the transaction and latches the transaction in the respective memory transaction queue.
2. Once the MARB selects the highest priority transaction from the memory transaction queues, it forwards the transaction to the DDR SDRAM Control Block. The DDR SDRAM Control Block decodes the address to determine whether or not any of the open pages are hit.
   — The ECC logic generates the ECC code for the data to be written.

A write that misses the open page encounters a miss penalty because the currently open page must be closed before the write is issued to the new page. Refer to Section 11.2.2.7, "Page Hit/Miss Determination" for the paging algorithm details. If a page hit occurs, steps 2-3 are skipped by the MCU.

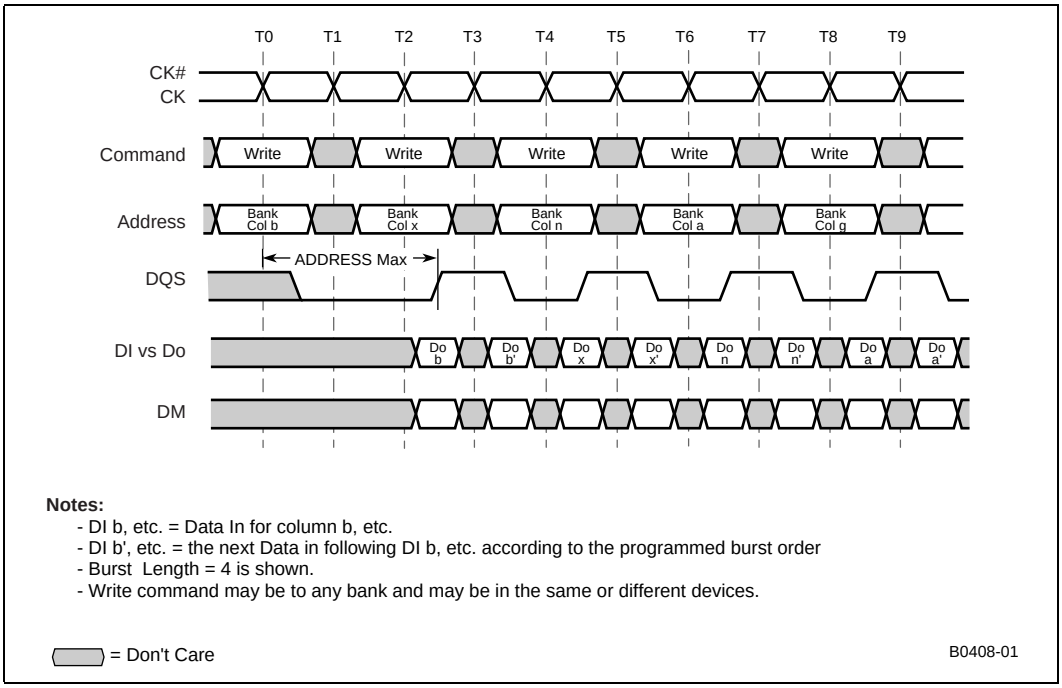
3. The DDR SDRAM Control Block closes the currently open page by issuing a **precharge** command to the currently open row.
   — The DDR SDRAM Control Block waits T_{rp}cycles after the precharge before issuing the **row-activate** command for the new write transaction.
4. The **row-activate** command enables the appropriate row.
   — The DDR SDRAM Control Block asserts DDR_RAS_N, de-asserts DDR_WE_N, and drives the row address on DDR_MA[13:0].
5. After T_{rcd} cycles in the case of a page miss, the DDR SDRAM Control Block asserts DDR_CAS_N, asserts DDR_WE_N, and places the column address on DDR_MA[13:0]. This initiates the burst write cycle. The DDR SDRAM Control Block drives the data to be written and its ECC code to the DDR SDRAM devices.

- The DDR SDRAM Control Block drives the new data to the corresponding memory transaction queue each cycle until the transaction is completed with the byte count expiring, or the transaction is interrupted if preemption conditions are met.
- For each burst issued, the DDR SDRAM Control Block increments the address by four.
- If ECC is enabled, when the data to write is not aligned on a 4 byte boundary for 32-bit data bus width, the DDR SDRAM Control Block performs a read-modify-write of the entire 4 byte aligned double-word for 32-bit data bus width and incorporate the new data when regenerating ECC.

Note: Even though the data is not written as the DDR SDRAM Control Block asserts DDR_DM[4:0], the burst cycle completes.

The MCU supports optimized performance for random address transactions. This optimization eliminates the need of the DDR SDRAM Control Block to issue the transaction command to the DDR array if the previous transaction is the same type (read or write). In addition, the DDR SDRAM Control Block supports pipelining of transactions that allows the column address of the next transaction to be issued before the current transaction's data transfer is completed by the DDR SDRAM devices. These optimizations are illustrated in Figure 131 for random write memory transactions.

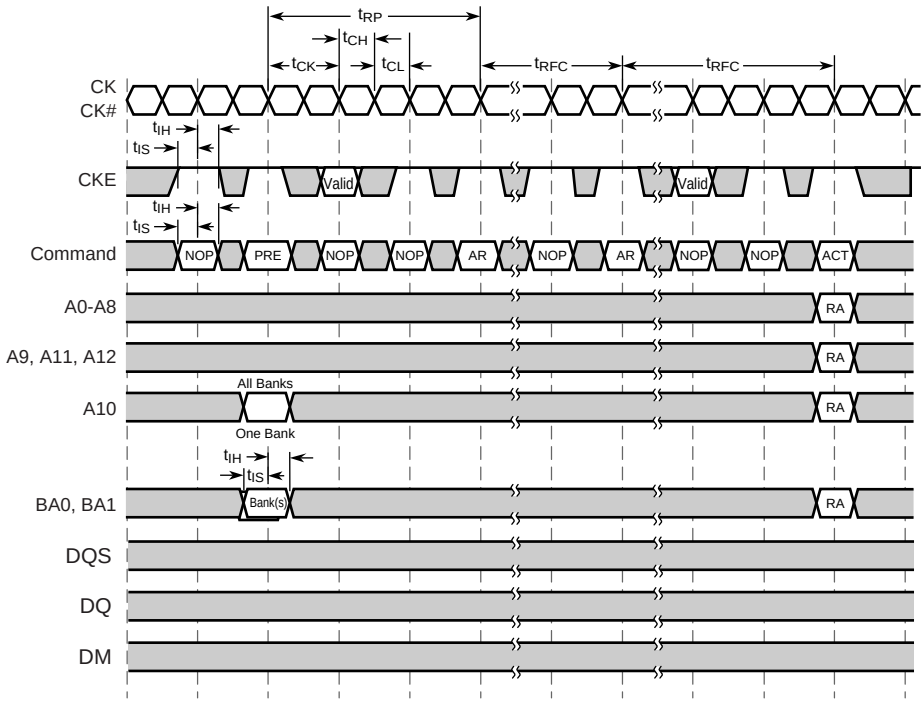
Figure 131. DDR SDRAM Pipelined Writes



11.2.2.14 DDR SDRAM Refresh Cycle

Since the DDR SDRAM is a dynamic memory, the MCU issues a refresh cycle periodically. The interval of these refresh cycles is programmable in the RFR register. The DDR SDRAM device generates the refresh address internally. The MCU initiates two sequential refresh cycles (one per bank) after the MCUs refresh timer expires and any current transaction is complete. To preserve the correct refresh period, the refresh timer continues counting after it expires to prevent a gradual skewing of the refresh interval. The waveform in Figure 132 illustrates the case where the refresh timer expires as the memory bus is not busy.

Figure 132. Refresh While the Memory Bus is Not Busy



Notes:
PRE = Precharge; ACT = Active; RA = Row address; BA = Bank address; AR = Autorefresh
NOP commands are shown for ease of illustration; other valid commands may be possible at these times.
DM, DQ, and DQS signals are all don't care/high-Z operations shown.
= Don't Care

A7799-01

- Once the refresh timer expires, the MCU knows that a refresh cycle is necessary.
  - The refresh timer continues to count for the next refresh cycle.
- The MARB allows the current transaction to complete.
  - If the DDR SDRAM Control Block and the DDR SDRAM array are transferring data, or a CMTQ tenure is ongoing, the refresh cycle is queued until the transaction is completed or the CMTQ tenure expires.
- The DDR SDRAM Control Block closes all open pages with a **precharge-all** command to all the populated DDR SDRAM banks.
  - The DDR SDRAM Control Block resets the page register valid bits.
- The DDR SDRAM Control Block issues an **auto-refresh** command to DDR SDRAM bank 0.
  - This command affects all internal leaves.
- In the next cycle, the DDR SDRAM Control Block issues an **auto-refresh** command to DDR SDRAM bank 1.
  - The MCUs internal 2-bit refresh counter is decremented by one. This is actually done when the DDRSM consumes the command from the ARB.
- After T_{rfc} cycles, the DDR SDRAM Control Block can service a new transaction or another refresh cycle.

It is recommended that the RFR (Section 11.6.13, "Refresh Frequency Register RFR") is programmed with the value to achieve 7.8 µs, though some DDR SDRAM devices may provide for the ability to refresh at a period of 15.6 us. The value is based on the frequency of the DDR SDRAM and Table 205 can provide for these two typical values.

Table 205. Typical Refresh Frequency Register Values

DDR Speed	7.8 µs Value	15.6 µs Value
266 MHz	0410H	0820H
400 MHz	0610H	0C30H

The longest possible internal bus transaction is writing a 32-byte burst where each data cycle results in a read-modify-write due to partial writes. See Section 11.2.3.2, "ECC Generation for Partial Writes" for details. The longest possible CMTQ tenure is 16 transactions where each of the transaction are page misses and partial writes.

The MCU uses a 2-bit counter to queue refresh cycles. If another refresh cycle is queued when a current refresh cycle completes, the MCU services the queued refresh cycle before servicing any internal bus requests. If the counter exceeds three (that should never happen), the counter does NOT roll over and three refreshes are serviced when the MCU is available.

11.2.2.15 DDR SDRAM Debugging Procedure

You may encounter error during READ / WRITE operation when the receive enable delay or/and drive strength settings are configured to the optimal value. It may be resolved by performing the procedure mentioned below:

1. Review the value configured in DDR SDRAM Control Register 0 SDCR0 and DDR SDRAM Control Register 1 SDCR1 registers. They must be programmed according to the JEDEC specs/ DRAM component datasheet.
2. Configure the Receive Enable Delay Register RCVDLY register bit[2:0] by incrementing/ decrementing the value. Perform a WRITE and then READ it back for data comparison. Repeat this procedure until an optimal value has been identified.

Note: The Receive Enable Delay Register RCVDLY register configuration must be optimized for the newly designed platform. The default value may not work.

3. Configure the DDR Drive Strength Control Register LEGOVERIDE register bit[3:0] by incrementing/ decrementing the value. The bit 4 should be set to '0'. Perform a WRITE and then READ it back for data comparison. Repeat this procedure until an optimal value has been identified.

Note: The DDR Drive Strength Control Register LEGOVERIDE register configuration must be optimized for the newly designed platform. The default value may not work.

11.2.3 Error Correction and Detection

The memory controller of the IXP43X network processors supports a 32-bit data bus width memory implementation with and without ECC, and supports a 16-bit data bus width memory implementation without ECC.

The MCU is capable of correcting any single bit errors and detecting any double bit errors in the IXP43X network processors DDR SDRAM memory subsystem. ECC enhances the reliability of a memory subsystem by correcting single bit errors caused by electrical noise or occasional alpha particle hits on the DDR SDRAM devices.

Similar to parity, that detects single bit errors, error correction requires an additional 8-bit code word for the 32-bit datum. This means that a memory must have the additional 8-bit error correction code (DDR_CB[7:0]) per 32-bit datum (DDR_DQ[31:0]) resulting in a 40-bit wide memory subsystem. During DDR SDRAM read cycles, the DDR SDRAM Control Block detects single-bit errors and corrects the data prior to returning the data to the respective memory transaction queue. DDR SDRAM write cycles generate the ECC and sends it with the data to the memories.

In 32-bit wide memory, the IXP43X network processors will zero extend the 32-bit datum to a 64-bit datum to generate, check and correct ECC. This means that a 32-bit datum memory with ECC results in a 40-bit wide memory since an 8-bit error correction code is still required.

The ECC algorithm of the IXP43X network processors use H-matrix/G-Matrix method used by the Intel® Pentium® Pro Processor (Figure 134 and Figure 137). The H-matrix reads a non-zero syndrome and indicates the bit that was incorrect for single-bit errors.

Scrubbing is the process of correcting an error in the memory array. The chance of an unrecoverable multi-bit error increases if the software does not correct a single-bit error in the array. For the IXP43X network processors, scrubbing is handled by software. If error reporting is enabled, the MCU logs the error type in ELOG0 or ELOG1 and the address in ECAR0 or ECAR1 when an error occurs.

11.2.3.1 ECC Generation

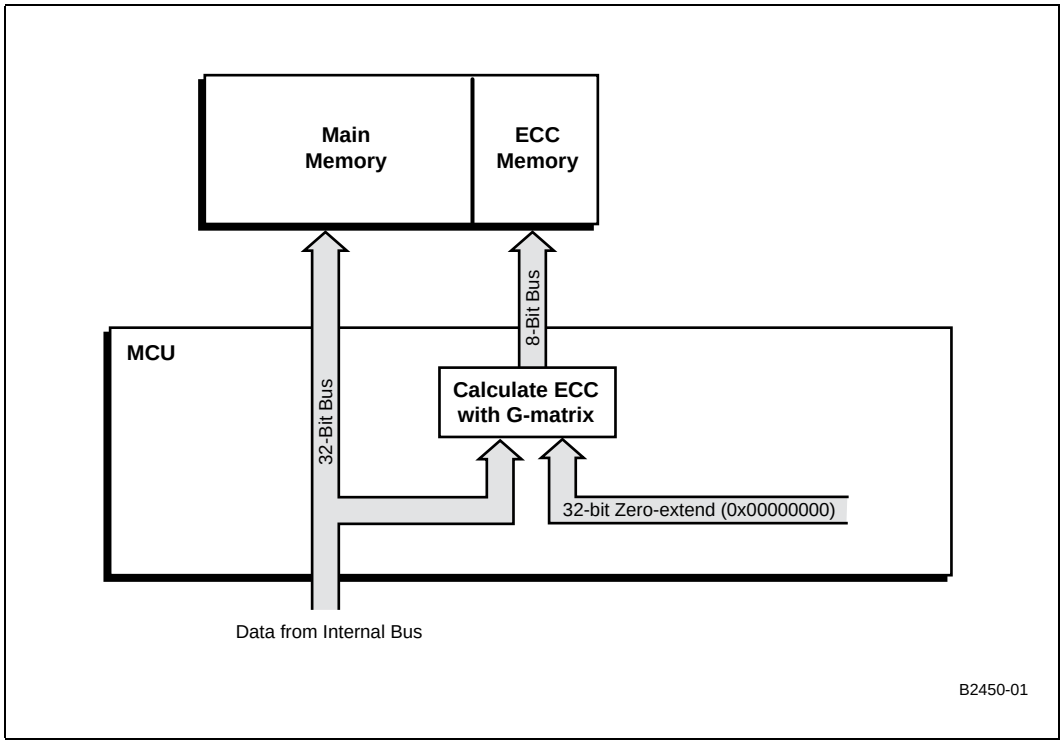
For write operations, the MCU generates the error correction code that is written along with the data. This section describes the operation of the DDR SDRAM Control Block for ECC generation in a 32-bit wide memory, where the MCU generates 8-bit wide ECC by internally zero extending the data to 64-bits.

The algorithm for a write transaction is:

```
if data to write is 32 bits wide

    Generate the ECC with the G-matrix

    Write the new data and ECC

else {Partial Write}

    Read entire 32-bit data word from memory

    Merge the new data portion with the data from memory

    Generate the new ECC with the G-matrix

    Write new data and ECC
```

Figure 133 shows how the data logically flows through the ECC hardware for a write transaction.

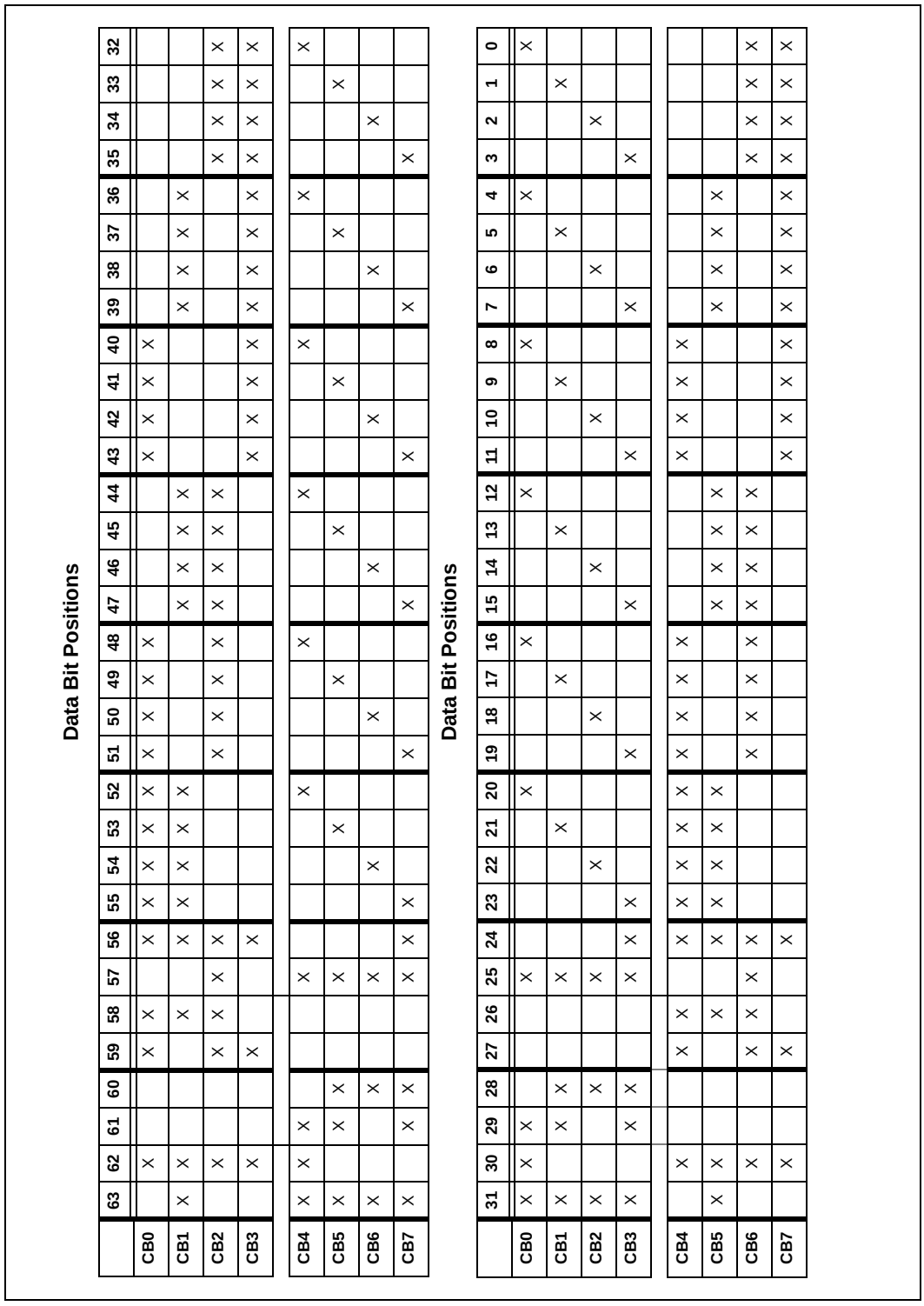
Figure 133. ECC Write Flow



The G-Matrix in Figure 134 generates the ECC. The data to be written is input to the matrix and the output is the ECC code. Each row of the G-Matrix indicates the data bits of DDR_DQ[31:0] zero extended to 64-bits must be XORed together to form the ECC bit. The resulting ECC bits are driven on DDR_CB[7:0].

11.2.3.2 ECC Generation for Partial Writes

Figure 134. Intel® IXP43X Product Line G-Matrix (Generates the ECC)



If the memory transaction writes less than the data bus width programmed in the DDR SDRAM Control Register 0 - SDCR0, then the DDR SDRAM Control Block translates the write transaction into a read-modify-write transaction. For a partial write, the DDR SDRAM Control Block calculates the ECC for the modified datum and writes it back. So, if an external unit issues a write cycle with partial data to an MCU port, the MCU:

1. Issues a 32-bit read.
2. Modifies the value with the new portion to be written.
3. Calculates the ECC on the modified value.
4. Writes the 32-bit value and ECC.

Note: If the MCU detects a single-bit error during the read, it is corrected BEFORE being merged with the write data so the corrected data is written back to the array. If a multi-bit error is detected, the MCU causes an interrupt to the core by writing to the MCISR. The memory location is overwritten by the MCU with the error data but valid ECC, making the contents of memory invalid. For more details on how the MCU handles error conditions, see Section 11.4, “Interrupts/Error Conditions”.

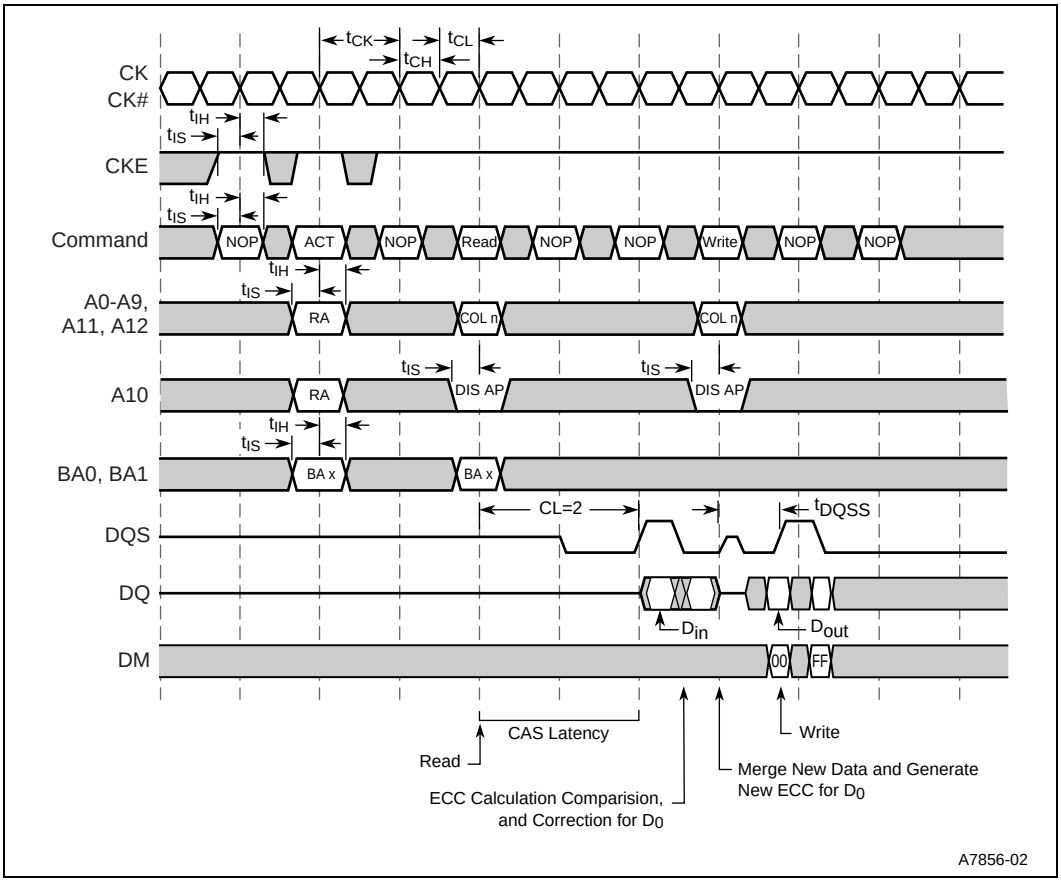
Figure 135 shows an example where the data of a write is less than 32-bits wide. The waveform illustrates how the DDR SDRAM Control Block issues a read-modify-write cycle for the data (D_1).

In 32-bit wide memory, the DDR SDRAM Control Block still generates 8-bit wide ECC by internally zero extending the data to 64-bits. A partial write is a write of less then 4 Bytes.

An undefined INCR read from the AHB bus causes the MCU to read an entire cache line (8 words). If ECC error correction and detection is enabled, it is necessary to initialize a complete cache line before performing an undefined INCR read on any words or bytes within that cache line. If the entire cache line is not initialized and an undefined INCR read occurs, any bytes or words that were not initialized are read, potentially causing an erroneous ECC error.

ECC is enabled or disabled in the ECCR MMR before any transaction to DDR occurs, and it's state must NOT subsequently be changed after DDR transactions have begun.

Figure 135. Sub 32-bit DDR SDRAM Write (D_0)



11.2.3.3 ECC Checking

If enabled, the ECC logic uses the following ECC read algorithm. This algorithm corrects the data before it's driven onto the internal bus.

The ECC algorithm for a read transaction is:

```
Read 32-bit data and 8-bit ECC (For the IXP43X network processors,32 bit read data
is internally zero extended to 64 bits)

Compute the syndrome by passing the 32-bit data through the G-Matrix and XORing
the 8-bit result with the 8-bit ECC

if the syndrome <> 0 {ECC Error}

    Look up in H-matrix to determine error type

    Register the address where the error occurred

    if error is correctable {single bit}

        if single-bit error correction is enabled

            Correct data

            Send corrected data to internal bus

        if single bit error reporting is enabled

            Interrupt core for software scrubbing

    else {uncorrectable}

        if the read cycle is not part of a RMW cycle {read}

            Target-Abort the Internal Bus read transaction.

        else {write requiring RMW}

            Merge the new data portion with the read data from memory

            Generate the new ECC with the G-matrix

            Write new data and ECC

        if multi-bit error reporting is enabled

                Interrupt the core for uncorrectable error
```

When the MCU reads the ECC code from the memory subsystem, it is compared (XORed) with an ECC that the MCU generates from the data read from the memory. The result is called the syndrome. Table 206 shows how the MCU decodes the syndrome for DDR SDRAM read cycles.

Table 206. Syndrome Decoding

Error Type	Symptom
None	The syndrome is 0000 0000.
Single-Bit	Use the H-Matrix in Figure 137 to determine the bit that the MCU inverts to fix the error.
Multi-Bit	If the Syndrome does not match an 8-bit value in the H-matrix, the error is uncorrectable.

Figure 136 shows how the data flows through the ECC hardware for a read transaction.

Figure 136. ECC Read Data Flow

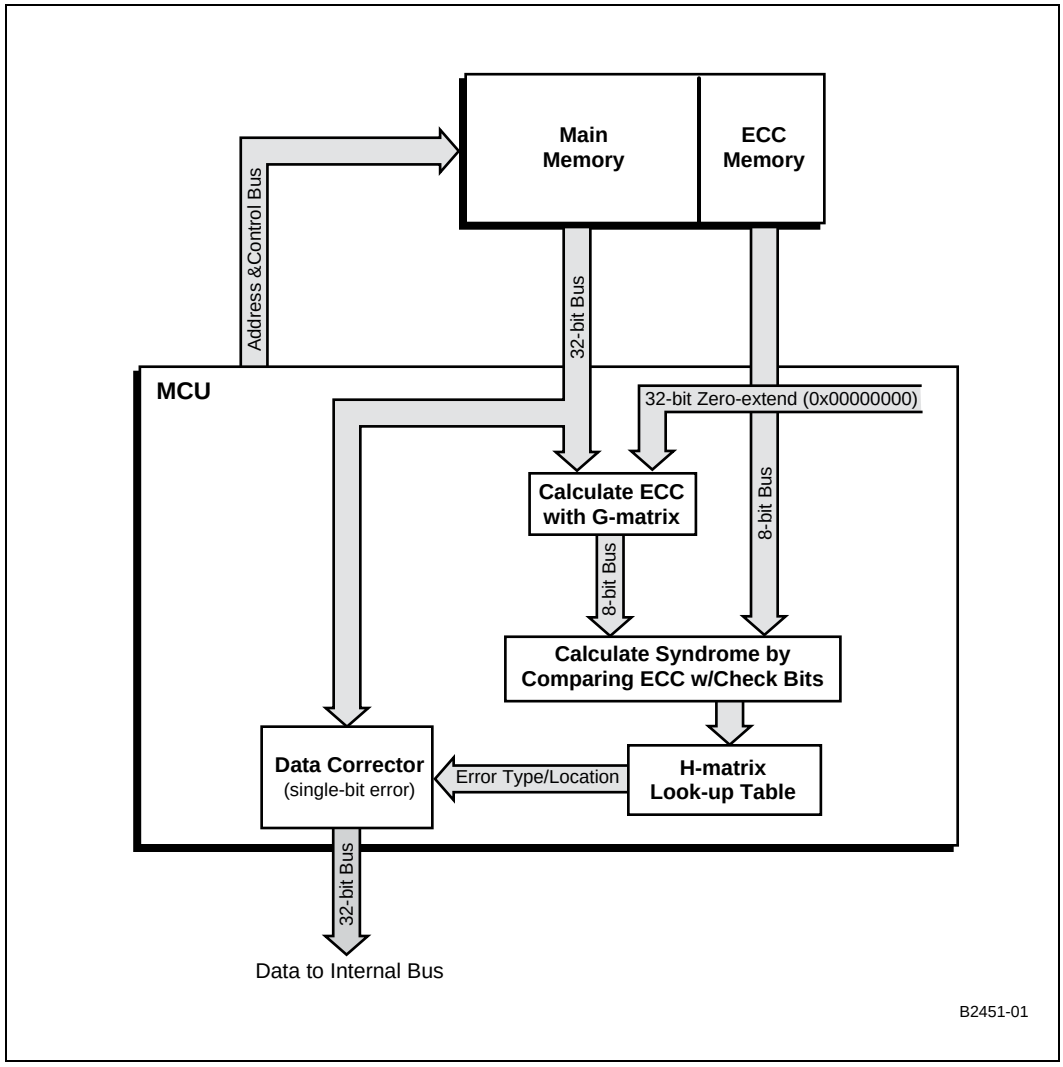
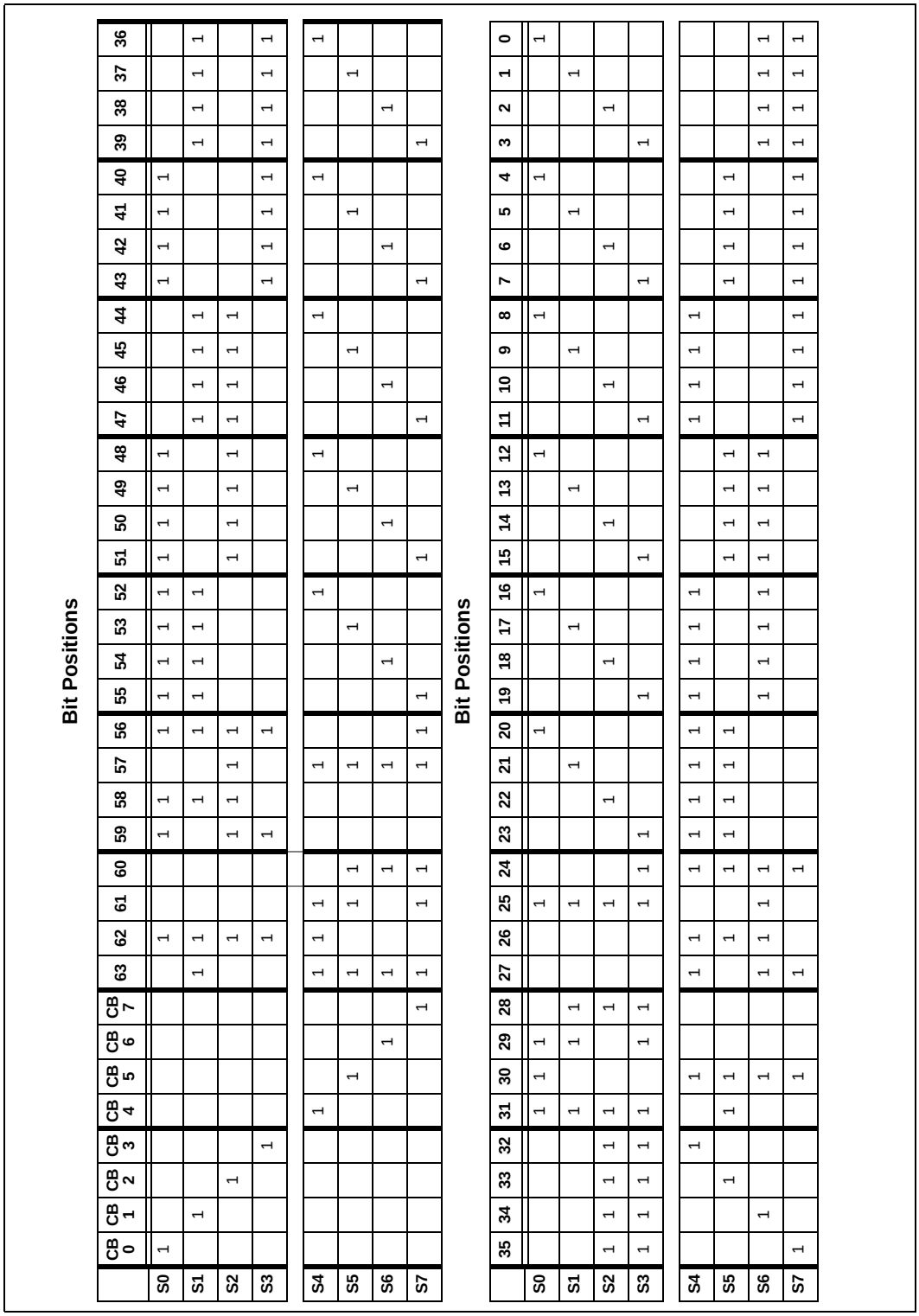


Figure 137 illustrates the H-Matrix used for decoding the syndrome. For single-bit errors, the H-Matrix indicates the bit that contains the error and consequently, the bit to fix.

Figure 137. Intel® IXP43X Product Line H-Matrix (Indicates the Single-Bit Error Location)



Referring to Figure 136, the syndrome bits are created by XORing the data bits as indicated by the appropriate row of the G-Matrix in Figure 134 with the corresponding ECC bit. For example, the MCU derives syndrome bit 0 by XORing data bits 0, 4, 8, 12, 16, 20, 25, 29..31, 40..43, 48..56, 58, 59, 62, and ECC bit 0 (physically read on DDR_CB[0]). For 32-bit memory, the upper 32-bits (bits 63 to 32) are internally zero extended. The MCU performs eight such XOR operations (one per syndrome bit).

If decoding the syndrome indicates multi-bit error (see Table 206), the transaction results in a target-error for Internal Bus transactions, or a multi-bit error in the BIU for core transactions. If an internal bus master detects a target-abort, the master asserts an interrupt to the core. Write cycles are posted to the memory transaction queues, and already completed to the initiating master. For write cycles with a multi-bit error and ECC Error reporting is enabled, the MCU reports the interrupt in the MCISR and interrupts the core.

If the syndrome indicates a single-bit error and single-bit error correction is enabled, the H-Matrix is used to determine the bit in error (see Figure 137). For example, if the syndrome was 1100 0001, the error is with bit 0 of DDR_DQ[31:0]. The MCU inverts bit 0 before driving the data on DDR_DQ[31:0].

If error reporting is enabled in the ECCR and the MCU detects a single-bit or multi-bit error, the MCU stores the address in ECARx and the syndrome in ELOGx. Then, the MCU signals an interrupt to the core. Software decides how to proceed through an interrupt handler. By registering the address in ECARx, software can identify the faulty DIMM.

For details about the MCU error conditions and how the MMR registers are affected, refer to Section 11.4, “Interrupts/Error Conditions”.

Note: In 32-bit wide memory, the DDR SDRAM Control Block still generates 8-bit wide ECC by internally zero extending the data to 64-bits. A partial write is a write of less then 4 Bytes.

11.2.3.4 Scrubbing

Fixing the data error in memory is called scrubbing. The IXP43X network processors rely on Intel XScale processor software to perform the scrubbing. When the MCU detects an error during a read, the MCU logs the address where the error occurred and interrupts the Intel XScale processor. The Intel XScale processor decides how to fix the error through an interrupt handler. Software could decide to perform the scrubbing on:

- the data location that failed
- the entire row of the data that failed
- the entire memory

For single-bit errors reported on a write transaction scrubbing is not required, as the MCU has scrubbed the data during the RMW operation. For single-bit errors, the error is fixed by reading the location that failed and writing back the data after the ECC hardware fixed it. The scrubbing routine should read a DWORD of the 32-bit memory space using a **ld** instruction and write the data back with a **st** instruction. Software should isolate activity on the memory location to guarantee atomicity.

Note: If the scrubbing routine reads the failed location to fix the single-bit error, a second error is reported. Therefore, software should disable single-bit ECC reporting (ECCR[0]) during the scrubbing routine. Also, the scrubbing routine should be aware that partial writes automatically scrubs the QWORD aligned location, if it contains a single-bit ECC error

Multi-bit errors cannot be fixed by the H-Matrix.

11.2.3.4.1 ECC Example Using the H-Matrix

Assume Intel XScale processor writes 1234 5678H to the SDRAM memory space. The Intel XScale processor Address Decoder decodes the address and determines the write should be sent to the Core Memory Transaction Queue. The CMTQ latches the transaction with data 1234 5678H on DDR_DQ[31:0].

During the next CMTQ tenure, this transaction is processed and the DDR SDRAM Control Block receives the data and must calculate the ECC code.

Using the G-Matrix in Figure 134, the DDR SDRAM Control Block creates each check bit by XORing the appropriate bits in the row. Using 0000 0000 1234 5678H (upper 32-bits are internally padded with zeros), the ECC code generated is 7AH. This code is written with the data to the SDRAM devices on DDR_CB[7:0].

Assume that bit 17 was corrupted in the array. Therefore, the bit has been inverted from 0 to 1.

At some later point in time, the core wishes to read from the same address. The Intel XScale processor issues a read transaction that is latched by the CMTQ after the Core Address Decoder decodes the address and determines the read targets the DDR SDRAM address space. Upon the receipt of 1236 5678H on DDR_DQ[31:0], the DDR SDRAM Control Block calculates the syndrome with the G-Matrix in Figure 134. The DDR SDRAM Control Block calculates a syndrome of 52H.

Note: During a memory write, ECC code is created by XORing the appropriate data bits indicated by the G-Matrix (upper 32-bits are internally padded with zeros for 32-bit memory). The syndrome is created during a memory read by XORing the 8-bit value generated by XORing appropriate data bits (DDR_DQ[31:0] with upper 32-bits internally padded with zeros) indicated by the G-Matrix with the check bits (DDR_CB[7:0]).

Referring to Table 206, if the syndrome is non-zero and matches a value in the H-Matrix, there is a single-bit error that is fixed. A syndrome of 52H matches a value in the H-Matrix (see Figure 137)and indicates that bit 17 has an error. The DDR SDRAM Control Block inverts bit 17 prior to returning the corrected data on DDR_DQ[31:0]. The MCU returns 1234 5678H on DDR_DQ[31:0].

Assuming this was the first error, the MCU records the address where the error occurred in ECAR0 and error type in ELOG0. If error reporting is enabled in the ECCR, the MCU writes a 1 to MCISR[0] and that generates an interrupt to the core. A software interrupt handler scrubs the array and fixes the error in bit 17. Unless more errors occur, future reads from this location do not result in an error.

11.2.3.5 ECC Disabled

If software disables ECC, the MCU does generate the ECC byte for writes, but does not check the ECC byte for reads.

11.2.3.6 ECC Testing

Section 11.2.3.4, “Scrubbing” explains how the software is responsible for correcting an error in the memory array once it has been detected by the ECC logic. The MCU implements the ECTST register providing the programmer the ability to test error handling software. For write transactions, the ECTST register value is XORed with the generated ECC. This inverts the bits where the mask is set prior to writing the ECC to memory. When the MCU reads the address later, the ECC mismatches and the error condition occurs (see “Interrupts/Error Conditions” on page 550).

11.2.4 Overlapping Memory Regions

The MCU supports two independent memory regions:

- Memory Mapped Register (MMR) Space
- DDR SDRAM memory space

The MMR memory space is fixed at CC00 E000H to CC00 FFFFH. Software programs the DDR SDRAM memory space by providing a base address in SDBR, each of the two bank boundaries in SBR0 and SBR1.

Note: The two memory regions must never be programmed to overlap for the IXP43X network processors.

11.2.5 DDR SDRAM Clocking

The MCU provides 6 clocks, that are three differential clocks pair, (DDR_CK[2:0]) and (DDR_CK_N[2:0]), to the DDR SDRAM memory subsystem at the selected DDR SDRAM command rate. The 72-bit 2-bank unbuffered *JEDEC Standard Double Data Rate (DDR) SDRAM Specification JESD79*, January 2004 requires 6 clocks to distribute the loading across eighteen x8 DDR SDRAM components.

Note: There are a maximum of 10 chips for the IXP43X network processors; two banks of x8 DRAM chips for a 40-bit bus.

11.2.6 Performance Monitoring

By setting up the IXP43X network processors system level PMU registers, the following parameters are monitored:

- Page hit and page miss counts for each of the 8 possible open pages of DDR SDRAM.
- AHB bus Read latency (from request to valid data back for each of the AHB bus ports)
- Core memory bus read latency (from Intel XScale processor to DDR request to valid data back for each of the eight outstanding read requests possible from the Intel XScale processor to memory).
- The PMU status comes into the PMU event counters in the form of three different buses. These are:
  - DDR_PMU_EVENT [4:0] where bit 4 = valid, bit 3 = hit/miss, bits 2:0 = page number
  - BIU_PMU_RDACTIVE[7:0] where each bit is the enable for an active read in progress
  - DDR_PMU_RDACTIVE[1:0] where bit 1 = South AHB read active and bit 0 = North AHB read active

11.3 Power Failure Mode

This mode is not supported by the IXP43X network processors as there is no support for self-refresh.

11.4 Interrupts/Error Conditions

The MCU has two conditions that require intervention from the Intel XScale processor. If a single-bit error is detected during a read cycle, the MCU can correct the data returned but software must fix the error in the memory array. If a multi-bit error is detected, the core decides how to handle the condition. For all ECC errors, the MCU records the requester of the transaction resulting in the error in ELOGx[23:16] and interrupts the core.

If the MCU detects an ECC error during a read or write cycle[1], MCISR[0] or MCISR[1] is set to 1. Whenever the MCU toggles one of the MCISR bits from 0 to 1, an interrupt is generated to the core.

Table 207 shows how the MCU responds to error conditions.

Table 207. MCU Error Response

Error Type	MCU Action[1]
Single-Bit during a read or write	Fix Error (if ECC error correction enabled in the ECC Control Register - ECCR)
Multi-bit during a read	Interrupt the Intel XScale® Processor through the Interrupt Controller or Terminate the core transaction, notify the BIU of multi-bit error
Multi-bit during a write	New ECC is generated with bad data and written to DDR SDRAM array. Data location is no longer valid.

1. The ECC Enable bit in the ECCR must be set in order for these actions to occur.

Note: If ECC reporting is enabled with ECCR[1] or ECCR[0] and an ECC error occurs, MCISR[1] or MCISR[0] is set and ELOGx/ECARx logs the error in addition to the actions in Table 207.

11.4.1 Single-Bit Error Detection

When enabled, the MCU interrupts the core when the ECC logic detects a single-bit error by setting the appropriate bit in the MCISR register. The Intel XScale processor knows the interrupt was caused by a single-bit error by polling the ELOG0 or ELOG1 register. The DDR SDRAM Control Block ensures that correct data is returned but the interrupt handler is responsible for scrubbing the error in the array (refer to Section 11.2.3.4, “Scrubbing”).

An example flow for a single-bit error with error detection and reporting enabled is:

- A single-bit ECC error is detected on the data bus by the MCU.
- The MCU fixes the error prior to returning the data.
- The MCU clears ELOG0[8] indicating a single-bit error.
- The MCU records the requester of the transaction that resulted in an error in ELOG0[23:16]
- The MCU loads ELOG0[7:0] with the syndrome that indicated the error.
- The MCU loads ECAR0[31:2] with address where the error occurred.
- Since the Intel XScale processor must scrub the error in the array, the MCU sets MCISR[0] to 1 (assuming it is not already set).
- Setting any bit in the MCISR causes an interrupt to the Intel XScale processor.

1. Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See “ECC Generation” on page 539 for details.

- Software polls the interrupt status register. Bit 0 set to 1 indicates that the first error has occurred.
- Software polls ELOG0 and ECAR0 and scrubs the error at the location specified by ECAR0.
- Software writes a 1 to MCISR[0] thereby clearing it.

If software does not perform error scrubbing, the probability of an unrecoverable multi-bit error increases for the memory location containing the single-bit error.

ECARx and ELOGx remain registered until software explicitly clears them.

If a second error occurs before software clears the first by resetting MCISR[0] or MCISR[1], the error is recorded in the remaining ELOGx/ECARx register. If none are available, the error is not logged but the MCU carries out the action described in Table 207.

11.4.2 Multi-Bit Error Detection

If a multi-bit error occurs during a read or write transaction and error reporting is enabled, the MCU sets MCISR[0] or MCISR[1] and that asserts an interrupt to the core. Upon receiving an interrupt, the core knows the interrupt was caused by a multi-bit error by polling the ELOGx registers.

When MCU detects a multi-bit error during a read cycle and ECC calculation is enabled in the ECCR, the MCU target aborts the transaction in the IBMTQ, indicating to the internal bus masters that an unrecoverable error has been detected. For core transactions issued by the CMTQ, when a multi-bit error is detected during a read cycle, the MCU signals a multi-bit error to the BIU. The MCU records the error type in ELOGx and the address in ECARx.

When MCU detects a multi-bit error during a write[1] cycle and error reporting is enabled in the ECCR, the MCU records the first multi-bit error by programming ELOGx and ECARx. The MCU generates new ECC with the data before sending it on DDR_DQ[31:0] so the contents of memory after the read-modify-write cycle is corrupted with correct ECC.

If a second error occurs before software clears the first by resetting MCISR[0] or MCISR[1], the error is recorded in the remaining ELOGx/ECARx register. If none are available, the error is not logged but the MCU carries out the action described in Table 207.

It is the interrupt handler's responsibility to decide how to handle this error condition and clear the MCISR.

11.5 Reset Conditions

Once RESET_N is deasserted after 200µs, software must issue the initialization sequence defined in Section 11.2.2.10, "DDR SDRAM Initialization". After initialization, the DDR SDRAM devices are ready to be written to or read from. Reads issued prior to a write to the same address results in an ECC error and are not recommended if ECC is enabled.

As RESET_N is asserted, the MCU initializes its MMR registers to the states defined in "Register Definitions" on page 552.

1. Any error condition during a write cycle actually occurs while performing the read portion of a read-modify-write on a partial write. See "ECC Generation" on page 539 for details.

11.6 Register Definitions

A series of configuration registers control the MCU. Software can determine the status of the MCU by reading the status registers. Table 208 lists all of the MCU registers that are detailed further in proceeding sections.

Note: When computing the register values based on JEDEC specifications, round your final answer up to the nearest integer value.Constant polling of MCU MMRs can result in inducing long latencies in peripheral unit DDR SDRAM transactions, and therefore may negatively impact performance. Polling of MCU MMRs should be avoided.

Table 208. Memory Controller Register Table

Address	Register Name	Description	Reset Value	Page with Details
CC00 E500H	DDR SDRAM Initialization Register - SDIR	DDR SDRAM Initialization Register	0x0000 000FH	553
CC00 E504H	DDR SDRAM Control Register 0 - SDCR0	DDR SDRAM Control Register 0	0x0001 0004H (DDRI) 0x0001 0000H (DDRII)	554
CC00 E508H	DDR SDRAM Control Register 1 - SDCR1	DDR SDRAM Control Register 1	0x0000 0000H	556
CC00 E50CH	SDRAM Base Register - SDBR	SDRAM Base Register	0x0000 0000H	558
CC00 E510H	SDRAM Boundary Register 0 - SBR0	SDRAM Boundary Register 0	0x0000 0000H	558
CC00 E514H	SDRAM Boundary Register - SBR1	SDRAM Boundary Register 1	0x0000 0000H	559
CC00 E51CH	ECC Control Register - ECCR	ECC Control Register	0x0000 0000H	560
CC00 E520H CC00 E524H	ECC Log Registers - ELOG0, ELOG1	ECC Log Registers	0x0000 0000H	561
CC00 E528H CC00 E52CH	ECC Address Registers - ECAR0, ECAR1	ECC Address Registers	0x0000 0000H	562
CC00 E530H	ECC Test Register - ECTST	ECC Test Register	0x0000 0000H	562
CC00 E534H	Memory Controller Interrupt Status Register - MCISR	Memory Controller Interrupt Status Register	0x0000 0000H	563
CC00 E53CH	MCU Port Transaction Count Register - MPTCR	MCU Port Transaction Count Register	0x0000 0011H	564
CC00 E548H	Refresh Frequency Register - RFR	Refresh Frequency Register	0x0000 0000H	564
0 — CC00 E550H 1 — CC00 E554H 2 — CC00 E558H 3 — CC00 E55CH 4 — CC00 E560H 5 — CC00 E564H 6 — CC00 E568H 7 — CC00 E56CH	SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7	SDRAM Page Registers	0x0000 0000H	565
CC00 F550H	Receive Enable Delay Register - RCVDLY	Receive Enable Delay Register	0x0000 0007H	566
CC00 F574H	DDR Drive Strength Control Register - LEGOVERIDE	DDR Drive Strength Control Register	0x0000 0009H	566

11.6.1 DDR SDRAM Initialization Register SDIR

The DDR SDRAM Initialization Register (SDIR) is responsible for programming the operation of the DDR SDRAM device state machines. The SDIR provides a method for software to execute the DDR SDRAM initialization sequence (see "DDR SDRAM Initialization" on page 524).

Register Name:	DDR SDRAM Initialization Register - SDIR		
Hex Offset Address:	CC00 E500H	Reset Hex Value:	0x0000 000FH
Register Description:	DDR SDRAM Initialization Register		

Access: See below.

31																								4	3			0
(Reserved)																												

Register		DDR SDRAM Initialization Register - SDIR		
Bits	**Name**	**Description**	**Default**	**Access**
31:04	(Reserved)		0	RO
03:00		**Special DDR SDRAM Command**: These bits are used for DDR SDRAM initialization. See "DDR SDRAM Initialization" on page 524 for details. When not in the initialization sequence, these bits must be set to $11xx_2$. For details on the exact DDR SDRAM commands, refer to Table 204, "DDR SDRAM Commands" on page 523. • 0000_2 - **Mode-Register-Set Command** where DLL is not Reset and CAS# Latency as specified in SDCR0 • 0001_2 - **Mode-Register-Set Command** where DLL is Reset and CAS# Latency and Burst Length of four • 0010_2 - **Precharge-All Command**: The MCU issues one **precharge-all** command to the DDR SDRAM devices. • 0011_2 - **NOP Command**: The MCU issues one **NOP** command to the DDR SDRAM devices. • 0100_2 - Extended **Mode-Register1 Set (EMRS) Command** where DLL is enabled on the DDR SDRAM devices. • 0101_2 - Extended **Mode-Register1 Set (EMRS) Command** where DLL is disabled on the DDR SDRAM devices. • 0110_2 - **Auto-Refresh Command**: The MCU issues one refresh sequence (that is, precharge-all followed by **auto-refresh** command) to the DDR SDRAM devices. • 0111_2 - Extended **Mode-Register2 Set (EMRS2) Command** • 1000_2 - Extended **Mode-Register3 Set (EMRS3) Command** • Others - **Normal DDR SDRAM Operation** ***Note:*** When value is set to 0111_2 or 1000_2 in SDIR[3:0], it will issue EMRS2 or EMRS3 command respectively to the DDR2 memory. However, there won't be any value being programmed into the EMRS2 and ERMS3 register. The command will be sent with value 0 for all the A0 to A15 bits.	1111_2	RW

11.6.2 DDR SDRAM Control Register 0 SDCR0

The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDR-I/II SDRAM state machines. The SDCR0 specifies the DIMM type, data bus width, and some SDRAM timing parameters required by the DDR-I/II SDRAM state machine as defined in Section 11.2.2.10, "DDR SDRAM Initialization" and Section 11.2.2.11, "DDR SDRAM Mode Programming". The remaining SDRAM timing parameters required by the DDR-I/II SDRAM state machine are set in SDCR1.

Note: Each parameter field contains either an IXP43X network processors-required value limit or an example value derived from a typical device data sheet.

Register Name:	DDR SDRAM Control Register 0 - SDCR0		
Hex Offset Address:	CC00 E504H	Reset Hex Value:	0x0001 0004H (DDRI) 0x0001 0000H (DDRII)
Register Description:	DDR SDRAM Control Register 0		
Access: See below.			

31			28	27	26		24	23	22		20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RAS				(Rsvd)	RP			(Rsvd)	RCD			(Rsvd)		EDP		(Rsvd)		WDL		(Rsvd)		CAS		(Rsvd)		ODT			DDR		(Rsvd)

† The reset value of this register is dependent on the **DDR_MODE** boot strap input to SDCR0. Bit [2] is '1' for DDRI-266 MHz and '0' for DDRII-400 MHz. The default **DDR_MODE** input is '1'.

Register	DDR SDRAM Control Register 0 - SDCR0 (Sheet 1 of 2)			
Bits	**Name**	**Description**	**Default**	**Access**
31:28	RAS	**RAS**: Active to Precharge duration in MCLK periods. This value is computed by using the t_{RAS} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 3. RAS = t_{RAS} - 1** NOTE: t_{RAS} must be converted to mclks for this equation. **For example**, a device specification yields that t_{RAS} is 40ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{RAS} = (40ns / 7.5ns) = 5.33, round up to 6 mclks. Thus, RAS = 6-1 = 5, so a value of 0101_2 should be programmed.	0H	RW
27	(Reserved)		0_2	RO
26:24	RP	**RP**: Precharge Command Period in MCLK periods. This value is computed by using the t_{RP} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 4. RP = t_{RP} - 1** NOTE: t_{RP} must be converted to mclks for this equation. **For example**, a device specification yields that t_{RP} is 20ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{RP} = (20ns / 7.5ns) = 2.67, round up to 3 mclks. Thus, RP = 3-1 = 2, so a value of 010_2 should be programmed.	000_2	RW
23	(Reserved)		0_2	RO
22:20	RCD	**RCD**: Active to Read or Write Delay in MCLK periods. This value is computed by using the t_{RCD} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 5. RCD = t_{RCD} - 1** NOTE: t_{RCD} must be converted to mclks for this equation. **For example**, a device specification yields that t_{RCD} is 20ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{RCD} = (20ns / 7.5ns) = 2.67, round up to 3 mclks. Thus, RCD = 3-1 = 2, so a value of 010_2 should be programmed.	000_2	RW
19:18	(Reserved)		00_2	RO
17:16	EDP	**EDP**: External Data Path Latency in MCLK periods. **A value of 10_2 should be programmed.**	01_2	RW
15:14	(Reserved)		00_2	RO

Register		DDR SDRAM Control Register 0 - SDCR0 (Sheet 2 of 2)		
Bits	**Name**	**Description**	**Default**	**Access**
13:12	WDL	**WDL**: Write Data Latency in MCLK periods used by the memory controller state machine using the following equation: **Equation 6. $t_{WDL} = t_{CAS} - 2$** NOTE: t_{CAS} must be converted to mclks for this equation. **For example**, a device specification yields that t_{CAS} is 3 mclks. Thus, WDL = 3 - 2 = 1, so a value of 01_2 should be programmed. • 00 = 0 MCLK periods (tCAS = 2 and 2.5) • 01 = 1 MCLK period (tCAS = 3) • 10 = 2 MCLK period (tCAS = 4) • 11 = RESERVED	00_2	RW
11:10	(Reserved)		00_2	RO
09:08	CAS	**CAS**: Indicates the CAS Latency of the memory device (provided in the JEDEC/Vendor specification). • 00 = 2 MCLK periods (DDR-I Type only) • 01 = 2.5 MCLK periods (DDR-I Type only) • 10 = 3 MCLK periods (DDR-II Type only) • 11 = 4 MCLK periods (DDR-II Type only)	00_2	RW
07:06	(Reserved)		00_2	RO
05:04	ODT	**ODT Termination Value**: Determines the termination value of the On Die Termination for both Banks (controlled by ODT[1:0]). Applies to DDR-II SDRAM memory type only. • 00 = Disabled • 01 = 75 ohm (memory); **150 ohm (on-chip)** • 10 = 150 ohm (memory); **150 ohm (on-chip)** • 11 = RESERVED (memory); **75 ohm (on-chip)** ***Note:*** It is possible to have a combination of memory and on-chip ODTs, which are not listed above. For example, 75 ohm memory and 75 ohm on-chip ODTs are allowed. The above combination can be achieved by first completing SDCR0[5:4] register write for the desired memory-side ODT -> EMRS -> followed by a rewrite of SDCR0[5:4] to the desired on-chip ODT. For example, if you want to set 75 ohm (memory) and 75 ohm (on-chip), they can follow the sequence of SDCR0[5:4] = 01 -> EMRS -> SDCR0[5:4] = 11.	00_2	RW
03	Data Bus Width	**Data Bus Width:** Indicates the width of the data bus. See Section 11.2.2.6, "16-bit Data Bus Width" on page 518. 0 = For enabling 32-bit selection 1 = 16 bits (Higher priority compared to 32-bit)	0_2	RW
02	DDR	**DDR:** Identifies the generation of DDR SDRAM selected by the **DDR_MODE** reset strap. 0 = DDRII-400MHz 1 = DDRI-266MHz	Varies with external state of DDR_MODE at reset Should default to 1	RO
01	Data Bus Width	**Data Bus Width:** Indicates the width of the data bus. See Section 11.2.2.5, "32-bit Data Bus Width" on page 517. 0 = RESERVED for 64 bits 1 = 32 bits	0_2	RW
00	(Reserved)	**DRAM Type:** Selects unbuffered or registered DRAM operating modes. 0 = Unbuffered 1 = RESERVED for registered. A value of 0_2 **must** be used for the IXP43X network processors.	0_2	RW

11.6.3 DDR SDRAM Control Register 1 SDCR1

The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDR SDRAM state machines as defined in Section 11.2.2.10, "DDR SDRAM Initialization" on page 524 and Section 11.2.2.11, "DDR SDRAM Mode Programming" on page 528. The SDCR1 specifies the remaining SDRAM timing parameters required by the DDR SDRAM state machine not specified in SDCR0.

Note: Each parameter field contains either an IXP43X network processors-required value limit or an example value derived from a typical device data sheet

Register Name:	DDR SDRAM Control Register 1 - SDCR1		
Hex Offset Address:	CC00 E508H	Reset Hex Value:	0x0000 0000H
Register Description:	DDR SDRAM Control Register 1		

Access: See below.

31	30		28	27			24	23	22		20	19		17	16				12	11		09	08				04	03			00
DQS	RTCMD			WTCMD				(Rsvd)	RTW			(Rsvd)			RFC					WR			RC					WTRD			

Register	DDR SDRAM Control Register 1 - SDCR1 (Sheet 1 of 2)			
Bits	**Name**	**Description**	**Default**	**Access**
31	DQS	**DQS# Disable**: Controls the behavior of the strobes and the configuration of the DIMM. 0 = DQS# Enabled for differential operation, EMRS bit 10 is programmed as zero. (DDR-II Type only) 1 = DQS# Disabled for single-ended operation, EMRS bit 10 is programmed as one. ***Note:*** Differential mode operation is only supported in DDRII SDRAM.	0_2	RW
30:28	RTCMD	**RTCMD**: Read to non-Write command turnaround period in MCLK periods. **Equation 7. RTCMD = t_{CAS} + 2** NOTE: t_{CAS} must be converted to mclks for this equation. **For example**, a device specification yields that t_{CAS} is 2.5 mclks. Thus, RTCMD = 2.5 + 2 = 4.5, round up to 5 mclks, so a value of 101_2 should be programmed.	000_2	RW
27:24	WTCMD	**WTCMD**: Write to command (non-Read) turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS})and Write Recovery (t_{WR}) measured for the device (provided in the JEDEC/ Vendor specification) using the following equation: **Equation 8. WTCMD = t_{CAS} + t_{WR} + 1** NOTE: t_{CAS} and t_{WR} must be converted to mclks for this equation. **For example**, a device specification yields that t_{CAS} is 2.5 mclks and t_{WR} is 15ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{WR} = (15ns / 7.5ns) = 2 mclks. Thus, WTCMD = 2.5 + 2 + 1 = 5.5, round up to 6 mclks, so a value of 0110_2 should be programmed.	000_2	RW
23	(Reserved)		0_2	RO

Register		DDR SDRAM Control Register 1 - SDCR1 (Sheet 2 of 2)		
Bits	**Name**	**Description**	**Default**	**Access**
22:20	RTW	**RTW**: Read to Write turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS}) for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 9. RTW = t_{CAS} + EDP + 1** NOTE: t_{CAS} must be converted to mclks for this equation. A programmed value of 000_2 represents a value of 8_{10}. **For example**, a device specification yields that t_{CAS} is 2.5 mclks and the suggested EDP value of 1 mclks is used. Thus, RTW = 2.5 + 1 + 1 = 4.5, round up to 5 mclks, so a value of 0101_2 should be programmed. NOTE: The MCU allows for back-to-back reads, so long as they are to open pages.	000_2	RW
19	(Reserved)	RESERVED. A value of 0_2 **must** be used for the IXP43X network processors.	0_2	RO
18:17	(Reserved)		00_2	RO
16:12	RFC	**RFC**: Auto Refresh Command Period in MCLK periods. This value is computed by using the t_{RFC} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 10. RFC = t_{RFC} - 1** NOTE: t_{RFC} must be converted to mclks for this equation. **For example**, a device specification yields that t_{RFC} is 120ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{RFC} = (120ns / 7.5ns) = 16 mclks. Thus, RFC = 16-1 = 15, so a value of 01111_2 should be programmed.	$0\ 0000_2$	RW
11:09	WR	**WR**: Write Recovery time in MCLK periods. • 000 = 0 MCLK period (DDRI-266MHz) • 010 = 3 MCLK period (DDRII-400MHz) all other values reserved	000_2	RW
08:04	RC	**RC**: Active to Active and Active to Auto Refresh command period in MCLK periods. This value is computed by using the t_{RC} measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 11. RC = t_{RC} - 1** NOTE: t_{RC} must be converted to mclks for this equation. **For example**, a device specification yields that t_{RC} is 65ns (and the device speed is 133MHz, so the mclk period is 7.5ns). t_{RC} = (65ns / 7.5ns) = 8.67, round up to 9 mclks. Thus, RC = 9-1 = 8, so a value of 01000_2 should be programmed.	$0\ 0000_2$	RW
03:00	WTRD	**WTRD**: Write-to-Read turnaround period in MCLK periods. This value is computed by using the CAS Latency (t_{CAS})and internal Write to Read command delay (t_{WTR}) measured for the device (provided in the JEDEC/Vendor specification) using the following equation: **Equation 12. WTRD = t_{CAS} + t_{WTR} + 1** NOTE: t_{CAS} and t_{WTR} must be converted to mclks for this equation. **For example**, a device specification yields that t_{CAS} is 2.5 mclks and t_{WTR} is 1 mclk. Thus, WTRD = 2.5 + 1 + 1 = 4.5, round up to 5 mclks, so a value of 101_2 should be programmed.	0000_2	RW

11.6.4 DDR SDRAM Base Register SDBR

This register indicates the beginning of SDRAM space. See Section 11.2.2.2, "DDR-I/II SDRAM Bank Sizes and Configurations" on page 511 for usage details. There are two contiguous physical banks defined by SBR0 and SBR1 in the DDR SDRAM subsystem starting at this address.

Note: DDR SDRAM memory space must **never** cross a 1 Gbyte boundary. This register should be read back after being written, before the Intel XScale processor performs transactions that address the DDR SDRAM.

Register Name:	SDRAM Base Register - SDBR		
Hex Offset Address:	CC00 E50CH	Reset Hex Value:	0x0000 0000H
Register Description:	SDRAM Base Register		

Access: See below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
(Reserved)																															

Register	SDRAM Base Register - SDBR			
Bits	Name	Description	Default	Access
31:25		**SDRAM Base Address:** These bits define the upper seven bits of the DDR SDRAM base address. These seven bits are compared with **ADDR[31:25]** to determine if the internal bus transaction hits SDRAM memory space. See Table 197.	00H	RW
24		**SDRAM Base Address for DDRI 16MB only:** This bit define the lowest bit of the DDR SDRAM base address. This bit is compared with **ADDR[24]** to determine if the internal bus transaction hits SDRAM memory space. See Table 197.	0_2	RW
23:00	(Reserved)		000000H	RO

11.6.5 DDR SDRAM Boundary Register 0 SBR0

This register indicates the upper boundary of SDRAM bank 0 and its memory technology. If bank 0 is unpopulated, SBR0[7:0] is programmed to the value in SDBR[31:24]. See "DDR-I/II SDRAM Bank Sizes and Configurations" on page 511 for more details and programming examples.

Note: DDR SDRAM memory space must **never** cross a 1-Gbyte boundary. This register should be read back after being written, before the Intel XScale processor performs transactions that address the DDR SDRAM.

Register Name:	SDRAM Boundary Register 0 - SBR0		
Hex Offset Address:	CC00 E510H	Reset Hex Value:	0x0000 0000H
Register Description:	SDRAM Boundary Register 0		

Access: See below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	(Reserved)																														

Register		SDRAM Boundary Register 0 - SBR0		
Bits	**Name**	**Description**	**Default**	**Access**
31:30		**SDRAM Address Translation: Based on** Table 200, " DDR SDRAM Address Decode Summary" on page 516. • 00 DDR SDRAM Address Translation #1 • 10 DDR SDRAM Address Translation #2 • 11 DDR SDRAM Address Translation #3 • 01 Reserved	00_2	RW
29:08	(Reserved)		0000000H	RO
7		**SDRAM Boundary for DDRI 16MB only:** Defines the upper limit of SDRAM bank 0. This value is compared with **ADDR[24]** to determine a bank hit or miss.	0_2	RW
06:00		**SDRAM Boundary:** Defines the upper limit of SDRAM bank 0. This value is compared with **ADDR[31:25]** to determine a bank hit or miss.	0000000_2	RW

11.6.6 DDR SDRAM Boundary Register 1 SBR1

This register indicates the upper boundary of SDRAM bank 1 and its memory technology. If bank 1 is unpopulated, SBR1[7:0] should be programmed to the value stored in SBR0[7:0]. If bank 1 is populated, SBR1[7:0] must be programmed greater than or equal to SBR0[7:0]. See "DDR-I/II SDRAM Bank Sizes and Configurations" on page 511 for more details and programming examples.

Note: DDR SDRAM memory space must never cross a 1-Gbyte boundary. This register should be read back after being written, before the Intel XScale processor performs transactions that address the DDR SDRAM.

Register Name:	SDRAM Boundary Register - SBR1		
Hex Offset Address:	CC00 E514H	Reset Hex Value:	0x0000 0000H
Register Description:	SDRAM Boundary Register 1		

Access: See below.

31	30	29																			08	07							00
		(Reserved)																											

Register		SDRAM Boundary Register - SBR1		
Bits	**Name**	**Description**	**Default**	**Access**
31:30		**SDRAM Address Translation: Based on** Table 200, " DDR SDRAM Address Decode Summary" on page 516. • 00 DDR SDRAM Address Translation #1 • 10 DDR SDRAM Address Translation #2 • 11 DDR SDRAM Address Translation #3 • 01 Reserved	00_2	RW
29:08	(Reserved)		0000000H	RO
7		**SDRAM Boundary for DDRI 16MB only:** Defines the upper limit of SDRAM bank 1. This value is compared with **ADDR[24]** to determine a bank hit or miss.	0_2	RW
06:00		**SDRAM Boundary:** Defines the upper limit of SDRAM bank 1. This value is compared with **ADDR[31:25]** to determine a bank hit or miss.	0000000_2	RW

11.6.7 ECC Control Register ECCR

This register programs the MCU error correction and detection capabilities. The configuration depends on the application's needs but a typical configuration is:

- ECC Mode Enabled
- Enable multi-bit error reporting
- Disable single-bit error reporting
- Enable single-bit error correcting

ECC must be enabled or disabled in the ECCR MMR before any transaction to DDR occurs, and it's state must NOT subsequently be changed. For more details, see "Error Correction and Detection" on page 538 and "Interrupts/Error Conditions" on page 550.

Register Name:	ECC Control Register - ECCR		
Hex Offset Address:	CC00 E51CH	Reset Hex Value:	0x0000 0000H
Register Description:	ECC Control Register		
Access: See below.			

31																									04	03	02	01	00
(Reserved)																													

Register		ECC Control Register - ECCR		
Bits	Name	Description	Default	Access
31:04	(Reserved)		000 0000H	RO
03		**ECC Enabled:** Enables ECC Read Modify Write sequence for ECC calculation and generation during sub 64-bit writes. See "ECC Disabled" on page 548. 0 = ECC Disabled (mode for Intel XScale® Processor ECC scrub) 1 = ECC Enabled (normal operation)	0_2	RW
02		**Single Bit Error Correction Enable:** Enables or disables the correction of a single bit error. 0 = Disable single bit error correction 1 = Enable single bit error correction	0_2	RW
01		**Multi-Bit Error Reporting Enable:** Enables or disables the reporting of a multi-bit error condition. 0 = Disable multi-bit error reporting 1 = Enable multi-bit error reporting	0_2	RW
00		**Single Bit Error Reporting Enable:** Enables or disables the reporting of a single bit error condition. 0 = Disable single bit error reporting 1 = Enable single bit error reporting	0_2	RW

Note: To ensure predictable ECC operation, Single Bit Error Correction Enable *must* be enabled whenever ECC is enabled. The reporting enables are configured as desired.

11.6.8 ECC Log Registers ELOG0, ELOG1

The ECC Log Registers are responsible for logging the error types detected on the local memory bus. Two errors are detected and logged. The error type is logged (single-bit or multi-bit) along with the syndrome that indicated the error. For a single-bit error,

software can read this syndrome and determine the bit that had the error to perform scrubbing. For a multi-bit error, the syndrome does not match an entry in the H-Matrix and thus, is uncorrectable. See Table 206, “Syndrome Decoding” on page 544.

The error recorded in ELOG0 corresponds to the address in ECAR0. ELOG1 corresponds to ECAR1.

The ELOGx registers comprise read-only bits and only have meaning if MCISR[0] or MCISR[1] is non-zero. For more details on error handling, see “Error Correction and Detection” on page 538.

Register Name:	ECC Log Registers - ELOG0, ELOG1		
Hex Offset Address:	CC00 E520H CC00 E524H	Reset Hex Value:	0x0000 0000H
Register Description:	ECC Log Registers		

Access: See below.

31							24	23							16	15			13	12	11			09	08	07							0
(Reserved)								(RO)								(Rsvd)				(RO)	(Rsvd)				(RO)	(RO)							

Register		ECC Log Registers - ELOG0, ELOG1		
Bits	Name	Description	Default	Access
31:24	(Reserved)		0	RO
23:16		**ECC Error Requester:** Indicates the requester of the logged error. 0000_0000 - Intel XScale® Processor BIU 1000_0000 - IB BUS (North or South AHB) All Other Codes Are Reserved Note: Intel XScale processor BIU is logged for core transactions directed to the MCU via the core MCU port only. Core transactions directed to the MCU via the IB port of the BIU is logged as the IB BUS.	00H	RO
15:13	(Reserved)		000_2	RO
12		**Read or Write**: Indicates if the error occurred during a read or write transaction. 0 = Read error 1 = Write Error	0_2	RO
11:09	(Reserved)		000_2	RO
08		**ECC Error Type:** Indicates the type of error that occurred at this address. 0 = Single Bit Error 1 = Multi-Bit Error	0_2	RO
07:00		**Syndrome:** Holds the syndrome value that indicated the error. Using the H-Matrix, this indicates the bit that caused the error if it was a single bit error.	00H	RO

11.6.9 ECC Address Registers ECAR0, ECAR1

These registers are responsible for logging the addresses where the errors were detected on the local memory bus. Two errors are detected and logged. The software knows the DDR SDRAM address that had the error by reading these registers and decoding the syndrome in the log registers. For error details, see “Error Correction and Detection” on page 538.

Register Name:	ECC Address Registers - ECAR0, ECAR1		
Hex Offset Address:	CC00 E528H CC00 E52CH	Reset Hex Value:	0x0000 0000H
Register Description:	ECC Address Registers		

Access: See below.

31	...	02	01	00
RO			(Rsvd)	

Register		ECC Address Registers - ECAR0, ECAR1		
Bits	Name	Description	Default	Access
31:02		Error Address: Stores the upper 30 bits of the address that resulted in a single bit or multi-bit error.	000 0000H	RO
01:00	(Reserved)		00_2	RO

11.6.10 ECC Test Register ECTST

This register allows testing between the ECC logic and the memory subsystem (Section 11.2.3.6, "ECC Testing"). To test error handling software, the programmer writes this register with a non-zero masking function. Any subsequent writes to memory stores a masked version of the computed ECC. Therefore, any subsequent reads to these locations result in an ECC error.

Register Name:	ECC Test Register - ECTST		
Hex Offset Address:	CC00 E530H	Reset Hex Value:	0x0000 0000H
Register Description:	ECC Test Register		

Access: See below.

31	...	08	07	...	0
(Reserved)					

Register		ECC Test Register - ECTST		
Bits	Name	Description	Default	Access
31:08	(Reserved)		00 0000H	RO
07:00		ECC Mask: 8-bit ECC mask. Each bit of the generated ECC is XORed with the appropriate bit in this mask field before the ECC is stored into memory. See "ECC Testing" on page 548.	00H	RW

11.6.11 Memory Controller Interrupt Status Register MCISR

Setting the MCISR asserts an interrupt to the core. Upon an interrupt, the Intel XScale processor polls the interrupt status register for each unit. The interrupt status register tells the core the reason for the interrupt. The MCU has three interrupt conditions: first ECC error (MCISR[0]), second ECC error (MCISR[1]), and more than two ECC errors (MCISR[2]).

If the MCU detects an ECC error and both MCISR[0] and MCISR[1] are cleared, the error is logged in ELOG0 and MCISR[0] is set to 1. If one of the MCISR bits are not clear and the MCU detects an error, the error is logged in the unused ELOGx register and the appropriate MCISR bit is set to 1. If both MCISR[0] and MCISR[1] are not clear, any additional ECC errors are not logged and MCISR[2] is set.

Bits 4:0 are read/clear bits that means, to clear them, software must write a one to these bits.

Register Name:	Memory Controller Interrupt Status Register - MCISR		
Hex Offset Address:	CC00 E534H	Reset Hex Value:	0x0000 0000H
Register Description:	Memory Controller Interrupt Status Register		

Access: See below.

31																							05	04	03	02	01	00
(Reserved)																												

Register		Memory Controller Interrupt Status Register - MCISR		
Bits	Name	Description	Default	Access
31:05	(Reserved)		0000 000H	RO
04		**IB Discard Timer Expired:** Indicates that the IB Port Delayed Read Completion Timeout has expired and a completion from the MCU has been discarded. 0 = No error detected 1 = Error detected	0_2	RW1C
03		**Address Region Error:** Indicates that a transaction to the invalid address region created with a 32-bit Region. (see Section 11.2.2.2, "DDR-I/II SDRAM Bank Sizes and Configurations" on page 511) 0 = No error detected 1 = Error detected	0_2	RW1C
02		**ECC Error N:** Indicates that the MCU detected an ECC error as MCISR[1] and MCISR[0] are both set. 0 = No error detected 1 = Error detected	0_2	RW1C
01		**ECC Error 1:** Indicates that the MCU detected an ECC error and recorded the error in ELOG1. 0 = No error detected 1 = Error detected and recorded in ELOG1	0_2	RW1C
00		**ECC Error 0:** Indicates that the MCU detected an ECC error and recorded the error in ELOG0. 0 = No error detected 1 = Error detected and recorded in ELOG0	0_2	RW1C

11.6.12 MCU Port Transaction Count Register MPTCR

Sets the number of transactions a given port can have processed during a single tenure. The 4-bit fields for each port allow up to 16 transactions to be processed by a port before the MCU arbiter selects another port for DDR SDRAM transactions. This register is used to optimize the memory controller operation.

Note: This value MUST remain programmed to 11H for the IXP43X network processors to prevent unfair arbitration and indeterminate results.

Register Name:	MCU Port Transaction Count Register - MPTCR		
Hex Offset Address:	CC00 E53CH	Reset Hex Value:	0x0000 0011H
Register Description:	MCU Port Transaction Count Register		

Access: See below.

31	...	08	07	...	04	03	...	00
(Reserved)								

Register	MCU Port Transaction Count Register - MPTCR			
Bits	Name	Description	Default	Access
31:08	(Reserved)		000000H	RO
07:04		North and South AHB Transaction Count: Number of transactions the IB MCU port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction 2H = 2 transaction 3H = 3 transaction ... FH = 15 transactions 0H = 16 transactions	0001_2	RW
03:00		Core Transaction Count: Number of transactions the Core Processor MCU port can have processed in a single tenure of the DDR SDRAM. 1H = 1 transaction ... FH = 15 transactions 0H = 16 transactions	0001_2	RW

11.6.13 Refresh Frequency Register RFR

The Refresh Frequency Register is programmed for refreshing the DDR SDRAM subsystem at the specified interval. Writing to the RFR programs the refresh counter with the Refresh Interval. Reading from the RFR results in the value currently within the refresh counter. Refer to Table 205, “Typical Refresh Frequency Register Values” on page 538 for recommended programmed values.

Register Name:	Refresh Frequency Register - RFR		
Hex Offset Address:	CC00 E548H	Reset Hex Value:	0x0000 0000H
Register Description:	Refresh Frequency Register		

Access: See below.

31	...	13	12	...	04	03	...	00
(Reserved)						(RO)		

Register		Refresh Frequency Register - RFR		
Bits	**Name**	**Description**	**Default**	**Access**
31:13	(Reserved)		00000H	RO
12:00		**Refresh Interval:** Programs the number of clocks that triggers a request for a refresh cycle on the DDR SDRAM interface. If all zeroes, refresh cycles are disabled. See "Refresh Counter" on page 508. ***Note:*** If the memory interface is busy when the refresh counter expires, it is possible for the MCU to generate more than one refresh cycle when the memory interface becomes available.	000H	12:04 RW 03:00 RO

11.6.14 SDRAM Page Registers SDPR0-7

These registers hold the addresses of the eight pages that may be open. The valid bit is included. For more details, see Section 11.2.2.7, "Page Hit/Miss Determination" on page 519. These read-only registers are provided for testability purposes. The page size is determined by the memory and data bus width as defined in Table 192.

Register Name:	SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7		
Hex Offset Addresses:	0 — CC00 E550H 1 — CC00 E554H 2 — CC00 E558H 3 — CC00 E55CH 4 — CC00 E560H 5 — CC00 E564H 6 — CC00 E568H 7 — CC00 E56CH	Reset Hex Value:	**0x0000 0000H**
Register Description:	SDRAM Page Registers		

Access: See below.

31						25	24											11	10								01	00
(Reserved)							(RO)												(Reserved)									(RO)

Register		SDRAM Page Registers - SDPR0, SDPR1, SDPR2, SDPR3, SDPR4, SDPR5, SDPR6, SDPR7		
Bits	**Name**	**Description**	**Default**	**Access**
31:25	(Reserved)		00H	RO
24:11		**Page Address:** Specifies the 14 bits row address of an open page.	0000H	RO
10:01	(Reserved)		000H	RO
00		**Page Valid:** Indicates if the page address in bits 31 to 12 is a valid page. 0 = Page Invalid (closed) 1 = Page Valid (open)	0_2	RO

11.6.15 Receive Enable Delay Register RCVDLY

This 32-bit register consists of a packed 3-bit field for DQS Receive Enable Calibration.

Register Name:	Receive Enable Delay Register - RCVDLY		
Hex Offset Address:	CC00 F550H	Reset Hex Value:	0x0000 0007H
Register Description:	Receive Enable Delay Register		

Access: See below.

31																										03	02		00
(Reserved)																													

Register	Receive Enable Delay Register - RCVDLY			
Bits	Name	Description	Default	Access
31:03	(Reserved)		0000000H	RO
2:0		Receive Enable Delay Value	111_2	R/W

11.6.16 DDR Drive Strength Control Register LEGOVERIDE

This 32-bit register consists of a 5-bit field to override the DDR pad automatic drive strength control mechanism.

Register Name:	DDR Drive Strength Control Register - LEGOVERIDE		
Hex Offset Address:	CC00 F574H	Reset Hex Value:	0x0000 0009H
Register Description:	DDR Drive Strength Control Register		

Access: See below.

31																									04				00
(Reserved)																													

Register	DDR Drive Strength Control Register - LEGOVERIDE			
Bits	Name	Description	Default	Access
31:5	(Reserved)		000000H	RO
Overdrive Enable = 0				
4		Overdrive Enable: Disabled, automatic pad control used for drive strength value Enabled, bits 3:0 used for pad control	0_2	R/W
3:0		Drive Strength Value Hint: Value used by DDR pad control drive strength	1001_2	R/W
Overdrive Enable = 1				

Register		DDR Drive Strength Control Register - LEGOVERIDE		
Bits	**Name**	**Description**	**Default**	**Access**
4		Overdrive Enable: Disabled, automatic pad control used for drive strength value Enabled, bits 3:0 used for pad control	0_2	R/W
3		**Pad Control Selection:** (Valid when Overdrive Enable is set) Pad control uses Drive Strength Overdrive Value (bits 2:0) Locks current drive strength value determined from automatic pad control (i.e. freeze automatic control). Value reflected in LEGREG.	n/a	R/W
2:0		Drive Strength Override Value: Value to use for DDR pad control drive strength in place of automatic determined value. Requires bit 4 to be set and bit 3 to be clear.	n/a	R/W

§ §

12.0 Expansion Bus Controller

12.1 Overview

The Expansion Bus Controller provides an interface from the internal AHB in the Intel® IXP43X Product Line of Network Processors to external Expansion target devices.

The Expansion Bus Controller includes a 24-bit address bus and a 16-bit wide data path. The Expansion Bus Controller maps transfers between the internal AHB and external devices. The expansion bus supports Intel multiplexed, Intel non-multiplexed, Intel StrataFlash®, Synchronous Intel StrataFlash® Memory, Motorola* multiplexed and Motorola* non-multiplexed target devices.

Applications having less than 16-bit external target devices may connect to an 8-bit interface.

12.2 Feature List

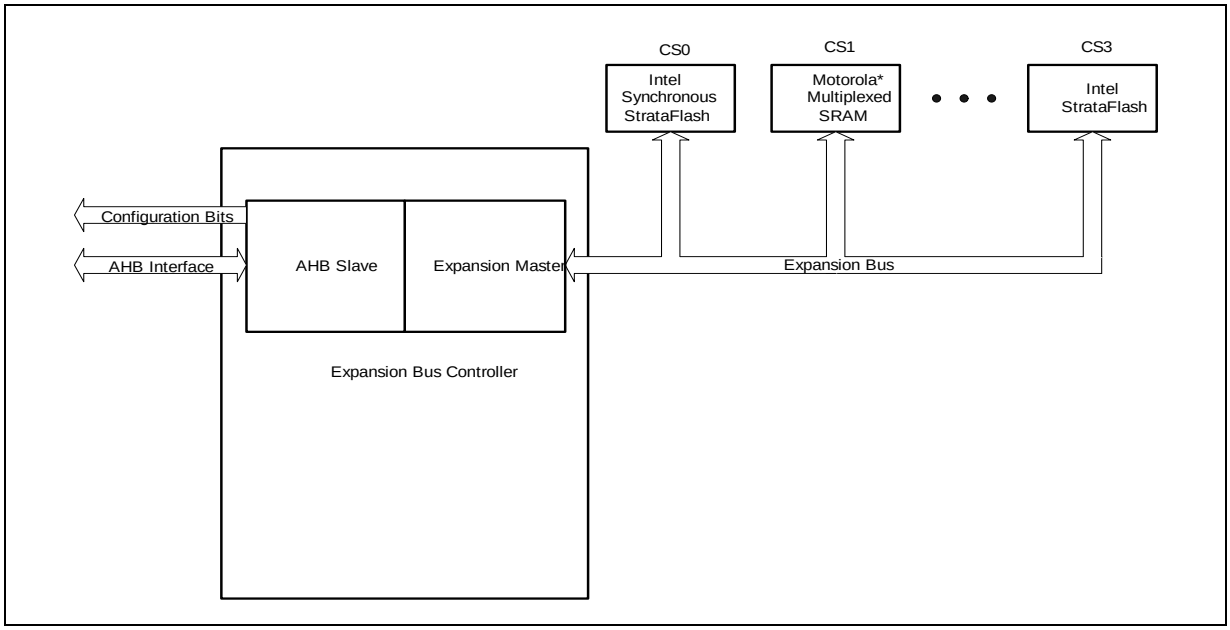
The features of Expansion Bus Controller are:

- Outbound transfers (IXP43X network processors are the master to an external target device)
- Four programmable target chip selects
- Twenty four bits of address; sixteen bits of data
- Supports Intel mode and Motorola mode bus cycles
- Supports Intel StrataFlash
- Supports 66-MHz Synchronous Intel StrataFlash Memory (16-bit only)
- Multiplexed or non-multiplexed address / data busses for Intel/Motorola bus cycles
- Maximum clock input frequency of 80 MHz

12.3 Block Diagram

Figure 138 explains the details of Expansion Bus Controller.

Figure 138. Expansion Bus Controller



12.4 Theory of Operation

The Expansion Bus Controller on the IXP43X network processors supports outbound transfers that are initiated by an AHB master that targets Expansion bus targets. The Expansion data bus is 16 bits wide and the address bus is 24 bits wide.

Since the Expansion Bus Controller has only 1 outbound transaction queue, outbound accesses all complete in order.

12.4.1 Outbound Transfers

For outbound data transfers, the Expansion Bus Controller occupies 256 Mbytes of address space in the memory map of IXP43X network processors and contains a 1-deep address queue, an 8-word write data FIFO, and an 8-word read data FIFO. Four chip selects are supported to allow up to four independent external devices to be connected. The address space for each chip select is up to 16 Mbytes.

A clock input, EX_CLK, is required to operate the Expansion interface. The maximum clock frequency supported by the Expansion Bus Controller is 80 MHz. The clock input is provided to allow various peripherals to be connected to the Expansion interface. The GPIO unit provides a clock output after reset and is used. Refer to the GPIO chapter for additional details. If a GPIO clock is used, it must be externally routed on the board to connect to EX_CLK. This implementation gives the designer the option to choose between a lower part count and the speed of the interface operations.

To provide a glue-less interface to a wide variety of devices, the Expansion Bus Controller supplies four chips selects to a 16-bit wide external bus, and is configured as Intel, Synchronous Intel or Motorola. The signaling characteristics and timing for each chip select is individually programmable. After chip reset, chip-select 0 defaults to conservative timing values for controlling an asynchronous flash device and the data width of the flash is determined by the value of Expansion Bus Address bits 0 during the reset sequence. The remaining chip selects are un-programmed. Refer to “Configuration Register 0” on page 592 for additional details.

For Synchronous Intel StrataFlash Memory, the Expansion Bus Controller only supports single word asynchronous page-mode read and synchronous burst-mode read (1-8 words). It does not support page mode read mode or single word latched asynchronous read mode. When configuring a Synchronous Intel StrataFlash Memory, wait polarity must be programmed to active low, data hold programmed to one clock, wait delay be deasserted with valid data and clock edge programmed to rising edge. For 16-bit Synchronous Intel devices, the burst length must be programmed to 16-word bursts. For 32-bit Synchronous Intel devices, the interconnect must be through the 16-bit interface and the burst length must be programmed to 8-word bursts. The latency count must be programmed to the appropriate code that is defined in the Synchronous Intel StrataFlash Memory specification.

The Expansion Bus interface signals must be connected based upon the device type (Intel, Synchronous Intel, or Motorola) and a sample mapping of the pins are shown in Table 209.

Table 209. Example Expansion Bus Pin Mappings to Target Devices

Pin	Intel® Embedded Flash[2] 28F128J3D	Intel StrataFlash® Embedded Memory 28F128P30	Synchronous Intel StrataFlash® 28F256K3	Motorola* MCM6946
EX_ALE	OPEN	OPEN	ADV#	OPEN
EX_ADDR[23:0]	A[23:0]	A[23:1]	A[23:1]	A[18:0]
EX_CS_N[3:0]	CE#	CE#	CE#	EN
EX_DATA[15:0]	D[15:0]	D[15:0]	D[15:0]	DQ[7:0]
EX_IOWAIT_N	OPEN	OPEN	OPEN	OPEN
EX_RD_N	OE#	OE#	OE#	G_N
EX_WR_N	WE#	WE#	WE#	W_N

Notes:
1. The # symbol or an _N suffix in a signal name indicates that the signal is asserted low.
2. The Intel® Embedded Flash Memory product family is offered in 32-Mb, 64-Mb and 128-Mb densities

The EX_IOWAIT_N signal is available to be shared by the devices attached to chip 0 through 3, when the chip selects are configured in Intel or Motorola mode of operation. The EX_IOWAIT_N signal allows an external device to hold off completion of the read or write phase of a transaction until the external device is ready to complete the transaction.

12.4.1.1 Expansion Bus Address Space

As seen in Table 210, on the AHB, the lowest 256 Mbytes of address space (0x00000000 to 0x0FFFFFFF) is overlapped with the DDRII/DDRI SDRAM address space (0x00000000 to 0x3FFFFFFF). The actual interface that is accessed when the overlapped region is addressed is configurable based on the value of a configuration register bit located in the Expansion Bus Controller.

Table 210. Trimmed Version of IXP43X network processors Memory Map (Sheet 1 of 2)

Start Address	End Address	Size	Use
0000_0000	0FFF_FFFF	256 Mbyte	Expansion bus (Boot up)
0000_0000	3FFF_FFFF	1 Gbyte	DDRII/DDRI SDRAM
4000_0000	47FF_FFFF	128 Mbyte	(Reserved)

Table 210. Trimmed Version of IXP43X network processors Memory Map (Sheet 2 of 2)

Start Address	End Address	Size	Use
4800_0000	4FFF_FFFF	128 Mbyte	PCI
5000_0000	5FFF_FFFF	256 Mbyte	Expansion bus
6000_0000	63FF_FFFF	64 Mbyte	Queue manager

When bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1, the Expansion Bus accesses occupy the lowest 256-Mbytes of address space. When bit 31 of the Configuration Register 0 (EXP_CNFG0) is cleared to logic 0, the DDRII/DDRI SDRAM occupies the lowest 256-Mbytes of address.

On reset, bit 31 of the Configuration Register 0 (EXP_CNFG0) is set to logic 1. This setting is required to allow the boot memory to be accessed and is located at hexadecimal address 0x00000000 in non-volatile storage on the Expansion Bus.

The first instruction execution of the Intel XScale® Processor is located at address 0x00000000. Once the boot sequence starts, the Intel XScale processor switches bit 31 of the Configuration Register 0 (EXP_CNFG0) from logic 1 to logic 0 at an appropriate time.

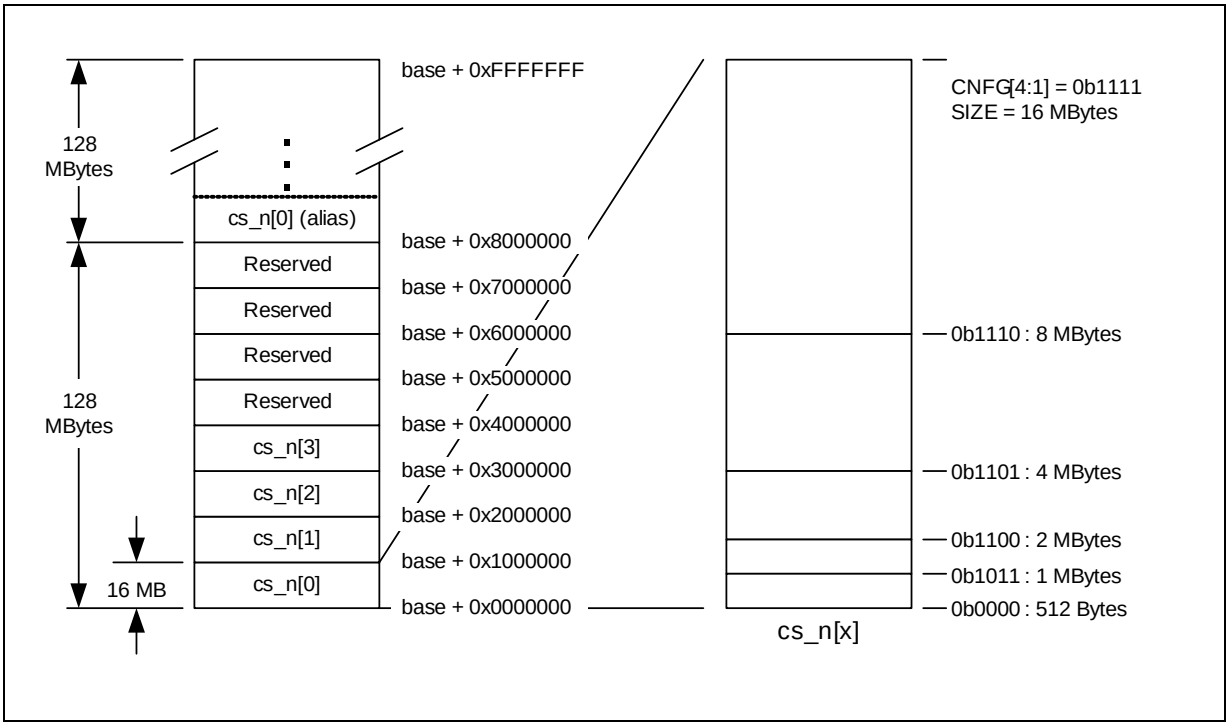
The information transfer from the flash to the DDRII/DDRI SDRAM is completed this way:

The configuration bit is swapped to allow the DDRII/DDRI SDRAM to have access at address 0x00000000 and the remainder of the flash information is retrieved from the expansion bus address location 0x50000000 to 0x5FFFFFFF

12.4.1.2 Chip Select Address Allocation

The Expansion Bus Controller occupies 256 Mbytes of address space in the memory map of the IXP43X network processors. The Expansion Bus Controller uses bits 27:0, from the AHB, to determine how to translate the AHB address to the Expansion Bus Address. The lower 24 bits of the AHB address are translated to the lower 24 bits of the Expansion Bus address, EX_ADDR [23:0]. EX_ADDR[24] is always zero. Bits 26:24 of the AHB are used to decode one of four chip-select regions implemented by the Expansion bus, each region being 16 Mbytes. Access to the Reserved chip select address results in unpredictable behavior of the Expansion Bus Controller. Address bit 27 is not used and currently aliases each chip select region as shown on the left side of Figure 139.

Figure 139. Chip Select Address Allocation



The right side of Figure 139 shows the implementation of bit 13:10 of the each Timing and Control (EXP_TIMING_CS) Register. A Timing and Control (EXP_TIMING_CS) Register is implemented for each of the four chip selects. Each chip select defines a base region size of 512 bytes with the actual size of the region given by the formula shown in Figure 140. If the AHB address is outside of the programmed region, the Expansion Bus Controller responds with an error response.

Figure 140. Expansion Bus Memory Sizing

Region Size = $2^{(9+CNFG[4:1] + 16*CNFG[0])}$

For Examples of how to use this feature:

If bits 13:9 of Timing and Control (EXP_TIMING_CS0) Register 0 = “00000” an address space of 2^9 = 512 Bytes is defined for chip select 0 (EX_CS0_N).

If bits 13:9 of Timing and Control (EXP_TIMING_CS1) Register 1 = “10000” an address space of 2^{17} = 128KBytes is defined for chip select 1 (EX_CS1_N).

If bits 13:9 of Timing and Control (EXP_TIMING_CS2) Register 2 = “11110” an address space of 2^{24} = 16Mbytes is defined for chip select 2 (EX_CS2_N).

12.4.1.3 Address and Data Byte Steering

Table 211 shows the address and data mapping from the AHB to the Expansion Bus. This table applies to Intel, Synchronous Intel and Motorola defined cycles only. For 32-bit read operations to a byte/halfword wide interface, multiple bytes are collected and then transferred as a complete 32-bit word. This pattern occurs as shown below for any allowable sub-length read access. Four- and eight-word reads are also supported and generate multiple accesses to the target device. Four- and eight-word reads to

Synchronous Intel generate one burst access to the device. Byte enables are not generated for both reads and writes. Byte write devices (devices that need byte enabled, EX_BE_N asserted in the same exact cycles that EX_WR_N is asserted) are not supported. The AHB bus is always big-endian format and the Expansion bus is little-endian. The conversions for each cycle are shown in Table 211.

For 8-bit devices, EX_DATA[15:8] must not toggle to conserve power (it should be terminated when unutilized).

Table 211. Expansion Bus Address and Data Byte Steering (Sheet 1 of 2)

AHB Bus Cycle	Device Width Connected to Expansion Bus (8-bit or 16-bit)	AHB Address Value (AHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and AHB Data Bus
Word write	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0]
			01	AHB data bus [23:16] = Expansion data bus [7:0]
			10	AHB data bus [15:8] = Expansion data bus [7:0]
			11	AHB data bus [7:0] = Expansion data bus [7:0]
Word write	16-bit	00	00	AHB data bus [31:16] = Expansion data bus [15:0]
			10	AHB data bus [15:0] = Expansion data bus [15:0]
Word read	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0]
			01	AHB data bus [23:16] = Expansion data bus [7:0]
			10	AHB data bus [15:8] = Expansion data bus [7:0]
			11	AHB data bus [7:0] = Expansion data bus [7:0]
Word read	16-bit	00	00	AHB data bus [31:16] = Expansion data bus [15:0]
			10	AHB data bus [15:0] = Expansion data bus [15:0]
Halfword read	8-bit	0x	00	AHB data bus [31:24] = Expansion data bus [7:0]
			01	AHB data bus [23:16] = Expansion data bus [7:0]
Halfword read	8-bit	1x	10	AHB data bus [15:8] = Expansion data bus [7:0]
			11	AHB data bus [7:0] = Expansion data bus [7:0]
Halfword read	16-bit	0x	00	AHB data bus [31:16] = Expansion data bus [15:0]
Halfword read	16-bit	1x	10	AHB data bus [15:0] = Expansion data bus [15:0]
Halfword write	8-bit	0x	00	AHB data bus [31:24] = Expansion data bus [7:0]
			01	AHB data bus [23:16] = Expansion data bus [7:0]
Halfword write	8-bit	1x	10	AHB data bus [15:8] = Expansion data bus [7:0]
			11	AHB data bus [7:0] = Expansion data bus [7:0]
Halfword write	16-bit	0x	00	AHB data bus [31:16] = Expansion data bus [15:0]
Halfword write	16-bit	1x	10	AHB data bus [15:0] = Expansion data bus [15:0]
Byte read	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0]
Byte read	8-bit	01	01	AHB data bus [23:16] = Expansion data bus [7:0]
Byte read	8-bit	10	10	AHB data bus [15:8] = Expansion data bus [7:0]

Table 211. Expansion Bus Address and Data Byte Steering (Sheet 2 of 2)

AHB Bus Cycle	Device Width Connected to Expansion Bus (8-bit or 16-bit)	AHB Address Value (AHB_ADDR[1:0])	Expansion Bus Address Value (EX_ADDR[1:0])	Data Location Translation Between Expansion Data Bus and AHB Data Bus
Byte read	8-bit	11	11	AHB data bus [7:0] = Expansion data bus [7:0]
Byte write	8-bit	00	00	AHB data bus [31:24] = Expansion data bus [7:0]
Byte write	8-bit	01	01	AHB data bus [23:16] = Expansion data bus [7:0]
Byte write	8-bit	10	10	AHB data bus [15:8] = Expansion data bus [7:0]
Byte write	8-bit	11	11	AHB data bus [7:0] = Expansion data bus [7:0]

12.4.1.4 Expansion Bus Interface Configuration

There are four registers called the Timing and Control (EXP_TIMING_CS) Registers that define the operating mode for each chip select. When designing with the Expansion Bus Interface, placing the devices on the correct chip selects is required.

Chip Select 0 through 3 is configured to operate with devices that require an Intel, Synchronous Intel, or Motorola Micro-Processor style bus accesses. These chip selects is configured to operate in a multiplexed or a simplex mode of operation for either Intel- or Motorola-style bus accesses. The mode of operation (Intel, Motorola, or Synchronous Intel) is set by bits 15,14, and 8 of each Timing and Control (EXP_TIMING_CS) Register. Table 214, “Bit Level Definition for each of the Timing and Control Registers” on page 591 shows the possible settings for the Cycle Type selection using bits 15, 14, and 8 of the Timing and Control (EXP_TIMING_CS) Register.

Once the cycle type has been determined, the mode of operation must be set. There are two configurable modes of operation for each chip select, multiplexed and non-multiplexed. Bit 4 of the Timing and Control (EXP_TIMING_CS) Registers is used to select this mode. If bit 4 of the Timing and Control (EXP_TIMING_CS) Register is set to logic 1, the access mode for that Chip Select is multiplexed. Likewise, if bit 4 of the Timing and Control (EXP_TIMING_CS) Register is cleared to logic 0, the access mode for that Chip Select is non-multiplexed. For Synchronous Intel, memories bit 4 must be programmed to logic 0. Multiplexed and non-multiplexed can imply various operations depending upon the Cycle Type that is selected. For more information refer to section “Expansion Bus Outbound Timing Diagrams” on page 579.

The size of the data bus for each device connected to the Expansion bus must be configured. The data bus size is selected on a per-chip-select basis, allowing the most flexibility when connecting devices to the Expansion bus. There are two valid selections that is configured for each data bus size, 8-bit or 16-bit. Bit 0 and bit 2 of each Timing and Control (EXP_TIMING_CS) Register is used to select the data bus size on a per-chip-select basis. For more information refer to Table 214, “Bit Level Definition for each of the Timing and Control Registers” on page 591. One special case for the data bus width selection is for chip select 0. Chip select 0 (data bus width) is selected by the value contained on Expansion Bus Address bit 0 at the assertion of PLL_LOCK. When PLL_LOCK is deasserted, Expansion Bus Address bit 0 is captured into Timing and Control (EXP_TIMING_CS) Register bit 0 for Chip Select 0. This feature allows either an 8-bit or 16-bit flash device to be connected to the Expansion Bus Interface for a boot device.

Each chip select is independently enabled or disabled by setting a value in bit 31 of each Timing and Control (EXP_TIMING_CS) Register. Clearing bit 31 of the Timing and Control (EXP_TIMING_CS) Register to logic 0 disables the corresponding chip select. Setting bit 31 of the Timing and Control (EXP_TIMING_CS) Register to logic 1 enables the corresponding chip select. Accesses to chip selects that are disabled result in an AHB error response.

Split transfers are supported for all read transfer types and controlled by setting bit 3 (SPLT_EN) of the Timing and Control (EXP_TIMING_CS) Register. Setting bit 3 of each Timing and Control (EXP_TIMING_CS) Register to logic 1 enables split transfers for accesses to the corresponding chip select. Clearing bit 3 of each Timing and Control (EXP_TIMING_CS) Register to logic 0 disables split transfers for accesses to the corresponding chip select. Multi-word read transfers requested by the AHB might be split. Only one access at a time is split.

Split transfers require that the read data from the Expansion bus be stored in an eight-word FIFO until all expansion bus transfers are complete before that data is forwarded on the AHB. When split transfers are initiated, the Expansion Bus Controller acknowledges the read request. The AHB is relinquished until all the data is acquired from the Expansion bus and stored in the eight-word FIFO contained in the Expansion Bus Controller. After all of the data has been acquired by the Expansion Bus Controller, the requesting master on the AHB is signaled that the read data is in the FIFO and the read transfer completes uninterrupted in its normal rotation in the arbitration scheme. This feature allows for slow devices connected to the Expansion bus not to impede the performance of data flow from high-speed peripherals (like PCI) on the AHB.

Retries also are supported and used predominately when expansion bus requests are issued as a split transfer is in progress. Retries may also occur when a write occurs when a previous read or write is in progress. A split response never occurs during a write transfer. The Expansion Bus Controller never deasserts AHB HREADY for writes; the writes is posted or retried. The Expansion Bus Controller also retries reads until all the words of the Expansion bus data transfer are in the data FIFO if SPLT_EN is clear. The Expansion Bus Controller does not deassert AHB HREADY for reads.

Each chip select region has the ability to be write-protected by setting bit 1 of each Timing and Control (EXP_TIMING_CS) Register. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is cleared to logic 0, writes to a specified chip select region results in an error response. When bit 1 of Timing and Control (EXP_TIMING_CS) Register is set to logic 1, writes are allowed to a specified chip select region. Chip select 0 is write-protected after reset.

One final set of parameters that is set prior to using Expansion Bus Interface Chip Select 1 through Chip Select 3. After boot up, these parameters are adjusted for Chip Select 0 as well. These five parameters are the timing extension parameters for each phase of an Expansion Bus access.

There are five phases to every Expansion Bus access:

- T1 – Address Timing
- T2 – Setup/Chip Select Timing
- T3 – Strobe Timing
- T4 – Hold Timing
- T5 – Recovery Phase

For Synchronous Intel mode, the T1,T2, T3, T4, T5 timing parameters are only used for writes. For Synchronous Intel reads, the Expansion Bus Controller uses the Count value programmed in the EXP_SYNCINTEL_COUNT register to determine how many cycles before data is valid.

The expansion bus address is used to present the 24 bits of the address [23:0] used for the Expansion bus access accompanied by an address latch enable output signal, EX_ALE for multiplexed devices. The address phase normally lasts two clock cycles in multiplexed mode. The address phase is extended by one to three clock cycles using the T1 - Address Timing parameter, bits 29:28 in the Timing and Control (EXP_TIMING_CS) Register for the particular Chip Select.

When the address phase T1 is extended, the ALE pulse is extended and always deasserts one cycle prior to the end of the T1 phase. The lower address bits are placed onto the data bus (that is, for a 16 bit data bus, EX_DATA contains EX_ADDR[15:0]) along with EX_ADDR[23:0] signals during the first cycle of the address phase. During the second cycle of the address phase, the data bus now outputs data when attempting to complete a write or tri-state when attempting to complete a read. The address signals retains their state.

For Synchronous Intel devices, EX_ALE acts as the address valid signal (ADV#) and is logic 0 during the address phase and logic 1 during the continuation of a burst or IDLE cycle.

The chip-select signal is presented for one Expansion bus phase before the Strobe Phase. The chip select is presented for the remainder of the Expansion bus cycles (setup, strobe, and hold phases).

The Setup/Chip Select Timing phase may also be extended by one to three clock cycles, using bits 27:26 of the Timing and Control (EXP_TIMING_CS) Register, T2 – Setup/Chip Select Timing parameter.

The Strobe Phase of an expansion bus access is when the read or write strobe is applied. The 24 Expansion Bus Interface Address bits are maintained in non-multiplexed mode or the Expansion Bus Interface Data bus is switched from address to data when configured in multiplexed mode during the Strobe Phase.

The Strobe Phase is extended from one to 15 clock cycles, as defined by programming bits 25:22 of the Timing and Control (EXP_TIMING_CS) Register, T3 – Strobe Timing parameter.

The Hold Phase of an expansion bus access is provided to allow a hold time for data to remain valid after the data strobe has transitioned to an invalid state. During a write access, the Hold Phase provides hold time for data written to an external device on the Expansion bus, after the strobe pulse has completed.

During a read access, the Hold Phase allows an external device time to release the bus after driving data back to the controller. The Hold Phase is extended one to three clock cycles, using bits 21:20 of the Timing and Control (EXP_TIMING_CS) Register, T4 – Hold Timing parameter.

After the address and chip select is de-asserted, the Expansion Bus Controller is programmed to wait a number of clocks before starting the next Expansion Bus access. This action is referred to as the Recovery Phase. The Recovery Phase is extended one to 15 clock cycles using bits 19:16 of the Timing and Control (EXP_TIMING_CS) Register, T5 – Recovery Timing parameter.

12.4.1.5 Using I/O Wait

The EX_IOWAIT_N signal is available to be shared by devices attached to chip selects 0 through chip select 3, when configured in Intel or Motorola modes of operation. The main purpose of this signal is to properly communicate with slower devices requiring more time to respond during data access. During idle cycles, the board is responsible for ensuring that EX_IOWAIT_N is pulled-up. The Expansion Bus Controller always ignores EX_IOWAIT_N for synchronous Intel mode writes.

As shown in Figure 141, a normal phase transaction is initiated during the T1 (address timing) period, where the processor drives the address lines with an address that is decoded by the peripheral being accessed.

The next segment of the transaction is the T2 (Chip Select Timing) period, where the processor asserts Chip Select and the Address signals have reached a stable state. EX_IOWAIT_N must be asserted during the T2 period. If not asserted at this time, the

processor ignores EX_IOWAIT_N and treats it as it never occurred. If EX_IOWAIT_N is asserted during T2, the processor expects the signal to be deasserted during the T3 (Strobe Timing) period.

T3 is programmed from 0 to F, where the value indicates the number of cycles that the processor waits for EX_IOWAIT_N to be deasserted. The counter starts at the rising edge of the clock when EX_RD_N or EX_WR_N is asserted. The following rules describe processor operation during T3:

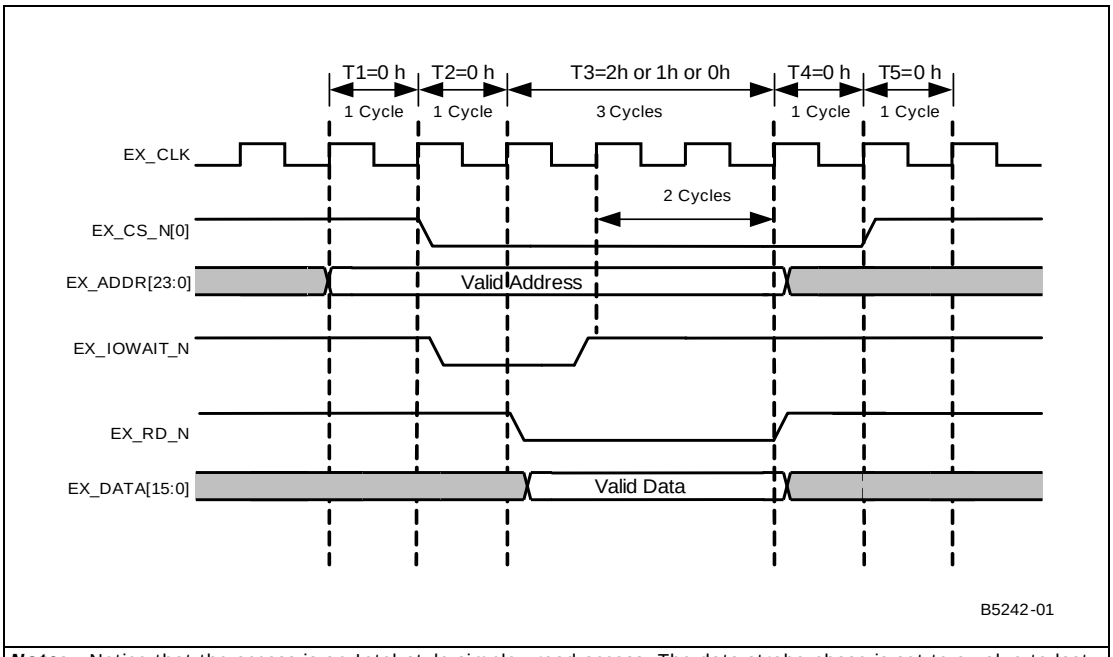
- If EX_IOWAIT_N is deasserted at least two clock cycles before the T3 counter expires, then the processor deasserts EX_RD_N/EX_WR_N at the end of the number of cycles programmed in T3.
- If EX_IOWAIT_N is deasserted after T3 counter has expired, it takes 2 more clock cycles before the processor deasserts the EX_RD_N/EX_WR_N signal.
- If EX_IOWAIT_N is not deasserted before the T3 counter expires, then EX_RD_N/EX_WR_N continues to be asserted for as long as EX_IOWAIT_N continues to be asserted, plus 2 more clock cycles.

The T4 (Hold Timing) period is the time interval where Chip Select is held after READ is deasserted. T4 prevents bus contention as Chip Select is asserted, in case the peripheral driving the bus continues to send data out after READ has been deasserted. During T4 no other transaction can start, since the current transaction does not finish until Chip Select is deasserted by the processor.

T5 is the recovery time required before the next transaction can start.

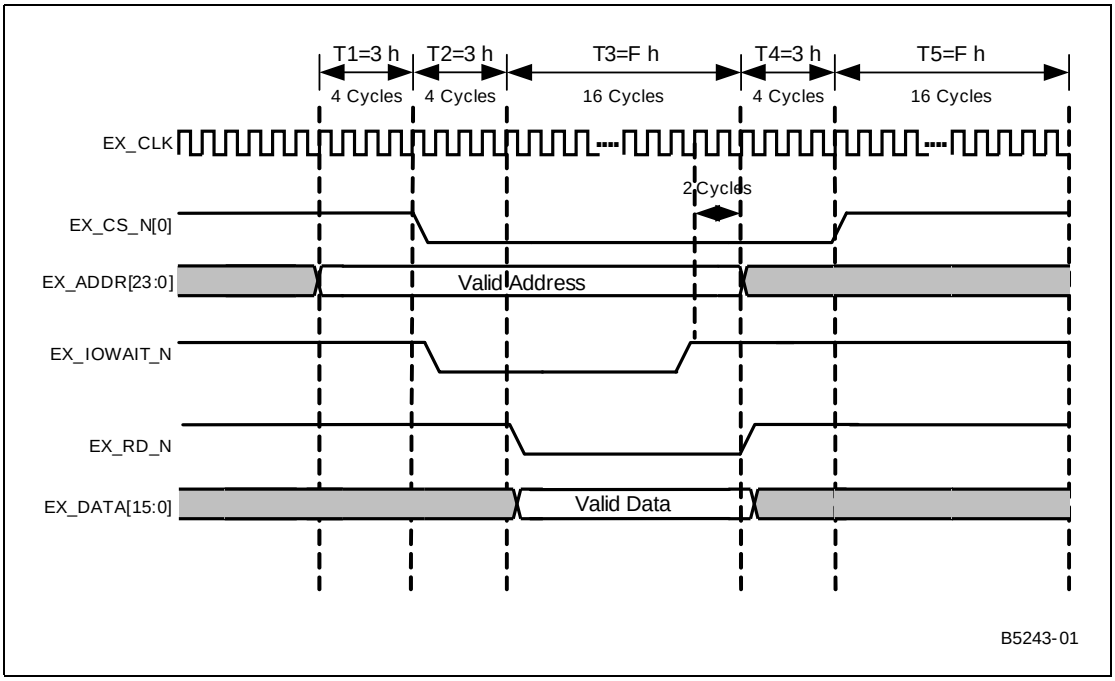
In the case of extended phase timing, EX_IOWAIT_N is used in the same way as the normal phase, but, the T1, T2, T4 and T5 periods take place over 4 cycles. T3 is still programmable but each value is a multiple of 4 cycles. See Figure 142 for details.

Figure 141. I/O Wait Normal Phase Timing



Note: Notice that the access is an Intel-style simplex read access. The data strobe phase is set to a value to last three clock cycles. The data is returned from the peripheral device prior to the three clocks and the peripheral device de-asserts EX_IOWAIT_N. The data strobe phase terminates after two clocks even though the strobe phase was configured to pulse for three clocks.

Figure 142. I/O Wait Extended Phase Timing

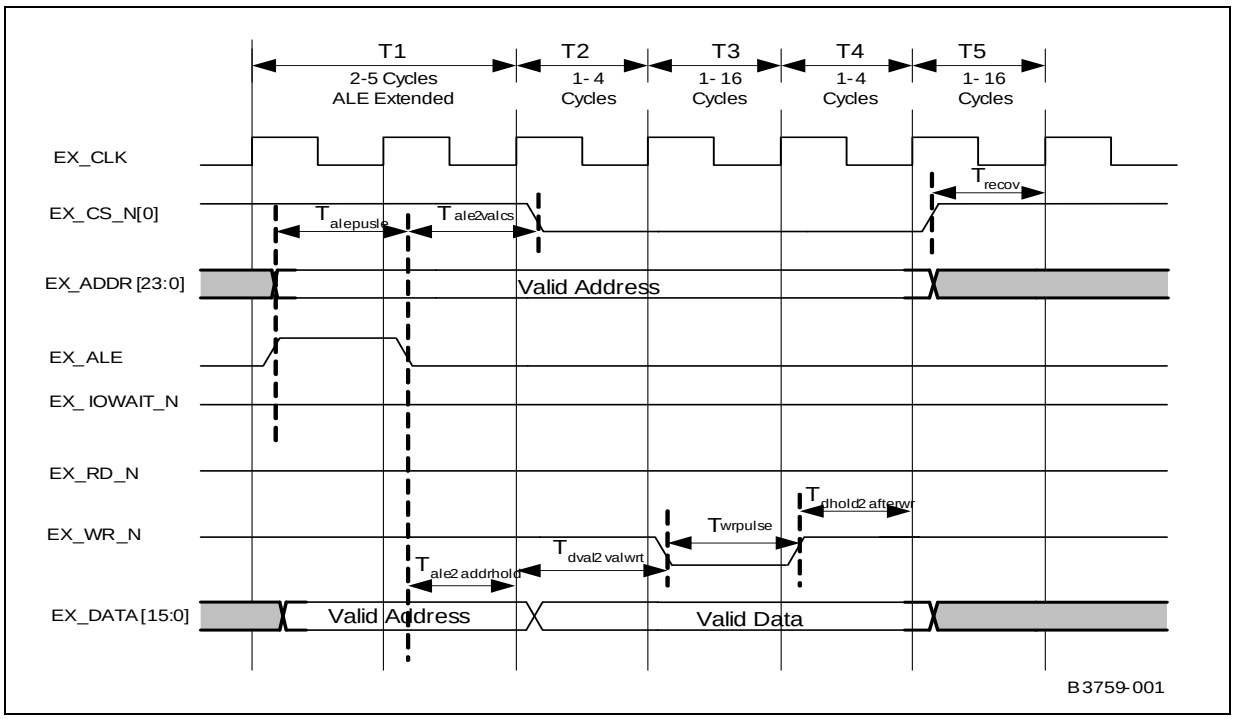


12.4.1.6 Expansion Bus Outbound Timing Diagrams

The STATE signal that is shown in some of the following timing diagrams is the internal state of the Expansion Bus Controller.

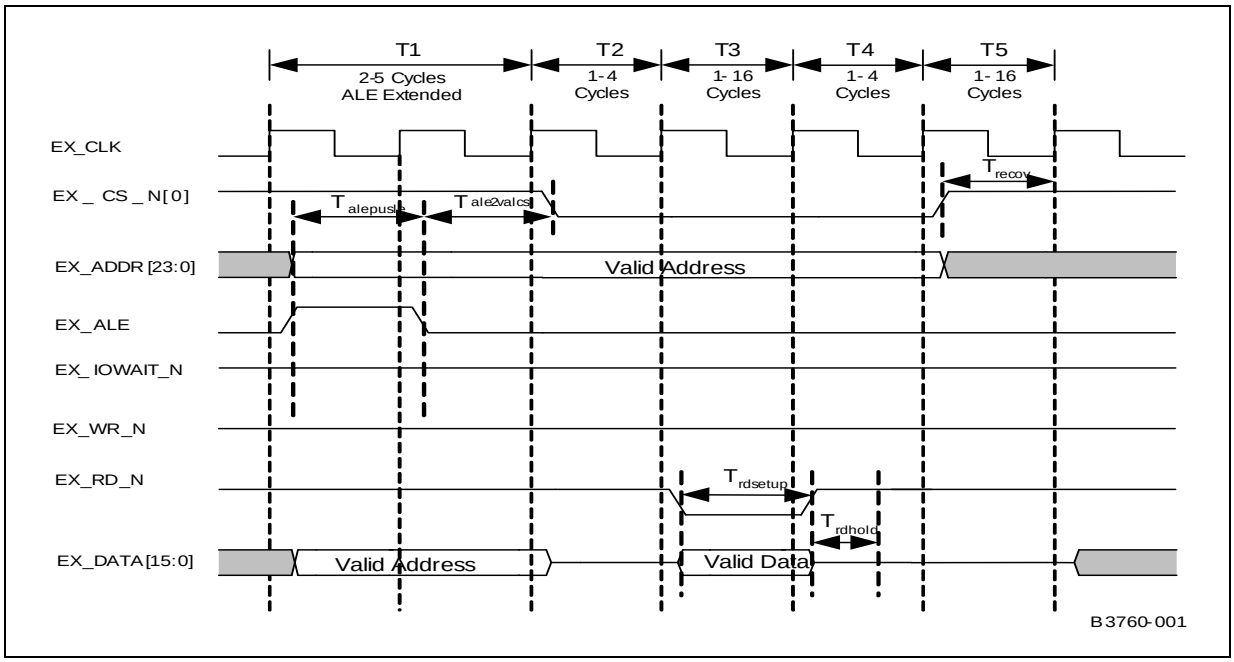
12.4.1.6.1 Intel, Multiplexed-Mode Write Access

Figure 143. Expansion Bus Write (Intel, Multiplexed Mode)



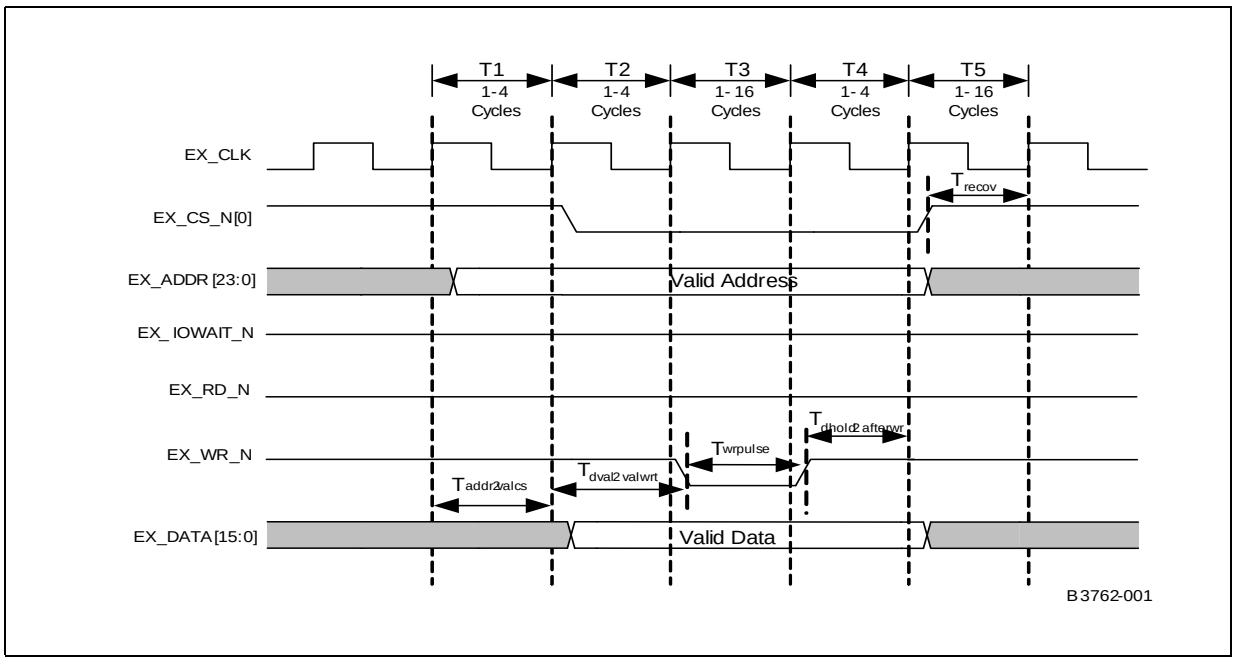
12.4.1.6.2 Intel, Multiplexed-Mode Read Access

Figure 144. Expansion Bus Read (Intel, Multiplexed Mode)



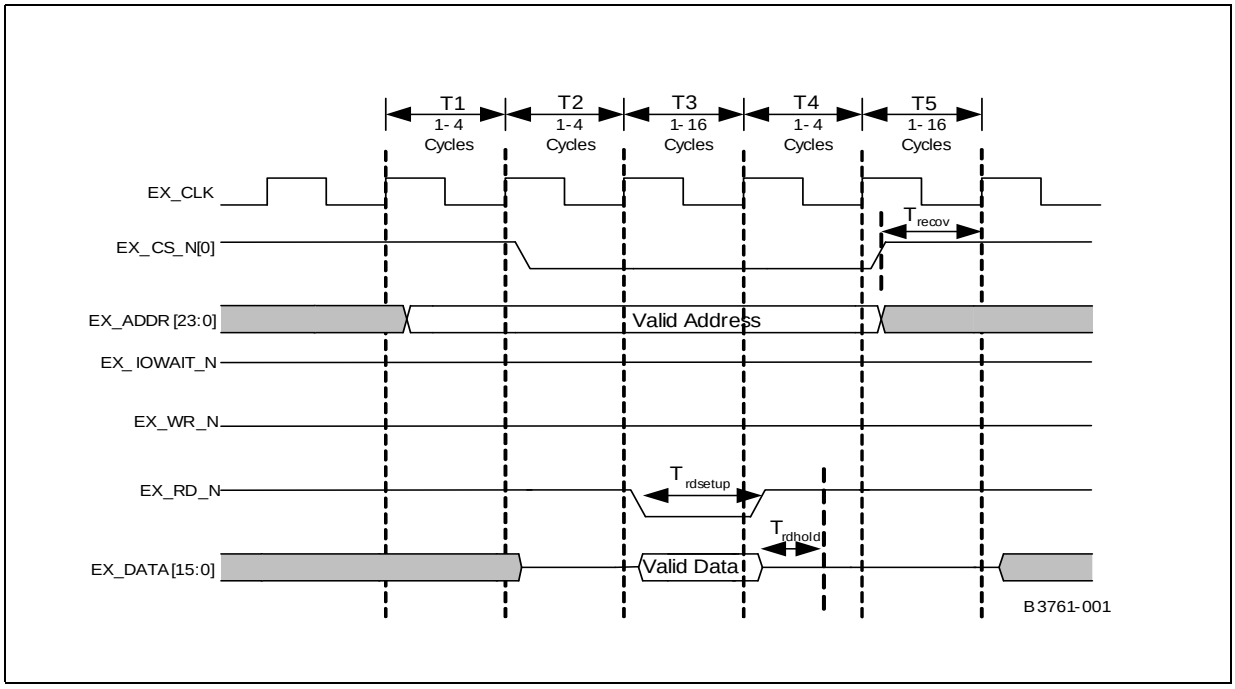
12.4.1.6.3 Intel Simplex-Mode and Synchronous Intel Write Access

Figure 145. Expansion Bus Write (Intel Simplex-Mode, Synchronous Intel)



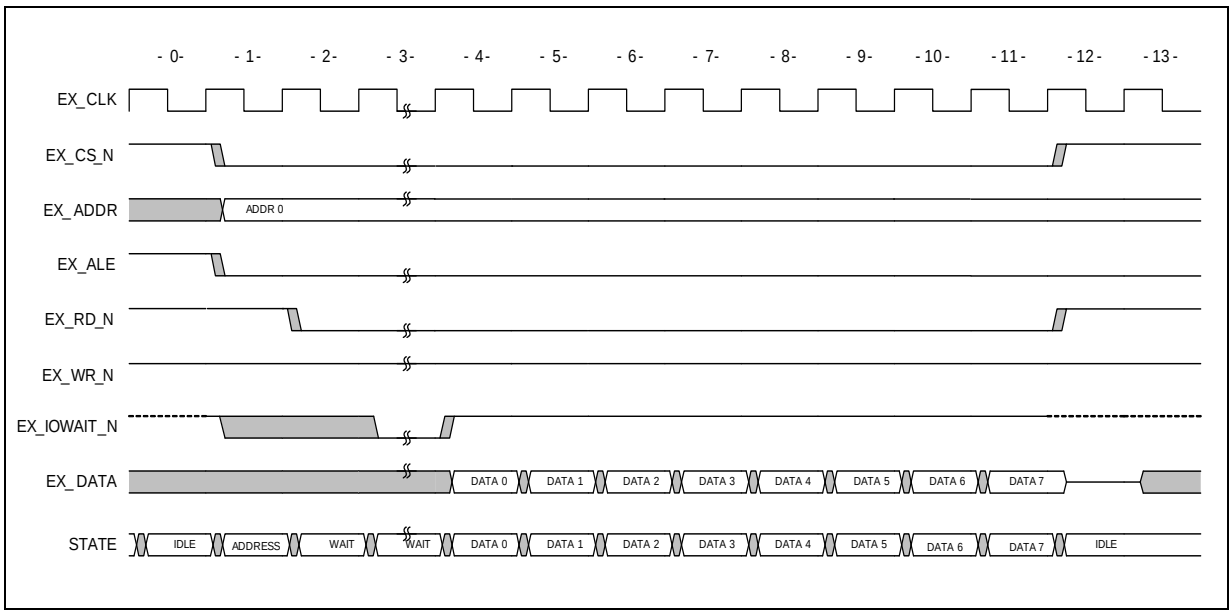
12.4.1.6.4 Intel Simplex-Mode Read Access

Figure 146. Expansion Bus Read (Intel Simplex-Mode)



12.4.1.6.5 Synchronous Intel 8-Word Read Access

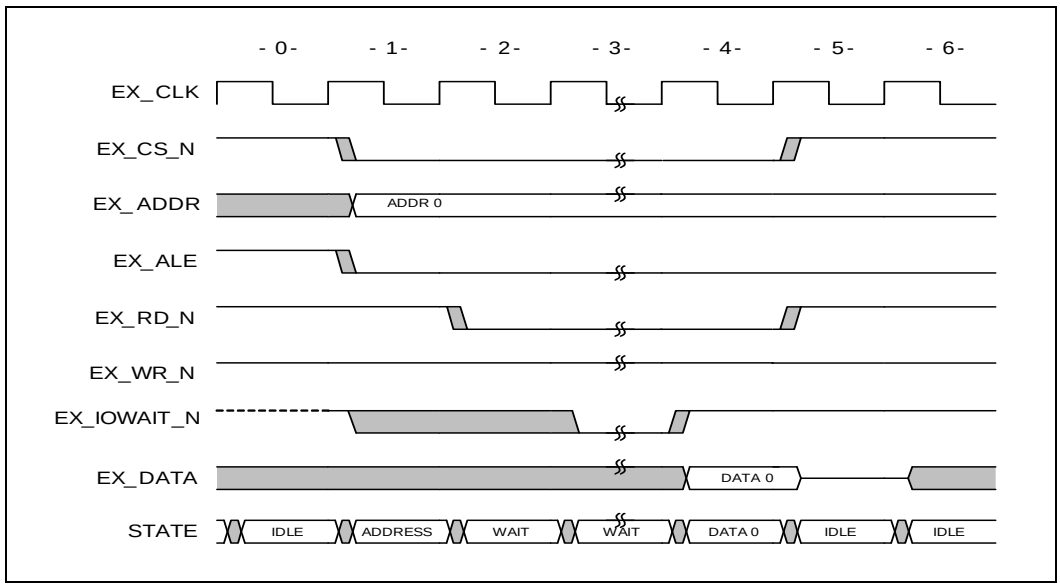
Figure 147. Intel Synchronous 8-Word Read



The Figure 147 shows an 8-word read to a Synchronous Intel device such as Synchronous Intel StrataFlash (for example, 32-bit / 8-bit device). This is also applicable to depending on the EX_CLK period, the latency count bits in the Intel Synchronous Device read configuration register must be programmed appropriately. The Expansion Bus Controller always waits in cycles 1 and 2, regardless of EX_IOWAIT_N. The device then asserts EX_IOWAIT_N for several cycles and deasserts EX_IOWAIT_N when its ready to transfer data. After the device deasserts EX_IOWAIT_N, it transfers the remaining words until all 8 words are transferred. The Expansion Bus Controller and Synchronous Intel device both support wrapping for 8-word reads, therefore ADDR0 is not always aligned to an 8-word boundary. The STATE signal shows the internal Expansion bus state. The diagram is applicable to other configuration of Synchronous Intel device, where after the deasserts EX_IOWAIT_N, it transfers the remaining words until all words are transferred, example 16-bit device (16-word read) and 8-bit device (1 to 8-word read).

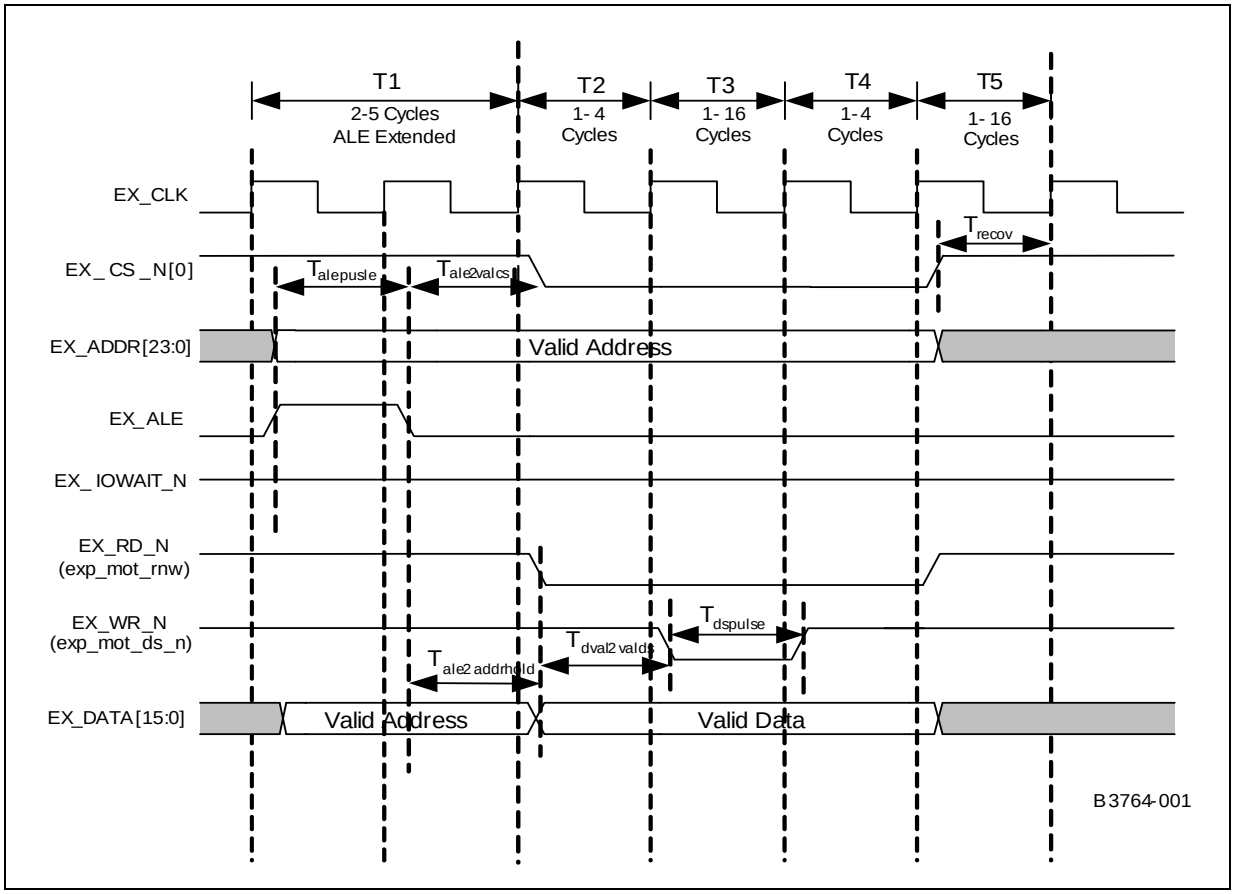
12.4.1.6.6 Synchronous Intel 1-Word Read Access

Figure 148. Intel Synchronous One-Word Read



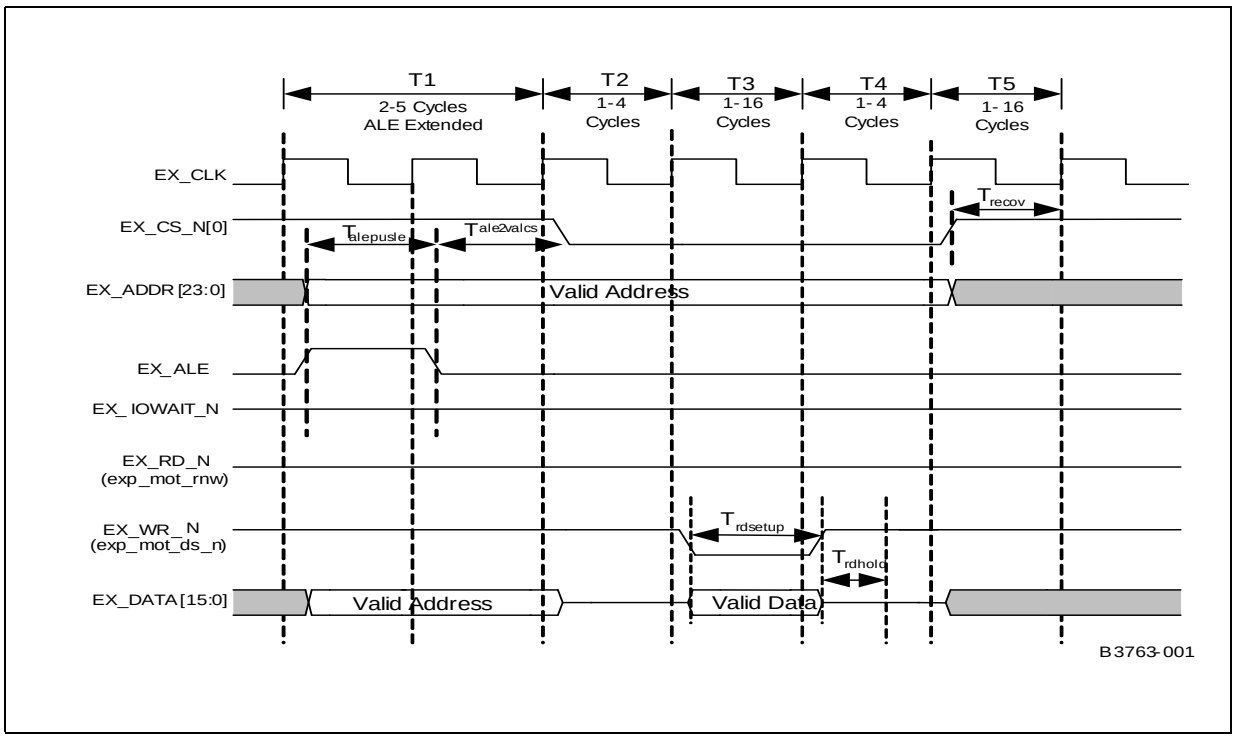
12.4.1.6.7 Motorola*, Multiplexed-Mode Write Access

Figure 149. Expansion Bus Write (Motorola*, Multiplexed Mode)



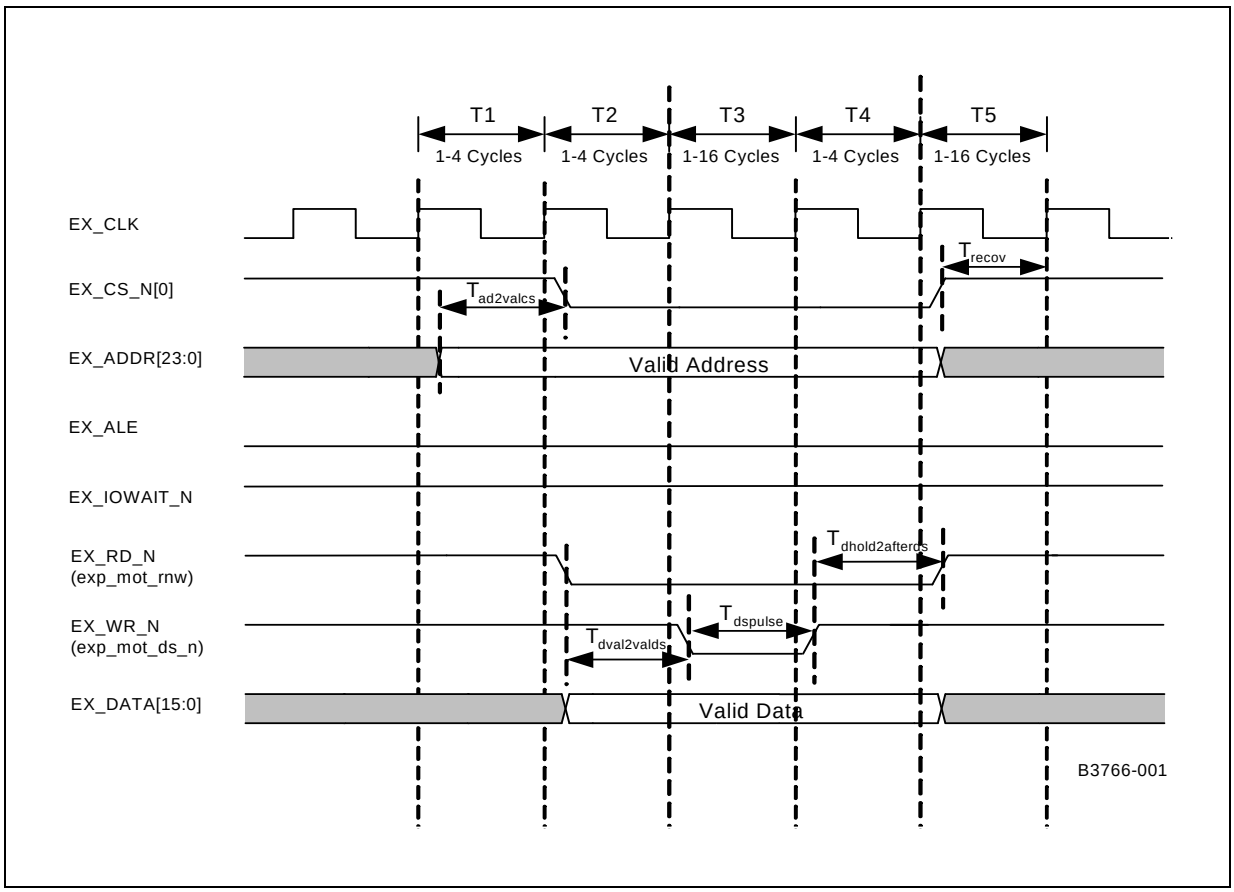
12.4.1.6.8 Motorola*, Multiplexed-Mode Read Access

Figure 150. Expansion Bus Read (Motorola*, Multiplexed Mode)



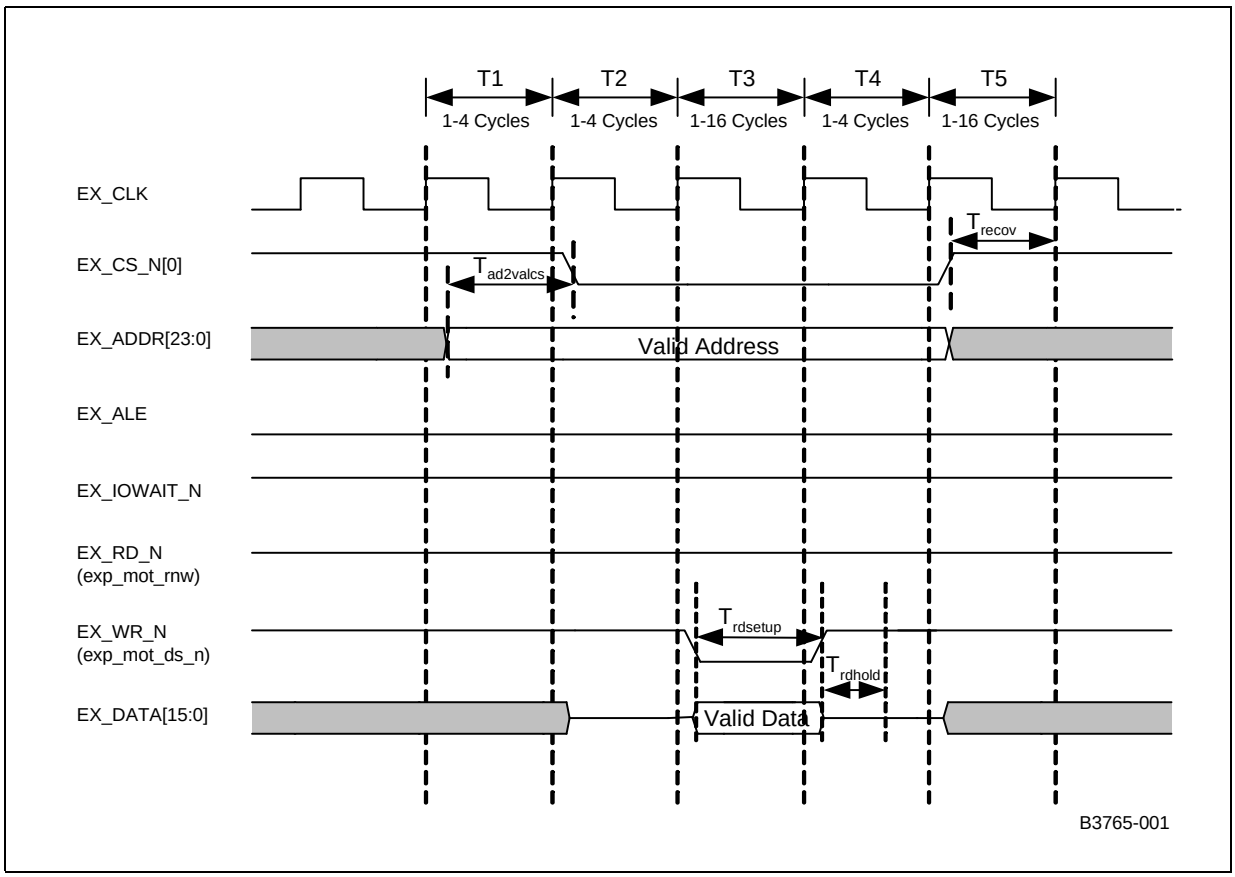
12.4.1.6.9 Motorola*, Simplex-Mode Write Access

Figure 151. Expansion-Bus Write (Motorola*, Simplex Mode)



12.4.1.6.10 Motorola*, Simplex-Mode Read Access

Figure 152. Expansion-Bus Read (Motorola*, Simplex Mode)



12.4.2 Configuration Straps

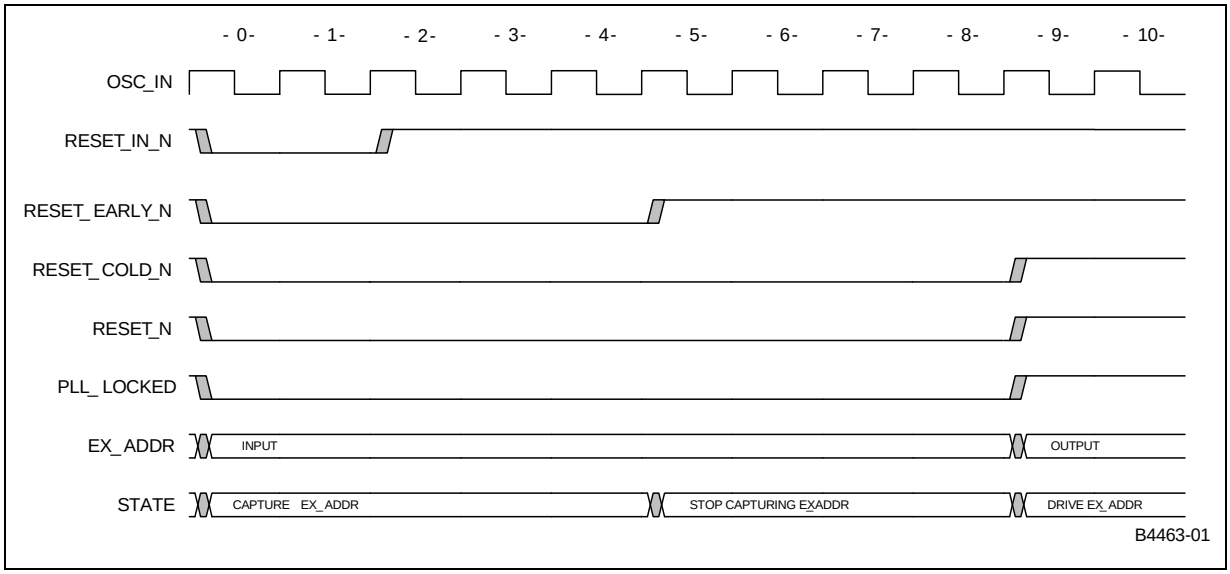
The Expansion Bus Controller contains configuration registers beyond what is required for its own configuration. There are several bits of configuration signals provided as output from the Expansion Bus Controller to the rest of the IXP43X network processors. These signals provide the AHB with functions like the software interrupt capabilities, location of Expansion Bus Controller in the processor memory map, PCI Host/Arbiter information, and configuration information on devices connected to the Expansion Bus.

One of these general purpose configuration registers is used to capture the value on the EX_ADDR pins immediately after reset. In the Expansion Bus Interface, 24 address lines are used to capture this configuration information during reset. When power up is complete and reset is asserted, the 24 address lines are configured as inputs. When internal reset is asserted (PLL_LOCK = 0), the configuration registers latch the values contained on the 24 address lines and the 24 address lines become configured as outputs. The Expansion Bus address signals have internal pull-up resistors of about 50Kohm that are only enabled when PLL_LOCK is deasserted. Pull-down resistors is placed on some of the 24 address lines to select the configuration values at reset. The pull-down resistors must be able to over-drive the internal pull-up resistors. Refer to Section 12.5.5, “Configuration Register 0” on page 592 for additional details on particular configuration options. The Expansion Bus Controller samples EX_ADDR when

reset_early_n = ‘0’, that occurs during power up or during a watch dog reset. But, the Expansion Bus Controller only samples the Intel XScale processor Clock Set bits, EX_ADDR[23:21], when reset_early_n = ‘0’ and reset_cold_n = ‘0’.

12.4.2.1 Sampling EX_ADDR During Reset

Figure 153. Sampling EX_ADDR During Reset



12.4.2.2 Expansion Bus Controller Operation

In this document, there are several occurrences of the word “must” in the description of the functional operation of the Expansion Bus Controller. Failure to comply to these requirements results in unpredictable behavior of the Expansion Bus Controller.

12.5 Detailed Register Descriptions

Accesses to Expansion bus registers is only word transfers. Byte and half-word accesses result in unpredictable operation. Accesses to all reserved bits must be written with zero unless otherwise specified. All reserved address spaces must not be read. All writes to reserved address spaces results in unpredictable operation.

Table 212. Register Legend

Attribute	Legend	Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear	RW1C	Normal Read/ Write ‘1’ to clear
PR	Preserved	RO	Read Only	RW1S	Normal Read/ Write ‘1’ to Set
RS	Read/Set	WO	Write Only	W1C	Write ‘1’ to clear
RW	Read/Write	NA	Not Accessible		

Table 213. Expansion Bus Register Summary

Address	Register Name	Description	Reset Value	Attribute
0XC4000000	EXP_TIMING_CS0	Timing and Control Register for Chip Select 0	0xBFFF3C0x	See Register Table
0XC4000004	EXP_TIMING_CS1	Timing and Control Register for Chip Select 1	0x00000000	See Register Table
0XC4000008	EXP_TIMING_CS2	Timing and Control Register for Chip Select 2	0x00000000	See Register Table
0XC400000C	EXP_TIMING_CS3	Timing and Control Register for Chip Select 3	0x00000000	See Register Table
0XC4000020	EXP_CNFG0	General Purpose Configuration Register 0	0x80XXXXXX	See Register Table
0XC4000024	EXP_CNFG1	General Purpose Configuration Register 1	0x00000000	See Register Table
0xC4000028	EXP_UNIT_FUSE_RESET	Specifies the value of the fuse register.	0xXXXXXXXX	See Register Table
0xC4000124	EXP_SYNCINTEL_COUNT	This register is used to set the read latency count when a Synchronous Intel Device is accessed.	0x00000000	See Register Table
0xC4000230	EXP_USBAFE_DBGCTRL1	This register is used to set the noise sensitivity.	0x170F0035	See Register Table

12.5.1 Timing and Control Registers for Chip Select 0

Register Name:	EXP_TIMING_CS0				
Hex Offset Address:	0XC4000000		Reset Hex Value:	0xBFFF3C0x	
Register Description:	Timing and Control Registers				
Access: See Table 214					

31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR	T1		T2		T3				T4		T5				CYC_TYPE[1]	CYC_TYPE[0]	CNFG[4:1]				CNFG[0]	Sync_Intel	EXP_CHIP	BYTE_RD16	(Reserved)	MUX_EN	SPLT_EN	WORD	WR_EN	BYTE_EN

Note: The undefined (X) in the reset value is dependent upon values supplied to the chip on the Expansion Bus address during the reset sequence. Refer to Section 12.5.5, “Configuration Register 0” on page 592 for additional details.

12.5.2 Timing and Control Registers for Chip Select 1

Register Name:	EXP_TIMING_CS1																															
Hex Offset Address:	0XC4000004															Reset Hex Value:	0x00000000															
Register Description:	Timing and Control Registers																															
Access: See Table 214																																
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR	T1		T2		T3				T4		T5				CYC_TYPE[1]	CYC_TYPE[0]	CNFG[4:1]				CNFG[0]	Sync_Intel	EXP_CHIP	BYTE_RD16	(Reserved)	MUX_EN	SPLT_EN	WORD	WR_EN	BYTE_EN

12.5.3 Timing and Control Registers for Chip Select 2

Register Name:	EXP_TIMING_CS2																															
Hex Offset Address:	0XC4000008															Reset Hex Value:	0x00000000															
Register Description:	Timing and Control Registers																															
Access: See Table 214																																
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR	T1		T2		T3				T4		T5				CYC_TYPE[1]	CYC_TYPE[0]	CNFG[4:1]				CNFG[0]	Sync_Intel	EXP_CHIP	BYTE_RD16	(Reserved)	MUX_EN	SPLT_EN	WORD	WR_EN	BYTE_EN

12.5.4 Timing and Control Registers for Chip Select 3

Register Name:	EXP_TIMING_CS3																															
Hex Offset Address:	0XC400000C															Reset Hex Value:	0x00000000															
Register Description:	Timing and Control Registers																															
Access: See Table 214																																
31	30	29	28	27	26	25			22	21	20	19			16	15	14	13				9	8	7	6	5	4	3	2	1	0
CSx_EN	PAR	T1		T2		T3				T4		T5				CYC_TYPE[1]	CYC_TYPE[0]	CNFG[4:1]				CNFG[0]	Sync_Intel	EXP_CHIP	BYTE_RD16	(Reserved)	MUX_EN	SPLT_EN	WORD	WR_EN	BYTE_EN

Table 214. Bit Level Definition for each of the Timing and Control Registers (Sheet 1 of 2)

Bits	Name	Description	Access
31	CSx_EN	0 = Chip Select x disabled 1 = Chip Select x enabled	RW
30	PAR	0 = Parity is not generated or compared Reset value for this bit is 0. Read-only, clear when a 1 is written to this bit.	RO
29:28	T1 – Address timing	00 = Generate normal address phase timing 01 - 11 = Extend address phase by 1 - 3 clocks	RW
27:26	T2 – Setup / Chip Select Timing	00 = Generate normal setup phase timing 01 - 11 = Extend setup phase by 1 - 3 clocks	RW
25:22	T3 – Strobe Timing	0000 = Generate normal strobe phase timing 0001-1111 = Extend strobe phase by 1 - 15 clocks	RW
21:20	T4 – Hold Timing	00 = Generate normal hold phase timing 01 - 11 = Extend hold phase by 1 - 3 clocks	RW
19:16	T5 – Recovery Timing	0000 = Generate normal recovery phase timing 0001-1111 = Extend recovery phase by 1 - 15 clocks	RW
15	CYC_TYPE[1]	0 = Configures the Expansion bus for Intel/Motorola cycles Reset value for this bit is 0. Read-only, clear when a 1 is written to this bit.	RO
14	CYC_TYPE[0]	0 = Configures the Expansion bus for Intel cycles if CYC_TYPE[1]=0 1 = Configures the Expansion bus for Motorola cycles if CYC_TYPE[1]=0.	RW
13:10	CNFG[4:1]	Device Configuration Size. Bit CNFG[0] must be 0. Calculated using the formula: SIZE OF ADDR SPACE = $2^{(9+CNFG[4:1]+16*CNFG[0])}$ For Example: 0000 = Address space of 2^9 = 512 Bytes 0001 = Address space of 2^{10} = 1024 Bytes ... 1000 = Address space of 2^{17} = 128 Kbytes ... 1110 = Address space of 2^{23} = 8 Mbytes 1111 = Address space of 2^{24} = 16 Mbytes	RW
9	CNFG[0]	0 = To configure device to address space of 16Mbytes or less Reset value for this bit is 0. Read-only, clear when a 1 is written to this bit.	RO
8	Sync_Intel	Synchronous Intel StrataFlash select. This bit must be 0 if CYC_TYPE is not programmed to Intel cycles. 0 = Target device is not a Synchronous Intel StrataFlash 1 = Target device is a Synchronous Intel StrataFlash	RW
7	EXP_CHIP	0 = Target device is not one of the IXP43X network processors. Reset value for this bit is 0. Read-only, clear when a 1 is written to this bit.	RO
6	BYTE_RD16	Byte read access to Half Word device 0 = Byte access disabled. 1 = Byte access enabled. Should always set to 0 as byte reads to 16-bit devices are not supported.	RW
5	(Reserved)	(Reserved)	RO

Table 214. Bit Level Definition for each of the Timing and Control Registers (Sheet 2 of 2)

Bits	Name	Description	Access
4	MUX_EN	0 = Separate address and data buses. 1 = Multiplexed address / data on data bus.	RW
3	SPLT_EN	0 = AHB split transfers disabled. 1 = AHB split transfers enabled.	RW
2	WORD	0 = Expansion bus uses 8/16 bit data bus based on BYTE_EN bit Reset value for this bit is 0. Read-only, clear when a 1 is written to this bit.	RO
1	WR_EN	0 = Writes to CS region are disabled. 1 = Writes to CS region are enabled.	RW
0	BYTE_EN	0 = Expansion bus uses 16-bit data bus if WORD = 0. 1 = Expansion bus uses only 8-bit data bus if WORD = 0.	RW

The EXP_TIMING_CS registers may only be written if there is not an outstanding Expansion bus transaction. Software must ensure that all outstanding Expansion bus transfers are complete before changing the EXP_TIMING_CS registers.

12.5.5 Configuration Register 0

At power up or whenever RESET_IN_N is asserted, the expansion bus address outputs are switched to inputs and the states of the bits are captured and stored in Configuration Register 0, bits 23 through 0. This occurs when PLL_LOCK is deasserted.

These configuration bits are made available to the system as outputs from the Expansion Bus Controller block. With the exception of bits 23, 22, 21 and bit 11 that are read only, all other bits is written and read from the AHB. But software should only modify the value of MEM_MAP (bit 31) and leave all other values as the captured value.

Register Name:	EXP_CNFG0				
Hex Offset Address:	0XC4000020		Reset Hex Value:		0x80XXXXXX
Register Description:	Configuration Register #0				

Access: See table below.

31	30						24	23	22	21	20			17	16	15				11	10	9	8	7	6	5	4	3	2	1	0
MEM_MAP	(Reserved)							Clock Set			Customer				(Reserved)					DDR_MODE	IOWAIT_CS0	EXP_MED_DRIVE	USB CLOCk	32_FLASH	(Reserved)	EXP_DRIVE	PCI_CLK		PCI_ARB	PCI_ HOST	8/16

Table 215. Configuration Register 0 Description (Sheet 1 of 3)

Bit	Name	Reset Value	Description	Access
31	MEM_MAP	1	Location of Expansion Bus in memory map space: 0 = Located at “50000000” (normal mode) 1 = Located at “00000000” (boot mode)	RW
30:24	(Reserved)	0x0	(Reserved)	RO
23:21	Intel XScale® Processor Clock Set[2:0]	EX_ADDR[23:21]	Allow a slower Intel XScale processor clock speed to override device fuse settings. But cannot be used to over clock core speed. Refer to Table 216, “Setting Intel XScale® Processor Operation Speed” on page 594 for additional details.	RO

Table 215. Configuration Register 0 Description (Sheet 2 of 3)

Bit	Name	Reset Value	Description	Access
20:17	Customer	EX_ADDR[20:17]	Customer defined bits.	RW
16:12	(Reserved)	EX_ADDR[16:12]	(Reserved)	RO
11	DDR_MODE	EX_ADDR[11]	DDRI or DDRII mode selection: 0 = DDRII mode (400MHz) 1 = DDRI mode (266MHz) DDR_mode or DDR clock speed selection bit is read only and strapped in from exp address bit 11 upon reset_early_n and reset_cold_n activated.	RO
10	IOWAIT_CS0	EX_ADDR[10]	1 = EX_IOWAIT_N is sampled during the read/write Expansion bus cycles as defined in Section 12.4.1.5, “Using I/O Wait” on page 577 for Chip Select 0. 0 = EX_IOWAIT_N is ignored for read and write cycles to Chip select 0 if EXP_TIMING_CS0 is configured to Intel mode. Typically, IOWAIT_CS0 must be pulled down to Vss when attaching a Synchronous Intel StrataFlash on Chip select 0 since the default mode for EXP_TIMING_CS0 is Intel mode and EX_IOWAIT_N is an unknown value for Synchronous Intel StrataFlash. If the board does not connect the Synchronous Intel StrataFlash WAIT pin to EX_WAIT_N (and the board guarantees EX_IOWAIT_N is pulled up), the value of IOWAIT_CS0 is a don’t care since EX_IOWAIT_N does not be asserted. When EXP_TIMING_CS0 is re-configured to Intel Synchronous mode during boot-up (for Synchronous Intel chips), the Expansion Bus Controller ignores EX_IOWAIT_N during read and write cycles since the WAIT functionality is determined from the EXP_SYNCINTEL_COUNT and EXP_TIMING_CS registers.	RW
9	EXP_MEM_DRIVE	EX_ADDR[9]	Refer to the table found in EXP_DRIVE bit (bit 5) of this register.	RW
8	USB Clock	EX_ADDR[8]	Controls the USB clock select 1 = USB Host clock is generated internally 0 = USB Host clock is generated from GPIO[1]. When generating a spread spectrum clock on OSC_IN, GPIO[1] can be driven from the system board to generate a 48 MHz clock for the USB Host. The USB clock select bit is preset to 1 upon reset_early_n activated to select internal PLL clock.	RW
7	32_FLASH	EX_ADDR[7]	Selects the data bus width of the FLASH memory device found on Chip Select 0. Refer to 8/16_ FLASH bit (Bit 0) of this register as well. 0 = 8 or 16-bit data bus size (must be pulled down during address strapping) 1 = not supported	RW
6	(Reserved)	EX_ADDR[6]	(Reserved).	RO
5	EXP_DRIVE	EX_ADDR[5]	Expansion bus low/medium/high drive strength. The drive strength depends on EXP_DRIVE and EXP_MEM_DRIVE configuration bits. EXP_MEM_DRIVE EXP_DRIVE Expansion drive strength 0 0 Reserved 0 1 Medium Drive 1 0 Low Drive 1 1 High Drive	RW
4	PCI_CLK	EX_ADDR[4]	Sets the clock speed of the PCI Interface 0 = 33MHz (must be pulled down during address strapping) 1 = not supported	RW
3	(Reserved)	EX_ADDR[3]	(Reserved). EX_ADDR[3] must not be pulled down during address strapping. This bit must be written to ‘1’ if performing a write to this register.	RO

Table 215. Configuration Register 0 Description (Sheet 3 of 3)

Bit	Name	Reset Value	Description	Access
2	PCI_ARB	EX_ADDR[2]	Enables the PCI Controller Arbiter 0 = PCI arbiter disabled 1 = PCI arbiter enabled	RW
1	PCI_HOST	EX_ADDR[1]	Configures the PCI Controller as PCI Bus Host 0 = PCI as non-host 1 = PCI as host	RW
0	8/16_FLASH	EX_ADDR[0]	Specifies the data bus width of the FLASH memory device found on Chip Select 0. 8/16_FLASH — Data bus size 0 — 16-bit 1 — 8-bit	RW

The chip-level memory map used is determined by the state of bit 31. At system reset this bit is a '1' and the memory map places the Expansion Bus at address 0x00000000 through 0x0FFFFFFF. This allows boot code stored in flash to be retrieved and executed as required.

Once the boot sequence completes this bit is written to a '0', switching the default system memory map to place the DDRII/DDRI SDRAM controller at address 0x00000000 to 0x0FFFFFFF. The Expansion Bus Controller now resides at address 0x50000000 to 0x5FFFFFFF. Weak pull-up resistors are placed on each expansion bus address pin.

Table 216. Setting Intel XScale® Processor Operation Speed

Intel XScale® Processor Speed (Factory Part Speed)	Cfg0 EX_ADDR[21]	Cfg1 EX_ADDR[22]	Cfg_en_n EX_ADDR[23]	Actual Core Speed (MHz)
667 MHz	X	X	1	667 MHz
667 MHz	0	0	0	667 MHz
667 MHz	1	0	0	533 MHz
667 MHz	1	1	0	400 MHz
533 MHz	X	X	1	533 MHz
533 MHz	0	0	0	533 MHz
533 MHz	1	1	0	400 MHz
400 MHz	X	X	1	400 MHz
400 MHz	0	0	0	400 MHz

Note: The Intel XScale processor can operate at slower speeds than the factory programmed speed setting. This is done by placing a value on Expansion bus address bits 23,22,21 when PLL_LOCK is deasserted and knowing the speed grade of the part from the factory. Column 1 above denotes the speed grade of the part from the factory. Column 2, 3, and 4 denotes the values captured on the Expansion Bus address bits when PLL_LOCK is deasserted. Column 5 represents the speed at which the Intel XScale processor speed is now operating at.

12.5.6 Configuration Register 1

One additional configuration register is defined within the Expansion Bus Controller for use by the IXP43X network processors.

Register Name:	EXP_CNFG1				
Hex Offset Address:	0XC4000024		Reset Hex Value:	0x00000000	
Register Description:	Configuration Register #1				
Access: See table below					

31	30:19	18	17	16	15	14	13	12	11	10	9	8	7	2	1	0
MPI_EN	(Reserved)	NPE-C MII	(Reserved)	NPE-A MII	(Reserved)	NPE-C ERR_EN	(Reserved)	NPE-A ERR_EN	(Reserved)		FORCE_BYTE_SWAP	BYTE_SWAP_EN	(Reserved)		SW_ INT1	SW_ INT0

Table 217. Configuration Register 1 Description

Bit	Name	Reset Value	Description	Access
31	MPI_EN	0x0	0 = DDR transactions are routed through the AHB 1 = DDR transactions are routed through the MPI port. This bit should always be set when configuring the DDR during boot-up. When this bit is set, the performance between the Intel XScale processor and DDR is increased.	RW
30:19	(Reserved)	0x0	(Reserved)	RO
18	NPE-C MII	0x0	0 = MII mode enabled for NPE-C Read-only, clear when a 1 is written to this bit.	RO
17	(Reserved)	0x0	(Reserved)	RO
16	NPE-A MII	0x0	0 = MII mode enabled for NPE-A Read-only, clear when a 1 is written to this bit.	RO
15	(Reserved)	0x0	(Reserved)	RO
14	NPE-C ERR_EN	0x0	0 = Error handling in NPE-C is disabled 1 = Error handling in NPE-C is enabled. For more information, see Section 21.0, “Error Handling”.	RW
13	(Reserved)	0x0	(Reserved)	RO
12	NPE-A ERR_EN	0x0	0 = Error handling in NPE-A is disabled 1 = Error handling in NPE-A is enabled. For more information, see Section 21.0, “Error Handling”.	RW
11:10	(Reserved)	0x0	(Reserved)	RO

Notes: For transactions initiated by the Intel XScale processor, the selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Intel® IXP4XX Product Line Memory Management Unit (MMU) and the BYTE_SWAP_EN bit (Bit 8). The BYTE_SWAP_EN resets to 0.

The default endian conversion method for the IXP43X network processors is address coherency. This was selected to enable backward compatible with the Intel® IXP42X and IXP46X Network Processors.

The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit.
When the bit is 0, address coherency is always performed.
When the bit is 1, the type of coherency depends on the P-attribute bit.

The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output from the Intel XScale processor, with any store or load access associated with that page.

Table 217. Configuration Register 1 Description

Bit	Name	Reset Value	Description	Access
9	FORCE_BYTE_SWAP	0x0	Forces byte swapping on Intel XScale processor initiated accesses in little-endian mode regardless of the P-attribute bit. 0 = do not force byte swapping 1 = force byte swapping on all transfers regardless of the P-attribute bit in little-endian mode. When this bit is set, BYTE_SWAP_EN is ignored and byte swapping always happens in little-endian mode.	RW
8	BYTE_SWAP_EN	0x0	Sets byte swapping at the on Intel XScale processor initiated accesses 0 = byte swapping disabled 1 = byte swapping enabled ***Note:*** See note, below.	RW
7:2	(Reserved)	0x0	(Reserved)	RO
1	SW_INT1	0x0	0 = Disable interrupt 1 = Generate interrupt	RW
0	SW_INT0	0x0	0 = Disable interrupt 1 = Generate interrupt	RW

Notes: For transactions initiated by the Intel XScale processor, the selection between address or data coherency is controlled by a software-programmable, P-attribute bit in the Intel® IXP4XX Product Line Memory Management Unit (MMU) and the BYTE_SWAP_EN bit (Bit 8). The BYTE_SWAP_EN resets to 0.

The default endian conversion method for the IXP43X network processors is address coherency. This was selected to enable backward compatible with the Intel® IXP42X and IXP46X Network Processors.

The BYTE_SWAP_EN bit is an enable bit that enables data coherency to be performed, based on the P-attribute bit.
When the bit is 0, address coherency is always performed.
When the bit is 1, the type of coherency depends on the P-attribute bit.

The P-attribute bit is associated with each 1-Mbyte page. The P-attribute bit is output from the Intel XScale processor, with any store or load access associated with that page.

Under software control, bits 0 and 1 allow interrupts to be generated to the Interrupt Controller.

When byte swapping is enabled, byte 3 of a a word is swapped with byte 0, and byte 1 is swapped with byte 2.

12.5.7 EXP_UNIT_FUSE_RESET

The EXP_UNIT_FUSE_RESET register is read by software to determine the features that are fused out or being software disabled. Software can choose to disable features for blocks that are not being used to lower power consumption by writing **1** to the appropriate bit for the feature that must be disabled. Disabling features by software is recommended to be done during the system initialization phase (that is, before the start of use of the related feature to be disabled). A feature cannot be enabled after being disabled without asserting system reset. Any block is independently disabled by software, but the HDLC and HSS must always be both enabled or both disabled by software since these blocks work together. For blocks that have external pin interfaces, pull-ups or pull-downs are enabled in the PAD I/O, and these pins should be left unconnected in the system.

Register Name:	EXP_UNIT_FUSE_RESET		
Physical Address:	0xC4000028	Reset Hex Value:	0xXXXXXXXX
Register Description:	Specifies the value of the fuse register.		
Access: See below.			

31							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
(Reserved)								SPEED		(Reserved)		NPE-A ETHERNET	USB HOST	UTOPIA PHY		ECC	PCI	NPE-C	(Reserved)	NPE-A	NPE-C ETHERNET	(Reserved)	UTOPIA	HSS	(Reserved)	HDLC	DES	AES	HASHING COP	(Reserved)

Register	EXP_UNIT_FUSE_RESET		
Bits	Name	Description	Access
31:24	(Reserved)	(Reserved)	RO
23:22	SPEED	Intel XScale processor FACTORY SPEED 00 = 533 MHz 01 = 400 MHz 10 = 667 MHz 11 = Reserved	RO
21:20	(Reserved)	(Reserved)	RW
19	NPE-A ETHERNET	0 = NPE-A Ethernet Enabled if UTOPIA=1 1 = NPE-A Ethernet Disabled Refer to Table 218 for more details.	RW
18	USB HOST	0 = USB HOST Enabled 1 = USB HOST Disabled and clock gated	RW
17:16	UTOPIA PHY	00 = Maximum of 31 UTOPIA Phys 01 = Maximum of 16 UTOPIA Phys 10 = Maximum of 8 UTOPIA Phys 11 = Maximum of 4 UTOPIA Phys	RW
15	ECC	0 = DDR ECC Enabled 1 = DDR ECC Disabled	RW
14	PCI	0 = PCI Enabled 1 = PCI Disabled and clock gated	RW
13	NPE-C	0 = NPE-C Enabled 1 = NPE-C Disabled and clock gated/Reset	RW
12	(Reserved)	(Reserved)	RW
11	NPE-A	0 = NPE-A Enabled 1 = NPE-A Disabled and clock gated/Reset	RW
10	NPE-C ETHERNET	0 = NPE-C Ethernet Enabled 1 = NPE-C Ethernet Disabled	RW
9	(Reserved)	(Reserved)	RW
8	UTOPIA	0 = UTOPIA Enabled 1 = UTOPIA Disabled/NPE-A Ethernet enabled if NPE-A ETHERNET=0 Refer to Table 218 for more details.	RW

Register		EXP_UNIT_FUSE_RESET	
Bits	**Name**	**Description**	**Access**
7	HSS	0 = HSS Enabled 1 = HSS Disabled and clock gated	RW
6	(Reserved)	(Reserved)	RW
5	HDLC	0 = HDLC Enabled 1 = HDLC Disabled and clock gated	RW
4	DES	0 = DES Enabled 1 = DES Disabled	RW
3	AES	0 = AES Enabled 1 = AES Disabled	RW
2	Hashing Cop	0 = HASH Enabled 1 = HASH Disabled	RW
1:0	(Reserved)	(Reserved)	RW

Each NPE can also be reset anytime through software by writing **1** to the appropriate NPE bit (NPE-A or NPE-C) in the EXP_UNIT_FUSE_RESET register. When a NPE is reset, all the coprocessors that belong to that NPE are also automatically reset and all the coprocessor data transfers to external interfaces are terminated immediately. While resetting the NPEs, software should do a read modify write to the EXP_UNIT_FUSE_RESET register and only set the NPE bit that has to be reset.

A read modify write must ensure that the NPE coprocessors that were previously disabled continue to be disabled and the coprocessors that are enabled continue to be enabled. Writing **1** to a NPE coprocessor bit that is already enabled when resetting the NPEs disables that coprocessor and cause the coprocessor interface pins to enter an unknown state. When software writes a **1** to reset a NPE, the EXP_UNIT_FUSE_RESET register does not reflect the reset status until that NPE has been reset. Typically software polls the EXP_UNIT_FUSE_RESET register until that NPE bit that is being reset equals **1**. Once software detects that the NPE has been reset, software then can clear the appropriate bits of the EXP_UNIT_FUSE_RESET register to deassert reset to that NPE.

When the software does a NPE reset, the coprocessor interfaces that are attached to that NPE could enter reset at any time, even during the middle of a data transfer. The system design must account for this abrupt reset on these busses. After being reset, the coprocessor interfaces goes back to the default reset mode for that interface.

The Intel XScale processor factory speed cannot be changed by software.

Table 218. UTOPIA/Ethernet Configuration Options

UTOPIA Fuse	NPE-A Ethernet Fuse	Operation
0	0	UTOPIA interface enabled on NPE-A. NPE-A ethernet interface is disabled.
0	1	UTOPIA interface enabled on NPE-A. NPE-A ethernet interface is disabled.
1	0	UTOPIA interface is disabled. NPE-A ethernet interface is enabled in MII mode.
1	1	UTOPIA interface is disabled. NPE-A ethernet interface is disabled.

Table 219. NPE-C Ethernet Configuration Options

NPE-C Ethernet Fuse	Operation
0	NPE-C Ethernet is enabled in MII mode.
1	NPE-C Ethernet is disabled.

12.5.8 EXP_SYNCINTEL_COUNT

This register is used to set the read latency count when a Synchronous Intel Device is accessed.

Register Name:	EXP_SYNCINTEL_COUNT		
Physical Address:	0xC4000124	Reset Hex Value:	0x00000000
Register Description:	This register is used to set the read latency count when a Synchronous Intel Device is accessed.		

Access: See below.

31																								4	3			0
(Reserved)																									Count			

Register		EXP_SYNCINTEL_COUNT		
Bits	Name	Description	Reset Value	Access
31:4	Reserved	Reserved	0x0	RO
3:0	Count	The count bits tell the Expansion Bus Controller how many clock cycles must elapse before the first data word is sampled from a synchronous Intel StrataFlash memory. The value that has to be programmed is dependent on the Expansion bus clock frequency and must be set to the same value as the latency count bits defined in the Read Configuration Register in the Synchronous Intel StrataFlash Memory. Refer to the synchronous Intel StrataFlash memory datasheet for more information. 0000: Reserved 0001: Reserved 0010: Code 2 0011: Code 3 0100: Code 4 0101: Code 5 0110: Code 6 0111: Code 7 1000: Code 8 1001: Code 9 1010: Code 10 1011 - 1111: Reserved	0x0	RW

12.5.9 EXP_USBAFE_DBGCTRL1

Register Name:	EXP_USBAFE_DBGCTRL1		
Physical Address:	0xC4000230	Reset Hex Value:	0x170F0035
Register Description:	USB AFE Debug Control Register 1		

Access: Read/Write

31	...	7	6	5	...	0
(Reserved)			exp_ousblowpwrenb	(Reserved)		

Register		EXP_USBAFE_DBGCTRL1	
Bits	**Name**	**Description**	**Access**
31:7	Reserved	Reserved	RO
6	exp_ousblowpwrenb	This bit is set to '0' by default. This bit must be set to '1' before USBH enters Full Speed transmit mode for the first time, in order to reduce noise sensitivity.	RW
5:0	Reserved	Reserved	RO

§ §

13.0 Universal Asynchronous Receiver-Transmitter (UART)

13.1 Overview

The Intel® IXP43X Product Line of Network Processors supports one Universal Asynchronous Receiver-Transmitter (UART) interface. The UART is intended primarily to support connection to high-speed serial devices such as Bluetooth* or to support debug and control functions.

Each UART supports four pins:

- Transmit Data
- Receive Data
- Request to Send
- Clear to Send

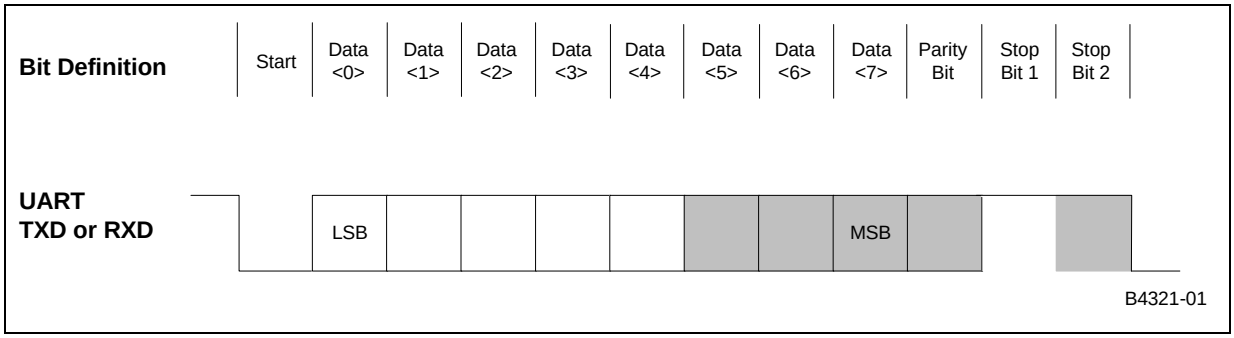
The UART performs serial-to-parallel conversion on data characters received from a peripheral device or a modem, and parallel-to-serial conversion on data characters received from Intel XScale® Processor.

The Intel XScale processor, within the IXP43X network processors reads the complete status of the UART at any time during functional operation. Available status information includes the type and condition of the transfer operations being performed by the UART, and any detected error conditions (such as parity, overrun, framing, or break interrupt).

The UART has enhancements in place to support higher speeds than defined by the 16550 UART specification. The UART is capable of supporting data transfers containing five, six, seven, or eight data bits. The data transfers may be configured to have one or two stop bits and supports even, odd, or no parity.

Figure 154 shows a functional waveform of the data that could be contained on the UART transmit and receive lines. Notice that Data bits 5 through 7, the Parity Bit, and Stop Bit 2 are shaded. The Data bits 5 through 7, Parity bit, and Stop bit 2 are all programmable and optional as previously described.

Figure 154. UART Timing Diagram



The serial port operates in either FIFO or non-FIFO mode. In FIFO mode, a 64-byte Transmit FIFO holds data coming from the Intel XScale processor to be transmitted on the serial link, and the 64-byte Receive FIFO, buffers data received from the serial link until the data is read by the Intel XScale processor. In non-FIFO mode, data is transmitted and received using two registers - the Transmit-Holding Register and the Receive-Buffer Register, along with the UART control, status and interrupt registers.

The UART includes a programmable baud rate generator capable of dividing the 14.7456-MHz, UART input clock by divisors of 1 to (2^{16} –1) and produces a 16X clock to drive the internal transmitter and receiver logic. The 14.7456-MHz, UART input clock is generated internally to the IXP43X network processors.

Interrupts are programmed to the requirements of the user thus minimizing the computing required to handle the communications link. The UART is operated in a polled or an interrupt driven environment as selected by software.

The maximum baud rate supported by the UART is 921.6 Kbps. The divisors programmed in divisor latch registers should be equal to or greater than 1 for proper operation. The device UART may be initialized by setting 13 configuration registers.

13.2 Feature List

The features of the UART interface are:

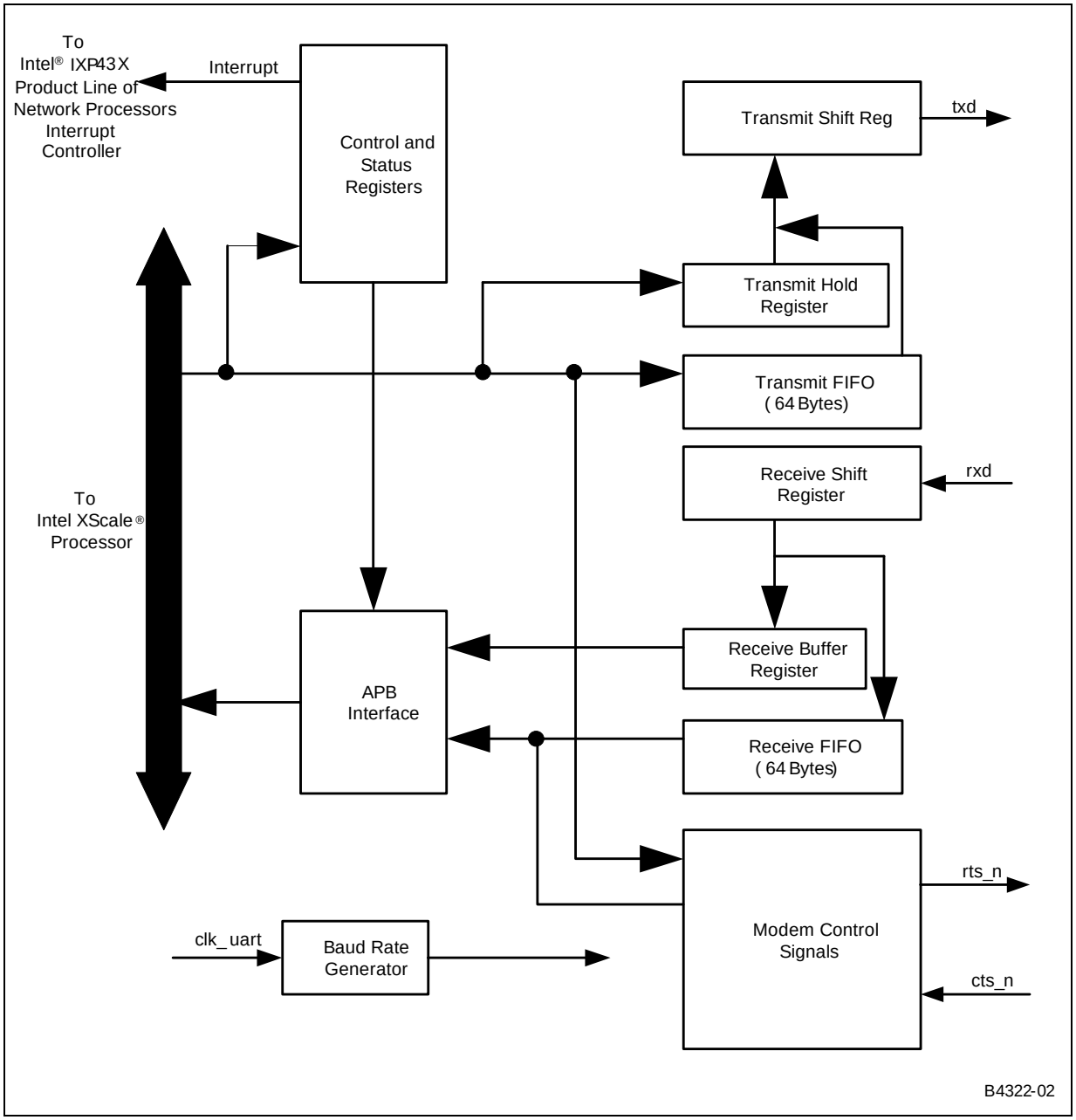
- Microprocessor Interface Control via Advanced Peripheral Bus (APB)
- Adds or deletes standard asynchronous communications bits (start, stop, and parity) to or from the serial data
- Independently controlled transmit, receive, line status and data set interrupts
- Programmable baud rate generator allows division of clock by 1 to (2^{16} –1) and generates an internal 16X clock
- Modem control functions (cts_n and rts_n)
- Fully programmable serial-interface characteristics:
- 5, 6, 7, or 8-bit characters
- Even, odd, or no parity detection
- 1, 1-1/2, or 2 stop bit generation
- Baud rate generation (up to 921kbps)
- False start bit detection
- 64-byte Transmit FIFO
- 64-byte Receive FIFO

- Complete status reporting capability
- Line break generation and detection
- Internal diagnostic capabilities include:
- Loopback controls for communications link fault isolation
- Break, parity, overrun, and framing error simulation
- Fully prioritized interrupt system controls
- Two Data Service requests for transmit and receive data service

13.3 Block Diagram

Figure 155 shows a functional block diagram of the UART interface.

Figure 155. UART Block Diagram



13.4 Theory of Operation

The following sections provide a detailed description of configuring the UART interface for operation.

13.4.1 Setting the Baud Rate

The UART contains a programmable baud-rate generator that is capable of taking the 14.7456 MHz, input clock (clk_uart) and dividing it by any divisor ranging from 1 to (2^{16} –1). The output frequency of the baud-rate generator is 16 times the baud rate. So, if a 1,200 Baud rate was required, the output frequency of the baud-rate generator would be 1,200 KHz *16 = 19,200 KHz.

The two 8-bit registers that store the divisor in a 16-bit binary format are the Divisor Latch Low Register (DLL) and the Divisor Latch High Register (DLH). The Divisor Latch Low register makes up the lower eight bits of the 16-bit divisor and the Divisor Latch High register makes up the upper eight bits of the 16-bit divisor.

The two Divisor Latch registers is loaded during initialization to ensure proper operation of the baud-rate generator. If both Divisor Latches are loaded with 0, the 16X output clock is stopped. A Divisor value of 0 in the Divisor Latch Low Register is not allowed.

The reset value of the divisor is hexadecimal 0x0002. The value of hexadecimal 0x0002 implies a value of hexadecimal 0x00 in the Divisor Latch High Register and a value of hexadecimal 0x02 in the Divisor Latch Low Register. The Divisor Latch High Register and Divisor Latch Low Register is only written after the DLAB bit (bit 7 of the Serial Line Control Register) is set to logic 1.

The baud rate of the UART transmit and receive data is given by:

Baud Rate = 14.7456 MHz/(16xDivisor)

Table 220 shows some commonly used baud rates.

Table 220. Typical Baud Rate Settings

Divisor Latch High Register	Divisor Latch Low Register	Divisor		Baud Rate Generator Clock Output	Baud Rate
		Hexadecimal	Decimal		
0x00	0x01	0x0001	1	14.7456 MHz	921,600
0x00	0x02	0x0002	2	7.3728 MHz	460,800
0x00	0x04	0x0004	4	3.6864 MHz	230,400
0x00	0x08	0x0008	8	1.8432 MHz	115,200
0x00	0x10	0x0010	16	921.6 KHz	57,600
0x00	0x20	0x0020	32	460.8 KHz	28,800
0x00	0x30	0x0030	48	307.2 KHz	19,200
0x00	0x40	0x0040	64	230.4 KHz	14,400
0x00	0x60	0x0060	96	153.6 KHz	9,600
0x00	0xC0	0x00C0	192	76.8 KHz	4,800
0x01	0x80	0x0180	384	38.4 KHz	2,400
0x03	0x00	0x0300	768	19.2 KHz	1,200

13.4.2 Setting Data Bits/Stop Bits/Parity

The Line Control Register (LCR) is an 8-bit register that enables the system programmer to specify the format of the asynchronous data communications exchange. The serial data format consists of a start bit (logic 0), five to eight data bits, an optional

parity bit and one or two stop bits (logic 1). The Line Control Register also contains a bit used for accessing the Divisor Latch Registers and causing a UART break condition. The programmer also has the ability to read the contents of the Line Control Register.

The 8-bit Line Control Register is broken up into seven smaller registers

- Divisor-Latch Access Bit
- Even-Parity Select Bit
- Word-Length Select Bits
- Set-Break Bit
- Parity-Enable Bit
- Sticky-Parity Bit
- Stop-Bits Bit

The Divisor-Latch Access bit is set to logic 1 to access the Divisor Latch Registers of the baud-rate generator. When the Divisor-Latch Access bit is set to logic 0, accessing the same addresses as defined for accessing the Divisor Latch Registers allows access to the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.

The Set-Break Control bit causes a UART break condition to be transmitted to the receiving UART. When Set-Break Control bit is set to logic 1, the serial output data (TXD) is forced to the spacing or logic 0 state and remains in the spacing state until Set-Break Control bit is set to logic 0. When the Set-Break Control bit is set to logic 0, the serial output data is transmitted normally from the Transmit Holding Register. The Set-Break feature enables the processor to alert a terminal in a computer communications system.

The Sticky-Parity bit (STKYP) is used to transmit the complement of the Even-Parity Select when parity is enabled. When the Parity-Enable bit and Sticky-Parity bit are logic 1, the bit that is transmitted, in the parity bit location of the serial output data stream, is the complement of the Even-Parity Select bit.

For example, the Even-Parity Select bit is logic 0 and the parity-bit location of the serial output data stream is transmitted as logic 1. When the Sticky-Parity bit and Parity-Enable bits are logic 1, the receiver logic compares the parity bit that is received, in the parity bit location of the serial data input stream, with the complement of the Even-Parity Select bit.

If the values being compared are not equal, the receiver sets the Parity-Error bit in line to the Status Register and causes an interrupt error if Line-Status interrupts are enabled.

If the Even-Parity Select bit is logic 0, the receiver expects the bit received at the parity-bit location of the serial data input stream to be logic 1. If the parity bit received is logic 0, the parity-error bit is set to logic 1. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers identifies some transmitted characters as receiver addresses and the rest of the characters as UART data.

If Parity enable is set to logic 0, Sticky Parity is ignored.

The Even-Parity Select (EPS) bit is used to determine the parity type to transmit or check on receive data when the Parity-Enable (PEN) bit in the Line-Control Register enables parity. When the Parity-Enable bit is logic 1 and the Even-Parity Select bit is logic 0, the parity generator transmits odd parity and the parity checker validates against odd parity on the received data.

When the Parity-Enable bit is logic 1 and the Even-Parity Select bit is logic 1, the parity generator transmits even parity and the parity checker validates against even parity on the received data.

Table 221 shows the serial output data configurations for transmission of the parity bit.

Table 221. UART Transmit Parity Operation

PEN	EPS	Data to be Transmitted (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit to be Transmitted
1	0	10101010	1
1	0	10101011	0
1	1	10101010	0
1	1	10101011	1
0	X	XXXXXXXX	No Parity Bit Sent

Table 222 shows the serial input data configurations for reception of data containing a parity bit.

Table 222. UART Receive Parity Operation

PEN	EPS	Data being Received + Parity Bit (Even or Odd Count of 1s to be Transmitted)	Value of Parity Bit Should Be
1	0	101010101	1
1	0	101010110	0
1	1	101010100	0
1	1	101010111	1
0	X	XXXXXXXX	Parity Checking Disabled

The Stop-Bits bit (STB) configures the number of stop bits to be transmitted or received in each serial character. When the Stop-Bits bit is logic 0, one stop bit is generated in the transmitted data. When the Stop-Bits bit is logic 1, and a 5-bit word length is selected via the Word Length select bits in the Line Control Register, 1.5 stop bits is transmitted.

When the Stop-Bits bit is logic 1 and the word length is selected as a 6, 7, or 8-bit word, 2 stop bits are transmitted. The UART receiver logic checks the first stop bit only, regardless of the number of stop bits configured by the Stop Bits bit.

The Word-Length Select (WLS) bits specify the number of data bits contained in each transmitted or received serial character. The Word-Length Select bits configuration is shown in Table 223.

Table 223. UART Word Length Select Configuration

WLS		Number of Data Bits Contained in Each Transmitted Or Received Character
Bit 1	Bit 0	
0	0	5-bit character (default)
0	1	6-bit character
1	0	7-bit character
1	1	8-bit character

The Line-Control Register is initialized to hexadecimal 0x00 after reset. The Line-Status Register is initialized to hexadecimal 0x60 after reset.

13.4.3 Using Modem Control Signals

The IXP43X network processors provide the following two modem control signals:

- Clear-to-Send input (CTS_N)
- Ready to Send output (RTS_N)

The Clear-to-Send input signal is sampled by reading the Modem Status Register and the Ready-to-Send output is controlled by the Modem Control Register. The Modem Control Register is a 5-bit register that provides control for four modem control signals:

- OUT1
- OUT2
- Data Terminal Ready
- Ready-to-Send
- Loop-back test control bit

Only one of the four modem-control signals (Ready-to-Send) is replicated to the external pins of the IXP43X network processors. The three other modem-control signals are utilized only when the UART is being used in loopback diagnostic mode. Bit 4 of the Modem-Control Register is the Loop Back Test Mode bit (LOOP). The Loop Back Test Mode bit provides a local loopback feature for diagnostic testing of the UART.

When the Loop Back Test Mode bit is set to logic 1, the following event occurs:

- The transmitter serial output is set to logic-1 state.
- The receiver serial input is disconnected from the pin.
- The output of the Transmitter Shift Register is **looped back** into the Receiver Shift Register input.
- The Clear-to-Send input signal is disconnected from the pin
- The Ready-to-Send output signal is forced to logic 1.
- The lower four bits of the Modem Control Register are connected to the upper four bits of the Modem Status Register

  Status register bit mapping, when in loop back mode:

  — DTR = 1 forces DSR to a 1

  — RTS = 1 forces CTS to a 1

  — OUT1 = 1 forces RI to a 1

  — OUT2= 1 forces DCD to a 1

Leaving loop-back mode and returning to normal mode may result in unpredictable activation of the Modem Status Register (MSR). It is recommended that the Modem Status Register be read once to clear the Modem Status bits in the Modem Status Register.

In the loopback diagnostic mode, data that is transmitted is immediately received. The ability to loopback transmit data path to the receive path allows the Intel XScale processor to verify the transmit data path and receive data path of the UART. The transmit interrupts, receive interrupts, and modem-control interrupts are operational, when placed in the loopback diagnostic mode. The Modem-Control Register bits, instead of the Modem Control inputs, activates the modem-control interrupts. A break signal is also transferred, from the transmitter section to the receiver section, when operating in loopback diagnostic mode.

Bit 1 of the Modem-Control Register is the Request-to-Send bit. The Request-to-Send control bit is used to program the RTS_N output pin. When the Request-to-Send control bit is programmed to logic 0, the RTS_N signal outputs logic 1. When the Request-to-Send control bit is programmed to logic 1, the RTS_N signal outputs logic 0.

The Modem-Status Register is used to monitor the status of an external modem or data set. The Modem Status Register is an 8-bit register that is used to detect when a modem is capable of accepting new data. The status of the modem to accept incoming data is monitored by reading the Clear-to-Send bit of the Modem Status Register and the Delta Clear-to-Send bit of the Modem Status Register. The other six bits of the Modem-Status Register is used for UART debug purposes only.

Modem Status Register bit 4 is the Clear-to-Send (CTS). The Clear-to-Send bit is the complement of the Clear-to-Send (CTS_N) input signal. The Clear-to-Send signal is connected to the Ready-to-Send bit of the Modem Control Register when the LOOP bit, in the Modem Control Register is set to logic 1.

- CTS = logic 0 = CTS_N pin is 1
- CTS = logic 1 = CTS_N pin is 0

Modem Status Register bit 0 is the Delta Clear-to-Send (DCTS). The Delta Clear-to-Send bit informs the IXP43X network processors that nothing has happened to the Clear-to-Send Status since the last time that the Modem-Status Register was read. The Delta Clear-to-Send bit is set to logic 0 after a read of the Modem-Status Register.

- DCTS = logic 0 = No change in CTS_N pin since last read of the Modem-Status Register
- DCTS = logic 1 = CTS_N pin has changed state since the last read of the Modem-Status Register

The Modem-Control Register is initialized to hexadecimal 0x00 after reset. The Modem-Status Register is initialized to hexadecimal 0x00 after reset.

13.4.4 UART Interrupts

The UART Interrupt Enable Register (IER) is an 8-bit register that enables five types of UART based interrupts, enables UART functionality and other control functionality that are not used by the IXP43X network processors.

UART Interrupt Enable Register bit 6 is the UART Unit Enable (UUE) bit. When the UART Unit Enable bit is set to logic 0, the UART is completely non-functional. Likewise, when the UART Unit Enable bit is set to logic 1, the UART is enabled.

UART Interrupt Enable Register bits 4 through 0 represents five various interrupt types that is individually enabled/disabled:

- Receiver Time Out Interrupt Enable (RTOIE)
- Modem Interrupt Enable (MIE)
- Receiver Line Status Interrupt Enable (RLSE)
- Transmit Data Request Interrupt Enable (TIE)
- Receiver Data Available Interrupt Enable (RAVIE)

The Receiver Line Status Interrupt Enable allows interrupts to be generated to the Interrupt Controller of the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR) when a receive error is detected. Such Receiver Line Status Conditions that will cause the interrupt to occur are:

- Overrun
- Break
- Parity
- FIFO error
- Framing

In FIFO mode, each received character carries the line status along with the character. When in FIFO Mode, the Receive Line Status Interrupt is active on the character located in the bottom of the Receive FIFO. If a Line Status error condition is detected on the character in the bottom of the Receive FIFO, a Receive Line Status Interrupt is generated. Reading the Line Status Register clears the Receive Line Status Interrupt.

When in non-FIFO Mode, the Receive Line Status Interrupt is active on the character located in the bottom of the Receive Buffer Register. If a line-status error condition is detected on the character in the Receive Buffer Register, a Receive Line Status Interrupt is generated.

The Receiver Data Available Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR) when UART data is available to be read by the Intel XScale processor.

When the UART is in FIFO mode, the Receive Data Available interrupt is encoded in the Interrupt Identification Register, after the FIFO trigger level defined in the FIFO Control Register (FCR) is reached. When the UART is in non-FIFO mode, the Receive Data Available interrupt is encoded in the Interrupt Identification Register after data is in the Receive Buffer Register (RBR).

When operating in FIFO Mode, the Receive Data Available Interrupt is cleared, when the FIFO drops below the programmed trigger level. When operating in non-FIFO Mode, the Receive Data Available Interrupt is cleared when the received character is read by the IXP43X network processors from the Receive Holding Register.

The Receiver Interrupt Time Out Enable is used only in FIFO Mode and allows interrupts to be generated to the Interrupt Controller for the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR). This happens when:

- At least one character is available in the Receive FIFO
- The last character received by the UART receive interface occurred more than four continuous character times ago
- The most recent read of the Receive FIFO by the IXP43X network processors was more than four continuous character times ago

For example, the maximum time between a received character and a Receive Character Time-Out Interrupt is 160 ms at 1,200 baud with a 12-bit receive character, that is, 1 start, 8 data, 1 parity and 2 stop bits.

```
(1/ ((1200characters/second)/12characters))) * 4
characters =

= 0.040 seconds
 = 40 ms
```

The time-out interrupt is cleared by the IXP43X network processors by reading the Receive FIFO or setting the RESETRF bit to logic 1, in the FIFO Control Register.

If a Receive Character Time-out Interrupt is active, the interrupt-time-out counter is reset only after the IXP43X network processors read a character from the Receive FIFO. If a Receive Character Time-Out Interrupt is non-active, the interrupt time-out counter is reset after the IXP43X network processors read a character from the Receive FIFO or when a new character is placed into the Receive FIFO by the receive interface.

The Transmit Data Request Interrupt Enable is used in non-FIFO mode or FIFO Mode.

In non-FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR), when the Transmit Holding Register is empty. Reading the Interrupt Identification Register or writing a new character into the Transmit Holding Register clears the Transmit Data Request Interrupt.

In FIFO Mode, the Transmit Data Request Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR), when the Transmit FIFO is half empty or less.

The Transmit FIFO size is 64 characters. When 32 or fewer characters are remaining in the Transmit FIFO to be transmitted, the Transmit Data Request Interrupt is generated. Reading the Interrupt Identification Register or writing a new data into the Transmit FIFO, clears the Transmit Data Request Interrupt.

The Modem Status Interrupt Enable allows interrupts to be generated to the Interrupt Controller for the IXP43X network processors and captures in the UART Interrupt Identification Register (IIR), when the Clear-to-Send, Data-Set-Ready, Ring-Indicator, or Received-Line Signal Detect bits in the Modem Status Register are set to logic 1. Reading the Modem Status Register clears the Modem Status Interrupt.

Clearing the appropriate bit of the Interrupt Enable Register may disable each of the interrupt types previously described. Similarly, by setting the appropriate bits, selected interrupts is enabled on a per-interrupt basis.

When the UART interrupts are disabled, the UART is placed into polled mode of operation. Since the UART receiver and the UART transmitter are controlled separately, either one or both interfaces is placed in the polled mode of operation.

In the polled mode of operation, software routines running on the Intel XScale processor checks receiver and transmitter status via the Line Status Register. Line Status Register bit 0 is logic 1, when a character is available to be read from the Receive Interface. Lines Status Register bits 1 through 4 specify the error(s) that have occurred, for the character at the bottom of the FIFO or in the Receive Buffer Register.

In FIFO mode, Line Status Register bit 1 through 3 is stored with each received character in the Receive FIFO. The Line Status Register shows the status bits of the character at the bottom of the Receive FIFO. When the character at the bottom of the FIFO has errors, the Line Status error bits are set and are not cleared until the Intel XScale processor reads the Line Status Register. Even if the character in the FIFO is read and a new character is now at the bottom of the FIFO, the interrupts is not cleared until the Line-Status Register is read.

Character-error status is handled in the same way as when the UART is operating in interrupt mode of operation. Setting Line-Status Register bit 5 to logic 1 indicates that the Transmit FIFO or the Transmit Holding Register is requesting data. Line Status Register bit 6 identifies that both the Transmit FIFO and the Transmit Shift Register have no data. Line Status Register bit 7 indicates the status of any errors in the Receive FIFO.

In non-FIFO mode, three of the LSR register bits, parity error, framing error, and break interrupt, show the error status of the character that has just been received.

The Receive Time-Out Interrupt is separated from the Receive-Data-Available Interrupt to prevent an Interrupt Controller routine and a Data Service controller routing from servicing the Receive FIFO at the same time. Bit 7 of the Interrupt Enable Register is used as the enable bit of Data Service requests. Bit 5 of the Interrupt Enable Register is used as the enable bit of NRZ-coding enable.

Bits 7 and 5 are not implemented by the IXP43X network processors. The use of bit 7 through bit 4, of the Interrupt Enable Register, is defined differently from the register definition of standard 16550 UART.

The Interrupt Enable Register is initialized to all zeros after receiving a reset. The Interrupt Identification Register is a hexadecimal 0x01. Bit 5 and bit 4 of the Interrupt Identification Register is always logic 0.

13.4.5 Transmitting and Receiving UART Data

Transmitting and receiving data using UART for the IXP43X network processors is accomplished in the following two modes:

- Non-FIFO
- FIFO

In non-FIFO mode, data is transmitted and received using two registers, the Transmit-Holding Register (THR) and the Receive-Buffer Register (RBR), along with the UART control, status, and interrupt registers.

In FIFO mode, data is transmitted and received using two 64-entry FIFOs, the Transmit FIFO and the Receive FIFO, along with the UART Control, Status, and Interrupt registers.

The Transmit FIFO is 64 entries deep by 8 bits wide. The Transmit FIFO sizing allows a complete 8-bit data character to be stored in each entry. When characters smaller than 8 bits are transmitted, they are right-justified.

If a 5-bit character is to be transmitted, the character is represented by a binary 10110. The value located in the FIFO entry is hexadecimal 0x16.

The Receive FIFO is 64 entries deep and 11 bits wide. The Receive FIFO sizing allows for an 8-bit character to be received along with the over-run flag, parity error flag, and framing error flag for each received character. Smaller characters is right-justified, as described for the Transmit FIFO.

The error flags position remains constant, independent of the character size. The mode of operation and FIFO control parameters is programmed using the FIFO Control Register (FCR).

The FIFO Control Register is an 8-bit register that configures the mode of operation of the UART. The Transmit and Receive FIFO Enable bit (TRFIFOE), bit 0 of the FIFO Control Register, determines the mode of operation of UART: FIFO Mode or non-FIFO Mode. When set to logic 0, the UART functions in non-FIFO Mode. When set to logic 1, the UART functions in FIFO Mode.

Two bits of the FIFO Control Register are used to reset the Transmit and Receive FIFO: the Reset Transmit FIFO bit (bit 2 of FCR) and the Reset Receive FIFO (bit 1 of FCR). Writing logic 0 to these bits has no effect. Writing logic 1 to the Reset Transmit FIFO bit causes the Transmit FIFO counter to be reset to 0 and the transmit-data request bit to be set in the Line-Status Register.

Writing logic 1 to the Reset Receive FIFO bit causes the Receive FIFO counter to be reset to 0 and the data ready bit in the Line-Status Register to be cleared. The Overrun Error Flag, Parity Error Flag, Framing Error Flag, and Break Interrupt Flag in the Line Status Register remains unaltered. The Reset Transmit FIFO and Reset Receive FIFO is cleared autonomously when the reset has been completed.

The Receive FIFO interrupt trigger level also is set in the FIFO Control Register. The Receive FIFO interrupt trigger level is used to generate an interrupt when the number of characters in the Receive FIFO is greater than to or equal to the trigger-level value. The Interrupt Trigger Level is defined as bit 6 and bit 7 of the FIFO Control Register. The bit definitions are shown in Table 224.

Table 224. UART FIFO Trigger Level

Interrupt Trigger Level [7:6]	Description
00	1 byte or more in the FIFO causes an interrupt
01	8 bytes or more in the FIFO causes an interrupt
10	16 bytes or more in the FIFO causes an interrupt
11	32 bytes or more in the FIFO causes an interrupt

The UART is configured prior to transmitting and receiving data to and from the UART. The Transmit Holding Register (THR) is used to transmit characters over the UART interface. The Receive Buffer Register is used to receive characters from the UART interface.

Transmitting UART data is implemented using FIFO Mode or non-FIFO mode. In FIFO mode, writing a character to the Transmit Holding Register puts data on the top of the Transmit FIFO. In non-FIFO mode, writing a character to the Transmit Holding Register puts data in the Transmit Holding Register. The next character transmitted is the character contained in the Transmit Holding Register.

If characters less than 8 bits are sent, the characters must be right-justified. For example, if a 5-bit data character is to be transmitted with a binary value of 01011. The data written to the Transmit Holding Register must be written as hexadecimal 0x0B.

Receiving UART data is implemented using FIFO Mode or non-FIFO mode. In FIFO mode, reading a character from the Receive Buffer Register reads a character from the bottom of the Receive FIFO. In non-FIFO mode, reading a character from the Receive Buffer Register reads the data contained in the Receive Buffer Register. The next character received is the character contained in the Receive Buffer Register.

If characters less than 8 bits are received, the characters must be right-justified. For example, if a 5-bit data character is received having a binary value of 00101. The data read from the Receive Buffer Register is a hexadecimal 0x05. (Notice that the three most-significant bits of the byte are filled with zeros.)

The UART transmit data pin is logic 1 and the Transmit FIFO and Receive FIFO pointers are initialized to the empty value after reset.

13.5 Register Descriptions

There are 13 registers to monitor and control the UART. The registers are all 32 bits in size, but only the lower 8 bits have valid data. The 13 UART registers share nine address locations in the I/O address space.

Note: The state of the Divisor Latch bit (DLAB), the most-significant bit of the Serial Line Control Register, affects the selection of certain of the UART registers. The DLAB bit is set high by the system software to access the Baud Rate Generator Divisor Latches.

The following sample register-summary table indicates, in parentheses, the paragraph tags that are cross-referenced in the individual register tables.

Table 225. UART Registers Overview

Address	DLAB	R/W	Name	Description
0xC800_X000†	0	R	RBR	Receive Buffer Register
	0	W	THR	Transmit Holding Register
	1	R/W	DLL	Divisor Latch Low Register
0x C800_X004	0	R/W	IER	Interrupt Enable Register
	1	R/W	DLH	Divisor Latch High Register
0x C800_X008	0/1	R	IIR	Interrupt Identification Register
	0/1	W	FCR	FIFO Control Register
0x C800_X00C	0/1	R/W	LCR	Line Control Register
0x C800_X010	0/1	R/W	MCR	Modem Control Register
0x C800_X014	0/1	R	LSR	Line Status Register
0x C800_X018	0/1	R	MSR	Modem Status Register
0x C800_X01C	0/1	R/W	SPR	Scratch-Pad Register
0x C800_X020	0	R/W	ISR	Slow Infrared Select Register

† The X in the value C800_X000 is used to denote that this could be a value of either 0 or 1. **For the IXP43X network processors, this value is 0**.

13.5.1 Receive Buffer Register

The following table provides details of the Receive Buffer Register:

Register Name:	RBR				
Hex Offset Address:	0xC800 X000		Reset Hex Value:	0x00000000	
Register Description:	Receive Buffer Register				
Access: Read Only.					

31																					8	7							0
(Reserved)																						RBR							

Register		RBR
Bits	Name	Description
31:8		(Reserved)
7:0	RBR	In non-FIFO mode, this register holds the character received by the UART's Receive Shift Register. If fewer than 8 bits are received, the bits are right-justified and the leading bits are zeroed. In FIFO mode, this register latches the value of the data byte at the bottom of the Receive FIFO. The DLAB bit in the Line Control Register is set to logic 0 to access this register.

13.5.2 Transmit Holding Register

The following tables provide details of the Transmit Holding Register:

Register Name:	THR				
Hex Offset Address:	0xC800 X000		Reset Hex Value:	0x00000000	
Register Description:	Transmit Holding Register				
Access: Write Only.					

31																				8	7							0
(Reserved)																					THR							

Register		THR
Bits	Name	Description
31:8		(Reserved)
7:0	THR	In non-FIFO mode, this register holds the next data byte to be transmitted. When the Transmit Shift Register becomes empty, the contents of the Transmit Holding Register are loaded into the shift register and the transmit data request (TDRQ) bit in the Line Status Register is set to 1. In FIFO mode, writing to THR puts data to the top of the Transmit FIFO. The data at the bottom of the Transmit FIFO is loaded to the shift register when the shift register becomes empty. The DLAB bit in the Line Control Register is set to logic 0 to access this register.

13.5.3 Divisor Latch Low Register

The following table provides details of the Divisor Latch Low Register:

Register Name:	DLL				
Hex Offset Address:	0xC800 X000		Reset Hex Value:	0x00000002	
Register Description:	Divisor Latch Low Register				
Access: Read/Write.					

31																				8	7							0
(Reserved)																					DLL							

Register		DLL
Bits	Name	Description
31:8		(Reserved)
7:0	DLL	Lower byte of compare-value used by the baud rate generator. The DLAB bit in the Line Control Register is set to logic 1 to access this register.

13.5.4 Divisor Latch High Register

The following table provides details of the Divisor Latch High Register:

Register Name:	DLH				
Hex Offset Address:	0xC800 X004		Reset Hex Value:	0x00000000	
Register Description:	Divisor Latch High Register				
Access: Read/Write.					

31																					8	7							0
(Reserved)																						DLH							

Register		DLH
Bits	Name	Description
31:8		(Reserved)
7:0	DLH	Upper byte of compare-value used by the baud-rate generator. The DLAB bit in the Line Control Register is set to logic 1 to access this register.

13.5.5 Interrupt Enable Register

The DLAB bit in the Line-Control Register is set to logic 0 to access this register. DMA is not supported on the IXP43X network processors.

Register Name:	IER				
Hex Offset Address:	0xC800 X004		Reset Hex Value:	0x00000000	
Register Description:	Interrupt Enable Register				
Access: See below.					

31																					8	7	6	5	4	3	2	1	0
(Reserved)																						DMAE	UUE	NRZE	RTOIE	RIE	RLSE	TIE	RAVIE

Register		IER (Sheet 1 of 2)
Bits	Name	Description
31:8		(Reserved)
7	DMAE	**DMA Requests Enable**: 0 = DMA requests are disabled 1 = DMA requests are enabled Not used on the IXP43X network processors
6	UUE	**UART Unit Enable**: 0 = the unit is disabled 1 = the unit is enabled
5	NRZE	**NRZ coding Enable**: 0 = NRZ coding disabled 1 = NRZ coding enabled Not used on the IXP43X network processors
4	RTOIE	Receiver Time Out Interrupt Enable: 0 = Receiver data Time out interrupt disabled 1 = Receiver data Time out interrupt enabled

Register		IER (Sheet 2 of 2)
Bits	**Name**	**Description**
3	RIE	**Modem Interrupt Enable**: 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled
2	RLSE	**Receiver Line Status Interrupt Enable**: 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled The DLAB bit in the Line Control Register is set to logic 0 to access this register.
1	TIE	**Transmit Data request Interrupt Enable**: 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled
0	RAVIE	**Receiver Data Available Interrupt Enable**: 0 = Receiver Data Available (Trigger level reached) interrupt disabled 1 = Receiver Data Available (Trigger level reached) interrupt enabled

13.5.6 Interrupt Identification Register

To minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels and records these in the Interrupt-Identification Register. The Interrupt-Identification Register (IIR) stores information indicating that a prioritized interrupt is pending and the source of that interrupt.

Priority Level	Interrupt origin
1 (highest)	**Receiver Line Status:** One or more error bits were set.
2	**Received Data is available:** In FIFO mode, trigger level was reached. In non-FIFO mode, RBR has data.
2	**Receiver Time out occurred:** It happens in FIFO mode only, when there is data in the Receive FIFO but no activity for a time period.
3	**Transmitter requests data:** In FIFO mode, the Transmit FIFO is half or more than half empty. In non-FIFO mode, THR is read already.
4 (lowest)	**Modem Status:** One or more of the modem input signals has changed state.

Register Name:	IIR				
Hex Offset Address:	0xC800 X008		Reset Hex Value:	0x00000001	
Register Description:	Interrupt Identification Register				
Access: See below.					

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
(Reserved)																								FIFOES		(Rsvd)		TOD	IID		IP_N

Register		IIR
Bits	**Name**	**Description**
31:8		(Reserved)
7:6	FIFOES	FIFO Mode Enable Status: 00 = Non-FIFO mode is selected 01 = (Reserved) 10 = (Reserved) 11 = FIFO mode is selected (TRFIFOE = 1)
5:4		(Reserved)
3	TOD	Time Out Detected: 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)
2:1	IID	Interrupt Source Encoded: 00 = Modem Status (CTS, DSR, RI, DCD modem signals changed state) 01 = Transmit FIFO requests data 10 = Received Data Available 11 = Receive error (Overrun, parity, framing, break, FIFO error)
0	IP_N	Interrupt Pending: 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending

Table 226. UART IDD Bit Mapping

Interrupt ID Bits				Interrupt SET/RESET Function			
3	**2**	**1**	**0**	**Priority**	**Type**	**Source**	**RESET Control**
0	0	0	1	-	None	No Interrupt is pending	-
0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt	Reading the Line Status Register.
0	1	0	0	Second-Highest	Received Data Available.	Non-FIFO mode: Receive Buffer is full. FIFO mode: Trigger level was reached.	Non-FIFO mode: Reading the Receiver Buffer Register. FIFO mode: Reading bytes until Receiver FIFO drops below trigger level or setting RESETRF bit in FCR register.
1	1	0	0	Second-Highest	Character Time-out indication.	FIFO Mode only: At least 1 character is in receiver FIFO and there was no activity for a time period.	Reading the Receiver FIFO or setting RESETRF bit in FCR register.
0	0	1	0	Third-Highest	Transmit FIFO Data Request	Non-FIFO mode: Transmit Holding Register Empty FIFO mode: Transmit FIFO has half or less than half data.	Reading the IIR Register (if the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (if the source of the interrupt) or writing to the Transmitter FIFO.
0	0	0	0	Fourth-Highest	Modem Status	Clear-to-Send, Data Set Ready, Ring Indicator, Received Line Signal Detect	Reading the Modem Status Register.

13.5.7 FIFO Control Register

FCR is a write-only register that is located at the same address as the IIR. (IIR is a read-only register.) FCR enables/disables the transmitter/receiver FIFOs, clears the transmitter/receiver FIFOs, and sets the receiver FIFO trigger level.

Register Name:	FCR				
Hex Offset Address:	0xC800 X008		Reset Hex Value:	0x00000000	
Register Description:	FIFO Control Register				
Access: Write Only.					

31	...	8	7	6	5	...	3	2	1	0
(Reserved)			ITL		(Rsvd)			RESETTF	RESETRF	TRFIFOE

Register		FCR
Bits	Name	Description
31:8		(Reserved)
7:6	ITL	**Interrupt Trigger Level:** When the number of entries in the Receive FIFO equals the interrupt trigger level programmed into this field and the Received Data Available Interrupt is enabled (via IER), an interrupt is generated and appropriate bits are set in the IIR. 00 = 1 byte or more in FIFO causes interrupt 01 = 8 bytes or more in FIFO causes interrupt 10 = 16 bytes or more in FIFO causes interrupt 11 = 32 bytes or more in FIFO causes interrupt
5:3		(Reserved)
2	RESETTF	**Reset Transmitter FIFO:** When RESETTF is logic 1, the transmitted FIFO count logic is set to 0, effectively clearing all the entries in the Transmit FIFO. The TDRQ bit in LSR are set and IIR shows a transmitter requests data interrupt if the TIE bit in the IER register is set. The transmitter shift register is not cleared; it completes the current transmission. After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0.
1	RESETRF	**Reset Receiver FIFO:** When RESETRF is set to 1, the Receive FIFO counter is reset to 0, effectively clearing all the entries in the Receive FIFO. The DR bit in the LSR is reset to 0. All the error bits in the FIFO and the FIFOE bit in LSR are cleared. Any error bits, OE, PE, FE or BI that had been set in LSR are still set. The receiver shift register is not cleared. If IIR had been set to Received Data Available, it is cleared. After the Receive FIFO is cleared, RESETRF is automatically reset to 0. 0 = Writing 0 has no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	TRFIFOE	**Transmit and Receive FIFO Enable:** TRFIFOE enables/disables the transmitter and receiver FIFO s. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFO s are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFO s. 0 = FIFOs are disabled 1 = FIFOs are enabled

13.5.8 Line Control Register

The following table provides details of the Line Control Register:

Register Name:	LCR				
Hex Offset Address:	0xC800 X00C		Reset Hex Value:	0x00000000	
Register Description:	Line Control Register				
Access: Read/Write.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			DLAB	SB	STKYP	EPS	PEN	STB	WLS	

Register		LCR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		(Reserved)
7	DLAB	**Divisor Latch Access Bit:** This bit is set to logic 1 to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It is set low (logic 0) to access the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register.
6	SB	**Set break:** This bit causes a break condition to be transmitted to the receiving UART. When SB is set to logic 1, the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is set to logic 0. This bit acts only on the TXD pin. 0 = No effect on TXD output 1 = Forces TXD output to 0 (space)
5	STKYP	**Sticky Parity:** This bit is the **sticky parity** bit, that is used in multiprocessor communications. When PEN and STKYP are logic 1, the bit that is transmitted in the parity bit location (the bit just before the stop bit) is the complement of the EPS bit. If EPS is 0, then the bit at the parity bit location is transmitted as a 1. In the receiver, if STKYP and PEN are logic 1, the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt if line status interrupts were enabled. For example, if EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, then the parity error bit is set. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers identifies some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, STKYP is ignored. 0 = No effect on parity bit 1 = Forces parity bit to be opposite of EPS bit value
4	EPS	Even-Parity Select: 0 = Sends or checks for odd parity 1 = Sends or checks for even parity

Register		LCR (Sheet 2 of 2)
Bits	**Name**	**Description**
3	PEN	**Parity enable:** This is the parity enable bit. When PEN is logic 1, a parity character is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. 0 = No parity function 1 = Allows parity generation and checking
2	STB	**Stop bits:** This bit specifies the number of stop bits transmitted and received in each serial character. If STB is a logic 0, one stop bit is generated in the transmitted data. If STB is a logic 1 when a 5-bit word length is selected via bits 0 and 1, then 1 and one half stop bits are generated. If STB is a logic 1 when either a 6, 7, or 8-bit word is selected, then two stop bits are generated. The receiver checks the first stop bit only, regardless of the number of stop bits selected. 0 = One stop bit 1 = Two stop bits, except for 5-bit character, then 1.5 bits
1:0	WLS	**Word-Length Select:** Specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character (default) 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

13.5.9 Modem Control Register

The following table provides details of the Modem Control Register:

Register Name:	MCR				
Hex Offset Address:	0xC800 X010		Reset Hex Value:	0x00000000	
Register Description:	Modem Control Register				
Access: Read/Write.					

31	...	5	4	3	2	1	0
(Reserved)			LOOP	OUT2	OUT1	RTS	DTR

Register		MCR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:5		(Reserved)

Register		MCR (Sheet 2 of 2)
Bits	**Name**	**Description**
4	LOOP	**Loop back test mode:** This bit provides a local Loop back feature for diagnostic testing of the UART. When LOOP is set to logic 1, the following occurs: • The transmitter serial output is set to a logic 1 state. • The receiver serial input is disconnected from the pin. • The output of the Transmitter Shift register is **looped back** into the receiver shift register input. • The four modem-control inputs (CTS_N, DSR_N, DCD_N, and RI_N) are disconnected from the pins and the modem control output pin RTS_N is forced to its inactive state. ***Note:*** Coming out of the loop back test mode may result in unpredictable activation of the delta bits (bits 3:0) in the Modem Status Register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. The lower four bits of the Modem Control register are connected to the upper four Modem Status register bits: • DTR = 1 forces DSR to a 1 • RTS = 1 forces CTS to a 1 • OUT1 = 1 forces RI to a 1 • OUT2= 1 forces DCD to a 1 — 0 = Normal UART operation — 1 = Test mode UART operation
3	OUT2	Interrupt Mask: The OUT2 bit is used to mask the UART's interrupt output to the Interrupt Controller unit. • OUT2 = 0 UART interrupt masked • OUT2 = 1 UART interrupt not masked When LOOP=1, this bit is not used as interrupt mask. The interrupt goes to the processor without mask in loop-back mode.
2	OUT1	**Test bit:** This bit is used only in loop-back test mode. See LOOP row, above.
1	RTS	**Request to Send:** This bit controls the Request to Send (RTS_N) output pin. 0 = RTS_N pin is 1 1 = RTS_N pin is 0
0	DTR	Data Terminal Ready: 0 = DTR_N pin is 1 1 = DTR_N pin is 0

13.5.10 Line Status Register

The following tables provide details of the Line Status Register:

Register Name:	**LSR**			
Hex Offset Address:	**0xC800 X014**		**Reset Hex Value:**	**0x00000060**
Register Description:	Line Status Register			
Access: Read Only.				

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			FIFOE	TEMT	TDRQ	BI	FE	PE	OE	DR

Register		LSR
Bits	**Name**	**Description**
31:8		(Reserved)
7	FIFOE	**FIFO Error Status:** In non-FIFO mode, this bit is 0. In FIFO Mode, FIFOE is set to 1 when there is at least a parity error, framing error, or break indication for any of the characters in the FIFO. Note that a processor read to the Line Status register does not reset this bit. FIFOE is reset when all error bytes have been read from the FIFO. 0 = Non-FIFO mode or no errors in receiver FIFO 1 = At least one character in receiver FIFO has errors
6	TEMT	**Transmitter Empty:** TEMT is set to a logic 1 when the Transmit Holding register and the Transmitter Shift register are both empty. It is reset to logic 0 when either the Transmit Holding register or the transmitter shift register contains a data character. In FIFO mode, TEMT is set to 1 when the transmitter FIFO and the Transmit Shift register are both empty.
5	TDRQ	**Transmit Data Request:** TDRQ indicates that the UART is ready to accept a new character for transmission. In non-FIFO mode, The TDRQ bit is set to logic 1 when a character is transferred from the Transmit Holding register into the Transmit Shift register. The bit is reset to logic 0 concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set to 1 when half of the characters in the FIFO have been loaded into the Shift register or the RESETTF bit in FCR has been set to 1. It is cleared when the FIFO has more than half data. If more than 64 characters are loaded into the FIFO, the excess characters are lost.
4	BI	**Break Interrupt:** BI is set to a logic 1 when the received data input is held in the spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + parity bit + stop bits). The Break indicator is reset when the processor reads the Line Status Register. In FIFO mode, only one character (equal to 00H) is loaded into the FIFO regardless of the length of the break condition. BI shows the break condition for the character at the bottom of the FIFO, not the most recently received character.
3	FE	**Framing Error:** FE indicates that the received character did not have a valid stop bit. FE is set to a logic 1 when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). The FE indicator is reset when the processor reads the Line Status Register. The UART re-synchronizes after a framing error. To do this, it assumes that the framing error was due to the next start bit, so it samples this **start** bit twice and then takes in the **data**. In FIFO mode, FE shows a framing error for the character at the bottom of the FIFO, not for the most recently received character.
2	PE	**Parity Error:** PE indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set to logic 1 upon detection of a parity error and is reset to logic 0 when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the bottom of the FIFO, not the most recently received character.
1	OE	**Overrun Error:** In non-FIFO mode, OE indicates that the processor did not read data the receiver buffer register before the next character was received. The new character is lost. In FIFO mode, OE indicates that all 64 bytes of the FIFO are full and the most recently received byte has been discarded. The OE indicator is set to logic 1 upon detection of an overrun condition and reset when the processor reads the Line Status register.
0	DR	**Data Ready:** Bit 0 is set to logic 1 when a complete incoming character has been received and transferred into the receiver buffer register or the FIFO. In non-FIFO mode, DR is reset to 0 when the receive buffer is read. In FIFO mode, DR is reset to a logic 0 if the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in the FCR. 0 = No data has been received 1 = Data is available in RBR or the FIFO

13.5.11 Modem Status Register

This register provides the current state of the control lines from the modem or data set (or a peripheral device emulating a modem) to the processor. In addition to this current state information, four bits of the Modem Status register provide change information. The 3:0 bits are set to a logic 1 when a control input from the modem changes state. The bits are reset to a logic 0 when the processor writes ones to the bits of the Modem Status Register.

Note: When bits 0, 1, 2, or 3 are set to logic 1, a Modem Status Interrupt is generated, if bit 3 of the Interrupt Enable Register is set.

Register Name:	MSR				
Hex Offset Address:	0xC800 X018		Reset Hex Value:	0x00000000	
Register Description:	Modem Status Register				
Access: Read Only.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)			DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

Register		MSR (Sheet 1 of 2)
Bits	**Name**	**Description**
31:8		(Reserved)
7	DCD	**Data Carrier Detect:** This bit is the complement of the Data Carrier Detect (DCD_N) input. This bit is equivalent to bit OUT2 of the Modem Control register if LOOP in the MCR is set to 1. 0 = DCD_N pin is 1 1 = DCD_N pin is 0
6	RI	**Ring Indicator:** This bit is the complement of the ring Indicator (RI_N) input. This bit is equivalent to bit OUT1 of the Modem Control register if LOOP in the MCR is set to 1. 0 = RI_N pin is 1 1 = RI_N pin is 0
5	DSR	**Data Set Ready:** This bit is the complement of the Data Set Ready (DSR_N) input. This bit is equivalent to bit DTR of the Modem Control register if LOOP in the MCR is set to 1. 0 = DSR_N pin is 1 1 = DSR_N pin is 0
4	CTS	**Clear to Send:** This bit is the complement of the Clear to Send (CTS_N) input. This bit is equivalent to bit RTS of the Modem Control register if LOOP in the MCR is set to 1. 0 = CTS_N pin is 1 1 = CTS_N pin is 0
3	DDCD	Delta Data Carrier Detect: 0 = No change in DCD_N pin since last read of MSR 1 = DCD_N pin has changed state

Register	MSR (Sheet 2 of 2)	
Bits	**Name**	**Description**
2	TERI	Trailing Edge Ring Indicator: 0 = RI_N pin has not changed from 0 to 1 since last read of MSR 1 = RI_N pin has changed from 0 to 1
1	DDSR	Delta Data Set Ready: 0 = No change in DSR_N pin since last read of MSR 1 = DSR_N pin has changed state
0	DCTS	Delta Clear To Send: 0 = No change in CTS_N pin since last read of MSR 1 = CTS_N pin has changed state

13.5.12 Scratch-Pad Register

This read/write register has no effect on the UART. The register is intended as a scratch-pad register for use by the programmer.

Register Name:	**SPR**				
Hex Offset Address:	**0xC800 X01C**		**Reset Hex Value:**	**0x00000000**	
Register Description:	Scratch-Pad Register				
Access: Read/Write.					

31																					**8**	**7**							**0**
(Reserved)																						SCR							

Register	SPR	
Bits	**Name**	**Description**
31:8		(Reserved)
7	SCR	No effect on UART functionality

13.5.13 Infrared Selection Register

The Slow Infrared (SIR) Interface is used in conjunction with a standard UART to support two-way, wireless communications using infrared radiation. It provides a transmit encoder and receive decoder to support a physical link that conforms to the Infrared Data Association Serial Infrared Specification.

Infrared mode is not supported in the IXP43X network processors. The reason for including this register in the address map and register description section is that, software ensures that this mode is never enabled.

Register Name:	ISR				
Hex Offset Address:	0xC800 X020		Reset Hex Value:	0x00000000	
Register Description:	Infrared Selection Register				
Access: Read/Write.					

31	...	8	7	6	5	4	3	2	1	0
(Reserved)						RXPL	TXPL	XMODE	(Reserved)	

Register		ISR
Bits	**Name**	**Description**
31:5		(Reserved)
4	RXPL	Receive Data Polarity: 0 = SIR decoder takes positive pulses as zeros 1 = SIR decoder takes negative pulses as zeros
3	TXPL	Transmit Data Polarity: 0 = SIR encoder generates a positive pulse for a data bit of zero 1 = SIR encoder generates a negative pulse for a data bit of zero
2	XMODE	**Transmit Pulse Width Select:** When XMODE is set to 0, clocking of the IRDA transmit and receive logic is done by the UART clock, that is operating in the 16X mode. When XMODE is set to a 1, the operation of the receive decoder does not change. The transmit encoder generates pulses 1.6 µs wide (that is three clock periods at frequency 1.8432 MHz), instead of the normal 3/16th of a bit time wide. The shorter infrared pulse, generated with XMODE set to 1, reduces the power consumed by the LEDs. At 2,400 bps, the LED would normally be activated for 78 µs for each 0 bit transmitted. With XMODE set, the LED would be activated for only 1.6 µs. This would reduce the power consumed by the LED by a factor of almost 48. 0 = Transmit pulse width is $3/16^{th}$ of a bit time wide 1 = Transmit pulse with is 1.6us
1		(Reserved)
0		(Reserved)

§ §

14.0 GPIO Controller

14.1 Overview

The purpose of this document is to outline the functional requirements of the general purpose input/output (GPIO) pins for the Intel® IXP43X Product Line of Network Processors.

The IXP43X network processors provide 16 general purpose input/output pins for generating and capturing application-specific input and output signals. Each pin is programmed as an input or output; and when GPIO0 through GPIO12 are programmed as an input, they can be used as an interrupt source. In addition, two of the pins are programmed to provide user programmable frequency source. During reset all pins are configured as inputs and remain in this state until configured otherwise with the exception of GPIO15, which by default provides a clock output. Each GPIO pin is capable of driving external LEDs. There are eight distinct register functions used in the GPIO module. When used as an interrupt source, each pin can detect interrupts as active high, active low, rising edge, falling edge, or transitional.

Another two signals are going to the Tsync unit through the GPIO Controller, these two are snapshot signals for the Tsync unit. They are wired through the GPIO, and goes to sync cells in the tsync unit. The Auxiliary slave snapshot signal is connected to GPIO_IN[8] and the Auxiliary master snapshot signal is connected to GPIO_IN[7].

14.2 Feature List

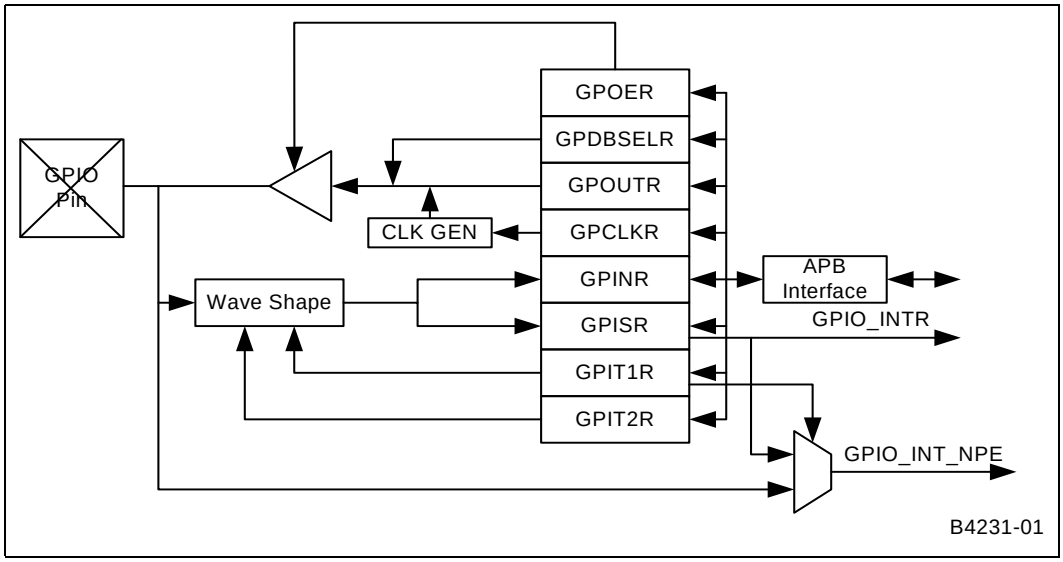
The features of the general purpose input/output pins are:

- Sixteen GPIO pins
- GPIO0 through GPIO12 are capable of being an interrupt source.
- Each pin is programmed to be an input or output.
- Two pins are programmed as a programmable clock source.
- Eight various functional registers.
- AMBA APB interface.
- Two snapshot trigger inputs
- GPIO_IN[7:0] is individually routed directly through to NPEs.

14.3 Block Diagram

The block diagram of the general purpose I/O block is shown in Figure 156.

Figure 156. GPIO Block Diagram



14.4 Theory of Operation

The GPIO pin is controlled through six registers and two more registers provide status. Each register is read through the APB interface and all registers except GPINR is written through the APB interface.

The GPIO on the IXP43X network processors is configured to be used as general purpose inputs or outputs. Three 16-bit registers are used to configure, initialize, and use the general purpose I/O. These registers are:

- General Purpose Data Output Register (GPOUTR)
- General Purpose Output Enable Register (GPOER)
- General Purpose Input Status Register (GPINR)

The General Purpose Output Enable Register is used to configure the GPIO pins as an input or an output. There is a one-to-one relationship between the register bit mapping and the GPIO. For example, bit 0 of the General Purpose Output Enable Register corresponds to GPIO 0 and bit 1 of the General Purpose Output Enable Register corresponds to GPIO 1.

When a bit of the General Purpose Enable Register contains logic 0, the corresponding GPIO is configured as an output. A logic 1 in the same bit of the General Purpose Enable Register causes the corresponding GPIO to be configured as an input.

For example, the General Purpose Output Enable Register contains a hexadecimal value of 0x00000500. GPIO 8 and GPIO 10 are configured as inputs and all other GPIO are configured as outputs. The GPIO that are configured as outputs by the values contained in the General Purpose Enable Register is driven by the values contained in the General Purpose Data Output Register.

The General Purpose Data Output Register is a 16-bit register with one-to-one correspondence between the 16 bits of the General Purpose Data Output Register and the 16-bit GPIO. When logic 1 is written to a bit in the General Purpose Data Output Register and the corresponding bit in the General Purpose Enable Register is set to logic 0, logic 1 is replicated to the corresponding GPIO. When logic 0 is written to the same

bit in the General Purpose Data Output Register and the corresponding bit in the General Purpose Enable Register is still set to a logic 0, logic 0 is replicated to the corresponding GPIO.

For example, the General Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General Purpose Data Output Register is set to hexadecimal 0x00000401. GPIO8 and GPIO10 is configured as output, GPIO8 drives logic 0, and GPIO10 drives logic 1.

Notice that bit 0 of the General Purpose Data Output Register is set to logic 1. But GPIO0 is configured as an input so the logic 1 in the General Purpose Data Output Register is not replicated to GPIO0.

Reading of the current status of the GPIO is obtained by reading the General Purpose Input Status Register. The General Purpose Input Status Register is a 16-bit register with one-to-one correspondence between the 16 bits of the General Purpose Input Status Register and the 16-bit GPIO.

When reading the General Purpose Input Status Register, the current logic value of the GPIO is reflected in the corresponding bits of the General Purpose Input Status Register. The values are replicated regardless of the configuration of the GPIO as defined by the General Purpose Enable Register.

If the GPIO is configured as an output, the value driven to the output is read while reading the corresponding bits of the General Purpose Input Status Register.

For example, the General Purpose Enable Register is set to hexadecimal 0xFFFFFAFF and the General Purpose Data Output Register is set to hexadecimal 0x00000401 and the GPIO pins have the following signals being supplied as inputs or driven as outputs, hexadecimal 0xAE37 with GPIO[15:12] = A and GPIO[3:0] = 7. GPIO8 and GPIO10 are configured as outputs as defined by the General Purpose Enable Register. All other GPIO pins are configured as inputs. When the General Purpose Input Status Register is read the value of hexadecimal 0x0000AE37 is returned.

14.4.1 Input Meta-Stability Protection, Edge Detect Logic, Pulse Discrimination

The GPIO input pin is sampled through a synchronizer cell to provide meta-stability protection. For the interrupt detection logic the pulse width of the synchronized GPIO input is monitored to ensure that it is greater than four pclk cycles. This decides against small pulses and ensures that glitches are not detected. A counter based approach is used to monitor the pulse width. A valid condition that depends on the interrupt detect type programmed in the GPIT registers is used to enable the counter. Once the count has reached four pclk cycles the appropriate bit is set in the GPISR register. Each programmed input pin of GPIO0 through GPIO12 can detect active high, active low, rising, falling or transitional (rising/falling) interrupts.

14.4.2 Clock Generation

The GPIO block provides two programmable clock outputs. The clocks are generated by counters based on the APB clock, pclk. There are two register settings that control the operation of the clock generator, one to define the terminal count (TC) of the counter and the other to define the duty cycle (DC).

The counters are 4-bit up counters, that is, counting from zero to the TC, free running at the pclk rate. Therefore, the TC defines the clock period. The duty cycle (DC) represents the number of counts for output clock is **low**. There is one special condition defined, if TC = 0xF and DC = 0xF, then the output of the clock generator block is pclk/2. If the duty cycle (DC) is programmed to be greater than or equal to the terminal count (TC), the generator outputs a ‘1’.

The clock generation logic is reset using an early reset, **early_reset_n**. This reset is deasserted prior to system reset being deasserted. This ensures that GPIO15 provides a clock out as system reset is still asserted.

14.4.3 APB Interface

The GPIO block interfaces with the Advanced Peripheral Bus, APB. The AHB/APB bridge provides the interface timing/signals required. Refer to the AMBA Rev 2.0 specification for a detailed description on the Advanced Micro controller Bus Architecture. The GPIO block responds to 8, 16, and 32 bit reads and writes.

14.5 Detailed Register Descriptions

The internal registers are accessed through the APB interface. Table 228 presents the registers.

Table 227. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
RW	Read/Write	RO	Read Only
RS	Read/Set	WO	Write Only
RW1C	Read/Write 1 to Clear	NA	Not Accessible

Table 228. Register Summary

Apb_paddr [31:0]	Register Name	Description	Reset Value	Attribute
"0xC8004000"	GPOUTR	GPIO pin data output register	"0x00000000"	RW
"0xC8004004"	GPOER	GPIO pin out enable register	"0x00007FFF"	RW
"0xC8004008"	GPINR	GPIO pin input status register	"0x00000000"	RO
"0xC800400C"	GPISR	GPIO interrupt status register	"0x00000000"	RW1C
"0xC8004010"	GPIT1R	GPIO interrupt type register, inputs 7:0	"0x00000000"	RW
"0xC8004014"	GPIT2R	GPIO pin interrupt type register, inputs 15:8	"0x00000000"	RW
"0xC8004018"	GPCLKR	GPIO Clock Control Register	"0x01100000"	RW

14.5.1 GPIO Output Register

The output data of each pin is controlled by programming this register. Each of the 16 bits in the register represents the data to be put on the output through a tri-state buffer, depending upon the status of the GPOER. The register is read through and written to through the APB interface on the rising edge of apb_pclk.

Register Name:	GPOUTR		
Physical Address:	0xC8004000	Reset Hex Value:	0x00000000
Register Description:	I/O Output register. Controls output value of GPIO pins, depending on tri-state control from GPOER.		
Access: Read/Write			

31															16	15							8	7							0
(Reserved)																DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Register	GPOUTR			
Bits	Name	Description	Reset Value	Access
[31: 16	(Reserved)	Reads back 0	0x0	RO
15	DO15	1 = Output a 1 on output pin, depends on GPCLKR24, GPOER15 0 = Output a 0 on output pin, depends on GPCLKR24, GPOER15	0	RW
14	DO14	1 = Output a 1 on output pin, depends on GPCLKR8, GPOER14 0 = Output a 0 on output pin, depends on GPCLKR8, GPOER14	0	RW
13:9	DO13:DO9	1 = Output a 1 on output pin, depends on GPDBSELR2, GPOER[13:9] 0 = Output a 0 on output pin, depends on GPDBSELR2, GPOER[13:9]	0	RW
8	DO8	1 = Output a 1 on output pin, depends on testmode_data, GPOER[8] 0 = Output a 0 on output pin, depends on testmode_data, GPOER[8]	0	RW
7:0	DO7:DO0	1 = Output a 1 on output pin GPOER[7:0] 0 = Output a 0 on output pin GPOER[7:0]	0	RW

14.5.2 GPIO Output Enable Register

The output tri-state buffer of each pin is controlled by programming this register. Each of the 16 bits in the register represents the control to the tri-state buffer of the output pin. The register is read through and written to through the APB interface on the rising edge of apb_pclk.

Register Name:	GPOER		
Physical Address:	0xC8004004	Reset Hex Value:	0x00007FFF
Register Description:	I/O Output Enable register. Turns on output driver.		
Access: Read/Write			

31															16	15							8	7							0
(Reserved)																OE15	OE14	OE13	OE12	OE11	OE10	OE9	OE8	OE7	OE6	OE5	OE4	OE3	OE2	OE1	OE0

Register		GPOER		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31: 16	(Reserved)	Reads back 0	0x0	RO
15	OE15	1 = Output pin is tri-stated or input 0 = Output pin is driven	0 (as clock is driven out on this pin during reset)	RW
14:0	OE14:OE0	1 = Output pin is tri-stated or input 0 = Output pin is driven	1	RW

14.5.3 GPIO Input Status Register

This is a read only register. This register contains the level of the I/O pin, as a '1' or a '0'.

Register Name:	GPINR		
Physical Address:	0xC8004008	Reset Hex Value:	0x00000000
Register Description:	This register is used to monitor input pins.		

Access: Read

31														16	15							8	7							0
(Reserved)															IN_LEV															

Register		GPINR		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:16	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
15:0	IN_LEV	Level of general purpose inputs 15-0 1 = 1 on GPIO 0 = 0 on GPIO	0x0000	RO

14.5.4 GPIO Interrupt Status Register

This register is used to store status of a GP input interpreted as an interrupt. The GP input interrupts are configured as active high, active low, rising edge, falling edge, or transitional depending upon the configuration of the GPIT1R or GPIT2R register. A **1** read from this register indicates a pending interrupt. Writing a **1** back to this register clears the interrupt provided the interrupting condition no longer exists. The interrupts are all masked in the Interrupt Controller block.

Note: GPIO0 through GPIO12 can be used as an interrupt (See Table 237, "Intel XScale® Processor Interrupt Mapping" on page 656.)

Register Name:	GPISR		
Physical Address:	0xC800400C	Reset Hex Value:	0x00000000
Register Description:	This register is used to store status of interrupts received on GP input pins		

Access: Read/Write 1C

31															16	15		13	12				8	7							0
(Reserved)																Not Used				INT_STAT											

Register	GPISR			
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
15:13		Not Used.	0x0	RO
12:0	INT_STAT	1 = Interrupt pending. 0 = No interrupt pending.	0x0000	RW1C

14.5.5 GPIO Interrupt Type Register 1

This register describes how to interpret GPIO [7:0] as interrupts, as level or as an edge, along with high, low, rising, falling, transitional. Three bits describe each GPIO pin, as described in the following table. The top [31:24] bits are used to control muxing between raw data from GPIO_IN[7:0] or GPISR[7:0] to the NPEs.

Register Name:	GPIT1R		
Physical Address:	0xC8004010	Reset Hex Value:	0x00000000
Register Description:	This register is used to control interrupt type for GPIO 7:0		

Access: Read/Write

31	30	29	28	27	26	25	24	23						16	15						8	7							0
gpio_npe_7	gpio_npe_6	gpio_npe_5	gpio_npe_4	gpio_npe_3	gpio_npe_2	gpio_npe_1	gpio_npe_0	GPIO7						GPIO6						GPIO5						GPIO4			
GPIO3								GPIO2								GPIO1								GPIO0			

Register	GPIT1R (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	Access
31	gpio_npe_7	When '1', a synchronized gpio_in[7] is muxed to gpio_int_npe[7], when '0', gpisr[7] is muxed to gpio_int_npe[7]	0	RW
30	gpio_npe_6	as per gpio_npe_7	0	RW
29	gpio_npe_5	as per gpio_npe_7	0	RW
28	gpio_npe_4	as per gpio_npe_7	0	RW

Register	GPIT1R (Sheet 2 of 2)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
27	gpio_npe_3	as per gpio_npe_7	0	RW
26	gpio_npe_2	as per gpio_npe_7	0	RW
25	gpio_npe_1	as per gpio_npe_7	0	RW
24	gpio_npe_0	as per gpio_npe_7	0	RW
23:21	GPIO7	000 – Active High 001 – Active Low 010 – Rising Edge 011 – Falling Edge 1xx – Transitional	000 Active High	RW
20:18	GPIO6	As per GPIO7	000 Active High	RW
17:15	GPIO5	As per GPIO7	000 Active High	RW
14:12	GPIO4	As per GPIO7	000 Active High	RW
11:9	GPIO3	As per GPIO7	000 Active High	RW
8:6	GPIO2	As per GPIO7	000 Active High	RW
5:3	GPIO1	As per GPIO7	000 Active High	RW
2:0	GPIO0	As per GPIO7	000 Active High	RW

14.5.6 GPIO Interrupt Type Register 2

This register describes how to interpret GPIO[15:8] as interrupts, as level or as an edge, along with high, low, rising, falling, transitional three bits describe each GPIO pin, as described in the following table.

Register Name:	**GPIT2R**		
Physical Address:	**0xC8004014**	**Reset Hex Value:**	**0x00000000**
Register Description:	This register is used to control interrupt type for GPIO 15:8		
Access: Read/Write			

31							**24**	**23**		**21**			**18**		**16**	**15**							**8**	**7**							**0**
(Reserved)								GPIO15			GPIO14			GPIO13			GPIO12			GPIO11			GPIO10			GPIO9			GPIO8		

Register		GPIT2R		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:24	(Reserved)	Not used. Ignored on writes and driven logic '0' on reads.	0x0	RO
23:21	GPIO15	Not Used	0x0	RO
20:18	GPIO14	Not Used	0x0	RO
17:15	GPIO13	Not Used	0x0	RO
14:12	GPIO12	000 – Active High 001 – Active Low 010 – Rising Edge 011 – Falling Edge 1xx – Transitional	000 Active High	RW
11:9	GPIO11	As per GPIO12	000 Active High	RW
8:6	GPIO10	As per GPIO12	000 Active High	RW
5:3	GPIO9	As per GPIO12	000 Active High	RW
2:0	GPIO8	As per GPIO12	000 Active High	RW

14.5.7 GPIO Clock Register

This register controls the use of GPIO15 and GPIO14 as clock outputs. GPOER defines the output enable for the driver, this register controls both the clock dividers and a MUX between the clock data and the data defined in GPOUTR. This register defines two four bit counts for a clock divider running from the pclk clock, one defines the period of the clock and the other defines the duty cycle. After early reset has been deasserted, GPIO 15 is defined as a pclk/2 output clock (33 MHz assuming 66 MHz pclk), with the expectation that this might be used to control the expansion bus peripherals.

The expansion bus requires an external clock to operate; it could be provided from GPIO15 or an external frequency source. As defined in the GPOER, GPIO14 is an input at reset. There are two special cases for these counters. First, if the TC is defined as '0x0' then the counter is disabled and the clock output is high. Second, if both TC and DC are '0xF', then the output clock is pclk/2. If DC is = TC, then the clock output is high.

Register Name:	**GPCLKR**		
Physical Address:	**0xC8004018**	**Reset Hex Value:**	**0x01100000**
Register Description:	This register controls the use of GPIO 15 and GPIO14 as clock sources		
Access: Read/Write			

31															16	15							8	7							0
(Reserved)								Mx15	CLK1TC			CLK1DC				(Reserved)							Mx14	CLK0TC				CLK0DC			

Register		GPCLKR		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:25	(Reserved)	Not used. Ignored on writes and driven logic ‘0’ on reads.	0x0	RO
24	MUX15	0 – Control from GPOUTR Register 1 – Clock output	1	RW
23:20	CLK1TC	Terminal count for a 4 bit up counter @ PCLK. An ‘F’ in this field and the CLK1DC field is a special case to provide PCLK/2.	0x1	RW
19:16	CLK1DC	Represents the number of counts for clock output should be low	0x0	RW
15:9	(Reserved)	Not used. Ignored on writes and driven logic ‘0’ on reads.		RW
8	MUX14	0 – Control from GPOUTR Register 1 – Clock Output	0	RW
7:4	CLK0TC	Terminal count for a 4 bit up counter @ PCLK. An ‘F’ in this field and the CLK0DC field is a special case to provide PCLK/2.	0x0	RW
3:0	CLK0DC	Represents the number of counts for clock output should be low	0x0	RW

§ §

15.0 Performance Monitoring Unit (PMU)

15.1 Overview

This chapter describes the components that comprise the Performance Monitoring Unit (PMU). These features aid in measuring and monitoring various system parameters that contribute to overall performance of the Intel® IXP43X Product Line of Network Processors. Operation modes, setup mechanisms, registers, and interrupts are also described in this chapter.

The performance monitoring (PMON) facility provided by the PMU comprises the following:

- Eight programmable event counters (PECx) clocked by the AHB clock (133MHz)
- Eight Programmable Event Counters () clocked by the MCU clock (133MHz/ 200MHz)
- Previous master/slave register
- Event-selection mux for 256x8 events
- Stop clock generation
- Simultaneous event counting

There are two types of counters, the PECx and the MPECx. The MPECx counters are similar to the PECx counters except that they are clocked by MCU clock, so that events referencing the MCU clock are captured correctly by the MPECx counters. The programmable event counters are 27 bits wide. Each counter is programmed to observe one event from a defined set of events. An event comprises a set of parameters and defines a condition. The monitored events are selected by programming the Event Select Registers (ESRx).

The PMU is inherently a model-specific function. This definition attempts to make the design and software model of the PMU more regular. The definition of events between instantiations of the PMU on various members of the family of network processors is expected to change. But this specification makes access to these events highly regular and allows a simple table-driven approach to associate the programming to event specification mapping.

Lots of empty space is available in the event address map that enables future expansion.

15.2 Feature List

The major features of the PMU are:

- Sixteen Programmable Event Count Registers
- Duration and Event detection logic
- Monitoring mechanisms
- Event selection logic

- Previous accessing/accessed Master/ Slave Register

15.3 Functional Description

The PMU comprises the following:

- Sixteen Programmable Event Counters (PEC).
- Event Selection logic
- Overflow detection and interrupt
- Previous Master and Slave information

The event counters are used to monitor events across various interfaces of the processor. The counters do nothing special to the events, only count the number of cycles that the event is “TRUE”. What makes an event a specific kind of performance event is the logic that decodes operations from the monitored buses and creates event streams to present to the event selection mux.

Unlike previous versions of the PMU, there are no **modes** or any meaningful difference between duration and occurrence events. There are only events, and each event has its description that defines what parameter is being measured. Some events are defined as occurrence events and some are defined as a duration event, but the implementation or programming of the PMU does not distinguish between any of the cases. From the software and implementation perspective, all events are uniform once they are presented to the selection mux. An occurrence event occurs only once per occurrence and back-to-back events represent two occurrences. A duration event is active the entire time the event is TRUE. Back-to-back events represent two disconnected events, back-to-back, or may represent consecutive cycles of the same event. Occurrence events have the semantics of a count whereas duration events have the semantics of time.

15.3.1 Programmable Event Counters

There are eight general purpose 27-bit wide Programmable Event Counters (PECx) clocked by the AHB clock, and another eight counters (MPECx) clocked by the MCU clock (with the same event inputs as the PECx). The reason for the size of the counter is to provide a count period of approximately one second at 133 MHz. Each counter is programmed to increment on detection of a rising edge or continuously with a high level on the selected input signal. If any counter generates an overflow, a flag is set and an interrupt is sent to the interrupt controller. The flag is cleared on a write to the PMU status register. The counters are reset whenever a new mode is selected, that is, North, South or DRAM. Mode HALT (that is, PMR set to disabled) causes all counters to halt allowing the same time snap-shot to be read across all counters.

The event that is being monitored by a counter is dependent on the ESR contents. There is an eight bit field per counter that selects the event being presented to each counter. In this way up to eight events is selected for monitoring. See Section 15.6.1, “Event Select Registers” for details. There is some redundancy between counter events to allow as many useful combinations to be monitored as possible (see Table 236 for combinations).

The programmable event counters provide real-time monitoring capability. They are memory mapped and is read directly. See Table 233 for details. All counters are reset and started whenever a value is written to the ESR and the mode is not HALT.

15.3.2 Occurrence Events

An occurrence event causes the counter to increase by one each time the event occurs. Table 229 presents the various occurrence events that are monitored on the IXP43X network processors.

Note: The PMU monitors two AHB buses, the north and the south. The logic performing the monitoring does not distinguish between the two buses, and the only effect that multiple buses has is to increase the number of events.

Table 229. Occurrence Events

Observed Interface	Monitored Events	Description
AHB (North or South)	Number of grants to the particular AHB device. Name: “xAHB[N] Grant Occur”	Monitors the number of times a master is granted the bus. It increments the counter when the master is the bus initiator. The counter is incremented once for every new transaction. For multi-cycle transactions, the counter increments once on the first cycle. Condition: HMASTER = MstrX
	Number of transactions initiated by the particular AHB device. Name: “xAHB[N] Xfer Occur”	Increments the counter every time that a master initiates a transaction on the bus. This event primitive is used in conjunction with the next primitive (retry), to calculate the effectiveness of transactions initiated on the AHB Bus by the master (that is,% transactions initiated by the master that are not retried.). Condition: HMASTER = MstrX & HTRANS = NonSeq
	Number of retries Signaled by the particular AHB device to the initiators. Note that not all AHB devices may initiate. Name: “xAHB[N] Retry Occur”	Increments the counter every time that a transaction initiated by the master is responded to with a retry by the target. Condition: HMASTER = MstrX & HRESP = Retry
	Number of split transfers claimed by the particular AHB device to the initiators. Note that not all AHB devices may initiate. Name: “xAHB[N] Split Occur”	Counts the number of times that SlaveX claims a split transaction. Condition: Hsel_SlaveX = 1 & HRESP = Split
MPI	Number of grants to the particular MPI port. Name: “MPI[N] Grant Occur”	Counts the number of times that PortX is granted an MPI transaction Condition: mpi_ackX
	Number of reads to the particular MPI port. Name: “MPI[N] Rd Occur”	Counts the number of times that PortX issues a read cycle Condition: mpi_ackX & !mpi_wr_rden_n
	Number of writes to the particular MPI port. Name: MPI[N] Wr Occur”	Counts the number of times that PortX issues a write cycle Condition: mpi_ackX & mpi_wr_rden_n
MEM	Number of occurrences of Event[N] from the MCU. Name: “MCU[N]	This event bus encodes up to 16 occurrence events where only one event is active in any given system clock cycle.
	Latency Events Name: “Universal Latency N”	Using the universal latency event bus things like the read latency of multiple outstanding reads is measured.

15.3.3 Duration Events

For a duration event, the counter counts the number of clocks when a particular condition or set of conditions is true. Duration measurements comprises the following:

- **Bus Acquisition Latency** — Represents the elapsed time between a bus master issuing a request for the bus to when it is granted the bus.
- **Split Transfer Latency** — Represents the elapsed time between the Requester receiving a Split Response and the Split Claimer transferring the first DWORD of Data Read.
- **Bus Ownership Time** — Represents the elapsed time when an initiator is in control of the bus.
- **Initiator Bus Data Transfer Time** — Represents the elapsed time when an initiator transfers data.
- **Split Data Transfer Time** — Represents the elapsed time when a Split Claimer transfers data (Data read) to a requester during a split transfer transaction. Please note that this metric is associated with the requester and **not** the Split Claimer.

Table 230 lists the various duration events that are monitored. The **Name** corresponds to the entry in Table 236.

Table 230. Duration Events (Sheet 1 of 2)

Observed Interface	Monitored Events	Description
AHB, Any Device	Number of clocks the AHB bus is doing Data Writes. Name: “AHBx Write”	Increments the counter on every AHB bus write data cycle. This enables calculation of data utilization of the Write data bus. Condition: HWrite = 1 & HReady = 1 & HTRANS = (seq OR NonSeq))
	Number of clocks the AHB bus is doing Data Reads. Name: “AHBx Read”	Increments the counter on every AHB bus read data cycle. This enables calculation of data utilization of the Read data bus. Condition: HWrite = 0 & HReady = 1 & HTRANS = seq or NonSeq
	Number of clocks the AHB bus is Idle. Name: “AHBx Idle”	Increments the counter every AHB bus idle cycle. An idle cycle occurs when there is no activity on the bus due to data being transferred and the bus is not in an overhead cycle. An overhead cycle is a cycle when an initiator owns the bus, but the initiator is unable to send data or the target is unable to receive data - hence no data is transferred. Condition: All Grant signals de-asserted

Table 230. Duration Events (Sheet 2 of 2)

Observed Interface	Monitored Events	Description
Specific AHB Device	Bus ownership for the AHB Masters. Name: xAHB[N] Grant Duration"	Counts the number of clocks spent by a master granted the AHB bus.
	Bus ownership metrics for the AHB masters. Name: "xAHB[N] Own Duration"	Counts the duration when the master is the initiator on the AHB bus. The counter increments on every clock cycle when the master is the bus initiator.
	Initiator data transfer time for Writes on the AHB masters. Name: "xAHB[N] WriteDuration"	Increments counter every AHB bus write data cycle when the master is the initiator.
	Initiator data transfer time for Reads on the AHB Masters. Name: "xAHB[N] Read Duration"	Increments counter every AHB bus read data cycle when the master is the initiator.
MPI	Number of clocks the MPI port is doing data writes. Name: "MPI[N] Wr Duration"	Increments counter for every cycle MPI port is transferring write data.
	Number of clocks the MPI port is doing data reads. Name: "MPI[N] rDDuration"	Increments counter for every cycle MPI port is receiving data.
	Number of clocks the MPI port is idle. Name: "MPI[N] IdleDuration"	Increments counter for every cycle MPI port is not requesting data nor transferring data.
	Bus acquisition latency for the MPI port. Name: "MPI[N] Latency Duration"	Increments counter for every cycle MPI port is requesting a transfer but not acknowledged.
FIFO	NPE coprocessor FIFO events	Full, empty, near full and near empty flags provided in the NPE Ethernet, Utopia, HSS and FIFO coprocessors are routed to the PMU for duration counting. Note that only specific flags were selected for implementation.
AQM	Queue Manager FIFO events	Full and near full flags in the Queue Manager (AQM) are routed to the PMU for duration counting.

15.3.4 Performance Monitoring

Performance monitoring consists of a collection of event primitives that is used for statistical calculations.

Various events on the AHBs are monitored by programming the ESR. Monitoring these events provide information about the AHB and the AHB initiators.

15.3.4.1 Halt: Performance Monitoring Disabled

Halting one or more counters disables performance monitoring on the halted counter. In this way any combination or all counters is halted at the same time. Counters are reset whenever a new mode (not Halt) is selected. Halt/enable is selected on a per counter basis by the enabled bits in the PMR. Halt/enable conditions are maintained in a separate register from the mux programming as not all of the counter mux registers is updated in the same cycle.

The normal expected mode of operation is to halt all counters, program in the counter event mux selects, and then enable the desired counters.

15.3.4.2 Cycle Count

Selecting mux input 0xFF (event select == 256) selects the event TRUE and has the effect of putting the counter into continuous count mode.

15.3.4.3 MCU: DRAM Transactions

Selecting MCU events enables performance monitoring of the DRAM. These transactions are measured using the mcu_pmu_event signal provided from the DRAM controller. Table 231 describes this signal. See the memory controller chapter for information on the semantics of these events.

Table 231. Signal Descriptions for mcu_pmu_event

Bit	Description
[4]	1 = Valid event
[3:0]	Event number.

15.3.4.4 Events

When not halted, a counter counts the cycles when an event is TRUE. The event descriptions are listed above. To alleviate the wiring of redundant events into a very large event mux, not all events are connected to every counter. Instead, the allocation of events is intended to be a compromise between flexibility and simplicity. The basic philosophy is that you should be able to measure all attributes about a given device, all of the same attributes about a number of devices, or a subset of the two.

An attempt is made to maximize the flexibility but any conceivable combination of performance metrics may not fit into 8 counter's worth of information or the chosen eight may not be all accessible at the same time. To acquire any given group of metrics, it is necessary to repeat two passes of the same operations. There are some natural inaccuracies that result as a consequence of multiple passes unless extreme care is taken to make the two passes identical.

15.4 Previous Master and Slave

The PMU provides a register that indicates the last masters and the slave they accessed on both the North and South AHB busses. The master value stored is determined from the HMASTER signal, that is the grants to the master devices. Whenever this signal changes and there is a bus transaction the register bits are updated. The slave value stored is determined from the HSELx signal. Whenever the slave selected changes and there is a bus transaction these register bits are updated.

The PMSR also locks the current master/slave in the case of an AHB error signal. There are three reasons for an AHB error signal - an unsupported bus operation to a slave, an uncorrectable error by a slave, and an out-of-range address. In the first two cases, the HSEL field contains the device that was being addressed, and in the third case the HSEL field contains 0xF.

The first occurrence of an error signal on a bus causes the master and slave fields to be updated to the current value, the **stuck** bit appropriate for the north and south fields is set, and further updates blocked until the stuck bit has been reset (by writing a '1' to the bit position). Multiple error conditions are not supported, and any subsequent errors after the first are ignored. See Section 15.6.6.1, "PMSR" for more details.

15.5 Miscellaneous

15.5.1 Interrupts

Each of the counters has an associated overflow flag. These flags are readable from the PMU Status Register ("PSR"). The pmu_overflow_int interrupt is the logic OR of all the overflow flags. The flags and interrupt are cleared when the register is written with ones in the overflow positions.

15.5.2 Reset Conditions

The ESR defaults to mode Halt upon reset. Performance monitoring is disabled and all counters are disabled in this mode. The Programmable Event Counters (PECRx) values are cleared upon reset.

15.6 Detailed Register Descriptions

This section gives the detailed register descriptions of the Performance Monitoring Unit.

Table 232. Register Legend

Attribute	Legend	Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read clear	RW1C	Normal read Write '1' to clear
PR	Preserved	RO	Read only	WO	Write only
RS	Read/Set	RW	Read/Write	NA	Not accessible

The performance monitoring facility in the IXP43X network processors comprises 13 memory-mapped registers for controlling operation and monitoring various events. Each register appears to be 32-bits wide to the APB bus. Each of these registers is accessed as a memory-mapped 32-bit register with a unique memory address. Access is accomplished through regular memory-format instructions from the Bus Interface Unit.

Table 233 presents the registers and their offset addresses.

Table 233. PMU Register Table (Sheet 1 of 2)

Register Name	Reset Hex Value	Hex Offset Address (Base Address: 0xC800_20XX)
ESR0	0x00000000	0x00
ESR1	0x00000000	0x04
(Reserved)		0x08
(Reserved)		0x0C
PSR	0x00000000	0x10
PMR	0x00000000	0x14
PMSR	0x00000000	0x18
(Reserved)		0x1C
PEC0	0x00000000	0x20
PEC1	0x00000000	0x24
PEC2	0x00000000	0x28
PEC3	0x00000000	0x2C

Table 233. PMU Register Table (Sheet 2 of 2)

Register Name	Reset Hex Value	Hex Offset Address (Base Address: 0xC800_20XX)
PEC4	0x00000000	0x30
PEC5	0x00000000	0x34
PEC6	0x00000000	0x38
PEC7	0x00000000	0x3C
MPEC0	0x00000000	0x40
MPEC1	0x00000000	0x44
MPEC2	0x00000000	0x48
MPEC3	0x00000000	0x4C
MPEC4	0x00000000	0x50
MPEC5	0x00000000	0x54
MPEC6	0x00000000	0x58
MPEC7	0x00000000	0x5C

15.6.1 Event Select Registers

Detailed descriptions of the Event Select Registers, ESR0 and ESR1 are given below.

15.6.1.1 ESR0 and ESR1

The ESR controls the specific item being monitored. Each PECx field is programmed according to Table 236, “Event Mux Programming” on page 650.

To change the monitored event, it is necessary to write the entire ESR. The programmable event counters are reset and started when a new value is written to the ESR.

Note: Since there are two registers and one cannot write both of them in the same cycle, it is preferable to halt the counters via the MODE register (that is, PMR), update the mux, selects via these registers (that resets the counters), and then enable the desired counters with the PMR.

Register Name:	ESR0		
Physical Address:	0xC8002000	Reset Hex Value:	0x00000000
Register Description:	Event Mux Select Register, counters 3-0		

Access: Read/Write

31															16	15							8	7							0
PEC3 ctrl								PEC2 ctrl								PEC1 ctrl								PEC0 ctrl							

Register	ESR (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	Access
31:24	PEC3 ctrl	Selects Enable conditions for counter PEC3/MPEC3.	0x00	RW

Register	ESR (Sheet 2 of 2)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
23:16	PEC2 ctrl	Selects Enable conditions for counter PEC2/MPEC2.	0x00	RW
15:8	PEC1 ctrl	Selects Enable conditions for counter PEC1/MPEC1.	0x00	RW
7:0	PEC0 ctrl	Selects Enable conditions for counter PEC0/MPEC0.	0x00	RW

Register Name:	ESR1		
Physical Address:	0xC8002004	Reset Hex Value:	0x00000000
Register Description:	Event Mux Select Register, counters 7-4		

Access: Read/Write

31															16	15							8	7							0
PEC7 ctrl								PEC6 ctrl								PEC5 ctrl								PEC4 ctrl							

Register	ESR			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:24	PEC7 ctrl	Selects Enable conditions for counter PEC7/MPEC7.	0x00	RW
23:16	PEC6 ctrl	Selects Enable conditions for counter PEC6/MPEC6.	0x00	RW
15:8	PEC5 ctrl	Selects Enable conditions for counter PEC5/MPEC5.	0x00	RW
7:0	PEC4 ctrl	Selects Enable conditions for counter PEC4/MPEC4.	0x00	RW

15.6.2 PMU Status Register

The detailed description of the PMU Status Register PSR is given below.

15.6.2.1 PSR

The PSR allows access to the over flow flags from the PECx and the MPECx counters. These flags remain set until cleared by writing a '1' to the bit. Setting an overflow condition from the counter takes precedence over resetting the interrupt. If the counter overflows on the exact same cycle the status is being reset, the overflow is set. If any of these status bits are set, the corresponding counter has overflowed and an interrupt continues to be generated until all the bits are clear.

Register Name:	PSR		
Physical Address:	0xC8002010	Reset Hex Value:	0x00000000
Register Description:	Counter Overflow Status Register		

Access: Read, Clear on write

31															16	15							8	7							0
Reserved																OFL15	OFL14	OFL13	OFL12	OFL11	OFL10	OFL9	OFL8	OFL7	OFL6	OFL5	OFL4	OFL3	OFL2	OFL1	OFL0

Register		PSR		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:16	Reserved	Always zero.		
15	OFL15	1 = MPEC7 has overflowed	0	RW1C
14	OFL14	1 = MPEC6 has overflowed	0	RW1C
13	OFL13	1 = MPEC5 has overflowed	0	RW1C
12	OFL12	1 = MPEC4 has overflowed	0	RW1C
11	OFL11	1 = MPEC3 has overflowed	0	RW1C
10	OFL10	1 = MPEC2 has overflowed	0	RW1C
9	OFL9	1 = MPEC1 has overflowed	0	RW1C
8	OFL8	1 = MPEC0 has overflowed	0	RW1C
7	OFL7	1 = PEC7 has overflowed	0	RW1C
6	OFL6	1 = PEC6 has overflowed	0	RW1C
5	OFL5	1 = PEC5 has overflowed	0	RW1C
4	OFL4	1 = PEC4 has overflowed	0	RW1C
3	OFL3	1 = PEC3 has overflowed	0	RW1C
2	OFL2	1 = PEC2 has overflowed	0	RW1C
1	OFL1	1 = PEC1 has overflowed	0	RW1C
0	OFL0	1 = PEC0 has overflowed	0	RW1C

15.6.3 PMU Mode Register

The detailed description of the PMU Mode Register PMR is given below.

15.6.3.1 PMR

The PMR controls the counters (both PECx and MPECx) that are enabled or halted. If the bit corresponding to a particular PEC or MPEC is false, the counter is not enabled to count and if the bit is true, the PEC or MPEC is enabled to count.

Register Name:	PMR		
Physical Address:	**0xC8002014**	**Reset Hex Value:**	**0x00000000**
Register Description:	Counter Enable Mode Register		

Access: Read, Clear on write

31															16	15							8	7							0
Reserved																Enable15	Enable14	Enable13	Enable12	Enable11	Enable10	Enable9	Enable8	Enable7	Enable6	Enable5	Enable4	Enable3	Enable2	Enable1	Enable0

Register		PMR		
Bits	Name	Description	Reset Value	Access
31:16	Reserved	Always zero.		
15	Enable15	1 = MPEC7 is Enabled	0	RW
14	Enable14	1 = MPEC6 is Enabled	0	RW
13	Enable13	1 = MPEC5 is Enabled	0	RW
12	Enable12	1 = MPEC4 is Enabled	0	RW
11	Enable11	1 = MPEC3 is Enabled	0	RW
10	Enable10	1 = MPEC2 is Enabled	0	RW
9	Enable9	1 = MPEC1 is Enabled	0	RW
8	Enable8	1 = MPEC0 is Enabled	0	RW
7	Enable7	1 = PEC7 is Enabled	0	RW
6	Enable6	1 = PEC6 is Enabled	0	RW
5	Enable5	1 = PEC5 is Enabled	0	RW
4	Enable4	1 = PEC4 is Enabled	0	RW
3	Enable3	1 = PEC3 is Enabled	0	RW
2	Enable2	1 = PEC2 is Enabled	0	RW
1	Enable1	1 = PEC1 is Enabled	0	RW
0	Enable0	1 = PEC0 is Enabled	0	RW

15.6.4 Programmable Event Counters

The detailed descriptions of the Programmable Event Counters (PECx) are given below.

15.6.4.1 PECx

There are eight programmable event counters (PEC0 – PEC7) that are available through the memory map. These counters are 27-bit wide and are read only.

The value in any register is incremented based on the current programmed ESR value and the descriptions shown in the following tables in this subsection. When a new event to monitor is chosen by writing a value to the ESR, these registers are reset to zero. The counters are enabled via the PMR.

Register Name:	PECx		
Physical Address:	0xC8002020 - 0xC800203C	Reset Hex Value:	0x00000000
Register Description:	Event Counter		
Access: Read			

31															16	15							8	7							0
Reserved															PECx																

Register		PECx		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:27	Reserved	Always zero.		
26:0	PECx	This is a 27-bit, read-only counter register.	0x0000000	R

15.6.5 MCU-Clock Programmable Event Counters

The detailed descriptions of the MCU-Clock Programmable Event Counters (MPECx) are given below.

15.6.5.1 MPECx

There are eight programmable event counters (MPEC0 – MPEC7) that are available through the memory map. These counters are 27-bit wide and are read only.

These counters are similar to the PECx counters. The only difference is that the counters are clocked by the Memory Controller (MCU) clock. Thus, events referencing the MCU clock (133MHz/200MHz) are counted using the MPECx counters to get an accurate count.

Register Name:	MPECx		
Physical Address:	0xC8002040 - 0xC800205C	Reset Hex Value:	0x00000000
Register Description:	Event Counter		

Access: Read

31															16	15							8	7							0
Reserved															MPECx																

Register		MPECx		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:27	Reserved	Always zero.		
26:0	MPECx	This is a 27-bit, read-only counter register.	0x0000000	R

15.6.6 Previous Master/Slave Register

The detailed description of the Previous Master/Slave Register is given below.

15.6.6.1 PMSR

This register encodes the device that was previously accessed or was being accessed on the monitored buses.

This register is reset, but the reset value is quickly overwritten with the bus activity, so the reset value is not relevant. This value reflects the last recorded activity. To build up a histogram of master/slave pairs this register is read periodically though there is no way to know that an unchanging value means that there was no bus activity.

Register Name:	PMSR		
Physical Address:	0xC8002018	Reset Hex Value:	0x00000000
Register Description:	Bus activity master/slave		

Access: Read

31														16	15							8	7							0
SE	NE	Reserved												MPI		PSS				PSN			PMS				PMN			

Register	PMSR			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31	South Error	When set, the South AHB fields have captured an AHB error and these fields are “stuck” at the first error condition. To clear the error, write a ‘1’ to this bit.	0	RW1C
30	North Error	When set, the North AHB fields have captured an AHB error and these fields are “stuck” at the first error condition. To clear the error, write a ‘1’ to this bit.	0	RW1C
29:20	Reserved			
19:16	MPI	Indicates the ports on the MPI that was previously accessing the MCU	0x0	RO
15:12	PSS	Indicates the Slaves on ARBS that was previously accessed the AHBS In the case of an address out-of-range error, this field is 0xF.	0x0	RO
11:8	PSN	Indicates the Slaves on ARBN that was previously accessed the AHBN In the case of an address out-of-range error, this field is 0xF.	0x0	RO
7:4	PMS	Indicates the masters on ARBS that was previously accessing the AHBS	0x0	RO
3:0	PMN	Indicates the master on ARBN that was previously accessing the AHBN	0x0	RO

15.7 Event Mapping

There are two connections to the PMU for monitoring the AHB system buses, one on the North bus and the other one on the South bus. Table 234 and Table 235 show how each bus maps to the PMU.

Table 234. AHB North PMU Mappings

PMU Device #	Name	Master	Slave	Retry	Split
0	NPE-A	Y	N	Y	Y
1	Reserved	NA	NA	NA	NA
2	NPE-C	Y	N	Y	Y
3	MCU	N	Y	N	N
4	AHB Bridge	N	Y	N	Y

Table 235. AHB South PMU Mappings

PMU Device #	Name	Master	Slave	Retry	Split
0	X-Scale BIU	Y	N	Y	Y
1	PCI	Y	Y	Y	N
2	AHB Bridge	Y	N	Y	Y
3	Expansion Bus	N	Y	Y	Y
4	USBH0	Y	Y	Y	N
5	MCU	N	Y	N	N
6	APB Bridge	N	Y	N	N
7	AQM	N	Y	N	N
8	USBH1	Y	Y	Y	N

In the following table, note that there appears to be too many entries. For example, there is space allocated for 16 north AHB devices even though these many entries may not be defined. To determine the characteristics of a particular model of PMU one must look at the Table 234 and Table 235 to determine the devices that are actually present and then in Table 236 to determine the event address in the ESR registers.

Table 236. Event Mux Programming (Sheet 1 of 5)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
MCU Events								
0	MCU[0]	MCU[1]	MCU[2]	MCU[3]	MCU[4]	MCU[5]	MCU[6]	MCU[7]
1	MCU[8]	MCU[9]	MCU[10]	MCU[11]	MCU[12]	MCU[13]	MCU[14]	MCU[15]
AHB North Device Events								
2	North AHB[0] Grant Occur	North AHB[0] Xfer Occur	North AHB[0] Retry Occur	North AHB[0] Split Occur	North AHB[0] Request Duration	North AHB[0] Own Duration	North AHB[0] Write Duration	North AHB[0] Read Duration
3	North AHB[1] Grant Occur	North AHB[1] Xfer Occur	North AHB[1] Retry Occur	North AHB[1] Split Occur	North AHB[1] Request Duration	North AHB[1] Own Duration	North AHB[1] Write Duration	North AHB[1] Read Duration
4	North AHB[2] Grant Occur	North AHB[2] Xfer Occur	North AHB[2] Retry Occur	North AHB[2] Split Occur	North AHB[2] Request Duration	North AHB[2] Own Duration	North AHB[2] Write Duration	North AHB[2] Read Duration
5	North AHB[3] Grant Occur	North AHB[3] Xfer Occur	North AHB[3] Retry Occur	North AHB[3] Split Occur	North AHB[3] Request Duration	North AHB[3] Own Duration	North AHB[3] Write Duration	North AHB[3] Read Duration
6	North AHB[4] Grant Occur	North AHB[4] Xfer Occur	North AHB[4] Retry Occur	North AHB[4] Split Occur	North AHB[4] Request Duration	North AHB[4] Own Duration	North AHB[4] Write Duration	North AHB[4] Read Duration
7	North AHB[5] Grant Occur	North AHB[5] Xfer Occur	North AHB[5] Retry Occur	North AHB[5] Split Occur	North AHB[5] Request Duration	North AHB[5] Own Duration	North AHB[5] Write Duration	North AHB[5] Read Duration
8	North AHB[6] Grant Occur	North AHB[6] Xfer Occur	North AHB[6] Retry Occur	North AHB[6] Split Occur	North AHB[6] Request Duration	North AHB[6] Own Duration	North AHB[6] Write Duration	North AHB[6] Read Duration

Table 236. Event Mux Programming (Sheet 2 of 5)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
9	North AHB[7] Grant Occur	North AHB[7] Xfer Occur	North AHB[7] Retry Occur	North AHB[7] Split Occur	North AHB[7] Request Duration	North AHB[7] Own Duration	North AHB[7] Write Duration	North AHB[7] Read Duration
10	North AHB[8] Grant Occur	North AHB[8] Xfer Occur	North AHB[8] Retry Occur	North AHB[8] Split Occur	North AHB[8] Request Duration	North AHB[8] Own Duration	North AHB[8] Write Duration	North AHB[8] Read Duration
11	North AHB[9] Grant Occur	North AHB[9] Xfer Occur	North AHB[9] Retry Occur	North AHB[9] Split Occur	North AHB[9] Request Duration	North AHB[9] Own Duration	North AHB[9] Write Duration	North AHB[9] Read Duration
12	North AHB[10] Grant Occur	North AHB[10] Xfer Occur	North AHB[10] Retry Occur	North AHB[10] Split Occur	North AHB[10] Request Duration	North AHB[10] Own Duration	North AHB[10] Write Duration	North AHB[10] Read Duration
13	North AHB[11] Grant Occur	North AHB[11] Xfer Occur	North AHB[11] Retry Occur	North AHB[11] Split Occur	North AHB[11] Request Duration	North AHB[11] Own Duration	North AHB[11] Write Duration	North AHB[11] Read Duration
14	North AHB[12] Grant Occur	North AHB[12] Xfer Occur	North AHB[12] Retry Occur	North AHB[12] Split Occur	North AHB[12] Request Duration	North AHB[12] Own Duration	North AHB[12] Write Duration	North AHB[12] Read Duration
15	North AHB[13] Grant Occur	North AHB[13] Xfer Occur	North AHB[13] Retry Occur	North AHB[13] Split Occur	North AHB[13] Request Duration	North AHB[13] Own Duration	North AHB[13] Write Duration	North AHB[13] Read Duration
16	North AHB[14] Grant Occur	North AHB[14] Xfer Occur	North AHB[14] Retry Occur	North AHB[14] Split Occur	North AHB[14] Request Duration	North AHB[14] Own Duration	North AHB[14] Write Duration	North AHB[14] Read Duration
17	North AHB[15] Grant Occur	North AHB[15] Xfer Occur	North AHB[15] Retry Occur	North AHB[15] Split Occur	North AHB[15] Request Duration	North AHB[5] Own Duration	North AHB[15] Write Duration	North AHB[15] Read Duration
AHB South Device Events								
18	South AHB[0] Grant Occur	South AHB[0] Xfer Occur	South AHB[0] Retry Occur	South AHB[0] Split Occur	South AHB[0] Request Duration	South AHB[0] Own Duration	South AHB[0] Write Duration	South AHB[0] Read Duration
19	South AHB[1] Grant Occur	South AHB[1] Xfer Occur	South AHB[1] Retry Occur	South AHB[1] Split Occur	South AHB[1] Request Duration	South AHB[1] Own Duration	South AHB[1] Write Duration	South AHB[1] Read Duration
20	South AHB[2] Grant Occur	South AHB[2] Xfer Occur	South AHB[2] Retry Occur	South AHB[2] Split Occur	South AHB[2] Request Duration	South AHB[2] Own Duration	South AHB[2] Write Duration	South AHB[2] Read Duration
21	South AHB[3] Grant Occur	South AHB[3] Xfer Occur	South AHB[3] Retry Occur	South AHB[3] Split Occur	South AHB[3] Request Duration	South AHB[3] Own Duration	South AHB[3] Write Duration	South AHB[3] Read Duration
22	South AHB[4] Grant Occur	South AHB[4] Xfer Occur	South AHB[4] Retry Occur	South AHB[4] Split Occur	South AHB[4] Request Duration	South AHB[4] Own Duration	South AHB[4] Write Duration	South AHB[4] Read Duration

Table 236. Event Mux Programming (Sheet 3 of 5)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
23	South AHB[5] Grant Occur	South AHB[5] Xfer Occur	South AHB[5] Retry Occur	South AHB[5] Split Occur	South AHB[5] Request Duration	South AHB[5] Own Duration	South AHB[5] Write Duration	South AHB[5] Read Duration
24	South AHB[6] Grant Occur	South AHB[6] Xfer Occur	South AHB[6] Retry Occur	South AHB[6] Split Occur	South AHB[6] Request Duration	South AHB[6] Own Duration	South AHB[6] Write Duration	South AHB[6] Read Duration
25	South AHB[7] Grant Occur	South AHB[7] Xfer Occur	South AHB[7] Retry Occur	South AHB[7] Split Occur	South AHB[7] Request Duration	South AHB[7] Own Duration	South AHB[7] Write Duration	South AHB[7] Read Duration
26	South AHB[8] Grant Occur	South AHB[8] Xfer Occur	South AHB[8] Retry Occur	South AHB[8] Split Occur	South AHB[8] Request Duration	South AHB[8] Own Duration	South AHB[8] Write Duration	South AHB[8] Read Duration
27	South AHB[9] Grant Occur	South AHB[9] Xfer Occur	South AHB[9] Retry Occur	South AHB[9] Split Occur	South AHB[9] Request Duration	South AHB[9] Own Duration	South AHB[9] Write Duration	South AHB[9] Read Duration
28	South AHB[10] Grant Occur	South AHB[10] Xfer Occur	South AHB[10] Retry Occur	South AHB[10] Split Occur	South AHB[10] Request Duration	South AHB[10] Own Duration	South AHB[10] Write Duration	South AHB[10] Read Duration
29	South AHB[11] Grant Occur	South AHB[11] Xfer Occur	South AHB[11] Retry Occur	South AHB[11] Split Occur	South AHB[11] Request Duration	South AHB[11] Own Duration	South AHB[11] Write Duration	South AHB[11] Read Duration
30	South AHB[12] Grant Occur	South AHB[12] Xfer Occur	South AHB[12] Retry Occur	South AHB[12] Split Occur	South AHB[12] Request Duration	South AHB[12] Own Duration	South AHB[12] Write Duration	South AHB[12] Read Duration
31	South AHB[13] Grant Occur	South AHB[13] Xfer Occur	South AHB[13] Retry Occur	South AHB[13] Split Occur	South AHB[13] Request Duration	South AHB[13] Own Duration	South AHB[13] Write Duration	South AHB[13] Read Duration
32	South AHB[14] Grant Occur	South AHB[14] Xfer Occur	South AHB[14] Retry Occur	South AHB[14] Split Occur	South AHB[14] Request Duration	South AHB[14] Own Duration	South AHB[14] Write Duration	South AHB[14] Read Duration
33	South AHB[15] Grant Occur	South AHB[15] Xfer Occur	South AHB[15] Retry Occur	South AHB[15] Split Occur	South AHB[15] Request Duration	South AHB[5] Own Duration	South AHB[15] Write Duration	South AHB[15] Read Duration
	AHB Miscellaneous							
34	AHBN Write Duration	AHBN Read Duration	AHBN Idle Duration	AHBN Any Cycle Duration	AHBS Write Duration	AHBS Read Duration	AHBS Idle Duration	AHBN Any Cycle Duration
	MPI							
35	MPI[0] Grant Occur	MPI[0] Rd Occur	MPI[0] Wr Occur	MPI[0] Done Occur	MPI[0] Wr Duration	MPI[0] Rd Duration	MPI[0] Latency Duration	MPI[0] Idle Duration
36	MPI[1] Grant Occur	MPI[1] Rd Occur	MPI[1] Wr Occur	MPI[1] Done Occur	MPI[1] Wr Duration	MPI[1] Rd Duration	MPI[1] Latency Duration	MPI[1] Idle Duration

Table 236. Event Mux Programming (Sheet 4 of 5)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
37	MPI[2] Grant Occur	MPI[2] Rd Occur	MPI[2] Wr Occur	MPI[2] Done Occur	MPI[2] Wr Duration	MPI[2] Rd Duration	MPI[2] Latency Duration	MPI[2] Idle Duration
38	MPI[3] Grant Occur	MPI[3] Rd Occur	MPI[3] Wr Occur	MPI[3] Done Occur	MPI[3] Wr Duration	MPI[3] Rd Duration	MPI[3] Latency Duration	MPI[3] Idle Duration
39	Universal Latency 0	Universal Latency 1	Universal Latency 2	Universal Latency 3	Universal Latency 4	Universal Latency 5	Universal Latency 6	Universal Latency 7
40	Universal Latency 8	Universal Latency 9	Universal Latency 10	Universal Latency 11	Universal Latency 12	Universal Latency 13	Universal Latency 14	Universal Latency 15
41	Universal Occurrence 0	Universal Occurrence 1	Universal Occurrence 2	Universal Occurrence 3	Universal Occurrence 4	Universal Occurrence 5	Universal Occurrence 6	Universal Occurrence 7
42	Universal Occurrence 8	Universal Occurrence 9	Universal Occurrence 10	Universal Occurrence 11	Universal Occurrence 12	Universal Occurrence 13	Universal Occurrence 14	Universal Occurrence 15
43	NPE A ffcp_fifo0_ne Duration	NPE A ffcp_fifo1_ne Duration	NPE A ffcp_fifo2_ne Duration	NPE A ffcp_fifo3_ne Duration	NPE A ffcp_fifo4_ne Duration	NPE A ffcp_fifo5_ne Duration	NPE A ffcp_fifo6_ne Duration	NPE A ffcp_fifo7_ne Duration
44	NPE A ffcp_fifo0_nf Duration	NPE A ffcp_fifo1_nf Duration	NPE A ffcp_fifo2_nf Duration	NPE A ffcp_fifo3_nf Duration	NPE A ffcp_fifo4_nf Duration	NPE A ffcp_fifo5_nf Duration	NPE A ffcp_fifo6_nf Duration	NPE A ffcp_fifo7_nf Duration
45	NPE C ffcp_fifo0_ne Duration	NPE C ffcp_fifo1_ne Duration	NPE C ffcp_fifo2_ne Duration	NPE C ffcp_fifo3_ne Duration	NPE C ffcp_fifo4_ne Duration	NPE C ffcp_fifo5_ne Duration	NPE C ffcp_fifo6_ne Duration	NPE C ffcp_fifo7_ne Duration
46	NPE C ffcp_fifo0_nf Duration	NPE C ffcp_fifo1_nf Duration	NPE C ffcp_fifo2_nf Duration	NPE C ffcp_fifo3_nf Duration	NPE C ffcp_fifo4_nf Duration	NPE C ffcp_fifo5_nf Duration	NPE C ffcp_fifo6_nf Duration	NPE C ffcp_fifo7_nf Duration
47	NPE A ecp_rx_fifo_full Duration	NPE A hss_tx_va_empty Duration	NPE A ahb_f0_done Duration	NPE C ecp_rx_fifo_full Duration	NPE C ahb_f0_done Duration			
48	NPE A ecp_tx_fifo_full Duration	NPE A hss_tx_vb_empty Duration	NPE A ahb_f1_done Duration	NPE C ecp_tx_fifo_full Duration	NPE C ahb_f1_done Duration			
49	NPE A ucp_tx_fifo_empty Duration	NPE A hss_tx_h_empty Duration	NPE A ahb_fifo_free Duration		NPE C ahb_fifo_free Duration			
50	NPE A ucp_rx_fifo_full Duration	NPE A hss_rx_va_full Duration						
51		NPE A hss_rx_vb_full Duration						
52		NPE A hss_rx_h_full Duration						
53	aqm_fifo0_full duration	aqm_fifo1_full duration	aqm_fifo2_full duration	aqm_fifo3_full duration	aqm_fifo4_full duration	aqm_fifo5_full duration	aqm_fifo6_full duration	aqm_fifo7_full duration
54	aqm_fifo8_full duration	aqm_fifo9_full duration	aqm_fifo10_full duration	aqm_fifo11_full duration	aqm_fifo12_full duration	aqm_fifo13_full duration	aqm_fifo14_full duration	aqm_fifo15_full duration

Table 236. Event Mux Programming (Sheet 5 of 5)

Event	PEC0	PEC1	PEC2	PEC3	PEC4	PEC5	PEC6	PEC7
55	aqm_fifo0_nearfull duration	aqm_fifo1_nearfull duration	aqm_fifo2_nearfull duration	aqm_fifo3_nearfull duration	aqm_fifo4_nearfull duration	aqm_fifo5_nearfull duration	aqm_fifo6_fnearull duration	aqm_fifo7_nearfull duration
56	aqm_fifo8_nearfull duration	aqm_fifo9_nearfull duration	aqm_fifo10_nearfull duration	aqm_fifo11_nearfull duration	aqm_fifo12_nearfull duration	aqm_fifo13_nearfull duration	aqm_fifo14_nearfull duration	aqm_fifo15_nearfull duration
57-254								
255	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count	AHB Cycle Count

§ §

16.0 Interrupt Controller

The Intel XScale® Processor supports only a single IRQ and FIQ interrupt. Expanding the interrupt capabilities beyond a single source requires an Interrupt Controller external to the Intel XScale processor. The APB Interrupt Controller provides IRQ and FIQ interrupts to the Intel XScale processor. The central APB Interrupt Controller accepts all interrupts and depending on the software settings, the interrupts are passed to the Intel XScale processor.

This version of the Interrupt Controller extends the number of interrupt sources as compared with the controller on the Intel® IXP420 Network Processor and Intel® IXP425 Network Processor and introduces a new class of **error interrupts**. The modifications to this unit were designed with the following objectives:

- Absolute software compatibility is insured by the fact that no current definitions have been changed, only additional controllability has been added, by uniformly extending the number of interrupts that are accommodated
- A new class of interrupts representing error conditions have unconditional highest priority. An example of this is some kind of fatal device failure that requires immediate attention.

16.1 Overview

The Interrupt Controller takes 64 individual interrupts as inputs; refer to Figure 237 for configuration and ordering of all the interrupts. These interrupts originate in internal blocks or from GPIO pins. The controller outputs both an IRQ and an FIQ interrupt to the Intel XScale processor. Any of the 64 input interrupts may be enabled to produce the IRQ or FIQ output. The INTR_EN/INTR_EN2 control register(s) is used to enable an interrupt, and the INTR_SEL/INTR_SEL2 control register(s) is programmed to present an interrupt as an IRQ or an FIQ.

The Intel XScale processor has two methods available to determine the interrupt within the controller that caused the interrupt to assert. The status of the entire 64 bit interrupt vector may be read from the pair of registers representing INTR_ST/ INTR_ST2. The status of all the 64 interrupts enabled as IRQ by INTR_EN/INTR_EN2 may be read from the pair of registers representing INTR_IRQ_ST/INTR_IRQ_ST2. The highest priority interrupt pending on the IRQ may be read by INTR_IRQ_ENC_ST.

Note: This register contains an incremented version of the highest priority interrupt; for example, if the highest priority interrupt number was 3, the register, when read, returns 4. This incrementing is due to the fact that the number '0' is used to indicate a spurious or no-interrupt condition. This is also true for the INR_FIQ_ENC_ST register. Compared with the previous Interrupt Controller, each of these two vectors is one bit wider to enable specification of 64 conditions.

INTR_FIQ_ST/INTR_FIQ_ST2 contains all interrupts enabled as FIQ while the INTR_FIQ_ENC_ST holds the **incremented number** of the highest priority FIQ enabled interrupt pending. Access to all the registers is through the APB interface.

It is worth pointing that the priority of interrupts impacts only the value in the INTR_IRQ_ENC_ST and INTR_FIQ_ENC_ST registers. To accelerate the software stack, these registers encode the highest priority as defined by the priority register and the positional priority of the interrupts presented to the Interrupt Controller. Software does not have to abide by this priority, and is free to ignore the encoded status registers. Instead software may read the interrupt status registers directly, and may choose to service interrupts in various orders.

Table 237. Intel XScale® Processor Interrupt Mapping (Sheet 1 of 2)

Interrupt Bit	Default Priority†	Source	Description
Int0	0	NPE-A	NPE-A Interrupt (includes debug/execution/MBox interrupts)
Int1	1	Reserved	NA
Int2	2	NPE-C	NPE-C interrupt (includes debug/execution/MBox interrupts)
Int3	3	QM	Queue[1-32]
Int4	4	QM	Queue[33-64]
Int5	5	Timer	General purpose timer[0]
Int6	6	GPIO	GPIO[0]
Int7	7	GPIO	GPIO[1]
Int8	8	PCI	PCI interrupt
Int9	9	PCI	PCI DMA channel 1
Int10	10	PCI	PCI DMA channel 2
Int11	11	Timer	General purpose timer[1]
Int12	12	Reserved	NA
Int13	13	Reserved	NA
Int14	14	Timer	Timer[2], Time-stamp
Int15	15	UART0	UART0 interrupt
Int16	16	Timer	Time[3], Watchdog Timer
Int17	17	APB PMU	AHB performance monitoring unit counter rollover
Int18	18	Intel XScale processor PMU	Intel XScale processor PMU counter rollover
Int19	19	GPIO	GPIO[2]
Int20	20	GPIO	GPIO[3]
Int21	21	GPIO	GPIO[4]
Int22	22	GPIO	GPIO[5]
Int23	23	GPIO	GPIO[6]
Int24	24	GPIO	GPIO[7]
Int25	25	GPIO	GPIO[8]
Int26	26	GPIO	GPIO[9]
Int27	27	GPIO	GPIO[10]
Int28	28	GPIO	GPIO[11]
Int29	29	GPIO	GPIO[12]
Int30	30	SW Interrupt	SW interrupt 0
Int31	31	SW Interrupt	SW interrupt 1
†	Priorities of interrupts 0 through 7 are programmable, 8 through 63 are fixed. Priority 0 is highest priority, 63 is lowest.		

Table 237. Intel XScale® Processor Interrupt Mapping (Sheet 2 of 2)

Interrupt Bit	Default Priority†	Source	Description
Int32	32	USB 2.0 Host0	USB 2.0 Host0 interrupt
Int33	33	USB 2.0 Host1	USB 2.0 Host1 interrupt
Int34	34	SSP	SSP Interrupt
Int35	35	Reserved	NA
Int36	36	Reserved	Reserved
Int37	37	Reserved	Reserved
Int38	38	Reserved	Reserved
Int39	39	Reserved	Reserved
Int40	40	Reserved	Reserved
Int41	41	Reserved	Reserved
Int42	42	Reserved	Reserved
Int43	43	Reserved	Reserved
Int44	44	Reserved	Reserved
Int45	45	Reserved	Reserved
Int46	46	Reserved	Reserved
Int47	47	Reserved	Reserved
Int48	48	Reserved	Reserved
Int49	49	Reserved	Reserved
Int50	50	Reserved	Reserved
Int51	51	Reserved	Reserved
Int52	52	Reserved	Reserved
Int53	53	Reserved	Reserved
Int54	54	Reserved	Reserved
Int55	55	Reserved	Reserved
Int56	56	Reserved	Reserved
Int57	57	Reserved	Reserved
Int58	58	Reserved	Reserved
Int59	59	Reserved	Reserved
Int60	60	QM	Queue manager parity error
Int61	61	MCU	Multi-Bit ECC error
In62	62	Reserved	Reserved
Int63	63	Reserved	Reserved
†	Priorities of interrupts 0 through 7 are programmable, 8 through 63 are fixed. Priority 0 is highest priority, 63 is lowest.		

16.2 Features List

The features of the Interrupt Controller are:

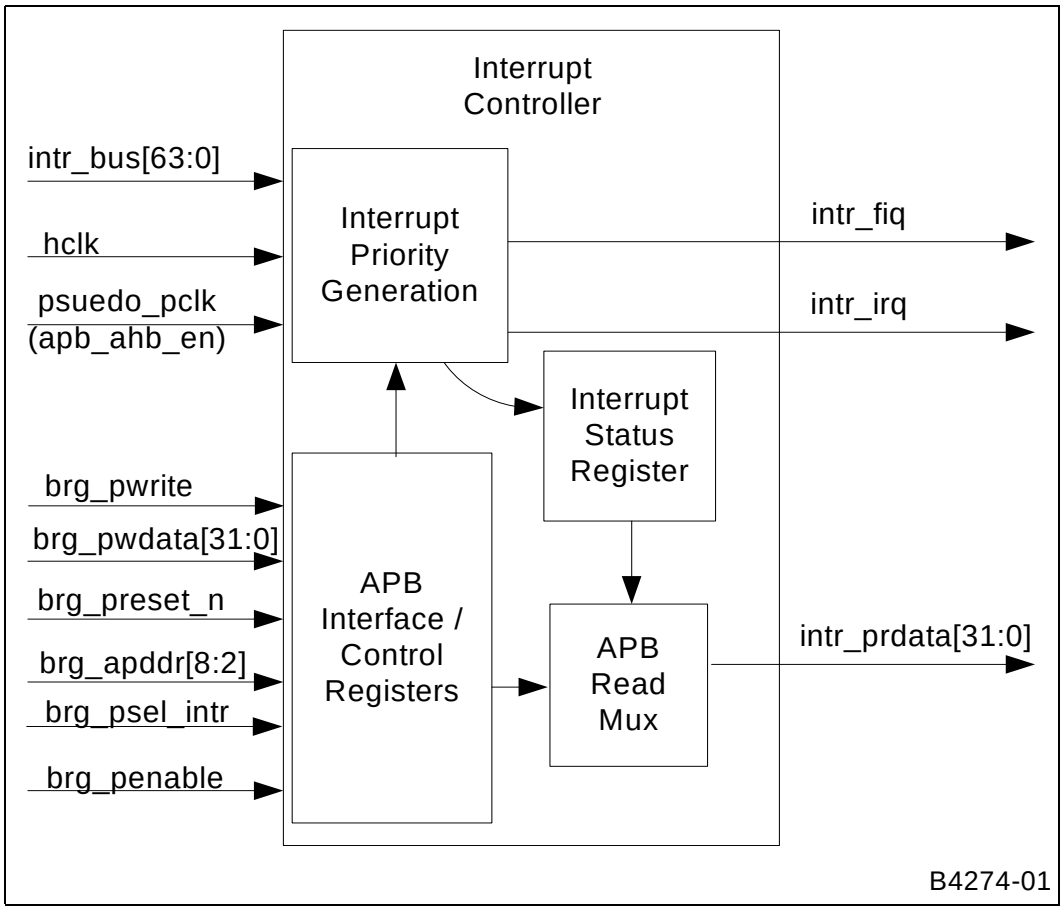
- 64 Supported interrupts
- Priority encoded servicing of interrupts
- Ability to program the order of the 8 highest priority interrupts

- Handling of pclk- and hclk-generated interrupts
- Standard AMBA APB interface clocked from qualified pclk

16.3 Block Diagram

This is the top level block diagram for a module. It shows the inputs, outputs and major internal blocks.

Figure 157. Interrupt Controller Block Diagram



16.4 Theory of Operation

The Interrupt Controller takes 63 individual interrupts as inputs with interrupt 64 reserved. These 63 individual interrupts originate from internal blocks or dedicated GPIO pins on the Intel® IXP43X Product Line of Network Processors. The interrupts are asserted if set (1) and de-asserted if reset (0).

There are two classes of interrupts; both are defined by their positional priority (for example, position 12 is of higher priority than position 42). The **error** class of the interrupts have unconditional priority over **normal** (that is, non **error** class). The Interrupt Error Enable Register is a 32-bit register that assigns each of the [63:32] interrupts as the special error interrupts.

For the **normal** class, the highest positional priority interrupt is bit [0] and the lowest priority interrupt (of the first group) is bit [31] assigned to a software interrupt. The Interrupts [63:32] are lower priority than the software interrupt to preserve the current functionality, but, **error** class of interrupt [63:32] has unconditional priority. The enhancements to the Interrupt Controller are not intended to alter connections to the interrupt sources as the Interrupt Controller does not influence that mapping. The Interrupt Controller for the IXP43X product line of network processors is implemented to expand the interrupt capabilities of the Intel XScale processor.

The Intel XScale processor has two kinds of interrupts, the first being a special high performance interrupt (FIQ) and a **normal** interrupt (IRQ). The interrupts collected by the Interrupt Controller are combined and configured to be an FIQ interrupt or an IRQ. The FIQ signal going to the Intel XScale processor is set when any of the interrupts assigned to be an FIQ is set. The IRQ signal going to the Intel XScale processor is set when any of the interrupts assigned to be an IRQ are set.

The Interrupt Controller does not imply any semantic meaning to the IRQ or FIQ interrupts, only the Intel XScale processor does. From the perspective of the Interrupt Controller, there are only two interrupt outputs, and any interrupt source is funneled to either interrupt. It may be worth pointing out that the FIQ interrupt is not more important than the IRQ interrupt, but instead the intent of the Intel XScale processor FIQ interrupt is to provide a low latency interrupt source. The division between IRQ and FIQ is strictly a latency and system software implementation decision.

The Interrupt Controller consists of:

- Two 32-bit interrupt status registers
- Two 32-bit interrupt select registers
- Two 32-bit FIQ status registers
- A 7-bit IRQ highest priority register
- A 32-bit high priority enable register
- Two 32-bit interrupt enable registers
- Two 32-bit IRQ status registers
- A 32-bit interrupt priority registers
- A 7-bit FIQ highest priority register

The Interrupt Controller has no concept of setting or clearing any interrupts and in fact is level sensitive only. The intent of the Interrupt Controller is to collect and prioritize the received interrupts from other sources. To set an interrupt, the device connected to the assigned interrupt line must assert the interrupt.

Clearing of the interrupt is made at the device that caused the interrupt. If the interrupt is not cleared at the device that asserted the interrupt, the interrupt is serviced again. There is a slight delay between resetting the interrupt at the device that caused the interrupt and resetting of the interrupt due to pipeline delay.

The Interrupt Controller has no semantic concept for any of the interrupts. The only semantics associated with the interrupts relate to the choices to funnel an interrupt as an error interrupt or normal one and as a FIQ or IRQ interrupt.

16.4.1 Interrupt Priority

Selecting the priority of interrupts is done through the assignment of the interrupt priority register and natural interrupt priority assigned by the interrupt number. As described, the interrupts follow a natural priority. The interrupt connected to interrupt 0 has the highest priority and the interrupt connected to interrupt 63 has the lowest priority.

In addition to the natural priority the lowest eight interrupts, interrupt 0 to interrupt 7, is assigned a priority value by writing the interrupt priority register (INTR_PRTY). The interrupt priority register is broken up into eight 3-bit registers.

Bits 0 through 2 of the interrupt priority register assign a priority value to the interrupt 0. Bits 3 through 5 of the interrupt priority register assign a priority to the interrupt 1. The interrupt priority values are assigned in a similar pattern to the assignments above for the first eight interrupts with the last interrupt priority assignment being bits 21 through 23 of the interrupt priority register assigning a priority value to interrupt 7.

The 3-bit interrupt priorities for each of the first eight interrupts takes on a value from 0 to 7. A value of 0, located at the 3-bit interrupt priority register for each of the first eight interrupts signifies the highest priority interrupt. A value of 7, located at the 3-bit interrupt priority register for each of the first eight interrupts, signifies the lowest priority interrupt. In the case of two interrupts being assigned the same priority, the interrupts natural priority assigns who has the highest priority.

For example, interrupt number 1 and interrupt number 3 both have a value of 0, written as their 3-bit interrupt priorities. The Interrupt number 1 would take priority over interrupt number 3 due to its individual natural priorities.

The priorities assigned to each of the 3-bit interrupt priorities for the first eight interrupts is set to a value of the corresponding interrupt number (interrupt number 0 gets assigned a value of 0 in the associated 3-bit interrupt priority register, interrupt number 1 gets assigned a value of 1, and so on.) when receiving a reset. Thus allowing natural priorities to be the default after a reset. The same effect could be achieved by resetting the register to all zeros (where the positional priority takes precedence) and the reset value reflects the *intent* of the priority.

16.4.2 Assigning FIQ or IRQ Interrupts

The Interrupt Controller for the IXP43X network processors provide the capability to assign each interrupt as an FIQ or an IRQ interrupt. As discussed earlier, the Intel XScale processor receives only a single FIQ interrupt signal and a single IRQ interrupt signal.

The Interrupt Controller allows multiple interrupts to be sent to the Intel XScale processor as FIQ interrupts or IRQ interrupts. Each interrupt may be assigned as an FIQ interrupt or an IRQ interrupt but never both. The interrupts are assigned as an FIQ interrupt or an IRQ interrupt by writing bits in the interrupt select register (INTR_SEL/ INTR_SEL2).

The Interrupt Select Register is a pair of 32-bit registers that assigns each of the 64 interrupts to a FIQ interrupt or an IRQ interrupt. Bit 0 of the Interrupt Select Register corresponds to the interrupt number 0 while bit 31 of the Interrupt Select Register 2 corresponds to the interrupt number 63.

Logic 1 written to a bit in the Interrupt Select Register assigns the corresponding interrupt number as an FIQ interrupt. Writing logic 0 to the same bit in the Interrupt Select Register assigns the corresponding interrupt number as an IRQ interrupt.

For example, the Interrupt Select Register is written with a hexadecimal value of 0x00000005. The result of this write would set interrupt number 0 as an FIQ and interrupt number 2 as an FIQ.

All other interrupt numbers would be assigned as IRQ interrupts. All interrupts are assigned as IRQ interrupts upon receiving a reset because the register is cleared to 0x00000000.

The Interrupt Error Enable Register is a 32-bit register that assigns each of the [63:32] interrupts as special error interrupts. These interrupts take unconditional priority over all other interrupts. Logic 1 written to a bit in the Interrupt Error Enable Register implies that the interrupt takes unconditional priority over all other interrupts. A Logic 0 written to the same bit implies that the interrupt has a simple positional priority.

16.4.3 Enabling and Disabling Interrupts

The interrupts on the IXP43X network processors can be individually enabled or disabled by writing to the pair of Interrupt-Enable Registers (INTR_EN/INTR_EN2). By disabling an interrupt, the Intel XScale processor is not interrupted by the FIQ interrupt signal or the IRQ interrupt signal when an interrupt occurs on the corresponding disabled interrupt input.

For instance, interrupt number 0 is disabled and an interrupt occurs on the interrupt number 0; the interrupt generated by the interrupt number 0 is not seen by the Intel XScale processor.

The Interrupt-Enable Register is a pair of 32-bit registers that individually enables or disables each of the 64 interrupts. Bit 0 of the Interrupt-Enable Register corresponds to the interrupt number 0 while bit 31 of the Interrupt-Enable Register 2 corresponds to the interrupt number 63.

Logic 1 written to a bit in the Interrupt-Enable Register enables the corresponding interrupt number. Writing logic 0 to the same bit in the Interrupt-Enable Register disables the corresponding interrupt number.

For example, the Interrupt-Enable Register is written with a hexadecimal value of 0x0000000A. The result of this write would enable interrupt number 1 and interrupt number 3. All other interrupt numbers would be disabled. All interrupts are disabled upon receiving a reset because the register is cleared to 0x00000000.

16.4.4 Reading Interrupt Status

The IXP43X network processors provide several mechanisms through which interrupt status can be obtained from the Interrupt Controller. One method of obtaining interrupt status is to read the interrupt status register directly (INTR_ST/INTR_ST2).

The Interrupt Status Register is a pair of 32-bit registers that has a one-for-one relationship with the interrupt number. Interrupt number 0 is the status represented on bit 0 of the Interrupt Status Register and the interrupt number 31 is the status represented on bit 31 of the Interrupt Status Register. Interrupt numbers 32 to 63 are represented in the second status register.

Reading Logic 1 from a bit in the Interrupt Status Register represents that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller. For example, a read is performed on the Interrupt Status Register and the result returned is a hexadecimal 0x00000002. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 1 has caused an interrupt.

The Intel XScale processor services the interrupt and clears the interrupt by updating the register in the device that caused the interrupt condition. To clear this specific interrupt, you must update the registers in the device that caused the interrupt condition, and not this Interrupt Controller. Using the Interrupt Status Register allows an interrupt service routine to assign relative priorities to the interrupts and removes all relationship from priority algorithms or interrupt types assigned by the Interrupt Controller for the IXP43X network processors. The Interrupt Status Register is set to all zeros upon reset.

All other methods of reading interrupt status involves the use of the register sets provided by the IXP43X network processors to aid in determination of interrupts that should be serviced first.

The IXP43X network processors provide the capability of reading the interrupt status of the interrupt numbers that have been assigned as FIQ interrupts or reading the interrupt status of the interrupt numbers that have been assigned as IRQ interrupts.

The status of the interrupt numbers that have been assigned as FIQ interrupts are read by reading the FIQ status register (INTR_FIQ_ST/INTR_FIQ_ST2). The status of the interrupt numbers that have been assigned as IRQ interrupts are read by reading the IRQ status register (INTR_IRQ_ST/INTR_IRQ_ST2).

The FIQ Status Register and the IRQ Status Register are 32-bit registers that have a one-for-one relationship with the interrupt number. Interrupt number 0 is the status represented on bit 0 of both the FIQ Status Register and the IRQ Status Registers. Interrupt number 31 is the status represented on bit 31 of both the FIQ Status Register and the IRQ Status Registers. Interrupt numbers 32 to 63 are represented in the second status registers for both the IRQ and the FIQ.

Reading logic 1 from a bit in the FIQ Status Register(s) or the IRQ Status Register(s) represent that the device connected to that particular interrupt number has asserted an interrupt to the Interrupt Controller and that the interrupt is enabled for the FIQ or IRQ interrupts respectively. For example, a read is performed on the FIQ Status Register and the result returned is an hexadecimal 0x00000001. The Interrupt Status Register is telling the Intel XScale processor that the interrupt number 0 has caused an interrupt and the interrupt is enabled as a FIQ interrupt.

The Intel XScale processor services the interrupt and clear the interrupt by updating the register in the device that caused the interrupt condition. The same action is applied to an interrupt that would be caused by an IRQ interrupt. Allowing the capability to separate the FIQ and IRQ interrupts allows separate interrupt service routines to be built based on the type of interrupt received. This allows greater control in applications that are developed for the IXP43X network processors.

The FIQ Status Registers and IRQ Status Registers are set to all zeros upon reset.

The IXP43X network processors also allow the capability to read the highest-priority IRQ interrupt or the highest-priority FIQ interrupt as determined by the priority algorithm described in “Interrupt Priority” on page 659.

The highest-priority IRQ interrupt is read by reading the IRQ Highest-Priority Register (INTR_IRQ_ENC_ST). The highest-priority FIQ interrupt is read by reading the FIQ Highest-Priority Register (INTR_FIQ_ENC_ST). The IRQ Highest-Priority Register and the FIQ Highest-Priority Registers are 7-bit registers that returns the highest-priority interrupt number (that is, its position) of each the IRQ interrupts and the FIQ interrupts.

Note: The value of the encoding status registers is always the positional value, and this is true even if the lower bits are programmed in another priority order. For example, if position 0 is actually set to priority 7 via the INTR_PRTY register, and it is the highest priority interrupt (because positions 7:1 are not interrupting), the value of the encoding status register is zero, because that is the position of the interrupt, and not the value 7, which is its priority.

The value obtained by reading the IRQ Highest-Priority Register is the interrupt number of the highest-priority IRQ interrupt, incremented by one and the sum of the add left shifted by two bits. Therefore, the seven bits that contain the IRQ highest priority are actually located in bits 2 through 8 of the IRQ Highest-Priority Register.

For example, interrupt number 1 is the highest-priority IRQ interrupt, the value obtained when reading the IRQ Highest-Priority Interrupt Register would be hexadecimal 0x00000008. A value of 0, returned when reading the IRQ Highest-Priority Register, signifies that no IRQ interrupts are pending.

The IRQ Highest-Priority Register is reset to a value of 0. The FIQ Highest-Priority Register behaves in an identical fashion to the IRQ Highest-Priority Register.

16.5 Interrupt Status Shadow Registers

Shadow registers are introduced to improve the reading of status registers by the Intel XScale processor. The problem is due to non-sequential of address allocation to registers INTR_ST and INTR_ST2, INTR_IRQ_ST and INTR_IRQ_ST2, and INTR_FIQ_ST and INTR_FIQ_ST2 due to which it takes very long time to read all the status information.

Shadow registers are introduced to take advantage of the APB burst read capability by introducing additional address for each of the existing status registers and these new addressed registers are aligned into cacheline boundary.

16.6 Error Enable Register

The ERROR_EN2 register is what defines the difference between the two so-called classes of interrupts. Interrupts [31:0] are by definition (for compatibility reasons) never of the error class. Only the new interrupts [63:32] are in this class. The purpose of the error class of interrupts is to force servicing of error conditions above normal conditions because they usually represent some kind of failure condition.

For example, an NPE may have parity bits added to its data memory. If a parity error is detected an interrupt is generated. But if that interrupt's error enable bit is TRUE, the parity error takes unconditional priority over the **normal** positional priority interrupts.

Note: This does not alter the value of the number reported in the encoding status registers; only the priority scheme is altered.

The bit in ERROR_EN2[0] corresponds to interrupt 32 and the bit in ERROR_EN2[31] corresponds to interrupt 63. If the interrupt is enabled, the error enable is TRUE, and the interrupt becomes active; that interrupt takes priority over all other interrupts. If more than one error class of interrupts is being reported, the highest positional priority is reported in the encoding status register assigned to that interrupt, that is, FIQ or IRQ.

16.7 Interrupt Controller Register Descriptions

Table 238 shows the attributes used and their legend. Table 239 shows the different Interrupt Controller Memory Mapped Registers.

Table 238. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible

Table 239. Interrupt Controller Memory Mapped Registers

Address	Access	Name	Description
0xC8003000	RO	INTR_ST	Interrupt Status Register
0xC8003004	RW	INTR_EN	Interrupt Enable Register
0xC8003008	RW	INTR_SEL	Interrupt Select Register
0xC800300C	RO	INTR_IRQ_ST	IRQ Status register
0xC8003010	RO	INTR_FIQ_ST	FIQ status Register
0xC8003014	RW	INTR_PRTY	Interrupt Priority Register
0xC8003018	RO	INTR_IRQ_ENC_ST	IRQ Highest Priority Register
0xC800301C	RO	INTR_FIQ_ENC_ST	FIQ Highest Priority Register
0xC8003020	RO	INTR_ST2	Interrupt Status Register 2
0xC8003024	RW	INTR_EN2	Interrupt Enable Register 2
0xC8003028	RW	INTR_SEL2	Interrupt Select Register 2
0xC800302C	RO	INTR_IRQ_ST2	IRQ Status register 2
0xC8003030	RO	INTR_FIQ_ST2	FIQ status Register 2
0xC8003034	RW	ERROR_EN2	Error Priority Enable Register
0xC8003400	RO	INTR_ST_SDW	Interrupt Status Shadow Register
0xC8003404	RO	INTR_ST2_SDW	Interrupt Status Shadow Register 2
0xC8003420	RO	INTR_FIQ_ST_SDW	FIQ Status Shadow Register
0xC8003424	RO	INTR_FIQ_ST2_SDW	FIQ Status Shadow Register 2
0xC8003440	RO	INTR_IRQ_ST_SDW	IRQ Status Shadow Register
0xC8003444	RO	INTR_IRQ_ST2_SDW	IRQ Status Shadow Register 2

16.7.1 Interrupt Status Register

The tables below describe the Interrupt Status Register and Interrupt Status Register 2:

Register Name:	INTR_ST				
Physical Address:	0xC800 3000		Reset Hex Value:		
Register Description:	This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but provides visibility into the individual interrupt outputs from each source. For this reason, there is no **reset value**.				
Access: Read.					

31																												0
Incoming Interrupt Status [31:0]																												

Register Name:	INTR_ST2				
Physical Address:	0xC800 3020		Reset Hex Value:		
Register Description:	This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but provides visibility into the individual interrupt outputs from each source. For this reason, there is no **reset value**.				
Access: Read.					

31																												0
Incoming Interrupt Status [63:32]																												

16.7.2 Interrupt Enable Register

The tables below describe the Interrupt Enable Register and Interrupt Enable Register 2:

Register Name:	INTR_EN				
Physical Address:	0xC800 3004		Reset Hex Value:	0x00000000	
Register Description:	Provides enables for the interrupts. This register allows the Intel XScale processor to disable interrupts from selected blocks. To enable an interrupt, a 1 is written into corresponding bit, to disable it, a 0 is written.				
Access: Read/Write.					

31																												0
Interrupt Enables[31:0]																												

Register Name:	INTR_EN2				
Physical Address:	0xC800 3024		Reset Hex Value:	0x00000000	
Register Description:	Provides enables for the interrupts. This register allows the Intel XScale processor to disable interrupts from selected blocks. To enable an interrupt, a 1 is written into corresponding bit, to disable it, a 0 is written.				
Access: Read/Write.					

31																												0
Interrupt Enables[63:32]																												

16.7.3 Interrupt Select Register

The tables below describe the Interrupt Select Register and Interrupt Select Register 2:

Register Name:	INTR_SEL				
Physical Address:	0xC800 3008		Reset Hex Value:	0x00000000	
Register Description:	This register decides if an interrupt is to be presented to the Intel XScale processor as an FIQ or an IRQ. If a bit corresponding to an interrupt is set (to 1), that interrupt is presented as a FIQ. If the bit is reset to 0, the interrupt is presented as an IRQ.				
Access: Read/Write.					

31																												0
Interrupt Selects[31:0]																												

Register Name:	INTR_SEL2				
Physical Address:	0xC800 3028		Reset Hex Value:	0x00000000	
Register Description:	This register decides if an interrupt is to be presented to the Intel XScale processor as an FIQ or an IRQ. If a bit corresponding to an interrupt is set (to 1), that interrupt is presented as a FIQ. If the bit is reset to 0, the interrupt is presented as an IRQ.				
Access: Read/Write.					

31																												0
Interrupt Selects[63:32]																												

16.7.4 IRQ Status Register

The tables below describe the IRQ Status Register and IRQ Status Register 2:

Register Name:	INTR_ IRQ_ST				
Physical Address:	0xC800 300C		Reset Hex Value:	0x00000000	
Register Description:	This register is an **AND** of the incoming status with the INTR_EN and the inverted version of the INTR_SEL. The INTR_IRQ_ST indicates the incoming interrupts that are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled.				
Access: Read.					

31																												0
IRQ Status Information[31:0]																												

Register Name:	INTR_ IRQ_ST2				
Physical Address:	0xC800 302C		Reset Hex Value:	0x00000000	
Register Description:	This register is an **AND** of the incoming status with the INTR_EN2 and the inverted version of the INTR_SEL2. The INTR_IRQ_ST2 indicates the incoming interrupts that are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled.				
Access: Read.					

31																												0
IRQ Status Information[63:32]																												

16.7.5 FIQ Status Register

The tables below describe the FIQ Status Register and FIQ Status Register 2:

Register Name:	INTR_FIQ_ST				
Physical Address:	0xC800 3010		Reset Hex Value:	0x00000000	
Register Description:	This register is an **AND** of the incoming status with the INTR_EN and the INTR_SEL. The INTR_FIQ_ST indicates the incoming interrupts that are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled.				
Access: Read.					

31																												0
FIQ Status Info[31:0]																												

Register Name:	INTR_FIQ_ST2				
Physical Address:	0xC800 3030		Reset Hex Value:	0x00000000	
Register Description:	This register is an **AND** of the incoming status with the INTR_EN2 and the INTR_SEL2. The INTR_FIQ_ST indicates the incoming interrupts that are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled.				
Access: Read.					

31																												0
FIQ Status Info[63:32]																												

16.7.6 Interrupt Priority Register

The table below describes the Interrupt Priority Register:

Register Name:	INTR_PRTY				
Physical Address:	0xC800 3014		Reset Hex Value:	0x00FAC688	
Register Description:	The highest eight priority interrupts is programmed via this register, each of the 3-bit sets is programmed to any priority from 0(000) through 7(111). This register applies to both IRQ and FIQ interrupts.				
Access: Read/Write.					

31							24	23																				0
Reserved								Interrupt Priority selects																				

Register		INTR_PRTY		
Bits	Name	Description	Reset Value	Access
31:24	Reserved	Reserved, Read as undefined, write as 0	0x00	RW
23:21	Prior_Intbus7 [2:0]	Set the priority of the Intr_bus [7]; default is 7	111	RW
20:18	Prior_Intbus6 [2:0]	Set the priority of the Intr_bus [6]; default is 6	110	RW
17:15	Prior_Intbus5 [2:0]	Set the priority of the Intr_bus [5]; default is 5	101	RW
14:12	Prior_Intbus4 [2:0]	Set the priority of the Intr_bus [4]; default is 4	100	RW
11:9	Prior_Intbus3 [2:0]	Set the priority of the Intr_bus [3]; default is 3	011	RW
8:6	Prior_Intbus2 [2:0]	Set the priority of the Intr_bus [2]; default is 2	010	RW
5:3	Prior_Intbus1 [2:0]	Set the priority of the Intr_bus [1]; default is 1	001	RW
2:0	Prior_Intbus0 [2:0]	Set the priority of the Intr_bus [0]; default is 0	000	RW

16.7.7 IRQ Highest-Priority Register

The table below describes the IRQ Highest-Priority Register:

Register Name:	INTR_IRQ_ENC_ST				
Physical Address:	0xC800 3018		Reset Hex Value:	0x00000000	
Register Description:	This register returns the **incremented number** of the highest-priority interrupt that is pending for the IRQ. For example, if interrupt 0 is the highest IRQ pending, the register returns 1. If the register returns 0, it means that there is no interrupt pending or a spurious interrupt. Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read. This allows the register's contents to be directly used as an offset into a jump table for interrupt vectoring.				
Access: Read.					

31																						8						2	1	0
Reserved																						IRQ_ENC_ST							Rsvd	

Register		INTR_IRQ_ENC_ST		
Bits	Name	Description	Reset Value	Access
31:9	Reserved	Reserved, Read as undefined, write as 0	0x000000	RO
8:2	IRQ_ENC_ST	Indicates the highest priority pending interrupt (the interrupt **number** incremented by 1)	0x00	RO
1:0	Reserved	Reserved, Read as undefined, write as 0	0x0	RO

16.7.8 FIQ Highest-Priority Register

The table below describes the FIQ Highest-Priority Register:

Register Name:	INTR_FIQ_ENC_ST				
Physical Address:	0xC800 301C		Reset Hex Value:	0x00000000	
Register Description:	This register returns the **incremented number** of the highest-priority interrupt that is pending for the FIQ. For example, if interrupt '0' is the highest FIQ pending, the register returns 1. Note that the encoded number is shifted left by two bits, a software requirement for the value to be multiplied by 4 before being read. This allows the register's contents to be directly used as an offset into a jump table for interrupt vectoring.				
Access: Read.					

31	...	8	...	2	1	0
Reserved		FRQ_ENC_ST			Rsvd	

Register	INTR_FIQ_ENC_ST			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:9	Reserved	Reserved, Read as undefined, write as 0	0x000000	RO
8:2	FIQ_ENC_ST	Indicates the highest-priority, pending fast interrupt (the interrupt **number** incremented by 1)	0x00	RO
1:0	Reserved	Reserved, Read as undefined, write as 0	0x0	RO

The additional bit defined (that is, bit 8) of the highest priority register is in the same bit position as the extended register address. The higher bit of the status is used as part of the address calculation to determine the group of registers to access.

16.7.9 Error High Priority Enable Register

The table below describes the Error High Priority Enable Register:

Register Name:	ERROR_EN2				
Physical Address:	0xC800 3034		Reset Hex Value:	0x00000000	
Register Description:	This register decides if an interrupt is to be presented to the Intel XScale processor above all other priorities. This affects the priority only, not the number reported in the *_ENC_ST registers. Bit 0 of this register corresponds to interrupt 32 and bit 31 corresponds to interrupt 63.				
Access: Read/Write.					

31	...	0
Error High Priority Enable[63:32]		

16.7.10 Interrupt Status Shadow Register

The table below describes the Interrupt Status Shadow Register:

Register Name:	INTR_ST_SDW				
Physical Address:	0xC800 3400		Reset Hex Value:		
Register Description:	This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but simply provides visibility into the individual interrupt outputs from each source. For this reason, there is no “reset value”. This is the shadow register of INTR_ST.				
Access: Read.					
31					0
Incoming Interrupt Status [31:0]					

Register Name:	INTR_ST2_SDW				
Physical Address:	0xC800 3404		Reset Hex Value:		
Register Description:	This register indicates the state of each incoming interrupt. Note that this is not a register in the normal sense, but simply provides visibility into the individual interrupt outputs from each source. For this reason, there is no “reset value”. This is the shadow register of INTR_ST2.				
Access: Read.					
31					0
Incoming Interrupt Status [63:32]					

16.7.11 IRQ Status Shadow Register

The table below describes the IRQ Status Shadow Register:

Register Name:	INTR_ IRQ_ST_SDW				
Physical Address:	0xC800 3440		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN and the inverted version of the INTR_SEL. The INTR_IRQ_ST_SDW indicates which of the incoming interrupts are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled. This is a shadow register of INTR_IRQ_ST.				
Access: Read.					
31					0
IRQ Status Information[31:0]					

Register Name:	INTR_ IRQ_ST2_SDW				
Physical Address:	0xC800 3444		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN2 and the inverted version of the INTR_SEL2. The INTR_IRQ_ST2_SDW indicates which of the incoming interrupts are enabled as an IRQ. An Interrupt is enabled if the corresponding bit is set, else it is disabled. This is a shadow register of INTR_IRQ_ST2.				
Access: Read.					
31					0
IRQ Status Information[63:32]					

16.7.12 FIQ Status Shadow Register

The table below describes the FIQ Status Shadow Register:

Register Name:	INTR_FIQ_ST_SDW				
Physical Address:	0xC800 3420		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN and the INTR_SEL. The INTR_FIQ_ST_SDW indicates which of the incoming interrupts are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled. This is a shadow register of INTR_FIQ_ST.				
Access: Read.					

31																												0
FIQ Status Info[31:0]																												

Register Name:	INTR_FIQ_ST2_SDW				
Physical Address:	0xC800 3424		Reset Hex Value:	0x00000000	
Register Description:	This register is an “AND” of the incoming status with the INTR_EN2 and the INTR_SEL2. The INTR_FIQ_ST_SDW indicates which of the incoming interrupts are enabled as a FIQ. An Interrupt is enabled if the corresponding bit is set. Otherwise, it is disabled. This is a shadow register of INTR_FIQ_ST2.				
Access: Read.					

31																												0
FIQ Status Info[63:32]																												

§ §

17.0 Operating System Timer

17.1 Overview

The Operating System Timer (OST) serves the function of a watchdog timer and a general purpose timer and is capable of generating interrupts at predetermined intervals. The module contains several registers that are written and read via the APB interface. All four timers can generate interrupts that are independently cleared.

All timers in the OST run off the AMBA APB clock by default. An additional feature to the OST makes it possible for each timer to use a prescaled clock. The timers may also enable a 3/4 scaler to allow the emulation of a 50-MHz clock from a APB 66.667-MHz input clock. All accesses to the OST are 32 bits (word) wide. Bits that correspond to reserved register bits are ignored on writes. These same bits return zero on reads.

17.2 Features List

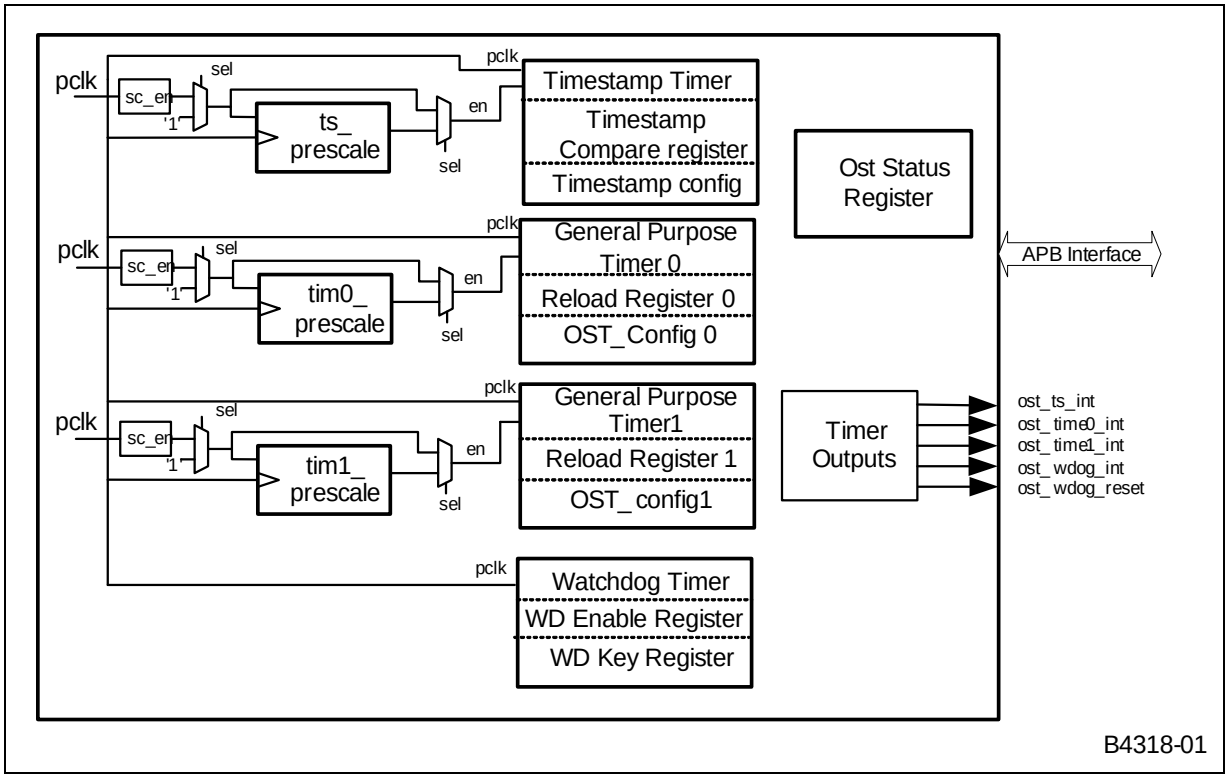
The features of the OST are:

- Two readable decrementing general purpose timers with read/writable reload registers (each suitable for use as an OST)
- 32-bit timestamp incrementing timer, with timestamp compare register.
- 32-bit watchdog timer (decrementing)
- Interrupt, reset enable, and watchdog count for watchdog timer
- Key protected access to watchdog register
- A 1-bit, warm-reset register that signifies when a warm reset has occurred from this unit
- Peripheral on the AMBA APB
- Each timer except the watchdog timer can enable a 16-bit Prescaler.
- Each timer except the watchdog can enable a 3/4 scale function (to emulate a 20-ns clock [50 MHz] off a 66.66667-MHz clock).
- Separate configuration registers for each of the timers.
- The timestamp timer and the two general purpose counters is paused only if the prescaler or the 3/4 scale function on the respective timer is enabled.
- Compatible with Intel® IXP400 Software.

17.3 Block Diagram

A block diagram of the OST unit is shown in Figure 158.

Figure 158. Operating System Timer Block Diagram



17.4 Theory of Operation

The OST supports four timers, the first is a watchdog timer, the second a free-running timestamp timer and the last two are general purpose timers capable of generating interrupts at predetermined intervals.

All the registers/counters operate on the APB bus clock by default. By writing a value into separate configuration and prescale registers, a prescaled enable is used for each timer. Each timer, except for watchdog timer, can also enable a 3/4 scale function to emulate a 20-ns clock from a 66.667-MHz input clock. The 3/4 scaler can also be used in combination with the prescaler.

17.4.1 Watchdog Timer Operation

The watchdog timer is used by the software to monitor inactivity. The watchdog timer is composed of four components:

- A 32-bit writable down counter — (ost_wdog)
- A 3-bit enable register — (ost_wdog_enab)
- A 16-bit key register — (ost_wdog_key)
- A 5-bit status register — (ost_sts)

The ost_wdog_key register is written at any time over the APB bus. But, all the watch-dog registers can only be written to when the ost_wdog_key register contains the value 0x482E (this value referred to from here on as “key-value”). A write to these registers has no effect unless ost_wdog_key = key_value. This is to prevent accidental

writes to these registers. The typical operation will be for the software to write the key-value into the ost_wdog_key register, write to the ost_wdog and/or ost_wdog_enab, then the software would write a value other than key-value into the ost_wdog_key register. It periodically (before the ost_wdog reaches zero) writes the key-value into the ost_wdog_key register, writes a new count value into the ost_wdog, then writes a value other than the key-value into the ost_wdog_key register.

Note: The following steps are recommended to safely update the ost_wdog register:

1. Disable interrupts
2. Enable ost_wdog_key by writing 0x482E
3. Write new user defined value to ost_wdog register
4. Disable ost_wdog_key by writing any value that is not 0x482E
5. Re-Enable interrupts

Upon reset the ost_wdog register is set to all ones, the ost_wdog_enab register is disabled.

The wdog_cnt_ena, wdog_int_ena and wdog_reset_n_ena are the three bits of ost_wdog_enab register. When enabled (by the wdog_cnt_ena bit) the ost_wdog decrements. If it reaches zero it stops decrementing and drives a ost_wdog_int signal and/or a ost_wdog_reset_n signal, depending on the values of wdog_int_ena and wdog_reset_n_ena. These signals continues to be driven active as the count in the ost_wdog equals zero. If the ost_rst_enb bit is set, the entire chip is reset and has the effect of setting all the OST registers to their initial state values as described in the previous paragraph. If the wdog_rst_ena bit is not set, a reset does not occur. In this case the ost_wdog remains zeros until the software writes another value into it.

If the ost_wdog ever reaches zero when wdog_rst_en is enabled, the warm_reset register bit is set. This register bit is cleared by writing 1 to it or by the assertion of reset_cold_n that is derived from the power on reset.

17.4.2 Timestamp Timer Operation

The timestamp timer is a readable 32-bit free-running counter. Upon reset it is by default set to 32'h0000_0001 and starts counting up as soon as the reset is released. Unless the timestamp compare register value is different from 32'h0000_0000, the timer rolls over and generates an interrupt signal. The ost_ts_int signal is asserted until cleared by writing a 1'b1 to the appropriate bit in ost_sts register. If the compare register has a different value than 32'h0000_0000, the timer generates an interrupt when the timestamp counter equals the compare register value, the counter continues to run. The ost_ts_int signal stays asserted until cleared by writing a 1'b1 to the appropriate bit in the ost_sts register.

A prescale is activated by writing a value other than 16'h0000 into the prescale value of the timestamp prescale register. The 3/4 scale_en is activated by writing a 1'b1 to ts_scale_en in the timestamp configuration register. The timer is paused by setting the ts_pause_en bit in the timestamp configuration register. The timer is restarted after a pause by clearing the ts_pause_en bit. The Timestamp timer can only use the pause/ restart feature if prescale or 3/4scale_en is activated. When using the prescaler, the prescale register must be updated before the counter is restarted to ensure the correct division rate. No extra action is necessary if only the 3/4 scale is used (ts_scale_en is 1'b1 and ost_ts_pre = 16'h0000).

Example:

Assume the ost_ts_pre register holds the value 16'h0001, and ts_scale_en is 1'b0. This means the timestamp is counting with a APB clk divided by 2.

To pause the timer.

1. Set ts_pause_en to 1'b1.

When desirable to let the counter continue:

2. Refresh the ost_ts_pre register with a 16'h0001, then clear ts_pause_en.

17.4.3 General Purpose Timers Operation

There are two 32-bit general purpose timers. The general purpose timers have the following six components:

- A 32-bit down-counter
- A one_shot control bit
- A cnt_enable control bit
- A 32-bit reload value register
- A 16-bit prescale register
- A 4-bit configuration register

The purpose of the timers are to generate timed or periodic interrupts to the Intel XScale® Processor.

All the registers are initialized to zero upon reset.

The 32-bit reload value register is packed into the same APB addressable word with the one_shot control bit and the cnt_enable control bit. The reload value occupying bits 31-2, one_shot in bit 1 and cnt_enable in bit 0. To achieve higher granularity the two least significant bits are fetched from bit 0 and bit 1 in the General Purpose Configuration Register.

For predictable operation, stop the timer before writing a new value into the reload register.

The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, and the counter reaches zero, the counter loads the reload value register and start counting down. If the one_shot control register is set to one, it loads the reload value register and stops counting.

The cnt_enable control bit specifies whether the counter is enabled for counting. When set to one the counter is enabled.

The timer counts down when the enable bit is set to one. The ost_tim*_int signal is asserted when the count reaches zero and remains asserted until cleared by writing a 1'b1 to the appropriate bit in the ost_sts register.

The general purpose timers have the option of using the prescaled clock, and/or the 3/4 scale_enable logic. The value of prescale is contained in the general purpose prescale registers. Only when a prescaled clock is used, the users have an option to pause/restart the timers. This option is controlled in the general purpose configuration registers. By writing a 1'b1 to tim0_pause_en or tim1_pause_en, the respective counter stops. To wake up the counter, refresh the prescale register for the respective counter by re-writing the same prescale value to the associated prescale register ost_timx_pre, and then clear the respective timx_pause_en.

Example on how to operate the general purpose timer0:

1. ost_tim0_cfg <= 0x07 Configures the timer with 3/4 scale_en and the two lsb bits in the reload value to be ones.
2. ost_tim0_pre <= 0x19 loads the prescaler with 25, that means a divide by 26.

3. ost_tim0_rl <= 0x15 Sets the reload value to be 0x14 and the lsb enables the counter. The reload value becomes 0x14 plus 0x03, set in step 1, a total of 0x17 (23). Since 0 is included in decrementing, the real value is N+1 = 24. This setup now produces an interrupt interval of

$$I_{period} = 3/4scale * prescale\_value * reload\_value$$
$$= 20ns * 26 * 24 = 12.48us$$

If it is desirable to pause the timer, then use the tim0_pause_en to halt counting:

4. ost_tim0_cfg <= 0x0F This sets tim0_pause_en

To enable the counting such that the counter continues from its paused value:

5. ost_tim0_pre <= 0x19 This is a refresh write that is necessary for the prescale to properly work.
6. ost_tim0_cfg <= 0x07 Clear tim0_pause_en

If no prescaler is used, step 5 need not be performed.

17.4.4 Clock Prescale

Each timer except the watchdog is provided with a programmable prescaler that divides the input clock with a 16-bit value. The input clock is the APB system clock or a 20-ns version of it (3/4scale). All timers use the APB clock by default. The 16-bit prescale value ranges from divide by 2 to 65,536, and results in a new clock enable available for the timers that ranges from 33.33 MHz down to 1017.26 Hz, the divider is a divide by N+1, where N={0,1,2,3...}. When N = 0, that means the divide factor is equal to '1', in other words the APB clock is used. The value for the new available clock enable is written in the internal register for the desired timer, the new value is updated on the next positive edge of pclk.

17.5 Detailed Register Descriptions

The registers are accessible via the APB bus interface.

Table 240. Register Legend

Attribute	Legend	Attribute	Legend
RV	(Reserved)	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	RW1C	Normal Read Write '1' to clear

Table 241. Register Summary

Address	Register Name	Description	Reset Value	Attribute	Counter type
"0xC8005000"	"ost_ts"	"Timestamp timer"	"0x00000001"	RW	Up
"0xC8005004"	"ost_tim0"	"General Purpose Timer 0"	"0x00000000"	RO	Down
"0xC8005008"	"ost_tim0_rl"	"General Purpose Timer 0 Reload"	"0x00000000"	RW	n/a
"0xC800500C"	"ost_tim1"	"General Purpose Timer 1"	"0x00000000"	RO	Down
"0xC8005010"	"ost_tim1_rl"	"General Purpose Timer 1 Reload"	"0x00000000"	RW	n/a

† Only through watchdog protection mechanism

Table 241. Register Summary

Address	Register Name	Description	Reset Value	Attribute	Counter type
"0xC8005014"	"ost_wdog"	"Watchdog Counter"	"0xFFFFFFFF"	RW†	Down
"0xC8005018"	"ost_wdog_enab"	"Watchdog Enable Register"	"0x00000000"	RW†	n/a
"0xC800501C"	"ost_wdog_key"	"Watchdog Key Register"	"0x00000000"	RW	n/a
"0xC8005020"	"ost_status"	"Timer Status Register"	"0x00000000"	RW1C	n/a
"0xC8005024"	"ost_ts_cmp"	"Timestamp Compare Register"	"0x00000000"	RW	n/a
"0xC8005028"	"ost_ts_cfg"	"Timestamp Configuration Register"	"0x00000000"	RW	n/a
"0xC800502C"	"ost_ts_pre"	"Timestamp Prescale Register"	"0x00000000"	RW	n/a
"0xC8005030"	"ost_tim0_cfg"	"General Purpose Timer 0 Configuration"	"0x00000000"	RW	n/a
"0xC8005034"	"ost_tim0_pre"	"General Purpose Timer 0 Prescale Register"	"0x00000000"	RW	n/a
"0xC8005038"	"ost_tim1_cfg"	"General Purpose Timer 1Configuration"	"0x00000000"	RW	n/a
"0xC800503C"	"ost_tim1_pre"	"General Purpose Timer 1Prescale Register"	"0x00000000"	RW	n/a

† Only through watchdog protection mechanism

17.5.1 Timestamp Timer

Register Name:	ost_ts		
Physical Address:	0xC8005000	Reset Hex Value:	0x00000001
Register Description:	Timestamp timer		
Access: Read/Write			

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
timer_val																															

Register	ost_ts			
Bits	Name	Description	Reset Value	Access
31:0	timer_val	Current value of the timer.	32'h0	RW

17.5.2 General Purpose Timer 0

Register Name:	ost_tim0		
Physical Address:	0xC8005004	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 0		
Access: Read			

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
timer_val																															

Register	ost_tim0			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:0	timer_val	Current value of the timer.	32'h0	RO

17.5.3 General Purpose Timer 0 Reload

Register Name:	ost_tim0_rl		
Physical Address:	0xC8005008	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 0 Reload		

Access: Read/Write

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
reload_val																														one_shot	enable

Register	ost_tim0_rl			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:2	reload_val	Reload value. Value to be reloaded into ost_tim0. Bits 1 and 0 are fetched from ost_tim0_cfg. For predictable operation, stop the timer before writing a new value into the reload register. ***Note:*** As the counter counts to zero, writing a value of X to these bits generates interrupts at intervals of (X*4)+1 cycles, unless the two least significant bits in ost_tim0_cfg are filled.	30'h0	RW
1	tim1_one_shot	The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, when the counter reaches zero it loads the reload value register and start counting down. If the one_shot control register is set to one it loads the reload value register and stop counting.	0	RW
0	tim0_enable	The cnt_enable register controls whether the counter is enabled for counting. When set to one the counter is enabled.	0	RW

17.5.4 General Purpose Timer 1

Register Name:	ost_tim1		
Physical Address:	0xC800500C	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 1		

Access: Read

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
timer_val																															

Register	ost_tim1			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:0	timer_val	Current value of the timer	32'h0	RO

17.5.5 General Purpose Timer 1 Reload

Register Name:	ost_tim1_rl		
Physical Address:	0xC8005010	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 1 Reload		

Access: Read/Write

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
reload_val																														one_shot	enable

Register	ost_tim1_rl			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:2	reload_val	Reload value. Value to be reloaded into ost_tim1. Bits 1 and 0 are fetched from ost_tim1_cfg. For predictable operation, stop the timer before writing a new value into the reload register. ***Note:*** As the counter counts to zero, writing a value of X to these bits generates interrupts at intervals of (X*4)+1 cycles, Unless the two least significant bits in ost_tim0_cfg is filled.	30'h0	RW
1	tim1_one_shot	The one_shot control bit specifies what action should be taken when the counter reaches zero. If the one_shot control bit is set to zero, when the counter reaches zero loads the reload value register and start counting down. If the one_shot control register is set to one it loads the reload value register and stops counting.	0	RW
0	tim1_enable	The cnt_enable register controls whether the counter is enabled for counting. When set to one the counter is enabled.	0	RW

17.5.6 Watchdog Timer

Register Name:	ost_wdog		
Physical Address:	0xC8005014	Reset Hex Value:	0xFFFFFFFF
Register Description:	Watchdog Counter		

Access: Read/Write

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
tim_val																															

Register	ost_wdog			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:0	tim_val	Timer value. *Write has no effect unless ost_wdog_key=0x482E	0xffffffff	RW

17.5.7 Watchdog Enable Register

Register Name:	ost_wdog_enab		
Physical Address:	0xC8005018	Reset Hex Value:	0x00000000
Register Description:	Watchdog Enable Register		

Access: Read/Write

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
(Reserved)																													cnt_en	int_en	rst_en

Register	ost_wdog_enab			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:3	(Reserved)		29'h0	RO
2	wdog_cnt_ena	Count Enable. '1' enables ost_wdog to decrement.Transmit FIFO * Write has no effect unless ost_wdog_key=0x482E.	0	RW
1	wdog_int_ena	Interrupt Enable. '1' enables ost_wdog_int signal to be generated. *Only writable when ost_wdog_key=0x482E.	0	RW
0	wdog_rst_ena	Watchdog Reset Enable. '1' enables ost_wdog_reset_n signal to be generated. *Write has no effect unless ost_wdog_key=0x482E.	0	RW

17.5.8 Watchdog Key Register

Register Name:	ost_wdog_key		
Physical Address:	0xC800501C	Reset Hex Value:	0x00000000
Register Description:	Watchdog Key Register		

Access: Read/Write

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
(Reserved)																key_value															

Register	ost_wdog_key			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:16	(Reserved)		16'h0	RO
15:0	key_value	Reset Key Value. Value of the reset key that allows access to ost_wdog_enab and ost_wdog, ost_wdog_key=0x482E	16'h0	RW

17.5.9 Timer Status

Register Name:	ost_status		
Physical Address:	0xC8005020	Reset Hex Value:	0x00000000
Register Description:	Timer Status Register		

Access: Read/Write Clear

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
(Reserved)																											ost status bits				

Register	ost_status			
Bits	Name	Description	Reset Value	Access
31:5	(Reserved)	Reserved. Returns 0 if read	27'h0	RO
4	warm_reset	'1' if warm reset has occurred. Writing a '1' to this bit clears it, if set.	0	RW1C
3	ost_wdog_int_val	'1' if ost_wdog_int has occurred. Writing a '1' to this bit clears it, if the condition that caused it is no longer present.	0	RW1C
2	ost_ts_int_val	'1' if ost_ts_int has occurred. Writing a '1' to this bit clears it, if the condition that caused it is no longer present.	0	RW1C
1	ost_tim1_int_val	'1' if ost_tim1_int has occurred. Writing a '1' to this bit clears it, if the condition that caused it is no longer present.	0	RW1C
0	ost_tim0_int_val	'1' if ost_tim0_int has occurred. Writing a '1' to this bit clears it, if the condition that caused it is no longer present.	0	RW1C

17.5.10 Timestamp Compare Register

Register Name:	ost_ts_cmp		
Physical Address:	0xC8005024	Reset Hex Value:	0x00000000
Register Description:	Timestamp Compare Register		

Access: Read/Write

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
compare_val																															

Register	ost_ts_cmp			
Bits	Name	Description	Reset Value	Access
31:0	compare_val	Value of the compare register. When the time stamp register reaches the value of the compare value, an interrupt is set and is not cleared until it is cleared by writing to the timer status register.	32'h0	RW

17.5.11 Timestamp Configuration Register

Register Name:	ost_ts_cfg		
Physical Address:	0xC8005028	Reset Hex Value:	0x00000000
Register Description:	Timestamp Configuration Register		

Access: Read/Write

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
(Reserved)																														ts_pasue_en	ts_scale_en

Register	ost_ts_cfg			
Bits	Name	Description	Reset Value	Access
31:2	(Reserved)		30'h0	RO
1	ts_pause_en	If '0' the timestamp counter is running, when '1' the counter is paused. Only when the prescale/scale_en is active. This bit has no effect if the timer is not using the prescaler/scale_en	0	RW
0	ts_scale_en	If this field is '1', the 20-ns scale is enabled. It makes timestamp timer count with 3/4 of the frequency the timestamp timer is driven by. If no prescaler is used apb_clk.	0	RW

17.5.12 Timestamp Prescale Register

Register Name:	ost_ts_pre		
Physical Address:	0xC800502C	Reset Hex Value:	0x00000000
Register Description:	Timestamp Prescale Register		

Access: Read/Write

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
(Reserved)																ts_presc															

Register	ost_ts_pre			
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)		16'h0000	RO
15:0	ts_presc	Contains the divide factor for the Timestamp Timer. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor, that is N+1. The prescaling starts when the field is written.	16'h0000	RW

17.5.13 General Purpose Timer 0 Configuration Register

Register Name:	ost_tim0_cfg		
Physical Address:	0xC8005030	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 0 Configuration		

Access: Read/Write

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
(Reserved)																												tim0_pause_en	tim0_scale_en	tim0_bit1	tim0_bit0

Register	ost_tim0_cfg			
Bits	Name	Description	Reset Value	Access
31:4	(Reserved)		28'h0	RO
3	tim0_pause_en	If '0' the timer0 counter is running, when '1' the counter is stopped. Only when the prescale/scale_en is active. This bit has no effect if the timer is not using the prescaler/scale_en	0	RW
2	tim0_scale_en	If this field is '1', the 3/4 scale is enabled. It makes the tim0 count with 3/4 of the frequency the timer0 is driven by. If no prescaler is used, it is 3/4 of the apb_clk.	0	RW
1	tim0_bit1	This is bit 1 in a higher granularity for the reload counter for tim0.	0	RW
0	tim0_bit0	This is bit 0 in a higher granularity for the reload counter for tim0.	0	RW

17.5.14 General Purpose Timer 0 Prescale Register

Register Name:	ost_tim0_pre		
Physical Address:	0xC8005034	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 0 Prescale Register		

Access: Read/Write

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
(Reserved)																tim0_presc															

Register	ost_tim0_pre			
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)		16'h0000	RO
15:0	tim0_presc	Contains the divide factor for the timer0. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor, that is N+1, so when tim0_presc = 0x01 it is divide by 2. The prescaling starts when the field is written with a value other than 0x00.	16'h0000	RW

17.5.15 General Purpose Timer 1 Configuration Register

Register Name:	ost_tim1_cfg		
Physical Address:	0xC8005038	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 1Configuration		

Access: Read/Write

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
(Reserved)																												tim1_pause_en	tim1_scale_en	tim1_bit1	tim1_bit0

Register	ost_tim1_cfg			
Bits	Name	Description	Reset Value	Access
31:4	(Reserved)		28'h0	RO
3	tim1_pause_en	If '0' the timer1 counter is running, when '1' the counter is stopped. Only when the prescale/scale_en is active. This bit has no effect if the timer uses apb_clk.	0	RW
2	tim1_scale_en	If this field is '1', the 3/4 scale is enabled. It makes timer1 count with 3/4 of the frequency the timer1 is driven by. If no prescaler is used, it is 3/4 of the apb_clk.	0	RW
1	tim1_bit1	This is bit 1 in a higher granularity for the reload counter for tim1.	0	RW
0	tim1_bit1	This is bit 0 in a higher granularity for the reload counter for tim1.	0	RW

17.5.16 General Purpose Timer 1 Prescale Register

Register Name:	ost_tim1_pre		
Physical Address:	0xC800503C	Reset Hex Value:	0x00000000
Register Description:	General Purpose Timer 1Prescale Register		

Access: Read/Write

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
(Reserved)																tim1_presc															

Register	ost_tim1_pre			
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)		16'h0000	RO
15:0	tim1_presc	Contains the divide factor for timer1. The usable range is 2 - 65,535. The generated frequency equals apb_clk divided by the divide factor, that is N+1, so when tim1_presc = 0x01 it is divide by 2. The prescaling starts when the field is written with a value other than 0x00	16'h0000	RW

§ §

18.0 Time Synchronization Hardware Assist (TSYNC)

18.1 Overview

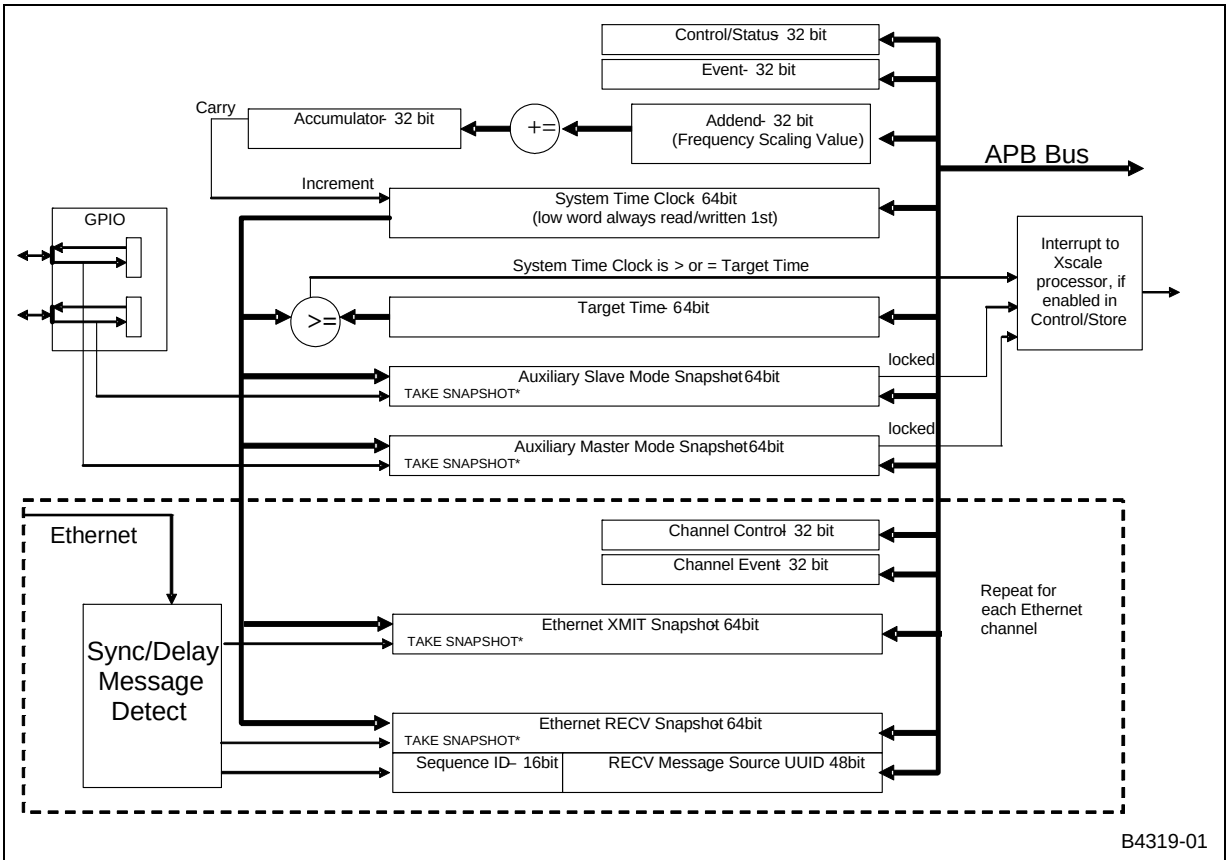
In a distributed control system containing multiple clocks, individual clocks tend to drift apart. Some kind of correction mechanism is necessary to synchronize the individual clocks to maintain global time, which is accurate to some requisite clock resolution. The IEEE* 1588 standard defines a precision clock synchronization protocol for networked measurement and control systems, including several messages used to exchange timing information. The hardware assist logic required to achieve precision clock synchronization using the IEEE 1588 standard depends on the implementation.

This document describes the hardware assist logic developed to achieve time synchronization on Ethernet.

18.2 Block Diagram

The following diagram provides a programming model of the TSync block, showing registers and interconnections.

Figure 159. Block Diagram of TSync Circuit



Notes:

1. If Master, the XMIT snapshot holds the “SYNC” msg time and the RECV snapshot holds the: DELAY” msg time. If Slave, the XMIT snapshot holds the “DELAY” msg time and the RECV snapshot holds the “SYNC” msg time
2. Take snapshot = Sync, edge detect, & lock until reset by write of ‘1’ to corresponding bit in event register

18.3 Theory of Operation (Ethernet Interfaces)

Time synchronization adjusts the rate that System Time increases in a time slave so that it is synchronized to the time master’s System Time. System Time is incremented by an overflow of the Accumulator. The Accumulator increments due to the repetitive addition of the Addend register to itself on every system clock. Therefore, periodically adjusting the value of the Addend register controls the rate that System time increments. System Time snapshots are taken in hardware by both master and slave when certain messages are detected. These snapshots are used to calculate the skew between master and slave, and the slave is then adjusted accordingly.

According to the IEEE-1588 protocol, four messages form the framework of the protocol: Sync, Follow_up, Delay_Req, and Delay_Resp. Because the messages for Ethernet use UDP/IP, this implies that messages can be lost and firmware must compensate for this.

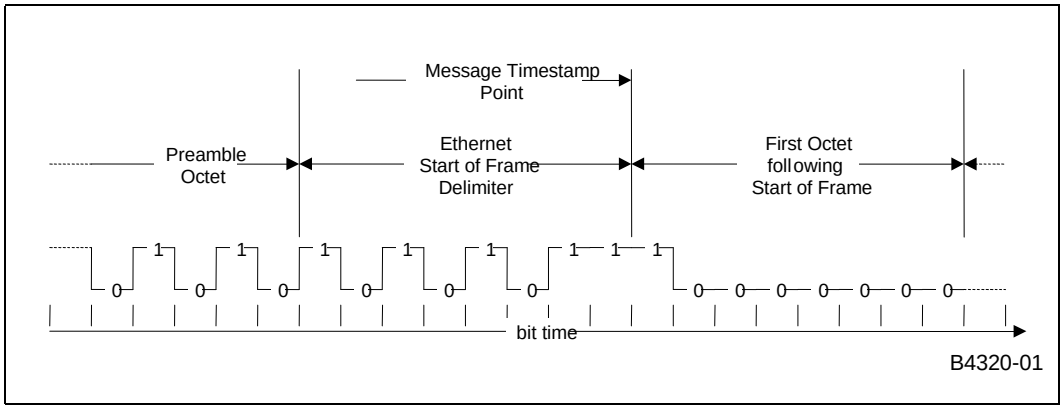
Per the 1588 specification, synchronized time is referenced to the end of the “start of frame delimiter” (SFD) as shown in Figure 160. Therefore, the time sync hardware captures the system time immediately upon detection of the SFD in the appropriate snapshot register (two per channel, one for transmit, one for receive). Due to PHY and

synchronization delays, the actual timestamp will be slightly later than the desired reference point. However, allowing for 1 pclk synchronization jitter, this is a fixed delay, easily nulled out in the software portion of the algorithm. This fixed delay is dependent on the 10/100 MHz selection at the PHY and therefore software needs to be able to access this information from each of the PHYs via the shared Management Data Interface (MDI).

When a Sync or Delay_Req messages is detected, the timestamp that was captured at the SFD in the snapshot register is frozen or “locked” until acknowledged by the firmware.

Each message is detected by identifying key bytes in the packet. The byte offsets identified below are numbered starting with the 1st byte after the SFD and the numbering begins at 0. Therefore “byte 74” refers to the 75th byte after the SFD.

Figure 160. Time Stamp Reference Point



18.3.1 Priority Message Support

Since there exists a wide variety of data on an industrial Ethernet network, including both time-critical and non-time-critical data, it is desirable to support “Tagged MAC Frames” from 802.3 which define priority based messages. A Tagged MAC Frames is identified by the “length/type” field. If byte 12 = 0x81 and byte 13=0x00, then the message is using the Tagged MAC Frame format, in which 4 additional bytes need to be accounted for in the header. Therefore, if a Tagged MAC Frame is detected, all the byte offsets mentioned below are incremented by 4.

18.3.2 Sync Message

Firmware in a time master transmits a multicast Sync message periodically over the network at 1, 2, 8, 16, or 64 second intervals. A Sync message is defined as a value of 0x00 in byte 74 of the Ethernet frame after the start of frame delimiter.

The TSync logic will monitor the MII signals, detect when the channel has transmitted or received a Sync message, and lock the timestamp. Furthermore, the TSync logic captures the Sequence ID and Source UUID if a Sync message is received by a channel configured as a time slave.

18.3.3 Follow_Up Message

Firmware in a time master transmits the timestamp, captured during a previously sent Sync message, using a multicast Follow_Up message. No time stamping is done by the master or slave with the Follow_Up message. A Follow_Up message is defined with a value of 0x02 in byte 74 of the Ethernet frame after the start of frame delimiter.

18.3.4 Delay_Req Message

Firmware in a time slave can transmit a Delay_Req message to the master for the purpose of determining propagation delays in the network. A Delay_Req message is defined as a value of 0x01 in byte 74 of the Ethernet frame after the start of frame delimiter.

The TSync logic will monitor the MII signals, detect when the channel has transmitted or received a Delay_Req message, and lock the timestamp. Furthermore, the TSync logic captures the Sequence ID and Source UUID if a Sync message is received by a channel configured as a time master.

18.3.5 Delay_Resp Message

Upon receipt of a Delay_Req message, firmware in a time master will transmit a Delay_Resp message that includes the timestamp of the Delay_Req message it received. No time stamping is done with the Delay_Resp message itself. A Delay_Resp message is defined with a value of 0x03 in byte 74 of the Ethernet frame after the start of frame delimiter.

The 1588 standard uses UDP/IP protocol, which implies that messages can be lost. Because a Delay_Req message can be locked out until firmware in the master channel enables it, the slave channel firmware will have to retry sending the Delay_Req message to the master if the slave does not receive a Delay_Resp message within some timeout period.

18.3.6 IPv6 Compatibility

A time sync message is always defined using an IPv4 message format. To avoid the potential of mistaking an IPv6 packet with a time sync packet, the byte at offset 14 will be checked for a value of 0x45. If not 0x45, the packet will be ignored. IPv6 is not supported

18.3.7 Traffic Analyzer Support

In a traffic analyzer type application, it is often desirable to timestamp every message on the network. An optional mode inhibits the “locking” of the time snapshot so that the SFD of every message triggers a time snapshot. In this mode the snapshot must be read (presumably by the NPE) during the message before the next SFD is received.

18.3.8 MII Clocking Methods

According to the IEEE 802.3 specification, the MII TX/RX data transitions synchronously with the MII clock (rising edge is implied). For the TX case, txclk is driven by the PHY, but txdata is driven by MAC. The txdata bus transitions some time after rising edge of txclk and is sampled at next rising edge by the PHY. For the RX case, the PHY drives rxclk and rxdata. The rxdata bus transitions on the falling edge of rxclk to provide sufficient setup time for the MAC to sample at next rising edge. The characteristics of the MII in cases of transmit and receive should be taken into consideration when constraining the design for synthesis.

18.3.9 System Time Clock Rate Set by Addend Register

The FreqOscillator is the frequency of all portions of the frequency compensated clock, or time synchronization circuit. The frequency compensation value (FreqCompensationValue) is the number held in the Addend register, which is added to the accumulator once every 1/FreqOscillator. To determine the value that should be placed in the Addend register, the following equation is used:

FreqDivisionRatio = FreqOscillator/FreqClock

where FreqClock is the nominal frequency at which the clock counter is to be incremented.

The equation for the FreqCompensationValue utilizes the precision of the accumulator and the FreqDivisionRatio. Since the accumulator is 32 bits, the following equation applies:

FreqCompensationValue = 2^{32} / FreqDivisionRatio

The hexidecimal representation of the FreqCompensationValue is the value that is written to the Addend register. The following table gives examples of addend values based on a 66 MHz FreqOscillator.

Table 242. System Time Clock Rates

FreqOscillator	FreqClock	FreqDivisionRatio	FreqCompensationValue
66 MHz	40 MHz	1.65	0x9B26C9B3
66 MHz	50 MHz	1.32	0xC1F07C1F
66 MHz	60 MHz	1.1	0xE8BA2E8C

18.3.10 MII Message Detection

Two state machines for each Ethernet channel are implemented in the Time Sync logic: one of the pair is for transmit message detection and the other is for received message detection. Both are required for full duplex operation of the channel.

Mastership of a channel is indicated per channel in the TS_Channel_Control register. Based on this information, the Time Sync logic knows the mastership of the channel, and it monitors the MII signals for transmission or reception of Ethernet messages accordingly. A master channel will time stamp Sync messages that are transmitted and Delay_Req messages that are received. Conversely, a slave channel will time stamp Delay_Req messages that are transmitted and Sync messages that are received.

The timestamp point is immediately after the SFD and this value is frozen in the snapshot register when the last nibble of the frame CRC is transmitted or received and the overall message is detected. Because the Sync and Delay_Req messages are of fixed length, the location of the last nibble of CRC is known. Byte 169 corresponds to the last byte of the CRC. The snapshot is locked and this value is frozen until the software acknowledges it. Therefore, a constant can be subtracted from the snapshot to compensate for PHY and synchronization delays to arrive at the IEEE-1588 specified time stamp point.

An explanation of time stamping of messages and time stamp lockout is necessary to assist the user with the implementation of the TSync registers. Time stamping means that the current value in the system time register is captured in a second register, generally called a snapshot register. Each Ethernet channel has two snapshot registers, and there are two auxiliary snapshot registers controlled by general-purpose I/O.

The time stamping will occur when the appropriate conditions exist such as a general-purpose input is received or when a particular type of message is transmitted or received by the channel. Once a timestamp of system time is taken and locked, a unique indication for the snapshot is set in the appropriate Event register. No interrupt is sent to the Intel XScale® Processor upon timestamp capture/lock on the MII interface, this is due to being to early as the MII messages would not have propagated up the network protocol stack, thus the poling of the event register is necessary to determine if the timestamp is captured. No further timestamps of that type can be received until the snapshot indication is cleared by firmware. Thus, the setting of the indication is a lockout of further snapshots of a particular type until firmware takes action (unless the traffic analyzer lock inhibit feature is enabled).

Below is a description of the messages and how they are detected and utilized in the TSync block. For simplification, the description will reference a single channel and leave off the numeric descriptors as defined in the logic and register definitions.

18.3.10.1 Sync Message

The master channel multicasts a Sync message periodically over the network at 1-, 2-, 8-, 16-, or 64-second intervals. A Sync message is defined as a value of 0x00 in byte 74 of the Ethernet frame after the start of frame delimiter.

If the channel is a master, the Time Sync logic will monitor the MII signals and detect when the master channel has transmitted a Sync message. When the SFD is detected and the XMIT_Snapshot is not locked out, the message is time stamped and the current system time is captured in the XMIT_Snapshot register.If the channel is a slave, Sync messages will be received (not transmitted). When the SFD is detected and the RECV_Snapshot is not locked out, the message is time stamped and the current system time is captured in the RECV_Snapshot register.When the snapshot of system time related to the Sync message has occurred, an indication asserts in the TS_Channel_Event register and remains set until firmware explicitly writes a ‘1’ back to that bit. Until the Sync message snapshot indication is cleared, no further Sync messages will be time stamped. Locking can be inhibited by setting the TS_Channel_Control register appropriately.

18.3.10.2 Delay_Req Message

Slave channels transmit a Delay_Req message to the master in response to receiving a Sync message. A Delay_Req message is defined as a value of 0x01 in byte 74 of the Ethernet frame after the start of frame delimiter.

If the channel is a master, the Time Sync logic will monitor the MII signals and detect when the master channel has received a Delay_Req message. When the SFD is detected and the RECV_Snapshot is not locked out, the message is time stamped and the current system time is captured in the RECV_Snapshot register.If the channel is a slave, Delay_Req messages will be transmitted (not received). When the SFD is detected and the XMIT_Snapshot is not locked out, the message is time stamped and the current system time is captured in the XMIT_Snapshot register.When the snapshot of system time related to the Delay_Req message has occurred, an indication asserts in the TS_Channel_Event register and remains set until firmware explicitly writes a ‘1’ back to that bit. Until the Delay_Req message snapshot indication is cleared, no further Delay_Req messages will be time stamped. This is important to note since multiple slave channels may try to send Delay_Req messages simultaneously. Locking can be inhibited by setting the TS_Channel_Control register appropriately.

18.3.10.3 Errors in Messages

The TSync logic will ignore 1588 time-sync messages that are not properly formatted as described above. CRC errors or other errors detected by the MAC or higher levels of the protocol will cause firmware to delete the message and any time snapshots incorrectly captured by the TSync logic to be ignored.

Note: On the Intel® IXP43X Product Line of Network Processors, the TSync logic does not check if properly formatted messages have PHY errors.

18.4 Theory of Operation (Auxiliary Snapshots)

18.4.1 Master Mode Programming Considerations

In master mode, the host processor firmware has complete control over both the incoming auxiliary snapshot signal and the clearing of the snapshot lock. Firmware will assert the signal, wait for the snapshot lock to set, read the snapshot, negate the signal, and then clear the lock. Since it has control over the environment, firmware will not clear the lock before it negates the signal. If the firmware does not clear the incoming auxiliary snapshot signal before the TSYNC lock is cleared, then a second / redundant snapshot event will be generated.

18.4.2 Slave Mode Programming Considerations

In slave mode, the host processor firmware typically knows the parameters of the signal coming from the master. The pulse per second signal from the GPS, for example, has a documented width, and the firmware must be designed to wait that amount of time after it detects the snapshot lock to the time it clears the lock. The firmware "wait" is facilitated by the fact that GPIO[8:7] should be configured as inputs to read this signal and assure that it is de-asserted before clearing the lock. Furthermore, the GPIO input can be configured to interrupt the CPU on the falling edge of the auxiliary snapshot signal, again making it easy for the firmware to know when to clear the lock. Hardware filtering and edge-detection were considered but not implemented because the signal quality from the master could be bad enough to cause spurious locks. For example, cables to a GPS could be a kilometer or more in length and the type of cable could be a factor as well.

Note: The host processor firmware handles the filtering and MUST not clear the lock until after the master has negated the snapshot input.

18.5 Detailed Register Descriptions

Table 243. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible
RW1C	Normal Read Write ‘1’ to clear	RW1S	Normal Read Write ‘1’ to set

18.5.1 Register Map

The registers of the TSync registers reside within the memory map of the IXP43X network processors. Table 244 presents the address offset for the TSync registers, the names and mnemonics of the registers, and their access capability. Subsequent sections describe the contents of these registers in greater detail. The mnemonic names match names used in RTL code.

Table 244. Register Summary Table (Sheet 1 of 2)

Address Offset paddr[11:0]	Function	Mnemonic	Access
000	Time Sync Control	TS_Control	RW
004	Time Sync Event	TS_Event	RW
008	Addend	TS_Addend	RW
00C	Accumulator	TS_Accum	RW
010	Time Sync Test	TS_Test	RW
014	Unused	-	-
018	RawSystemTime_Low	TS_RSysTime_Lo	RO
01C	RawSystemTime_High	TS_RSysTimeHi	RO
020	SystemTime_Low	TS_SysTimeLo	RW
024	SystemTime_High	TS_SysTimeHi	RW
028	TargetTime_Low	TS_TrgtLo	RW
02C	TargetTime_High	TS_TrgtHi	RW
030	AuxSlaveModeSnap_Low	TS_ASMSLo	RO
034	AuxSlaveModeSnap_High	TS_ASMSHi	RO
038	AuxMasterModeSnap_Low	TS_AMMSLo	RO
03C	AuxMasterModeSnap_High	TS_AMMSHi	RO
040	TS_Channel0_Control	TS_Ch0_Control	RW
044	TS_Channel0_Event	TS_Ch0_Event	RW
048	XMIT_Snaphot0_Low	TS_TxSnap0Lo	RO
04C	XMIT_Snapshot0_High	TS_TxSnap0Hi	RO
050	RECV_Snapshot0_Low	TS_RxSnap0Lo	RO
054	RECV_Snapshot0_High	TS_RxSnap0Hi	RO
058	SourceUUID0_Low	TS_SrcUUID0Lo	RO
05C	SequenceID0/SourceUUID0_High	TS_ SrcUUID0Hi	RO
060	TS_Channel1_Control	TS_Ch1_Control	RW
064	TS_Channel1_Event	TS_Ch1_Event	RW
068	XMIT_Snapshot1_Low	TS_TxSnap1Lo	RO
06C	XMIT_Snapshot1_High	TS_TxSnap1Hi	RO
070	RECV_Snapshot1_Low	TS_RxSnap1Lo	RO
074	RECV_Snapshot1_High	TS_RxSnap1Hi	RO
078	SourceUUID1_Low	TS_SrcUUID1Lo	RO
07C	SequenceID1/SourceUUID1_High	TS_SrcUUID1Hi	RO
080	TS_Channel2_Control	TS_Ch2_Control	RW
084	TS_Channel2_Event	TS_Ch2_Event	RW
088	XMIT_Snapshot2_Low	TS_TxSnap2Lo	RO
08C	XMIT_Snapshot2_High	TS_TxSnap2Hi	RO
090	RECV_Snapshot2_Low	TS_RxSnap2Lo	RO
094	RECV_Snapshot2_High	TS_RxSnap2Hi	RO
098	SourceUUID2_Low	TS_SrcUUID2Lo	RO

Table 244. Register Summary Table (Sheet 2 of 2)

Address Offset paddr[11:0]	Function	Mnemonic	Access
09C	SequenceID2/SourceUUID2_High	TS_SrcUUID2Hi	RO
0A0-0BF	Reserved for Channel 3	-	-
0C0-0DF	Reserved for Channel 4	-	-
0E0-0FF	Reserved for Channel 5	-	-
100-11F	Reserved for Channel 6	-	-
120-13F	Reserved for Channel 7	-	-
140-FFF	Unused	-	-

18.5.2 Register Descriptions

Table 245. Register Summary

Block Address	Offset Address	Register Name	Description	Reset Value	Page Number
	0x000	TS_Control	Time Sync Control Register	x0000	693
	0x004	TS_Event	Time Sync Event Register	x0010	694
	0x008	TS_Addend	Time Sync Addend Register	0x0	694
	0x00C	TS_Accum	Time Sync Accumulator Register	0x0	695
	0x010	TS_Test	Time Sync Test Register	x000	696
	0x018	TS_RSysTimeLo	RawSystemTime_Low Register	0x0	697
	0x01C	TS_RSysTimeHI	RawSystemTime_High Register	0x0	697
	0x020	TS_SysTimeLo	SystemTime_Low Register	0x0	698
	0x024	TS_SysTimeHi	SystemTime_High Register	0x0	698
	0x028	TS_TrgtLo	TargetTime_Low Register	0x0	699
	0x028	TS_TrgtHi	TargetTime_High Register	0x0	699
	0x030	TS_ASMSLo	Auxiliary Slave Mode Snapshot Low Register	0x0	700
	0x034	TS_ASMSHi	Auxiliary Slave Mode Snapshot High Register	0x0	700
	0x038	TS_AMMSLo	Auxiliary Master Mode Snapshot Low Register	0x0	701
	0x03C	TS_AMMSHi	Auxiliary Master Mode Snapshot High Register	0x0	701
	0x040*	TS_Ch_Control	Time Synchronization Channel Control Register	x00	702
	0x044*	TS_Ch_Event	Time Synchronization Channel Event Register	x00	703
	0x048*	TS_TxSnapLo	Transmit Snapshot Low Register	0x0	704
	0x04C*	TS_TxSnapHi	Transmit Snapshot High Register	0x0	705
	0x050*	TS_RxSnapLo	Receive Snapshot Low Register	0x0	706
	0x054*	TS_RxSnapHi	Receive Snapshot High Register	0x0	707
	0x058*	TS_SrcUUID0Lo	Source UUID0 Low Register	0x0	708
	0x05C*	TS_SrcUUIDHi	Sequence Identifier/Source UUID0 High Register	0x0	709

18.5.2.1 Time Sync Control Register

Register Name:	TS_Control																														
Block Base Address:	RegBlockAddress			Offset Address			0x000					Reset Value			x0000																
Register Description:	Time Sync Control Register																							Access:	(See below.)						
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
(Reserved)																												amm	asm	ttm	rst

Register		TS_Control		
Bits	Name	Description	Reset Value	Access
31:4	(Reserved)	Reserved for future use.	x	x
3	amm	**AMMS Interrupt Mask.** Controls whether the Auxiliary Master Mode snapshot indication, which is the snm bit in the Time Sync Event register, should interrupt the Host processor. • When this bit is set, the interrupt to the Host is enabled. • When cleared, the AMMS interrupt to the Host is disabled.	0	RW
2	asm	**ASMS Interrupt Mask.** Controls whether the indication that an Auxiliary Slave Mode snapshot, which is the sns bit in the Time Sync Event register, has been taken should interrupt the Host processor. • When this bit is set, the interrupt to the Host is enabled. • When cleared, the ASMS interrupt to the Host is disabled.	0	RW
1	ttm	**Target Time Interrupt Mask.** Controls whether the Target Time interrupt is passed to the Host processor. • When this bit is set, the interrupt to the Host is enabled. • When cleared, the Target Time interrupt to the Host is disabled.	0	RW
0	rst	Reset. • When a '1' is written to this bit, all logic is returned to the same default state as when a power-on reset occurs. • After writing a '1' to this bit to reset the logic, the firmware must write a '0' to the bit to indicate the end of the reset.	0	RW

18.5.2.2 Time Sync Event Register

Register Name:	TS_Event				
Block Base Address:	RegBlockAddress	Offset Address	0x004	Reset Value	x0010
Register Description:	Time Sync Event Register			Access:	(See below.)

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
(Reserved)																												amm	asm	ttm	rst

Register	TS_Event			
Bits	Name	Description	Reset Value	Access
31:4	(Reserved)	Reserved for future use.	x	x
3	snm	**AMMS Snapshot.** This event bit sets when the system time register value is captured in the Auxiliary Master Mode Snapshot register upon an active high level on a general-purpose input, ammssig. • When this signal is asserted high, an interrupt will be generated to the Host on the ts_intreq if the amm bit in the Time Sync Control register is also set. • To clear snm, write a '1' to it.	0	RW
2	sns	**ASMS Snapshot.** This event bit sets when the system time register value is captured in the Auxiliary Slave Mode Snapshot register upon detection of an active high level on a general-purpose input, asmssig. • When this signal is asserted high, an interrupt will be generated to the Host on the shared interrupt signal (ts_ntreq) if the asm bit in the Time Sync Control register is set. • To clear the sns bit, write a '1' to it.	0	RW
1	ttipend	**Target Time Interrupt Pending.** This bit is the Target Time interrupt pending indication. When this bit is set, it indicates that the Target Time interrupt condition has occurred, which means that the System Time value has reached the 64-bit Target Time register value. • If ttm in the Time Sync Control register is set, the interrupt will be passed to the Host processor. • To clear this condition, the firmware must write a '1' to the ttipend bit. To prevent an immediate reoccurrence of the target time interrupt, the processor should first write a new value to the Target Time register and then clear the condition. This bit is set at power-up since both the System Time and the Target Time are reset at power-up to 0.	1	RW
0	(Reserved)	Reserved for future use.	0	RW

18.5.2.3 Addend Register

Register Name:	TS_Addend				
Block Base Address:	RegBlockAddress	Offset Address	0x008	Reset Value	0x0
Register Description:	Time Sync Addend Register			Access:	(See below.)

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
Addend[31:0]																															

Register		TS_Addend		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:0	Addend	The Addend register contains the frequency scaling value used by a firmware algorithm to achieve time synchronization in the module. The value in this register is added to the value in the Accumulator. When the Accumulator rolls over, an overflow pulse is asserted and increments system time. Because the Addend register is cleared at reset, it must be written with a non-zero value to allow system time to increment.	0	RW

18.5.2.4 Accumulator Register

Register Name:	TS_Accum																														
Block Base Address:	RegBlockAddress					Offset Address	0x00C					Reset Value	0x0																		
Register Description:	Time Sync Accumulator Register																							Access:	(See below.)						
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
Accumulator[31:0]																															

Register		TS_Accum		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:0	Accumulator	The Accumulator register serves as the frequency divider in the time synchronization logic. Firmware calculates a frequency scaling value to be written to the Addend register. The data in the Accumulator register is added to the value in the Addend register once every period of the system clock. When the Accumulator rolls over, an overflow pulse is asserted which increments the value in the system timer. This register is not read or written to in normal operation.	0	RW

18.5.2.5 Test Register

Register Name:	TS_Test				
Block Base Address:	RegBlockAddress	Offset Address	0x010	Reset Value	x000
Register Description:	Time Sync Test Register			Access:	(See below.)

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
(Reserved)																													tenb		tm

Register	TS_Test			
Bits	Name	Description	Reset Value	Access
31:3	(Reserved)	Reserved for future use.	x	x
2:1	tenb	**Test Enable.** These bits define what signals drive the ts_testmode_data pin when the tm bit in this register is set. The target time interrupt pending signal (readable in the TS_Event register) is driven if tenb[1:0] is '00' to support future applications. Specific system timer bits drive ts_testmode_data for the remaining settings of tenb[1:0]. tenb[1:0] ts_testmode_data 00 TS_Event.ttipend TS_SysTimeLo[10] 10 TS_SysTimeLo[12] 11 TS_SysTimeLo[14]	0	
0	tm	**Test Mode.** This bit, which defaults to '0' at reset, is the test mode bit. • When this bit is set, the TSync logic outputs one of four possible signals on the ts_testmode_data pin. The tenb[1:0] bits select the data. This data appears on a GPIO pin when the Test Mode bit is set.	0	

18.5.2.6 RawSystemTime_Low Register

Register Name:	TS_RSysTimeLo				
Block Base Address:	RegBlockAddress	Offset Address	0x018	Reset Value	0x0
Register Description:	RawSystemTime_Low Register			Access:	(See below.)

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
RawSystemTime_Low[31:0]																															

Register	TS_RSysTimeLo			
Bits	Name	Description	Reset Value	Access
31:0	RawSystemTime_Low	This system time register is a read-only register of the raw system time. It is, therefore, not loadable and reflects the local time in the module. • The lower 32 bits of the 64-bit system time are read in this register. • The upper 32 bits are read in the RawSystemTime_High register. When a user reads system time with this pair of registers, no latching of system time occurs, which means that the system time could increment between the reading of the lower 32 bits in this register and the upper 32 bits in the RawSystemTime_High register. The user must account for this and deal with possible increments between reads of the two registers in firmware.	0	RO

18.5.2.7 RawSystemTime_High Register

Register Name:	TS_RSysTimeHI				
Block Base Address:	RegBlockAddress	Offset Address	0x01C	Reset Value	0x0
Register Description:	RawSystemTime_High Register			Access:	(See below.)

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
RawSystemTime_High[31:0]																															

Register	TS_RSysTimeHi			
Bits	Name	Description	Reset Value	Access
31:0	RawSystemTime_High	This register contains the upper 32 bits of system time. When you want to read or write the system time, this register typically first accesses the RawSystemTime_Low Register. This register pair contains the raw system timer value, and no latching of system time occurs when the lower half is read. Time could increment between the reading of the lower 32 bits in the RawSystemTime_Low register and the reading of this register.	0	RO

18.5.2.8 SystemTime_Low Register

Register Name:	TS_SysTimeLo				
Block Base Address:	RegBlockAddress	Offset Address	0x020	Reset Value	0x0
Register Description:	SystemTime_Low Register			Access:	(See below.)

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
SystemTime_Low[31:0]																															

Register	TS_SysTimeLo			
Bits	Name	Description	Reset Value	Access
31:0	SystemTime_Low	The system timer is a loadable up-counter, and reflects the local time in the module. While the system timer is 64 bits wide, the lower 32 bits reside in this register. The system timer is clocked by the module system clock and incremented when the Accumulator register rolls over. To read the entire system time value, the user must read this location first. Reading this location captures the upper 32 bits of the system time in a temporary register, which is accessed when the user reads the SystemTime_High Register next. Likewise, the SystemTime_Low Register must be written first when the user wants to write a new 64-bit value to system time. The data written to this register is captured in a holding register. When the user writes to the SystemTime_High Register, all 64 bits are then written to the system timer. Updating the system time with a direct write has precedence over increments to the system time based on an Accumulator rollover.	0	RW

18.5.2.9 SystemTime_High Register

Register Name:	TS_SysTimeHi				
Block Base Address:	RegBlockAddress	Offset Address	0x024	Reset Value	0x0
Register Description:	SystemTime_High Register			Access:	(See below.)

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
SystemTime_High[31:0]																															

Register	TS_SysTimeHi			
Bits	Name	Description	Reset Value	Access
31:0	SystemTime_High	This register contains the upper 32 bits of system time. When the user wants to read or write the system time, this register must first access the SystemTime_Low Register. See Section 18.5.2.8, "SystemTime_Low Register" on page 698 for more details.	0	RW

18.5.2.10 TargetTime_Low Register

Register Name:	TS_TrgtLo				
Block Base Address:	RegBlockAddress	Offset Address	0x028	Reset Value	0x0
Register Description:	TargetTime_Low Register			Access:	(See below.)

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
TargetTime_Low[31:0]																															

Register	TS_TrgtLo			
Bits	Name	Description	Reset Value	Access
31:0	TargetTime_ Low	The Target Time register set contains 64 bits of a time value. When the system time is greater than or equal to the target time value, an interrupt is generated to the Host on the **ts_intreq** signal if the **ttm** bit in the Time Sync Control register is set. For more information about the Target Time interrupt, see Section 18.5.2.1, "Time Sync Control Register" on page 693.	0	RW

18.5.2.11 TargetTime_High Register

Register Name:	TS_TrgtHi				
Block Base Address:	RegBlockAddress	Offset Address	0x028	Reset Value	0x0
Register Description:	TargetTime_High Register			Access:	(See below.)

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
TargetTime_Low[31:0]																															

Register	TS_TrgtLo			
Bits	Name	Description	Reset Value	Access
31:0	TargetTime_ High	The Target Time register set contains 64 bits of a time value. When the system time is greater than or equal to the target time value, an interrupt is generated to the Host on the **ts_intreq** signal if the **ttm** bit in the Time Sync Control register is set. For more information about the Target Time interrupt, see Section 18.5.2.1, "Time Sync Control Register" on page 693.	0	RW

18.5.2.12 Auxiliary Slave Mode Snapshot Low Register – ASMS_Low

Register Name:	TS_ASMSLo				
Block Base Address:	RegBlockAddress	Offset Address	0x030	Reset Value	0x0
Register Description:	Auxiliary Slave Mode Snapshot Low Register			Access:	(See below.)

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
ASMS_Low[31:0]																															

Register	TS_ASMSLo			
Bits	Name	Description	Reset Value	Access
31:0	ASMS_Low	When the board is operating in Slave mode, an active high level on a general-purpose input, **asmssig**, triggers a snapshot of System Time into the ASMS_Low and ASMS_High registers. The general-purpose input is synchronized by the Time Sync logic before it is used. Note: The processor can configure the GPIO bit as an output, but it will always be input-only to the Time Sync block. When the ASMS snapshot occurs, the **sns** indication in the Time Sync Event register is set. Writing a logic 1 to that bit clears the snapshot indication and allows a new snapshot to occur on the next rising transition of **asmssig**.	0	RO

18.5.2.13 Auxiliary Slave Mode Snapshot High Register – ASMS_High

Register Name:	TS_ASMSHi				
Block Base Address:	RegBlockAddress	Offset Address	0x034	Reset Value	0x0
Register Description:	Auxiliary Slave Mode Snapshot High Register			Access:	(See below.)

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
ASMS_High[31:0]																															

Register	TS_ASMSHi			
Bits	Name	Description	Reset Value	Access
31:0	ASMS_High	When the board is operating in Slave mode, an active high level on a general-purpose input, **asmssig**, triggers a snapshot of System Time into the ASMS_Low and ASMS_High registers. The general-purpose input is synchronized by the Time Sync logic before it is used. **Note:** The processor can configure the GPIO bit as an output, but it will always be input-only to the Time Sync block. When the ASMS snapshot occurs, the **sns** indication in the Time Sync Event register is set. Writing a logic 1 to that bit clears the snapshot indication and allows a new snapshot to occur on the next rising transition of **asmssig**.	0	RO

18.5.2.14 Auxiliary Master Mode Snapshot Low Register – AMMS_Low

Register Name:	TS_AMMSLo				
Block Base Address:	RegBlockAddress	Offset Address	0x038	Reset Value	0x0
Register Description:	Auxiliary Master Mode Snapshot Low Register			Access:	(See below.)

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
AMMS_Low[31:0]																															

Register	TS_AMMSLo			
Bits	Name	Description	Reset Value	Access
31:0	AMMS_Low	When the board is operating in Master mode, it receives a general-purpose input signal for synchronization of snapshots and time. This general-purpose input, **ammssig**, is synchronized by the system clock in the Time Sync logic before it is used. **Note:** The processor can configure the GPIO as an output, but it will always be an input-only to the Time Sync block. When the AMMS snapshot occurs, the **snm** indication in the Time Sync Event register is asserted. No new snapshots in the AMMS register pair are captured until the firmware writes a '1' back to the **snm** bit to clear the snapshot indication.	0	RO

18.5.2.15 Auxiliary Master Mode Snapshot High Register – AMMS_High

Register Name:	TS_AMMSHi				
Block Base Address:	RegBlockAddress	Offset Address	0x03C	Reset Value	0x0
Register Description:	Auxiliary Master Mode Snapshot High Register			Access:	(See below.)

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
AMMS_High[31:0]																															

Register	TS_AMMSHi			
Bits	Name	Description	Reset Value	Access
31:0	AMMS_High	When the board is operating in Master mode, it receives a general-purpose input signal for synchronization of snapshots and time. This general-purpose input, **ammssig**, is synchronized by the system clock in the Time Sync logic before it is used. ***Note:*** The processor can configure the GPIO as an output, but it will always be an input-only to the Time Sync block. When the AMMS snapshot occurs, the **snm** indication in the Time Sync Event register is asserted. No new snapshots in the AMMS register pair are captured until the firmware writes a '1' back to the **snm** bit to clear the snapshot indication.	0	RO

18.5.2.16 TS_Channel_Control Register (Per Channel)

Register Name:	TS_Ch_Control				
Block Base Address:	RegBlockAddress	Offset Address	0x040*	Reset Value	x00
Register Description:	Time Synchronization Channel Control Register			Access:	(See below.)

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
Reserved																														ta	mm

*Address offsets per channel...
Channel 0 = 0x040
Channel 1 = 0x060
Channel 2 = 0x080

Register	TS_Ch_Control			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:2	(Reserved)	Reserved for future use.	x	x
1	ta	**Timestamp All messages.** • When this bit is set, the locking of the time snapshot registers is inhibited. Each message is timestamped at the reception of a start of frame delimiter (SFD), regardless of whether the message is a Sync or Delay Request message. The timestamp is captured by the Snapshot register which is never locked and therefore must be read before the next SFD is received. • When this bit is cleared, the timestamp taken after the SFD is frozen or locked when a valid Sync or Delay Request message is detected, until the software resets it.	0	RW
0	mm	**Master Mode.** • When this bit is set, it indicates that this channel is a time master on the network. • When cleared, this bit indicates that this channel is in slave mode. The default after reset is slave mode.	0	RW

18.5.2.17 TS_Channel_Event Register (Per Channel)

Register Name:	TS_Ch_Event				
Block Base Address:	RegBlockAddress	Offset Address	0x044*	Reset Value	x00
Register Description:	Time Synchronization Channel Event Register			Access:	(See below.)

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
Reserved																														rxs	txs

*Address offsets per channel...
Channel 0 = 0x044
Channel 1 = 0x064
Channel 2 = 0x084

Register	TS_Ch_Event			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:2	Reserved	Reserved for future use.	x	x
1	rxs	**Receive Snapshot Locked.** This bit is automatically set when a Delay_Req message in Master mode, or a Sync message in Slave mode, is received and the **ta** bit in the corresponding TS_Channel_Control register is clear. It indicates that the current system time value has been captured in the RECV_Snapshot register and that further changes to the RECV_Snapshot are now locked out. To clear this bit, write a '1' to it.	0	RW
0	txs	**Transmit Snapshot Locked.** This bit is automatically set when a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted and the ta bit in the corresponding TS_Channel_Control register is clear. It indicates that the current system time value has been captured in the XMIT_Snapshot register and that further changes to the XMIT_Snapshot are now locked out. To clear this bit, write a '1' to it.	0	RW

18.5.2.18 XMIT_Snapshot_Low Register (Per Channel)

Register Name:	TS_TxSnapLo				
Block Base Address:	RegBlockAddress	Offset Address	0x048*	Reset Value	0x0
Register Description:	Transmit Snapshot Low Register			Access:	(See below.)

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
XMIT_Snapshot_Low[31:0]																															

*Address offsets per channel...
Channel 0 = 0x048
Channel 1 = 0x068
Channel 2 = 0x088

Register	TS_TxSnapLo			
Bits	Name	Description	Reset Value	Access
31:0	XMIT_ Snapshot_ Low	When a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted, the current system time is captured in this XMIT_Snapshot register. • The XMIT_Snapshot_Low register contains the lower 32 bits of the time value. • The XMIT_Snapshot_High register contains the upper 32 bits. After a XMIT_Snapshot has occurred, the **txs** indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the XMIT_Snapshot_Low and XMIT_Snapshot_High registers before it writes a '1' to the **txs** bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO

18.5.2.19 XMIT_Snapshot_High Register (Per Channel)

Register Name:	TS_TxSnapHi																															
Block Base Address:	RegBlockAddress						Offset Address						0x04C*						Reset Value						0x0							
Register Description:	Transmit Snapshot High Register																							Access:	(See below.)							
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
XMIT_Snapshot_High[31:0]																																
*Address offsets per channel... Channel 0 = 0x04C Channel 1 = 0x06C Channel 2 = 0x08C																																

Register	TS_TxSnapHi			
Bits	Name	Description	Reset Value	Access
31:0	XMIT_ Snapshot_ High	When a Sync message in Master mode, or a Delay_Req message in Slave mode, is transmitted, the current system time is captured in this XMIT_Snapshot register. • The XMIT_Snapshot_Low register contains the lower 32 bits of the time value. • The XMIT_Snapshot_High register contains the upper 32 bits. After a XMIT_Snapshot has occurred, the **txs** indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the XMIT_Snapshot_Low and XMIT_Snapshot_High registers before it writes a '1' to the **txs** bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO

18.5.2.20 RECV_Snapshot Low Register (Per Channel)

Register Name:	TS_RxSnapLo				
Block Base Address:	RegBlockAddress	Offset Address	0x050*	Reset Value	0x0
Register Description:	Receive Snapshot Low Register			Access:	(See below.)

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
RECV_Snapshot_Low[31:0]																															

*Address offsets per channel...
Channel 0 = 0x050
Channel 1 = 0x070
Channel 2 = 0x090

Register	TS_RxSnapLo			
Bits	Name	Description	Reset Value	Access
31:0	RECV_ Snapshot_ Low	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the current system time is captured in this RECV_Snapshot register. • The RECV_Snapshot_Low register contains the lower 32 bits of the time value. • The RECV_Snapshot_High register contains the upper 32 bits. After a RECV_Snapshot has occurred, the **rxs** indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the RECV_Snapshot_Low and RECV_Snapshot_High registers before it writes a '1' to the **rxs** bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO

18.5.2.21 RECV_Snapshot High Register (Per Channel)

Register Name:	TS_RxSnapHi																															
Block Base Address:	RegBlockAddress						Offset Address						0x054*										Reset Value						0x0			
Register Description:	Receive Snapshot High Register																						Access:						(See below.)			
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
RECV_Snapshot_High[31:0]																																
*Address offsets per channel... Channel 0 = 0x054 Channel 1 = 0x074 Channel 2 = 0x094																																

Register	TS_RxSnapHi			
Bits	Name	Description	Reset Value	Access
31:0	RECV_ Snapshot_ High	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the current system time is captured in this RECV_Snapshot register. • The RECV_Snapshot_Low register contains the lower 32 bits of the time value. • The RECV_Snapshot_High register contains the upper 32 bits. After a RECV_Snapshot has occurred, the **rxs** indication in the TS_Channel_Event register does not clear until the user writes a '1' to that bit in that register. Therefore, the firmware should read the RECV_Snapshot_Low and RECV_Snapshot_High registers before it writes a '1' to the **rxs** bit to clear the snapshot indication. In this way, the snapshot value cannot change between reads of the high and low locations.	0	RO

18.5.2.22 SourceUUID0_Low Register (Per Channel)

Register Name:	TS_SrcUUID0Lo				
Block Base Address:	RegBlockAddress	Offset Address	0x058*	Reset Value	0x0
Register Description:	Source UUID0 Low Register			Access:	(See below.)

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
SourceUUID0_Low[31:0]																															

*Address offsets per channel...
Channel 0 = 0x058
Channel 1 = 0x078
Channel 2 = 0x098

Register	TS_SrcUUID0Lo			
Bits	Name	Description	Reset Value	Access
31:0	SourceUUID0_Low	When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the Source UUID of the message is captured. The source UUID is located in bytes 64 through 69 of the Ethernet message, and this register contains the lower 32 bits of the source UUID. This register is read-only. At reset, the value in the register is 0, which is not a valid Source UUID value.	0	RO

18.5.2.23 SequenceID/SourceUUID_High Register (Per Channel)

When a Delay_Req message in Master mode, or a Sync message in Slave mode, is received, the source UUID and the sequence ID of the message are captured.

Register Name:	TS_SrcUUIDHi				
Block Base Address:	RegBlockAddress	Offset Address	0x05C*	Reset Value	0x0
Register Description:	Sequence Identifier/Source UUID0 High Register			Access:	(See below.)

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
SequenceID[15:0]																SourceUUID_High[47:32]															

*Address offsets per channel...
Channel 0 = 0x05C
Channel 1 = 0x07C
Channel 2 = 0x09C

Register	TS_SrcUUIDHi			
Bits	Name	Description	Reset Value	Access
31:16	SequenceID	The sequence ID is located in bytes 72 and 73 of the Ethernet message, and is captured in this register in bit locations [31:16].	0	RO
15:0	SourceUUID_High	This register contains the upper 16 bits (bits 47:32) of the source UUID in bit locations [15:0].	0	RO

§ §

19.0 Synchronous Serial Port

The Synchronous Serial Port (SSP) is a full-duplex synchronous serial interface. It can connect to a variety of external analog-to-digital (A/D) converters, audio and telecom codecs, and many other devices that use serial protocols for transferring data. It supports National Microwire*, Texas Instruments* synchronous serial protocol (SSP), and Motorola* serial peripheral interface (SPI) protocol.

The SSP operates in master mode, that is, with attached peripheral functions as a slave, and supports serial bit rates from 7.2 KHz to 1.84 MHz. Serial data formats may range from 4 to 16 bits in length. Two on-chip register blocks function as independent FIFOs for data, one for each direction. The buffers are 16 entries deep and 16 bits wide.

Buffers are loaded or emptied by the system processor using SRAM-like transfers. The transfers are always one word per transfer. Each 32-bit word from the system fills one entry in a FIFO using the lower half 16-bits of a 32-bit word.

19.1 SSP Operation

Serial data is transferred between the system and an external peripheral through FIFO buffers in the SSP Port. Transfers are initiated by the host processor to/from system memory. Operation is full duplex, separate buffers and serial data paths permit simultaneous transfers in both directions.

19.1.1 Processor-Initiated Data Transfer

Transmit data (system to peripheral) is written by host processor to SSP port transmit buffer. The buffer works as a FIFO, and is seen as one 32-bit location by the processor. The SSP Port then takes the data from the buffer, serializes it, and sends it over the serial wire (**SSP_TXD**) to the peripheral.

Received data from peripheral (on **SSP_RXD**) are converted to parallel words and stored in Receive FIFO buffer. When a programmable fullness threshold is passed, it triggers an interrupt to the interrupt controller (hence, if enabled, to an interrupt input to the CPU). Interrupt service routine responds by identifying source of interrupt and then doing an SRAM-like read from inbound FIFO buffer. All buffer reads and writes use SPB to transfer data between internal CPU system bus and the buffer.

The host processor differentiates between the two FIFOs based on whether it does a READ or a WRITE transfer. Read automatically targets the Receive FIFO, and WRITE, writes data to the Transmit FIFO. They are at the same address from memory map point of view.

FIFO buffers are 16 words deep x 16 bits wide. This stores up to 16 samples per buffer.

19.2 Data Formats

19.2.1 Serial Data Formats for Transfer to/from Peripherals

Four pins are used to transfer data between the CPU and external codecs or modems. Although there are several formats for serial data, they have the same basic structure:

- **SSP_SCLK** — Defines the bit rate at which the serial data is driven onto, and sampled from, the bus
- **SSP_SFRM** — Defines the boundaries of a basic data unit, comprised of multiple serial bits
- **SSP_TXD** — Is the serial data path for outbound data, from system to peripheral
- **SSP_RXD** — Is the serial data path for inbound data, from peripheral to system

A data frame may range from 4 to 16 bits, depending on the format selected. Serial data is transmitted to MSB first. Three formats are supported: Motorola SPI, Texas Instruments synchronous serial protocol (SSP), and National Microwire.

For SPI and Microwire formats, **SSP_SFRM** functions as a chip select to enable the external device (target of the transfer), and is held active-low during the data transfer. For SSP format, **SSP_SFRM** is pulsed high for one (serial) data period at the start of each frame.

Function and use of the serial clock **SSP_SCLK** is different for each format:

- For Microwire, both data sources switch (change to the next bit) outgoing data on the falling edge of **SSP_SCLK**, and sample incoming data on the rising edge.
- For SSP, data sources switch outgoing data on the rising edge of **SSP_SCLK**, and sample incoming data on the falling edge
- SPI gives the user the choice of the edge of **SSP_SCLK** to use for switching outgoing data, and for sampling incoming data. In addition, the user can move the phase of **SSP_SCLK**, shifting its active state one-half period earlier or later at the start and end of a frame.

As SSP and SPI are full-duplex protocols, Microwire uses a half-duplex master-slave messaging protocol. At the start of a frame, a single-byte control message is transmitted from the controller to the peripheral; no data is sent by the peripheral. The peripheral interprets the message and, if it is a READ request, responds with requested data, one clock after the last bit of the requesting message. Return data - part of the same frame - are from 4 to 16 bits in length. Total frame length is 13 to 25 bits.

Note: The serial clock (**SSP_SCLK**), if driven by the SSP Port alone toggles and it denotes that active data transfer is underway. Else, it is held in an inactive or idle state, as defined by the specified protocol under which it operates.

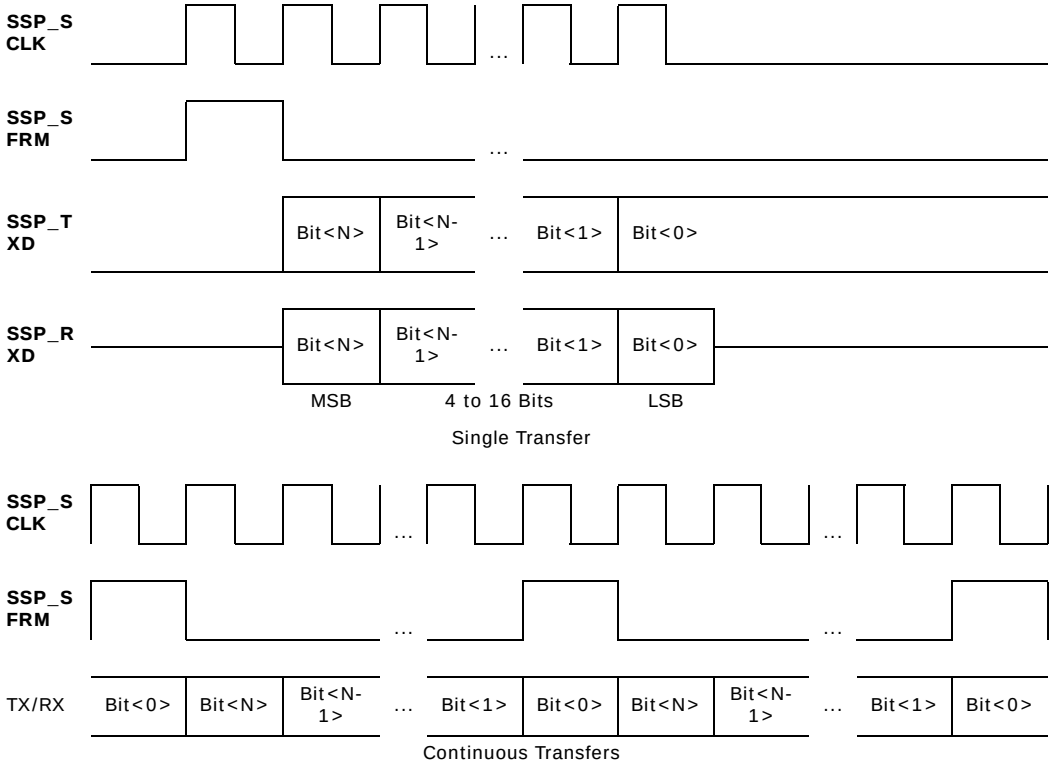
19.2.1.1 SSP Format

When outgoing data in the SSP controller is ready to transmit, **SSP_SFRM** asserts for one clock period. On the following clock, data to be transmitted is driven on **SSP_TXD** one bit at a time, most significant bit first. Similarly, the peripheral drives data on the **SSP_RXD** pin. Word length is from 4 to 16 bits. All transitions take place on the rising edge of **SSP_SCLK** and data sampling is done on the falling edge. At the end of the transfer, **SSP_TXD** retains the value of the last bit sent (bit 0) through the next idle period. If the SSP Port is disabled or reset, **SSP_TXD** is forced to zero.

Table 246 shows the Texas Instruments synchronous serial frame format for a single transmitted frame and when back-to-back frames are transmitted. Once the bottom entry of the Transmit FIFO contains data, **SSP_SFRM** is pulsed high for one **SSP_SCLK**

period and the value to be transmitted is transferred from the Transmit FIFO to the transmit logic's serial shift register. On the next rising edge of **SSP_SCLK**, the MSB of the 4 to 16-bit data frame is shifted to the **SSP_TXD** pin. Similarly, the MSB of the received data is shifted onto the **SSP_RXD** pin by the off-chip serial slave device. Both the SSP and the off-chip serial slave device then latch each data bit into their serial shifter on the falling edge of each **SSP_SCLK**. The received data is transferred from the serial shifter to the Receive FIFO on the first rising edge of **SSP_SCLK** after the LSB has been latched.

Table 246. Texas Instruments* Synchronous Serial Frame Format



19.2.1.2 SPI Format

Note that there are four possible sub-modes of the SPI format depending on the **SSP_SCLK** edges selected for driving data and sampling received data, and on the selection of the phase mode of **SSP_SCLK,** refer to "SSP Control Register 1 (SSCR1)" on page 719 for complete description of each mode.

For SPI format, **SSP_SCLK** and **SSP_TXD** are low, and **SSP_SFRM** is high, in idle mode or when the SSP is disabled. When transmit (outgoing) data is ready, **SSP_SFRM** goes low and stays low for the remainder of the frame. The MSB of serial data is driven on to **SSP_TXD** a half-cycle later, and halfway into the first bit period; **SSP_SCLK** asserts high and continues toggling for the remaining data bits. Data transitions on the falling edge of **SSP_SCLK**. 4 to 16 bits are transferred per frame.

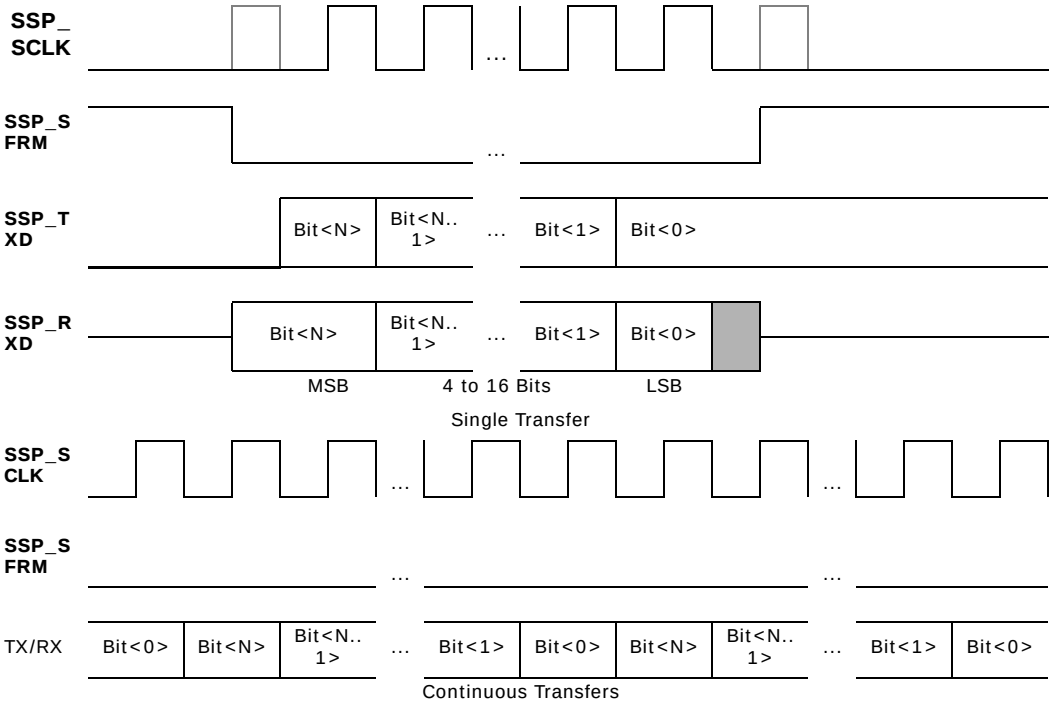
With assertion of **SSP_SFRM**, receive data is simultaneously driven from the peripheral on **SSP_RXD**, MSB first. Data transitions on **SSP_SCLK** falling edges and is sampled by the controller on rising edges.

At the end of the frame, **SSP_SFRM** is de-asserted high one clock period after the last bit has been latched at its destination, and the completed incoming word is shifted into the incoming FIFO. The peripheral can tri-state **SSP_RXD** after sending the last (LSB) bit of the frame. **SSP_TXD** retains the last value transmitted when the controller goes into idle mode, unless the SSP port is disabled or reset (that forces **SSP_TXD** to zero).

For back-to-back transfers, start and completion are like those of a single transfer, but **SSP_SFRM** does not deassert between words. Both transmitter and receiver knows the word length, and keeps track internally of the start and end of words (frames). There are no **dead** bits; the least significant bit of one frame is followed immediately by the most significant bit of the next.

Table 247 shows one of the four possible configurations for the Motorola SPI frame format for a single transmitted frame and when back-to-back frames are transmitted.

Table 247. Motorola* SPI Frame Format



Note: The phase and polarity of **SSP_SCLK** is configured for four different modes. This example shows just one of those modes.

19.2.1.3 Microwire* Format

This format is similar to SPI, except transmission is half-duplex instead of full-duplex, using master-slave message passing.

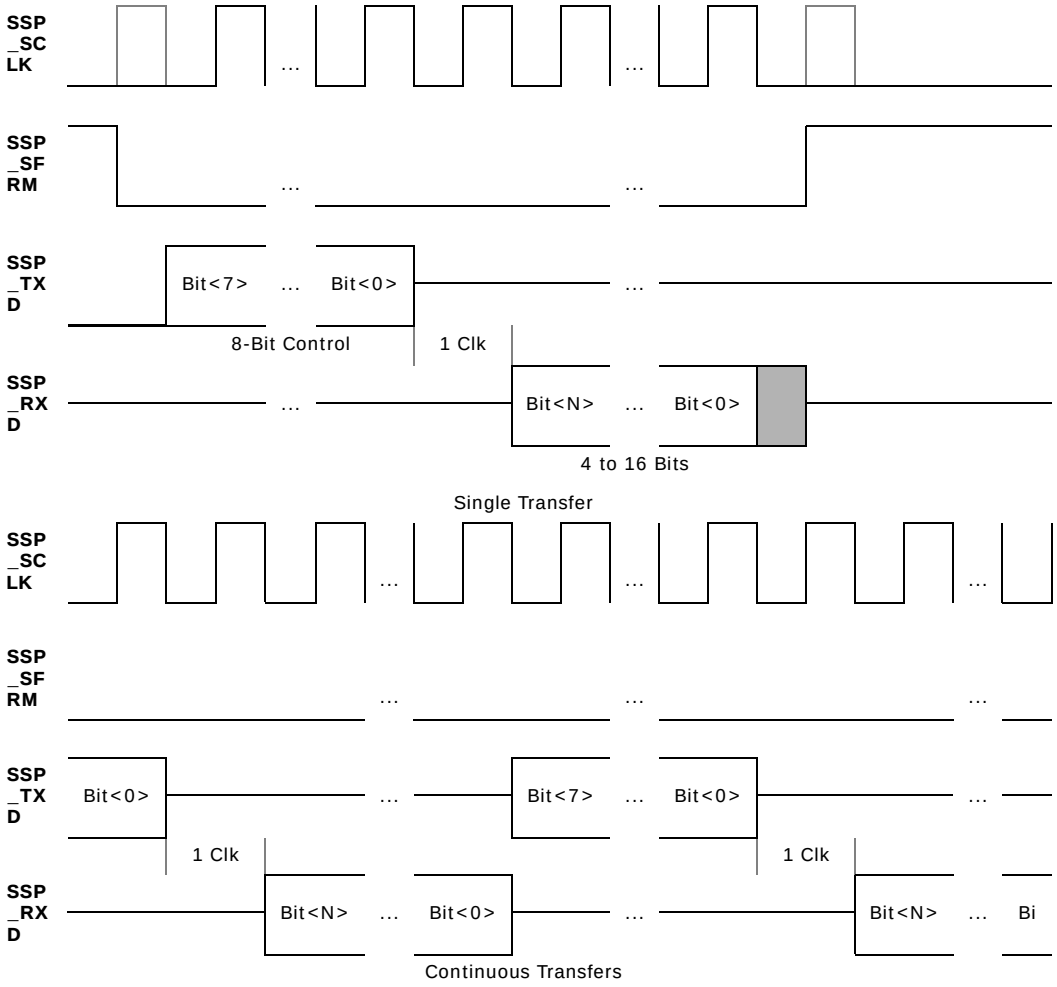
In idle state or when the SSP is disabled, **SSP_SCLK** is low, **SSP_SFRM** is high and **SSP_TXD** is low. Each serial transmission begins with **SSP_SFRM** assertion (low), followed by an 8-bit or 16-bit command word sent from controller to peripheral on **SSP_TXD**. **SSP_RXD**, controlled by the peripheral, remains tri-stated. **SSP_SCLK** goes high midway into the command MSB and continues toggling at the bit rate. One bit-period after the last command bit, the peripheral returns the serial data requested, MSB first, on **SSP_RXD**.

Data transitions on falling edge of **SSP_SCLK** and is sampled on the rising edge. The last falling edge of **SSP_SCLK** coincides with the end of the last data bit (LSB) on **SSP_RXD**, and it remains low after that (if the only, or last word, of the transfer). **SSP_SFRM** deasserts high one-half clock period later.

The start and end of a series of back-to-back transfers are like those of a single transfer; but, **SSP_SFRM** remains asserted (low) throughout the transfer. The end of a data word on **SSP_RXD** is followed immediately by the start of the next command byte on **SSP_TXD**, with no dead time.

Table 248 shows the National Microwire frame format for a single transmitted frame and when back-to-back frames are transmitted.

Table 248. National Microwire* Frame Format



19.2.2 Parallel Data Formats for Buffer Storage

Data in buffers is always stored with one sample per 16-bit word regardless of the format data word length. Within each 16-bit field, stored data sample is right-justified, with word LSB in bit 0, and unused bits packed as zeroes on left-hand (MSB) side. Logic in SSP automatically left-justifies the data in the Transmit FIFO so that the sample is properly transmitted on **SSP_TXD** in the selected frame format.

19.3 Buffer Operation

There are two separate and independent buffers for incoming (from peripheral) and outgoing (to peripheral) serial data. Buffers are filled or emptied by an SRAM-like transfer initiated by the system processor.

Although the system bus is 32 bits wide, only single samples are transferred. Thus, only the lower two bytes of the 32 bit word has valid data; the upper two bytes are not used and includes dummy or invalid data that should be discarded.

Each buffer comprises a dual-port register file with control circuitry to make it work as a FIFO, with independent read and write ports.

Buffer filling and emptying is performed by the system processor in response to an interrupt from the FIFO logic. Each FIFO has a programmable threshold when an interrupt is triggered. When the threshold value is exceeded, an interrupt is generated and, if enabled, signals the host processor to empty an inbound FIFO or to refill an outbound FIFO.

The system can also poll status bits to learn how full a FIFO is.

19.4 Baud-Rate Generation

The baud (or bit-rate clock) is generated internally by dividing the standard input clock (3.6864 MHz), that is first divided by 2. This feeds a programmable divider to generate baud rates from 7.2 KHz to 1.8432 Mbps. Optionally, an external clock (**SSP_EXTCLK**) is used to replace the 3.6864 MHz standard input clock.

19.5 SSP Serial Port Registers

There are five registers in the SSP block: two control, one data, one status register, and one test register.

- Control registers are used to program the baud rate, data length, frame format, data transfer mechanism, and port enabling. In addition, they permit setting the FIFO fullness threshold that triggers an interrupt.
- The Data Register is mapped as one 32-bit location, that physically points to either of the two 32-bit registers. One register is for WRITES, and transfers data to the Transmit FIFO; the other is for READS, and takes data from the Receive FIFO. A write cycle loads successive words into the SSP Write Register, from the lower half 2 bytes of a 32-bit word to the Transmit FIFO. A READ cycle similarly takes data from the SSP Read Register, and the Receive FIFO reloads it with available data bits it has stored.

  Read and writes should not increment the address; all accesses to the SSP Data Register memory address accesses the Read or Write Register.

  The FIFOs are independent buffers that allow full duplex operation.
- The Status Register signals the state of the FIFO buffers: whether the programmable threshold has been passed (transmit/receive buffer service request), and a value showing the actual fullness of the FIFO. There are flag bits to indicate when the SSP is actively transmitting data, when the Transmit Buffer is not full, and when the receive buffer is not empty. Error bits signal overrun errors.

Table 249. SSP Serial Port Register Summary

Block Address	Register Name	Description	Reset Value	Page Number
0xC801_20	SSCR0	SSC Control Register 0	0x0000_0000	718
0xC801_20	SSCR1	SSC Control Register 1	0x0000_0000	722
0xC801_20	SSSR	SSP Status Register	0x0000_0000	724
0xC801_20	SSITR	SSP Interrupt Test Register	0x0000_0000	725
0xC801_20	SSDR	SSP Data Register	0x0000_0000	726

Table 250. SSP Serial Port Register Table Legend

Attribute	Legend	Attribute	Legend	Attribute	Legend
NA	Not Accessible	RO	Read Only	RW	Read/Write
PR	Preserved	RS	Read/Set	WO	Write Only
RC	Read Clear	RV	Reserved	RW1C	Normal Read Write '1' to clear

19.5.1 SSP Control Register 0 (SSCR0)

The SSP control register 0 (SSCR0) contains five different bit fields that control various functions within the SSP.

19.5.1.1 Data Size Select (DSS)

The 4-bit data size select (DSS) field is used to select the size of the data transmitted and received by the SSP. Data is 4 to 16 bits in length. When data is programmed to be less than 16 bits, received data is automatically right-justified and the upper bits in the Receive FIFO are zero-filled by receive logic. Transmit data should not be left-justified by the user before being placed in the Transmit FIFO; transmit logic in the SSP automatically left-justifies the data sample according to the value of DSS before the sample is transmitted on **SSP_TXD**. Although it is possible to program data sizes of 1, 2, and 3 bits, these sizes are reserved and produce unpredictable results in the SSP.

When National Microwire frame format is selected, this bit field selects the size of the received data.

Note: The size of the transmitted data is always 8 bits in this mode.

19.5.1.2 Frame Format (FRF)

The 2-bit frame format (FRF) field is used to select the frame format to use: Motorola SPI (FRF=00), Texas Instruments synchronous serial (FRF=01), or National Microwire (FRF=10).

Note: FRF=11 is reserved and the SSP produces unpredictable results if this value is used.

19.5.1.3 External Clock Select (ECS)

The external clock select (ECS) bit selects whether the on-chip 3.6864-MHz clock is used by the SSP or if an off-chip clock is supplied via **SSP_EXTCLK**. When ECS=0, the SSP uses the on-chip 3.6864-MHz clock to produce a range of serial transmission rates ranging from 7.2 Kbps to a maximum of 1.8432 Mbps. When ECS=1, the SSP uses **SSP_EXTCLK** to input a clock supplied from off-chip. The frequency of the off-chip

clock is any value up to 33.33 MHz. This off-chip clock is useful when a serial transmission rate, that is not an even multiple of 3.6864 MHz, is required for synchronization with the target off-chip slave device.

19.5.1.4 Synchronous Serial Port Enable (SSE)

The SSP enable (SSE) bit is used to enable and disable all SSP operations. When SSE=0, the SSP is disabled; when SSE=1, it is enabled. When the SSP is disabled, all of its clocks are powered down to minimize power consumption.

Note: The SSE is within the SSP and is reset to a known state. It is cleared to zero to ensure the SSP is disabled following a reset.

When the SSE bit is cleared during active operation, the SSP is disabled immediately, causing the current frame being transmitted to be terminated. Clearing SSE resets the SSP's FIFOs. But the control and status registers of the SSP are not reset. The user ensures these registers are properly reconfigured before re-enabling the SSP.

19.5.1.5 Serial Clock Rate (SCR)

The 8-bit serial clock rate (SCR) bit-field is used to select the baud, or bit rate, of the SSP. A total of 256 various bit rates is selected, ranging from a minimum of 7.2 Kbps to a maximum of 1.8432 Mbps. The serial clock generator uses the 3.6864 MHz clock produced by the on-chip PLL divided by a fixed value of 2, and then divided by the programmable SCR value (0 to 255) plus 1 to generate the serial clock (**SSP_SCLK**). The resultant clock is driven on the **SSP_SCLK** pin and is used by the SSP's transmit logic to drive data on the **SSP_TXD** pin, and to latch data on the **SSP_RXD** pin. Depending on the frame format selected, each transmitted bit is driven on the rising or falling edge of **SSP_SCLK**, and is sampled on the opposite clock edge.

The following bit table presents the bit locations corresponding to the five different control bit fields within SSP control register 0.

Note: The SSE bit is the only control bit that is reset to a known state, to ensure that the SSP is disabled following a reset. The reset state of all other control bits is unknown and is initialized before enabling the SSP.

Register Name:	SSCR0				
Block Base Address:	0xC801_20	Offset Address	0x00	Reset Value	0x0000_0000
Register Description:	SSC Control Register 0			Access:	(See below.)

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
Reserved																SCR								SSE	ECS	FRF		SRS			

Register	SSCR0 (Sheet 1 of 2)			
Bit	Name	Description	Reset Value	Access
31:16	(Reserved)	(Reserved)	0x0000	RV
15:08	SCR	Serial Clock Rate Selection Value (0 to 255) used to generate transmission rate of SSP. Bit rate = 3.6864×10^6 / (2 x (SCR + 1)) where SCR is a decimal integer	X	RW
7	SSE	Synchronous Serial Port Enable bit. 0 = SSP operation disabled 1 = SSP operation enabled	0x0000	RW

Register		SSCR0 (Sheet 2 of 2)		
Bit	Name	Description	Reset Value	Access
6	ECS	External clock select bit. 0 = On-chip clock used to produce the SSP's serial clock (**SSP_SCLK**). 1 = **SSP_EXTCLK** is used to create the SSP's **SSP_SCLK**.	0x0000	RW
5:4	FRF	This field specifies the FRame Format. 00 - Motorola Serial Peripheral Interface (SPI) 01 - Texas Instruments Synchronous Serial Protocol (SSP) 10 - National Microwire 11 - Reserved, undefined operation	0x0000	RW
3:0	DSS	This field specifies the Data Size Selection. 0000 - Reserved, undefined operation 0001 - Reserved, undefined operation 0010 - Reserved, undefined operation 0011 - 4-bit data 0100 - 5-bit data 0101 - 6-bit data 0110 - 7-bit data 0111 - 8-bit data 1000 - 9-bit data 1001 - 10-bit data 1010 - 11-bit data 1011 - 12-bit data 1100 - 13-bit data 1101 - 14-bit data 1110 - 15-bit data 1111 - 16-bit data	0x0000	RW

19.5.2 SSP Control Register 1 (SSCR1)

The SSP Control Register 1 (SSCR1) contains nine bit fields that control various SSP functions.

19.5.2.1 Receive FIFO Interrupt Enable (RIE)

The Receive FIFO interrupt enable (RIE) bit is used to mask or enable the Receive FIFO service request interrupt. When RIE=0, the interrupt is masked, and the state of the Receive FIFO Service Request (RFS) bit within the SSP Status Register is ignored by the interrupt controller. When RIE=1, the interrupt is enabled, and whenever RFS is set to one an interrupt request is made to the interrupt controller.

Note: The programming RIE=0 does not affect the current state of RFS or the ability of the Receive FIFO logic to set and clear RFS; it only blocks the generation of the interrupt request.

19.5.2.2 Transmit FIFO Interrupt Enable (TIE)

The Transmit FIFO interrupt enable (TIE) bit is used to mask or enable the Transmit FIFO service request interrupt. When TIE=0, the interrupt is masked and the state of the Transmit FIFO service request (TFS) bit within the SSP status register is ignored by the interrupt controller. When TIE=1, the interrupt is enabled, and whenever TFS is set to one an interrupt request is made to the interrupt controller.

Note: The programming TIE=0 does not affect the current state of TFS or the ability of the Transmit FIFO logic to set and clear TFS; it only blocks the generation of the interrupt request.

19.5.2.3 Loop Back Mode (LBM)

The loop back mode (LBM) bit is used to enable and disable the ability of the SSP transmit and receive logic to communicate. When LBM=0, the SSP operates normally. The transmit and receive data paths are independent and communicate via their respective pins. When LBM=1, the output of the transmit serial shifter is directly connected to the input of the receive serial shifter internally.

19.5.2.4 Serial Clock Polarity (SPO)

The serial clock (**SSP_SCLK**) polarity bit (SPO) selects the polarity of the inactive state of the **SSP_SCLK** pin when Motorola SPI format is selected (FRF=00). For SPO=0, the **SSP_SCLK** is held low in the inactive or idle state when the SSP is not transmitting/ receiving data. For SPO=1, the **SSP_SCLK** is held high during the inactive/idle state. The programmed setting of the SPO alone does not determine the **SSP_SCLK** edge that is used to transmit or receive data. The SPO setting in combination with the **SSP_SCLK** phase bit (SPH) determines this.

Note: The SPO is ignored for all data frame formats except for the Motorola SPI format (FRF=00).

19.5.2.5 Serial Clock Phase (SPH)

The serial clock (**SSP_SCLK**) phase bit (SPH) determines the phase relationship between the **SSP_SCLK** and the serial frame (**SSP_SFRM**) pins when the Motorola SPI format is selected (FRF=00). When SPH=0, **SSP_SCLK** remains in its inactive/idle state (as determined by the SPO setting) for one full cycle after **SSP_SFRM** is asserted low at the beginning of a frame.

SSP_SCLK continues to transition for the rest of the frame and is then held in its inactive state for one-half of an **SSP_SCLK** period before **SSP_SFRM** is de-asserted high at the end of the frame. When SPH=1, **SSP_SCLK** remains in its inactive/idle state (as determined by the SPO setting) for one-half cycle after **SSP_SFRM** is asserted low at the beginning of a frame. **SSP_SCLK** continues to transition for the rest of the frame and is then held in its inactive state for one full **SSP_SCLK** period before **SSP_SFRM** is de-asserted high at the end of the frame.

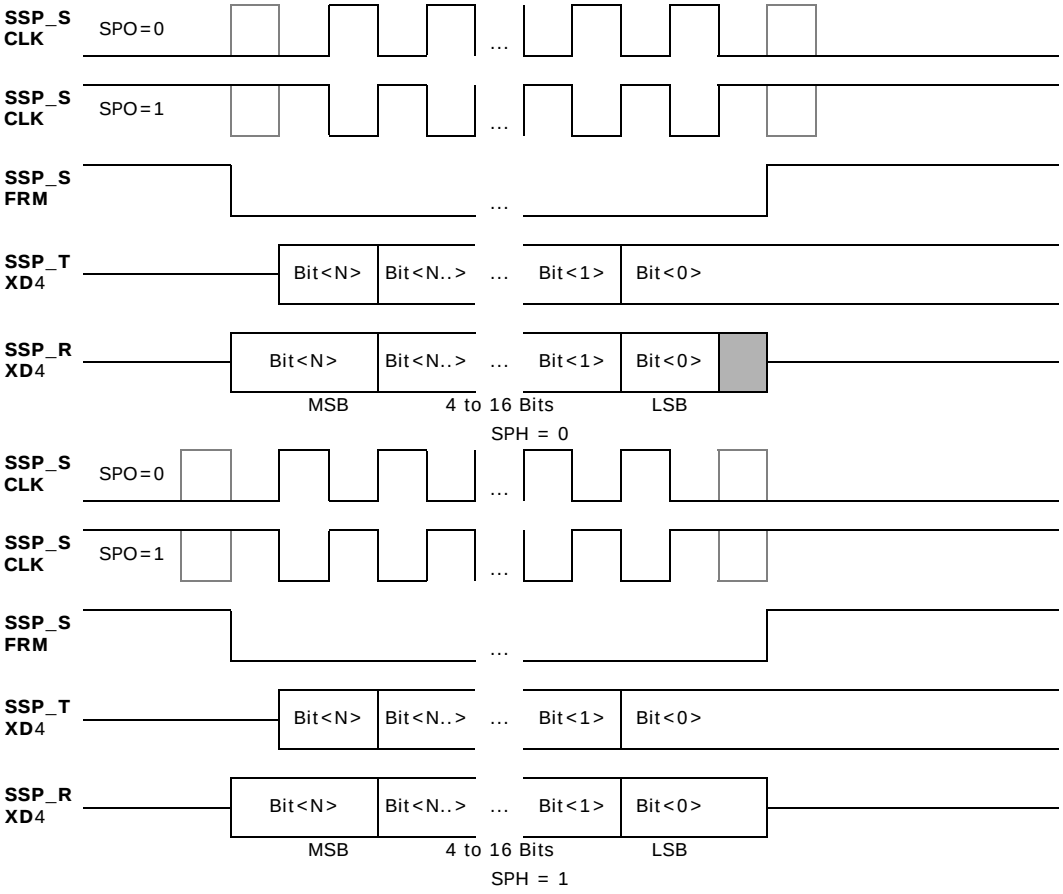
The combination of the SPO and SPH settings determines when **SSP_SCLK** is active during the assertion of **SSP_SFRM** and the **SSP_SCLK** edge that is used to transmit and receive data on the **SSP_TXD** and **SSP_RXD** pins. When SPO and SPH are programmed to the same value (both 0 or both 1), transmit data is driven on the falling edge of **SSP_SCLK** and receive data is latched on the rising edge of **SSP_SCLK**. When SPO and SPH are programmed to opposite values (one 0 and the other 1), transmit date is driven on the rising edge of **SSP_SCLK** and receive data is latched on the falling edge of **SSP_SCLK**.

Note: The SPH is ignored for all data frame formats except for the Motorola SPI format (FRF=00).

Table 251 shows the pin timing for all four programming combinations of SPO and SPH.

Note: The SPO inverts the polarity of the **SSP_SCLK** signal, and SPH determines the phase relationship between **SSP_SCLK** and **SSP_SFRM**, shifting the **SSP_SCLK** signal one-half phase to the left or right during the assertion of **SSP_SFRM**.

Table 251. Motorola* SPI Frame Formats for SPO and SPH Programming



19.5.2.6 National Microwire* Data Size (MWDS)

This bit sets the size of data in the Microwire format. If '1', a 16 bits data size is chosen for the Microwire format, else, an 8-bit data size.

19.5.2.7 Transmit FIFO Interrupt Threshold (TFT)

This 4-bit value sets the level at or below which the FIFO controller triggers a service interrupt.

19.5.2.8 Receive FIFO Interrupt Threshold (RFT)

This 4-bit value sets the level at or above which the FIFO controller triggers a service interrupt.

19.5.2.9 Enable FIFO Write/Read Function (EFWR)

This bit enables a special functional mode for the SSP. When EFWR = 0, then the SSP operates in the normal mode described in this document. When EFWR = 1, then the SSP enters a mode that whenever the CPU reads or writes to the SSP Data register it actually reads and writes exclusively to the Transmit FIFO or the Receive FIFO depending on the programmed state of the select FIFO for EFWR (STRF) bit. In this

special mode, data is not transmitted on the TXD pin and data input on the RXD pin is not stored. This mode is used to test through software, whether the Transmit FIFO or the Receive FIFO operates properly as a first-in-first-out memory stack.

19.5.2.10 Select FIFO for Enable FIFO Write/Read (STRF)

This bit selects whether the Transmit or Receive FIFO is enabled for writes and reads whenever the EFWR is programmed to one, that puts the SSP in a special functional mode.

Note: The following bit table shows bit locations corresponding to control bit fields in SSP Control Register 1. Note that writes to reserved bits are ignored, and reads of these bits return zero.

Register Name:	SSCR1				
Block Base Address:	0xC801_20	Offset Address	0x04	Reset Value	0x0000_0000
Register Description:	SSC Control Register 1			Access:	(See below.)

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
Reserved																STRF	EFWR	RFT				TFT			MWDS	SPH	SPO	LBM	TIE	RIE	

Register		SSCR1 (Sheet 1 of 2)		
Bits	**Name**	**Description**	**Reset value**	**Access**
31:16	(Reserved)	(Reserved)	0x0000	RV
15	STRF	Select FIFO for EFWR: 0 = Transmit FIFO is selected for enable FIFO write/read 1 = Receive FIFO is selected for enable FIFO write/read	0x0000	RW
14	EFWR	Enable FIFO Write/Read: 0 = FIFO write/read loopback function is disabled; normal operation enabled 1 = FIFO write/read loopback function is enabled	0x0000	RW
13:10	RFT	Receive FIFO Threshold. Sets threshold level to Receive FIFO asserts interrupt. This level should be set to the threshold value minus 1.	0x0000	RW
9:6	TFT	Transmit FIFO Threshold. Sets threshold level to Transmit FIFO asserts interrupt. This level should be set to the threshold value minus 1.	0x0000	RW
5	MWDS	National Microwire Data Size 0 = 8 bit Microwire format 1 = 16 bits Microwire format	0x0000	RW
4	SPH	Motorola SPI **SSP_SCLK** phase setting: 0 = **SSP_SCLK** is inactive one full cycle at the start of a frame and 1/2 cycle at the end of a frame. 1 = **SSP_SCLK** is inactive 1/2 cycle at the start of a frame and one full cycle at the end of a frame.	0x0000	RW
3	SPO	Motorola SPI **SSP_SCLK** polarity setting: 0 = The inactive or idle state of **SSP_SCLK** is low. 1 = The inactive or idle state of **SSP_SCLK** is high.	0x0000	RW
2	LBM	Loop Bank Mode Enable bit. 0 = Normal serial port operation enabled 1 = Output of transmit serial shifter connected to input of receive serial shifter, internally	0x0000	RW
1	TIE	Transmit FIFO Interrupt Enable 0 = Transmit FIFO level interrupt is disabled 1 = Transmit FIFO level interrupt is enabled	0x0000	RW

Register		SSCR1 (Sheet 2 of 2)		
Bits	Name	Description	Reset value	Access
0	RIE	Receive FIFO Interrupt Enable 0 = Receive FIFO level interrupt is disabled 1 = Receive FIFO level interrupt is enabled	0x0000	RW

19.5.3 SSP Status Register

The SSP status register (SSSR) contains bits that signal overrun errors and the Transmit and Receive FIFO service requests. Each of these hardware-detected events signal an interrupt request to the interrupt controller. The status register also contains flags that indicate when the SSP is actively transmitting characters, when the Transmit FIFO is not full, and when the Receive FIFO is not empty (no interrupt generated).

Bits that cause an interrupt signals the request as long as the bit is set. Once the bit is cleared, the interrupt is cleared. Read/write bits are called status bits, read-only bits are called flags. Status bits are referred to as sticky (once set by hardware, and is cleared by software). Writing a one to a sticky status bit clears it, writing a zero has no effect. Read-only flags are set and cleared by the hardware, writes have no effect. Additionally some bits that cause interrupts have corresponding mask bits in the control registers and are indicated in the section headings that follow.

19.5.3.1 Transmit FIFO Not Full Flag (TNF) (Read-Only, Non-Interruptible)

The Transmit FIFO not full flag (TNF) is a read-only bit that is set whenever the Transmit FIFO contains one or more entries that do not contain valid data. The TNF is cleared when the FIFO is completely full. This bit is polled when using programmed I/O to fill the Transmit FIFO over its half-way mark. This bit does not request an interrupt.

19.5.3.2 Receive FIFO Not Empty Flag (RNE) (Read-Only, Non-Interruptible)

The Receive FIFO not empty flag (RNE) is a read-only bit that is set whenever the Receive FIFO contains one or more entries of valid data and is cleared when it no longer contains any valid data. This bit is polled when using the programmed I/O to remove remaining bytes of data from the Receive FIFO since CPU interrupt requests are only made when the Receive FIFO threshold has been met or exceeded. This bit does not request an interrupt.

19.5.3.3 SSP Busy Flag (BSY) (Read-Only, Non-Interruptible)

The SSP busy (BSY) flag is a read-only bit that is set when the SSP is actively transmitting and/or receiving data, and is cleared when the SSP is idle or disabled (SSE=0). This bit does not request an interrupt.

19.5.3.4 Transmit FIFO Service Request Flag (TFS) (Read-Only, Maskable Interrupt)

The Transmit FIFO service request flag (TFS) is a read-only bit that is set when the Transmit FIFO is nearly empty and requires service to prevent an underrun. The TFS is set any time the Transmit FIFO has the same or fewer entries of valid data than indicated by the Transmit FIFO threshold, and it is cleared when it has more entries of valid data than the threshold value. When the TFS bit is set, an interrupt request is made unless the Transmit FIFO interrupt request enable (TIE) bit is cleared. As soon as the CPU fills the FIFO such that it exceeds the threshold, the TFS flag (and the service request and/or interrupt) is automatically cleared.

19.5.3.5 Receive FIFO Service Request Flag (RFS) (Read-Only, Maskable Interrupt)

The Receive FIFO service request flag (RFS) is a read-only bit that is set when the Receive FIFO is nearly filled and requires service to prevent an overrun. The RFS is set any time the Receive FIFO has the same or more entries of valid data than indicated by the Receive FIFO threshold, and it is cleared when it has fewer entries than the threshold value. When the RFS bit is set, an interrupt request is made unless the Receive FIFO interrupt request enable (RIE) bit is cleared. After the CPU reads the FIFO such that it has fewer entries than the RFT value, the RFS flag (and the service request and/or interrupt) is automatically cleared.

19.5.3.6 Receiver Overrun Status (ROR)

The Receiver Overrun status bit (ROR) is a read/write bit that is set when the receive logic attempts to place data into the Receive FIFO after it has been completely filled. Once the FIFO is filled, whenever a new piece of data is received, the set signal to the ROR bit is asserted, and the newly received data is discarded. This process is repeated for each new piece of data received until at least one empty FIFO entry exists. When the ROR bit is set, an interrupt request is made. Writing 1 to this bit resets ROR status and its interrupt request.

After this ROR interrupt occurs, it is recommended that the user read the Receive FIFO to empty it, then clear the ROR bit.

19.5.3.7 Transmit FIFO Level (TFL)

This 4-bit value shows the valid entries that are currently in the Transmit FIFO. See the register table below for details.

19.5.3.8 Receive FIFO Level (RFL)

This 4-bit value shows the valid entries that are currently in the Receive FIFO. See the register table below for details.

The following bit table shows the bit locations corresponding to the status and flag bits within the SSP status register. All the bits are read-only except ROR, that is read/write. Writes to TNF, RNE, BSY, TFS, and RFS have no effect.

Note: Writes to reserved bits are ignored and reads to those bits return zeros.

Register Name:	SSSR				
Block Base Address:	0xC801_20	Offset Address	0x08	Reset Value	0x0000_0000
Register Description:	SSP Status Register			Access:	(See below.)

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
Reserved																RFL				TFL				ROR	RFS	TFS	BSY	RNE	TNF	Rsvd.	

Register	SSSR (Sheet 1 of 2)			
Bits	Name	Description	Reset Value	Access
31:16	(Reserved)	(Reserved)	0x0000	RV

Register		SSSR (Sheet 2 of 2)		
Bits	**Name**	**Description**	**Reset Value**	**Access**
15:12	RFL	Receive FIFO Level: Number of entries minus one in Receive FIFO. ***Note:*** When the value 0xF is read, the FIFO is empty or full and the programmer should refer to the RNE bit.	0000	RO
11:08	TFL	Transmit FIFO Level: Number of entries in Transmit FIFO. ***Note:*** When the value 0x0 is read, the FIFO is empty or full and the programmer should refer to the TNF bit.	0000	RO
7	ROR	Receive FIFO Overrun: 0 = Receive FIFO has not experienced an overrun 1 = Attempted data write to full Receive FIFO, request interrupt	0	RW
6	RFS	Receive FIFO Service Request: 0 = Receive FIFO level is below RFT threshold, or SSP disabled. 1 = Receive FIFO level is at or above RFL threshold, request interrupt	0	RO
5	TFS	Transmit FIFO Service Request: 0 = Transmit FIFO level exceeds TFT threshold, or SSP disabled 1 = Transmit FIFO level is at or below TFL threshold, request interrupt	0	RO
4	BSY	SSP is busy 0 = SSP is idle or disabled 1 = SSP currently transmitting or receiving a frame	0	RO
3	RNE	Receive FIFO not empty. 0 = Receive FIFO is empty 1 = Receive FIFO is not empty	0	RO
2	TNF	Transmit FIFO not Full. 0 = Transmit FIFO is full 1 = Transmit FIFO is not full	1	RO
1:0	(Reserved)	(Reserved)	00	RV

19.5.4 SSP Interrupt Test Register (SSITR)

Writing 1 to the corresponding bit position of the SSP Interrupt Test Register generates an interrupt strobe signal to the Interrupt Controller or a DMA request for test purposes.

Note: SSITR functionality is available even when the SSP is disabled.

Register Name:	**SSITR**				
Block Base Address:	**0xC801_20**	**Offset Address**	**0x0C**	**Reset Value**	**0x0000_0000**
Register Description:	SSP Interrupt Test Register			**Access:**	(See below.)

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
Reserved																								TROR	TRFS	TTFS	Reserved				

Register		SSITR (Sheet 1 of 2)		
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:08	(Reserved)	(Reserved)	0x0000	RV
7	TROR	Test Receive FIFO overrun (ROR)	0x0000	RW

Register	SSITR (Sheet 2 of 2)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
6	TRFS	Test Receive FIFO service request (RFS)	0x0000	RW
5	TTFS	Test Transmit FIFO service request (TFS)	0x0000	RW
4:0	(Reserved)	(Reserved)	0x0000	RV

19.5.5 SSP Data Register (SSDR)

The SSP Data Register (SSDR) is a block of 32-bit locations that are accessed by 32-bit data transfers. The SSDR represents two physical registers, the first is temporary storage for data on its way out through the Transmit FIFO, while the other is temporary storage for data coming in through the Receive FIFO.

As the register is accessed by the system, the FIFO control logic transfers data automatically between register and FIFO as fast as the system moves it. Data in the FIFO shifts up or down to accommodate the new word (unless it is an attempted WRITE to a full Transmit FIFO). Status bits are available to show the system whether buffer is full, above/below a programmable threshold, or empty.

For outbound data transfers (WRITE from system to SSP peripheral), the register is loaded (written) by the system processor whenever it is empty.

When a data size of less than 16-bits is selected, the user should not left-justify data written to the Transmit FIFO. Transmit logic left-justifies the data and ignores any unused bits. Received data less than 16-bits is automatically right-justified in the receive buffer. When the SSP is programmed for National Microwire frame format, the default size for transmit data is 8-bits (the most significant byte is ignored), the receive data size is controlled by the programmer using the DSS field in SSCR0.

The following table shows the location of the SSP data register.

Note: Both the FIFOs are cleared when the block is reset, or by writing a zero to SSE (SSP disabled).

Register Name:	SSDR																													
Block Base Address:	0xC801_20						Offset Address						0x10						Reset Value						0x0000_0000					
Register Description:	SSP Data Register																					Access:	(See below.)							
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
Reserved																Data															

Register	SSDR			
Bits	**Name**	**Description**	**Reset Value**	**Access**
31:16	(Reserved)	(Reserved)	0x0000	
15:00	Data	Data (Low Word): Data word to be written to/read from the transmit/ Receive FIFO	0x0000	

20.0 AHB Queue Manager

20.1 Overview

The AHB Queue Manager (AQM) provides queue functionality for various cores. It maintains the queues as circular buffers in an embedded, 8-Kbyte SRAM. It also implements the status flags and pointers required for each queue.

The AQM manages 64 independent queues. Each queue is configurable for buffer and entry size. Additionally status flags are maintained for each queue.

The AQM interfaces include an advanced high-performance bus (AHB) interface to the NPEs and Intel XScale® Processor (or any other AHB bus master), a flag bus interface, an event bus (to the NPE condition select logic) and two interrupts to the Intel XScale processor. The AHB interface is used for configuration of the AQM and provides access to queues, queue status and SRAM. Individual queue status for queues 0-31 is communicated to the NPEs via the flag bus. Combined queue status for queues 32-63 are communicated to the NPEs via the event bus. The two interrupts, one for queues 0-31 and one for queues 32-63, provide status interrupts to the Intel XScale processor.

Read or write entries to a queue, is accomplished by performing AHB read/write accesses to any of the corresponding Queue Access Register addresses. The AQM intercepts these accesses, since no physical data resides at these addresses, and lookup the appropriate queue pointer to perform the requested read or write. Upon a read or write access to a queue, the pointers and status for the queue are updated as needed. Further detail is given in the following sections.

The lower queues provide individual status to the NPE Condition Coprocessor (CCP) via the flag bus interface. This means that any of these queues is individually assigned to events and scheduled accordingly. The upper queues have historically been managed as though they were a monolithic resource, where the resource is the entire group of queues and each queue is a sub-instance.

As the status for these upper queues is not directly connected to events, and the status flags are flattened in the status register, a software polling approach was used to determine the queues that requires attention. For this version of the AQM, this division between the upper and lower queues has been somewhat formalized. This definition has the upper queues providing all status, but in a flat fashion across each type of status register and the addition of the ability to schedule the upper queues as if they were a monolithic resource via a programmable event funnel into event bus flags. What these differences mean is discussed in later sections.

20.2 Feature List

The following are the features of the AHB Queue Manager:

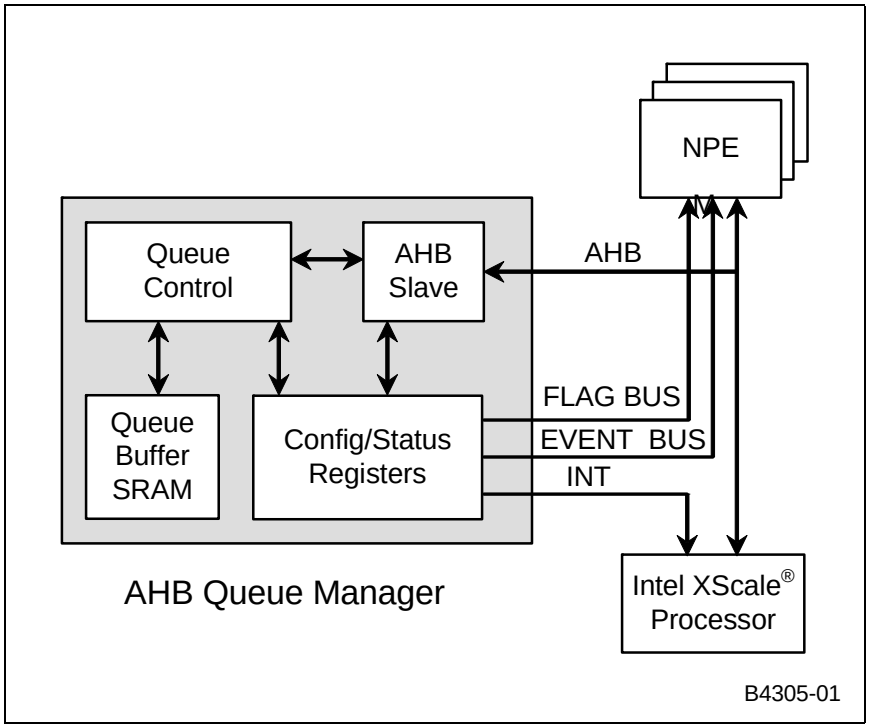
- Provides queue functionality for NPEs and the Intel XScale processor
- Manages 64 independent queues
- Implements queues as FIFOs with circular buffer rotation in SRAM
- Implements read/write pointers for each queue in SRAM

- Programmable queue entry size supported (queues are configured for 1, 2, or 4 word entries)
- Programmable queue size supported (queues are configured for 16, 32, 64 or 128 word buffer)
- Maintains empty (E), nearly empty (NE), nearly full (NF), and full (F) status flags on each of the queues 0-63
- Programmable queue watermarks for assessing NE and NF queue status flags
- Provides status flag information, E, NE, F and NF, for queues 0-31 to the NPEs via a common flag bus
- Provides Underflow and Overflow Status Flags for each of the queues 0-31
- Programmable event status enable for queues 32-63
- Two Intel XScale processor interrupts, one for queues 0-31 and one for queues 32-63
- Individual interrupt enables for each queue
- Programmable interrupt source for each of the queues 0-31 as the assertion or de-assertion of 1 of 4 status flags, E, NE, NF or F
- NE status flag used as the interrupt source for each of the queues 32-63
- Provides read/write access to all queues, queue pointers, status flags, configuration registers, interrupt registers and SRAM via the AHB
- SRAM core computes and checks parity and provides error registers

20.3 Block Diagram

Figure 161 shows a high level block diagram of the AQM and its connections:

Figure 161. AHB Queue Manager



The flag bus presents a unified flag bus to each of the NPE CCPs (Condition Coprocessors) that are to be connected. AQM instantiations are driven from the AHB bus and all operations are necessarily constrained to occur in the order they occur on the AHB. The timing between an AHB operation and a flag bus update is fixed. Not all NPEs are required to be connected to the flag bus via their CCP, so this connection scheme is very flexible.

The event outputs are also connected to the NPE CCP and wired to the appropriate events as the top level system requires. Each AQM produces three event outputs ‘A’, ‘B’ and ‘C’.

20.4 AHB Interface

The AHB interface provides read/write access to all AQM configuration/status registers, queues and SRAM. The AQM is a slave with a 32 bit data bus configuration on the AHB. The address map for the AQM is shown in Table 252, the addresses listed in the table are relative offsets from the from the AQM base address.

Unsupported exceptions to the AHB slave requirements include accesses with a data transfer size of byte or half-word, wrapping burst accesses and 16 beat incrementing burst accesses. These accesses results in an AHB Error response. Accesses to any unused locations within the AQM address space results in an OKAY response on the AHB. Read accesses to the unused address locations results in zeroes returned on the AHB. The AQM does not perform any internal operations on write accesses to any unused locations.The data formats for all registers accessible via the AHB is given in Section 20.6, “Register Descriptions” on page 740.

Table 252. AHB Queue Manager Memory Map (Sheet 1 of 2)

Address	AQM Function
0x03FFC 0x02100	64 Queue Buffer Space - SRAM 1,984 x 4 Bytes
0x020FC 0x02000	64 queue configuration words - SRAM 64 x 4 Bytes
0x01FFC 0x00468	UNUSED ADDRESS SPACE
0x00464	Reserved
0x00460	Error Control Register
0x0045C	Queue 0 to 31 Status Map Register
0x00458	Queue 32 to 63 Event Source Register
0x00454	Reserved
0x00450	Queue 32 to 63 Event ‘C’ Enable Register
0x0044C	Queue 32 to 63 Event ‘B’ Enable Register

Table 252. AHB Queue Manager Memory Map (Sheet 2 of 2)

Address	AQM Function
0x00448	Queue 32 to 63 Event 'A' Enable Register
0x00444	Queue 32 to 63 Nearly Full Status Register
0x00440	Queue 32 to 63 Empty Status Register
0x0043C 0x00438	Queue 0 to 63 Interrupt Register 2 x 4 Bytes
0x00434 0x00430	Queue 0 to 63 Interrupt Enable Register 2 x 4 Bytes
0x0042C 0x00420	Queue 0 to 31 Interrupt Status Flag Source Select Register 4 x 4 Bytes
0x0041C	Queue 32 to 63 Full Status Register
0x00418	Queue 32 to 63 Nearly Empty Status Register
0x00414 0x00410	Queue 0 to 31 Underflow/Overflow Status Register 2 x 4 Bytes
0x0040C 0x00400	Queue 0 to 31 Status Registers 4 x 4 bytes
0x003FC 0x00000	Queue 0 to 63 Read/Write Access 64 x 16 bytes

20.4.1 Queue Control

The queues are implemented as circular buffers where adding an entry is performed by a write to a queue and removing an entry is performed by a read from a queue. Entries are read from a queue in the same order that they were written to the queue. The read/write pointers track the removal/addition of entries from/to a queue. The queue control performs the autonomous access of the queues.

External agents wanting to access a queue should perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM performs the requested access to the queue in SRAM. Support is provided for 64 queues. Upon receiving a queue read or queue write from the AHB interface, queue control fetches the selected queue configuration from SRAM. The queues or circular buffers reside in internal SRAM. Configuration for each queue consists of:

- A Base Address — The address where the queue starts and is configurable for placing the queue buffer on any 16 word boundary within the SRAM address range of 000H to 7C0H (word address). These SRAM address ranges correspond to AQM address ranges 0x2000 to 0x3FFC respectively. Note that the lower SRAM addresses are used to store the queue control words and therefore should not be used for queue data storage.
- A Write Pointer — A pointer to the next queue location to be written and is maintained by queue control.
- A Read Pointer — A pointer to the next location to be read and is maintained by queue control.

- Queue Entry Size — The size of each queue entry. This is configurable for 1, 2, or 4 words.
- Queue Size — The number of words allocated to the queue and is configurable for 16, 32, 64, or 128 words.
  - — Examples of how to use the Queue Entry Size and Queue Size parameters to obtain number of Q-entries for each of the 64 queues when program a Q-Manager size of 8KB
    - Queue Size(128 words)/Queue Entry Size(4 words) = 32 entries of 4 words each
    - Queue Size(128 words)/Queue Entry Size(2 words) = 64 entries of 2 words each
    - Queue Size(128 words)/Queue Entry Size(1 word) = 128 entries of 1word each
    - Queue Size(32 words)/Queue Entry Size(4 words) = 8 entries of 4 words each
    - One more note, when the maximum Q-Manager SRAM utilization is 1984 Words (assuming all queues are being used), the Queue size cannot be chosen to be the maximum size for every queue. For example 1984Words/128 words = 15 queues worth of data with 64 words worth of data left over. Note that since one cannot program a queue size of 15, the next smallest queue depth is used or the queues are of various depths.
- NE watermark – this field indicates the maximum number of occupied entries that a queue is considered to be nearly empty. It is set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.
- NF watermark – this field indicates the maximum number of empty entries that a queue is considered to be nearly full. It is set to 0, 1, 2, 4, 8, 16, 32, or 64 entries.

Figure 162. Representative Logical Diagram of a Queue

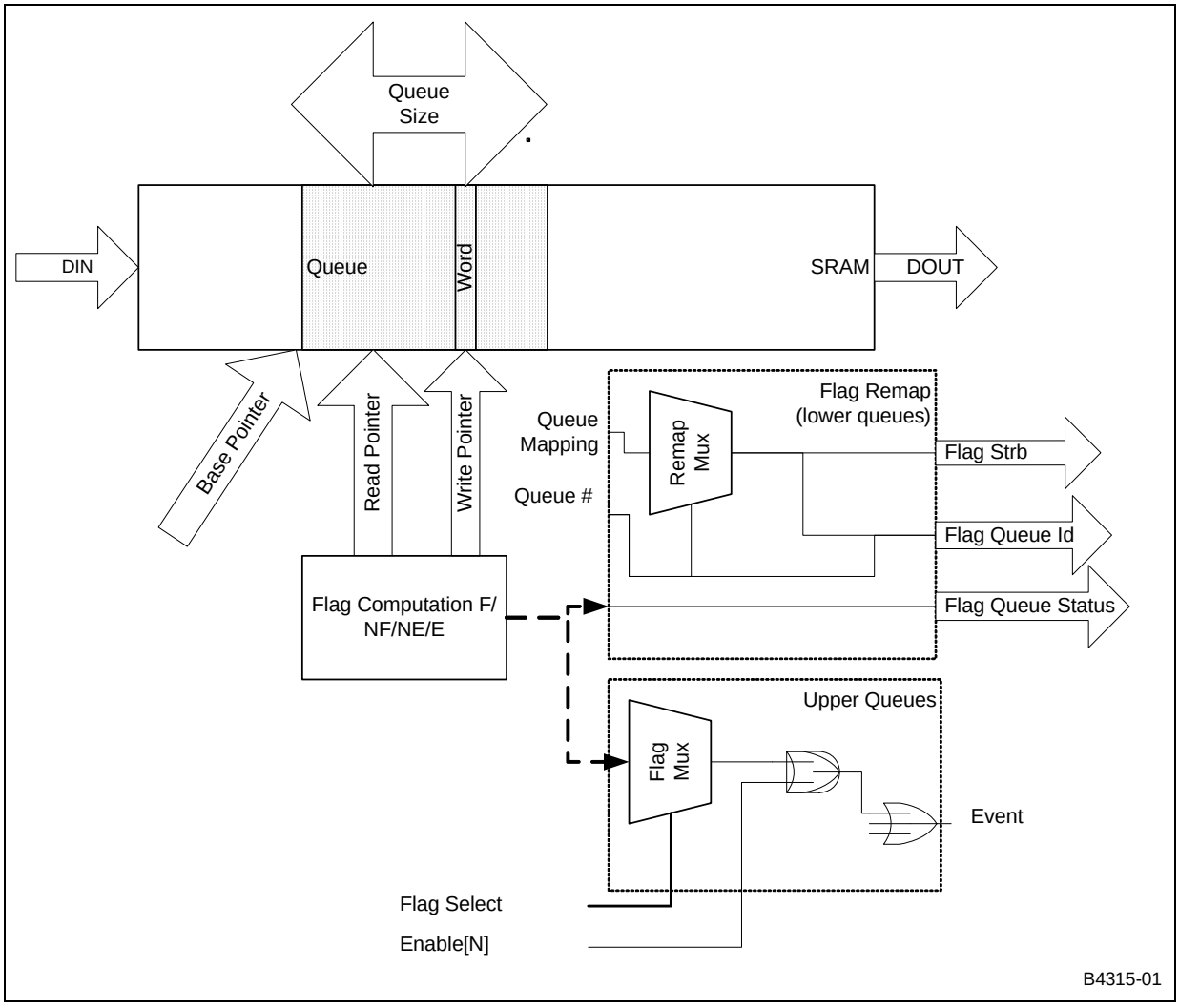


Figure 162 shows the logical diagram of what any given queue looks like. The actual implementation of the AQM is as a monolithic controller of queues, but you can look at each queue as if it stands alone. Each queue fits into the SRAM at a certain base address, is a certain size, and maintains its own flags. The output of those flags depends on whether the queue is a lower queue (0-31) or an upper queue (32-63). The detailed description follows.

To access any given queue, after the selected queue configuration is read from SRAM, the queue base address is summed with the read or write pointer to form the queue address. If the queue is not empty, when the request is a read, or full, when the request is a write, queue control performs the requested queue access at the calculated queue address.

If the request is a read, the queue data read from SRAM at the calculated queue address is passed to the AHB interface for the corresponding data acknowledge to the AHB read request. If the request is a write, the data from the AHB interface is written into SRAM at the calculated queue address. When the read and write pointers are equal, the queue is full or empty as determined by the full or empty status flags.

When a read request of an empty queue buffer is performed, the queue control returns zeroes in the data field to the AHB interface and sets the Underflow Status Flag. When a write request of a full queue is performed, the data is discarded and queue control sets the Overflow Status Flag. Underflow and Overflow Status Flags are maintained on queues 0-31 only. Otherwise for a read request of an empty queue or write request of a full queue, queue control does not perform any action upon the queue buffer or queue configuration word.

Following the queue access, the appropriate read or write pointer, as indicated by the type of queue access is incremented and the queue configuration word, with all other fields maintained, is written back into SRAM. The queue size is used in determining the number of active bits within the allocated 7 bit field for the read and write pointers. Configurable queue sizes of 16, 32, 64 and 128 directly correspond to read and write active bit widths of 4, 5, 6, and 7. The unused bits of the read and write pointers is zeroed prior to writing the queue configuration word back into SRAM.

One to four Queue Access Register addresses, 0, 4, 8, and 0xC, are allocated per queue as determined by the queue's programmed entry size. Thus a queue, with an entry size set to one word, supports accesses to the first location, 0, and a queue with an entry size set to two, supports accesses to the first and second locations, 0 and 4. A queue with an entry size set to four words support accesses to all four locations. Accesses to the non-supported locations is not performed and queue read/write pointers is unchanged.

A queue, with a programmed entry size of two or four words, requires the two or four accesses of each queue entry to be performed sequentially beginning with address 0. Out of order accesses are not performed. Incomplete accesses (For example, reading only the first two words of a four word entry) do not update the pointers. The sequential accesses is performed via multiple single word accesses or via a burst access on the AHB. If a Queue burst access of more than four words is attempted, the AQM performs the accesses to the first two or four locations, as determined by the queue's entry size, and the remainder of the accesses of the burst is not performed. Queue read accesses that are not performed, returns zeroes in the data field on the AHB.

Note: The entry size parameter does not affect the data type. The AQM does not know the semantic meaning of the entries in the queues, they are all 32-bit words. These entries are pointers, data blocks, big-endian packed data or little-endian packed data. The entry size does not really change anything except how the AHB can optimize a transfer. For example, if an entry size is set to two words, a device on the AHB can read and write this queue in a bus burst operation of length two, that optimizes bus access.

An entry size of four words enables a bus burst of length 4. To support bursting on AHB, one must have consecutive addresses since that is how the protocol works, and so to support this the entries have multiple addresses. Though individual reads and writes are performed to these consecutive addresses, unless there is some kind of bursting, the difference between individual reads to the same location and individual reads to consecutive locations is small. This means that unless a given AHB source device is using a burst access to read or write the AQM there is little AHB performance benefit to programming the word size of two or four (although there are software advantages).

20.4.2 Queue Status

Status information for the 64 queues is provided in the status registers. Six status flags is maintained for each of the queues 0-31, empty, nearly empty, full, nearly full, underflow, and overflow. Only the four empty, nearly empty, nearly full, and full status flags are provided for queues 32-63. The status flags is read/write accessible via the AHB, although the arrangement of flags differs between queues 0-31 and 32-63. The flag bus communicates status for queues 0-31 to the NPEs, the status funnel[1]

communicates status for queues 32-63 to the event inputs to the NPEs, and two interrupts provides status interrupting capability for the Intel XScale processor. The following sections outline the queue status requirements.

20.4.2.1 Status Update

Following any updates for a queue access, the read and write pointers is used in determination of status flag settings for the accessed queue. When the queue entry size is set to 1, status flags are updated following every queue access. If the queues are set for multi-word entry sizes, 2 or 4, the status flags is updated following the queue access, that completely fills or empties a queue entry.

If the read and write pointers are equal and the last access to the queue was a read, then the queue is empty. If the read and write pointers are equal and the last access was a write, then the queue is full. The nearly empty and nearly full configurable watermarks are used in determining the settings for the NE and NF Status flags. These watermarks is set to 0, 1, 2, 4, 8, 16, 32, or 64 entries. If the number of completely empty entries is less than or equal to the full watermark, the queue is considered nearly full. If the number of completely full entries is less than or equal to the empty watermark, the queue is considered nearly empty.

The status can be read at any time by an AHB read of the appropriate status register. The status read reflects the status of the queue at the time of the read, respecting all previous AHB operations. For example, if two closely spaced AHB reads are performed, one to a queue and another to that queue's status register, the status register read reflects the completion of the queue read. This requires that the second read should have additional wait states inserted (by hardware) into the bus operation to insure that the first operation has been completed.

The operation of the Nearly Full, Nearly Empty, Full and Empty Status Flags with the Nearly Empty and Nearly Full watermarks set to various levels is demonstrated in Table 253, "Queue Status Flags" on page 734. For this example, the buffer size is set to 64 and the entry size is set to 1. When the watermarks are set to zero, the Nearly Empty and Empty Flags is identical and the Nearly Full and Full Flags is identical.

Note: In Table 253, the **# Entries in the Queue** represents the cardinal number of entries in the queue for each watermark value.

Table 253. Queue Status Flags (Sheet 1 of 2)

Nearly Empty Watermark	Nearly Full Watermark	# Entries in the Queue	E	NE	NF	F
0 (000)	0 (000)	0	1	1	0	0
		1 – 63	0	0	0	0
		64	0	0	1	1
1 (001)	1 (001)	0	1	1	0	0
		1	0	1	0	0
		2 – 62	0	0	0	0
		63	0	0	1	0
		64	0	0	1	1

1. A **status funnel**, in this context, is the logical OR of a status word across the field. For example, if any bit of the Empty status register is set, the empty status funnel is true.

Table 253. Queue Status Flags (Sheet 2 of 2)

Nearly Empty Watermark	Nearly Full Watermark	# Entries in the Queue	E	NE	NF	F
2 (010)	2 (010)	0	1	1	0	0
		1 – 2	0	1	0	0
		3 - 61	0	0	0	0
		62 – 63	0	0	1	0
		64	0	0	1	1
4 (011)	4 (011)	0	1	1	0	0
		1 – 4	0	1	0	0
		5 – 59	0	0	0	0
		60- 63	0	0	1	0
		64	0	0	1	1
8 (100)	8 (100)	0	1	1	0	0
		1 - 8	0	1	0	0
		9 – 55	0	0	0	0
		56 - 63	0	0	1	0
		64	0	0	1	1

20.4.2.2 Status Interrupts

Two processor interrupts is provided, one for queues 0-31 and one for queues 32-63. Each of the interrupt signals is computed as a masked 32-way logical-OR of one edge-sensitive status bit per queue. In other words, each queue contributes a single edge-sensitive input into one of the 32-way logical-OR combinations. For queues 0-31, this input is independently configurable. It is a positive or negative edge-sensitive version of any one of the E, NE, NF or F status flag bits.

The selected status flag can be different for each queue. For queues 32-63, the input is always the NE status flag bit with a positive edge-sensitive version only. The set of selected 32 condition signals is masked by the corresponding interrupt enable register but the value of INT0SRCSELREG0 bit 3 affects how this happens in a subtle way.

There is a bit in INT0SRCSELREG0 that modifies the reset operation of the interrupts. If bit 3 of this register is set to 0, then these interrupts are generated for active high, level triggered usage. On occurrence of the selected transition of one or more of the status flag sources, an active high interrupt level is registered. Via the AHB, the processor can read a 32-bit Interrupt register to determine the source or sources for each interrupt.

Selective interrupt reset capability is provided for each of the queue sources via writing a one to the appropriate queue bit(s) within the interrupt register. Upon clearing that is, writing a ‘1’ to the appropriate bit(s) in the Interrupt Register, the interrupt cannot be generated again by the same source, until the active status flag condition is removed and then are asserted again. If the interrupt is reset, but the condition is still active, the interrupt register does not reflect the status. In this mode, the interrupt enable register affects whether the bit is set, and the interrupt is the logical ‘OR’ of the interrupt register.

If this bit is set to 1, then the interrupts resets if the interrupting condition has also been cleared when the write to the QUEINTREG occurs. In other words, this bit determines if the interrupt is globally rising edge sensitive (INT0SCRSELREG0[3] is ‘0’) or is level sensitive (INT0SCRSELREG0[3] is ‘1’). In addition, the interrupt mask is applied after the interrupt register. This means that the QUEINTREG register reflects the active status of masked interrupts.

Note: Notice that there is a subtle difference in how the interrupt mask is applied in these two cases. If the bit is ‘0’ the interrupt mask prevents the setting of the interrupt status

register bit whereas if the bit is '1' the interrupt mask prevents the propagation of the interrupt status register bit. This introduces a corner case. If the bit is set to a one, an interrupt is generated on a queue and then that interrupt is subsequently disabled, but it is not cleared (because there is no write to QUEINTREG) or is not clearable (because the condition is still true and in this mode it is level sensitive) then if you change the mode to '0' the interrupt mask is no longer applied, and a (spurious) interrupt is generated. The interrupt itself cannot be prevented, but after the mode switch the interrupt is cleared and subsequently remains masked.

20.4.3 AQM SRAM

The queue buffers and queue configuration words managed by the AQM reside in on-chip SRAM. The SRAM is a 2K by 36 bits wide implementation (where the **extra** 4 bits are used for parity protection). It is a single port synchronous SRAM that supports write accesses and a 133 MHz clock. The SRAM is fully accessible via the AHB interface to the AQM and the queue control for queue access.

The queue configuration words are located in the bottom 64 words of the SRAM with the remainder of the SRAM being allocated to queue buffers. The SRAM is freely written and read, although writes has obvious side effects. Overwriting the configuration words results in unpredictable behavior in the queues. Therefore, reads of the SRAM may not commingle with the normal usage of the queue manager's queues. Reading and writing the SRAM may be done as part of a diagnostic test or as part of the normal operation of the AQM.

The existence of parity requires that the memory be initialized to a defined pattern before it is read. The SRAM is divided into two regions, the queue configuration words and the buffer space. Reading a location directly or by operating the AQM in its natural mode without initializing the location is most likely going to produce a parity error. For this reason, software must take care to initialize the AQM SRAM or guarantee that no reads are ever performed before a write.

To initialize the SRAM for a strict queue usage model, all that is necessary is to initialize the queue control word for the specific queue and to guarantee that no reads are done before the first write. Accessing the SRAM for direct access operations that is, addresses > 0x2000 may produce corner cases where **left over** pointers (from previous operations) to uninitialized locations cause spurious and hidden reads.

For this reason, if the queue manager SRAM is expected to support direct access operations, the entire SRAM has to be initialized to guarantee that no spurious errors is detected. The safest solution is for the boot software to always initialize the SRAM to a constant value (such as 0x00000000) after power-up or reset. Parity errors is not reported if parity reporting is disabled in the address error register, and this bit should be left in the disable position until memory is initialized.

A parity error is signaled via an interrupt to the Intel XScale processor or some other mechanism. When the Error bit in the QUEADDERR register is true the aqm_parity_error port is true, and remains true until the Error bit is cleared. There is little or no recourse for parity errors, and the detection of a parity error indicates that a fatal and unrecoverable error has occurred in the AQM SRAM. But, the logical operation of the AQM is marginally intact. Should the parity error occur on a data fetch only the data is corrupted. Should the parity error occur on a configuration word fetch, the entire operation of that queue (and whether it interferes with the operation of another queue) is compromised.

It should be assumed that any parity error requires essentially a complete software reset of the AQM and in general any stored data is invalid. To accomplish this software reset, the AQM must have parity disabled and the entire memory should be reinitialized. After reinitialization normal operation may resume, although obviously any stored data is now destroyed.

20.4.4 Data Validity

The AHB queue manager can produce incorrect data under certain contexts, and these contexts must be understood. Some of these cases represent incorrect software usage of the AQM, others represent hardware conditions. The following table summarizes the cases where the AQM does not return valid data, how a software stack is notified and what (if anything) is done.

Table 254. Data Validity Cases and Their Handling

Case	Hardware Signal	Software Signal	Recovery
Unsupported bus operation	AHB Error response	Data Abort fault to Intel XScale processor Error to AHB COP Target Abort to PCI Forced parity error to expansion bus - may result in silent data corruption if expansion device ignores parity.	None
Out of range SRAM address	None	Zero value returned on all reads, writes ignored.	Detect zero value, handle as exception
Reading incorrect number of queue words	None	Zero value returned on all reads, writes ignored.	Detect zero value, handle as exception
Queue overflow on write	None	Status flag set.	None, data is lost.
Queue underflow on read	None	Status flag set, zero value returned	Detect zero value, handle as exception
Parity error without parity detection enabled	None	None, mostly results in silent corrupted data.	None
Parity error on data during queue read	AHB Error response	Data Abort fault to Intel XScale processor Error to AHB COP Target Abort to PCI Forced parity error to expansion bus - may result in silent data corruption if expansion device ignores parity. Interrupt signal to Intel XScale processor	Reset and restart application usage of AQM

As described earlier, not all AHB cycle types are supported; with certain ones automatically producing an Error response. The data returned for these cases is not defined.

Data returned for out-of-range SRAM access or MMR access is always zero and no error condition is signaled. If software should attempt reads or writes of out-of-range addresses, there is not any obvious notification except that the data read is zero and data written is ignored.

Reading (For example) two words out of a single word queue returns data for the first word and zeros for the second word. The fact that the second word is not defined is not signaled. In general, a mismatch between the number of words in the entry between the actual usage and the configuration word definition results in lost data (that is, as an overflow) or zeros.

On the overflow condition, the written data is permanently lost. On the underflow condition, the data returned is zero.

If there are parity errors where the parity notification is not enabled, the data returned represents the data containing the parity error. If parity notification is enabled and the entry contains a parity error, the error is signaled on that particular bus cycle and consistent with the AHB specification, the master has the option of terminating a burst operation on the first error.

In general, a software usage model should treat the return of zero data as a suspicious case and the data type stored in the AQM should take this into account. If the data type were to be memory address pointers, this is well behaved since a null pointer is not defined. Other data types should take this into account.

20.4.5 Burst Operations to Queues

Burst operations are useful for writing multiple entry queues. An INCR4 burst can fill an entire entry for a four entry queue without having to re-arbitrate for ownership of the AHB bus, that enhances performance. But, if the burst type does not match the entry size for the queue, the transaction data to the inactive queue entries is dropped. In general, software should avoid burst transactions to queues that have fewer words per entry than the burst because of the decreased performance.

20.5 Detailed Register Descriptions

This section provides the detailed register descriptions:

Table 255. Register Legend

Attribute	Legend	Attribute	Legend
RV	Reserved	RC	Read Clear
PR	Preserved	RO	Read Only
RS	Read/Set	WO	Write Only
RW	Read/Write	NA	Not Accessible

Table 256. Register Summary (Sheet 1 of 2)

Address	Register Name	Description	Reset Value	Attribute
0x6000_0000	QUEACC0_0	Queue 0 word 0 data register	N/A	RW
0x6000_0004	QUEACC0_1	Queue 0 word 1 data register (used only when Queue 0 has a entry size of 2 or 4)	N/A	RW
0x6000_0008	QUEACC0_2	Queue 0 word 2 data register (used only when Queue 0 has a entry size of 4)	N/A	RW
0x6000_000C	QUEACC0_3	Queue 0 word 3 data register (used only when Queue 0 has a entry size of 4)	N/A	RW
0x6000_0010	QUEACC1_0	Queue 1 word 0 data register	N/A	RW
0x6000_0014	QUEACC1_1	Queue 1 word 1 data register (used only when Queue 1has a entry size of 2 or 4)	N/A	RW
0x6000_0018	QUEACC1_2	Queue 1 word 2 data register (used only when Queue 1has a entry size of 4)	N/A	RW
0x6000_001C	QUEACC1_3	Queue 1 word 3 data register (used only when Queue 1has a entry size of 4)	N/A	RW
...	...	...	N/A	RW
0x6000_03F0	QUEACC63_0	Queue 63word 0 data register	N/A	RW
0x6000_03F4	QUEACC63_1	Queue 63 word 1 data register (used only when Queue 63 has a entry size of 2 or 4)	N/A	RW

Table 256. Register Summary (Sheet 2 of 2)

Address	Register Name	Description	Reset Value	Attribute
0x6000_03F8	QUEACC63_2	Queue 63 word 2 data register (used only when Queue 63 has a entry size of 4)	N/A	RW
0x6000_03FC	QUEACC63_3	Queue 63 word 3 data register (used only when Queue 63 has a entry size of 4)	N/A	RW
0x6000_0400	QUELOSTAT0	Queue 0-7 complete status	0x33333333	RW
0x6000_0404	QUELOSTAT1	Queue 8-15 complete status	0x33333333	RW
0x6000_0408	QUELOSTAT2	Queue 16-23 complete status	0x33333333	RW
0x6000_040C	QUELOSTAT3	Queue 24-31 complete status	0x33333333	RW
0x6000_0410	QUEUOSTAT0	Queue 0-15 Overflow/Underflow Status	0x00000000	RW
0x6000_0414	QUEUOSTAT1	Queue 16-31 Overflow/Underflow Status	0x00000000	RW
0x6000_0418	QUEUPSTATNE	Queue 32-63 Nearly Empty Status Flags	0x00000000	RW
0x6000_041C	QUEUPSTATF	Queue 32-63 Full Status Flags	0x00000000	RW
0x6000_0420	INTR0SRCSELREG0	Interrupt 0 Status Flag Source Select for Queues 0-7	0x00000000	RW
0x6000_0424	INTR0SRCSELREG1	Interrupt 0 Status Flag Source Select for Queues 8-15	0x00000000	RW
0x6000_0428	INTR0SRCSELREG2	Interrupt 0 Status Flag Source Select for Queues 16-23	0x00000000	RW
0x6000_042C	INTR0SRCSELREG3	Interrupt 0 Status Flag Source Select for Queues 24-31	0x00000000	RW
0x6000_0430	QUEIEREG0	Queue Interrupt Enable for Queues 0-31	0x00000000	RW
0x6000_0434	QUEIEREG1	Queue Interrupt Enable for Queues 32-63	0x00000000	RW
0x6000_0438	QUEINTREG0	Queue Interrupt for Queues 0-31	0x00000000	RW
0x6000_043C	QUEINTREG1	Queue Interrupt for Queues 32-63	0x00000000	RW
0x6000_0440	QUEUPSTATE	Queue 32-63 Empty Status Flags	0x00000000	RW
0x6000_0444	QUEUPSTATNF	Queue 32-63 Nearly Full Status Flags	0x00000000	RW
0x6000_0448	QUEUPEVA	Queue 32-63 Empty Event ‘A’ Enable	0x00000000	RW
0x6000_044C	QUEUPEVB	Queue 32-63 Empty Event ‘B’ Enable	0x00000000	RW
0x6000_0450	QUEUPEVC	Queue 32-63 Empty Event ‘C’ Enable	0x00000000	RW
0x6000_0454		(Reserved)		
0x6000_0458	QUEUPSTATSRC	Queue 32-63 Event Source Register	0x00000000	
0x6000_045C	QUELOSTATMAP	Flag bus mapping for queues 0-31		RW
0x6000_0460	QUEADDERR	Parity Error Address and Control	0x00010000	RW
0x6000_0464		(Reserved)		
0x6000_2000	QUECONFIG0	Queue 0 configuration	0	RW
...	...	...	0	RW
0x6000_20FC	QUECONFIG63	Queue 63 configuration	0	RW
0x6000_2100		SRAM queue data buffer	0	RW
...		SRAM queue data buffer	0	RW
0x6000_3FFC		SRAM queue data buffer	0	RW

20.6 Register Descriptions

The following sub-sections describes the AHB Queue Manager registers.

20.6.1 Queue Access Word Registers 0 - 63

External agents wanting to access a queue perform an AHB read or write to the Queue Access Register locations. As a result of the access to these locations, the AQM performs the requested access to the queue in SRAM. See Section 20.4.1, "Queue Control" on page 730 for clarification on AHB queue accesses to the AQM. As described above, these queue access registers are defined in a block of four 32-bit words, where only the first 32-bit word is defined for a word size of one, only the first two 32-bit words are defined for a word size of two and all four 32-bit words are defined for a word size of four.

Register Name:	QUEACC (0 <= n <=63)				
Block Base Address (BBA):	Queue #n 0x0000	Offset Address	BBA+ 16n + 4x	Reset Value	Not Applicable
Register Description:	Queue #n access register. There are 1-4 addresses (0 <= x <=3), as determined by the programmed entry size, for requesting read/write accesses to individual queues. No physical data resides at these addresses.			Access:	(See below.)

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
Queue Read/Write Data																															

Register	QUEACC (0 <= n <=63)			
Bits	Name	Description	Reset Value	Access
31:0	Queue Read/ Write Data	Queue data word. Addresses addressed consecutively program in multiple entry sizes. For example, QUEACC0 word 0 is at address offset 0x0000 and QUEACC0 word 1 is at address offset 0x0004.	N/A	RW

20.6.2 Queues 0-31 Status Register 0 - 3

The access to these status registers is read/write, except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually *change* the status, it only writes the shadow register that contains the status.

Register Name:	QUELOWSTAT (0 <= n <=3)				
Block Base Address:	Reg #n 0x0400	Offset Address	+ 4n	Reset Value	0x33333333
Register Description:	Queue status register for the queues 0-31. F/NF/NE/E: '1' – active flag.			Access:	(See below.)

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
Queue(8n+7)				Queue(8n+6)				Queue(8n+5)				Queue(8n+4)				Queue(8n+3)				Queue(8n+2)				Queue(8n+1)				Queue(8n)			

Register	QUELOWSTAT (0 <= n <=3)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
4k+3 :4k	Queue(8n+k) Status Flags	(0 <= k <= 7) Queue (8n+k) complete status flags of: Full / Nearly Full / Nearly Empty / Empty For each flag, '1' is active.	0x3	RW

20.6.3 Underflow/Overflow Status Register 0 - 1

The access to these status registers is read/write, except for initialization, diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register that contains the status.

Register Name:	QUEUOSTAT (0 <= n <=1)					
Block Base Address:	Reg #n 0x0410	Offset Address	+ 4n	Reset Value	0x00000000	
Register Description:	Queue underflow/overflow status register for the queues 0-31. OF/UF: '1' – Overflow/Underflow has occurred.			Access:	(See below.)	

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
Queue (16n + 15)		Queue (16n + 14)		Queue (16n + 13)		Queue (16n + 12)		Queue (16n + 11)		Queue (16n + 10)		Queue (16n + 9)		Queue (16n + 8)		Queue (16n + 7)		Queue (16n + 6)		Queue (16n + 5)		Queue (16n + 4)		Queue (16n + 3)		Queue (16n + 2)		Queue (16n + 1)		Queue (16n)	

Register	QUEUOSTAT (0 <= n <=1)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
2k+1 :2k	OF/UF	(0 <= k <= 7) Queue (16n+k) Overflow and Underflow bit, respectively.	'b00	RW

20.6.4 Queues 32-63 Empty Status Register

The access to these status registers is read/write, except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status; it only writes the shadow register that contains the status.

Register Name:	QUEUPPSTATE					
Block Base Address:	0x0440	Offset Address	+ 4n	Reset Value	0xFFFFFFFF	
Register Description:	Queue status register for queues 32-63. E: '1' – flag set.			Access:	(See below.)	

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
Q63 E	Q62 E	Q61 E	Q60 E	Q59 E	Q58 E	Q57 E	Q56 E	Q55 E	Q54 E	Q53 E	Q52 E	Q51 E	Q50 E	Q49 E	Q48 E	Q47 E	Q46 E	Q45 E	Q44 E	Q43 E	Q42 E	Q41 E	Q40 E	Q39 E	Q38 E	Q37 E	Q36 E	Q35 E	Q34 E	Q33 E	Q32 E

Register	QUEUPPSTATE			
Bits	**Name**	**Description**	**Reset Value**	**Access**
k	Empty	(0 <= k <= 31) Queue (k) Empty Status Flag.	1	RW

20.6.5 Queues 32-63 Nearly Empty Status Register

The access to these status registers is read/write, except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register that contains the status.

Register Name:	QUEUPPSTATNE																															
Block Base Address:	0x0418						Offset Address						+ 4n						Reset Value						0xFFFFFFFF							
Register Description:	Queue status register for queues 32-63. NE: '1' – flag set.																							Access:	(See below.)							
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**
Q63 NE	Q62 NE	Q61 NE	Q60 NE	Q59 NE	Q58 NE	Q57 NE	Q56 NE	Q55 NE	Q54 NE	Q53 NE	Q52 NE	Q51 NE	Q50 NE	Q49 NE	Q48 NE	Q47 NE	Q46 NE	Q45 NE	Q44 NE	Q43 NE	Q42 NE	Q41 NE	Q40 NE	Q39 NE	Q38 NE	Q37 NE	Q36 NE	Q35 NE	Q34 NE	Q33 NE	Q32 NE

Register	QUEUPPSTATNE			
Bits	**Name**	**Description**	**Reset Value**	**Access**
k	Nearly Empty	(0 <= k <= 31) Queue (k) Nearly Empty Status Flag.	1	RW

20.6.6 Queues 32-63 Nearly Full Status Register

The access to these status registers is read/write, except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register that contains the status.

Register Name:	QUEUPPSTATNF																															
Block Base Address:	0x0444						Offset Address						+ 4n						Reset Value						0x00000000							
Register Description:	Queue status register for queues 32-63. NE: '1' – flag set.																							Access:	(See below.)							
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**
Q63 NF	Q62 NF	Q61 NF	Q60 NF	Q59 NF	Q58 NF	Q57 NF	Q56 NF	Q55 NF	Q54 NF	Q53 NF	Q52 NF	Q51 NF	Q50 NF	Q49 NF	Q48 NF	Q47 NF	Q46 NF	Q45 NF	Q44 NF	Q43 NF	Q42 NF	Q41 NF	Q40 NF	Q39 NF	Q38 NF	Q37 NF	Q36 NF	Q35 NF	Q34 NF	Q33 NF	Q32 NF

Register	QUEUPPSTATNF			
Bits	**Name**	**Description**	**Reset Value**	**Access**
k	Nearly Full	(0 <= k <= 31) Queue (k) Nearly Full Status Flag.	0	RW

20.6.7 Queues 32-63 Full Status Register

The access to these status registers is read/write, except for diagnostic and test purposes, normal operation to these registers should be read only. Writing status does not actually change the status, it only writes the shadow register that contains the status.

Register Name:	QUEUPPSTATF																														
Block Base Address:	0x041C						Offset Address						+ 4n						Reset Value						0x00000000						
Register Description:	Queue status register for queues 32-63. F: '1' – flag set.																								Access:			(See below.)			
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
Q63 F	Q62 F	Q61 F	Q60 F	Q59 F	Q58 F	Q57 F	Q56 F	Q55 F	Q54 F	Q53 F	Q52 F	Q51 F	Q50 F	Q49 F	Q48 F	Q47 F	Q46 F	Q45 F	Q44 F	Q43 F	Q42 F	Q41 F	Q40 F	Q39 F	Q38 F	Q37 F	Q36 F	Q35 F	Q34 F	Q33 F	Q32 F

Register	QUEUPPSTATF			
Bits	Name	Description	Reset Value	Access
k	Nearly Full	(0 <= k <= 31) Queue (k) Nearly Full Status Flag.	0	RW

20.6.8 Interrupt 0 Status Flag Source Select Register 0 – 3

The interrupt source for each queue is selectable as the positive or negative (NOT) edge-sensitive version of any one of the E, NE, NF or F status flag bits on interrupt 0, aqm_int[0]. The selection is configurable for interrupt 0 only. Interrupt 1, aqm_int[1], is hard wired to the NE Status Flag bit.

Register Name:	INT0SRCSELREG (0 <= n <=3)																														
Block Base Address:	Reg #n 0x0420						Offset Address						+ 4n						Reset Value						0x00000000						
Register Description:	Status Flag selection for interrupt 0 source on queues 0-31.																								Access:			(See below.)			
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
Resv'd	Queue (8n + 7) Stat Src Sel			Resv'd	Queue (8n + 6) Stat Src Sel			Resv'd	Queue (8n + 5) Stat Src Sel			Resv'd	Queue (8n + 4) Stat Src Sel			Resv'd	Queue (8n + 3) Stat Src Sel			Resv'd	Queue (8n + 2) Stat Src Sel			Resv'd	Queue (8n + 1) Stat Src Sel			Spec for 0	Queue (8n) Stat Src Sel		

Register	INT0SRCSELREG (0 <= n <=3)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
4k+2 :4k	Status Source Select	(0 <= k <= 7) 'b000: Queue (8n+k) Empty going TRUE 'b001: Queue (8n+k) Nearly Empty going TRUE 'b010: Queue (8n+k) Nearly Full going TRUE 'b011: Queue (8n+k) Full going TRUE 'b100: Queue (8n+k) Empty going FALSE 'b101: Queue (8n+k) Nearly Empty going FALSE 'b110: Queue (8n+k) Nearly Full going FALSE 'b111: Queue (8n+k) Full going FALSE	000	RW

3	Clear Interrupt	For INT0SRCSELREG0 (only), bit 3 is a configuration for the interrupt operation. If bit 3 is a 0, its reset value, an interrupt bit clears when a 1 is written back to it in the QUEUEINTREG., even if the interrupting condition is still true. If bit 3 is a 1, then interrupt bits clears when a 1 is written to QUEUEINTREG *and* the interrupting condition has been satisfied. All 64 interrupts behave according to this bit.
		The impact of this bit is shown in the following pseudo-code:
		if (INT0SRCSELREG0[3] == FALSE) then QUEINT0REG bits are **set** when the selected state change happens AND the bit is enabled in the QUEIEREG register QUEINT0REG bits are **reset** when the bit is written with a 1 AQM_INT is true if any QUEINT0REG bits are true
		else
		QUEINT0REG bits are **set** when the selected state change happens QUEINT0REG bits are **reset** when the bit is written with a 1 AND the selected state is no longer true AQM_INT is true if any (QUEIEREG AND QUEINT0REG) bits are true
		endif
		But, in one corner case, if INT0SRCSELREG0[3] == TRUE and QUEINT0REG bits get set, if you then set INT0SRCSELREG0[3] == FALSE, the interrupt is generated even if the QUEIEREG bit corresponding to a QUEINT0REG bit is FALSE. This occurs because of the difference in the AQM_INT between the two cases.
			0	RW

20.6.9 Queue Interrupt Enable Register 0 – 1

The tables below describe the Queue Interrupt Enable Register 0 – 1:

Register Name:	QUEIEREG(0 <= n <=1)																													
Block Base Address:	0x0430						Offset Address						+ 4n						Reset Value						0x00000000			
Register Description:	Interrupt enables for the queues 0-63. IE: '1' – Enable.																						Access:				(See below.)			
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**
Q(32n + 31) IE	Q(32n + 30) IE	Q(32n + 29) IE	Q(32n + 28) IE	Q(32n + 27) IE	Q(32n + 26) IE	Q(32n + 25) IE	Q(32n + 24) IE	Q(32n + 23) IE	Q(32n + 22) IE	Q(32n + 21) IE	Q(32n + 20) IE	Q(32n + 19) IE	Q(32n + 18) IE	Q(32n + 17) IE	Q(32n + 16) IE	Q(32n + 15) IE	Q(32n + 14) IE	Q(32n + 13) IE	Q(32n + 12) IE	Q(32n + 11) IE	Q(32n + 10) IE	Q(32n + 9) IE	Q(32n + 8) IE	Q(32n + 7) IE	Q(32n + 6) IE	Q(32n + 5) IE	Q(32n + 4) IE	Q(32n + 3) IE	Q(32n + 2) IE	Q(32n + 1) IE	Q(32n) IE

Register	QUEIEREG(0 <= n <=1)			
Bits	**Name**	**Description**	**Reset Value**	**Access**
k	Interrupt Enable	(0 <= k <= 31) Queue (32n+k) Interrupt Enable.	0	RW

20.6.10 Queue Interrupt Register 0 – 1

Two interrupt registers correspond to the two AQM interrupts, aqm_int[0] and aqm_int[1]. Queue Interrupt register 0 represents queues 0-31, and register 1 represents queues 32-63. Following an interrupt, the appropriate register is read to determine the queues that caused the interrupt. To clear any bit in the interrupt register, write a 1 to the appropriate bit position.

Writing a 1 to a bit in the Interrupt register provides a reset-only operation for that bit. Clearing all set bits (by writing 1's in those locations) in the Interrupt Register removes the interrupt (de-assert). The interrupt cannot be generated again by the same source, until the active status flag condition is removed and then reasserted again. The INT0SRCSELREG0 bit in the INT0SRCSELREG0 register changes the operation of the interrupt from a rising-edge sensitive operation to a level sensitive operation.

Register Name:	QUEINTREG(0 <= n <=1)				
Block Base Address:	0x0438	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Interrupt Register for the 64 queues. INT: '1' – interrupt occurred.			Access:	(See below.)

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
Q(32n + 31) INT	Q(32n + 30) INT	Q(32n + 29) INT	Q(32n + 28) INT	Q(32n + 27) INT	Q(32n + 26) INT	Q(32n + 25) INT	Q(32n + 24) INT	Q(32n + 23) INT	Q(32n + 22) INT	Q(32n + 21) INT	Q(32n + 20) INT	Q(32n + 19) INT	Q(32n + 18) INT	Q(32n + 17) INT	Q(32n + 16) INT	Q(32n + 15) INT	Q(32n + 14) INT	Q(32n + 13) INT	Q(32n + 12) INT	Q(32n + 11) INT	Q(32n + 10) INT	Q(32n + 9) INT	Q(32n + 8) INT	Q(32n + 7) INT	Q(32n + 6) INT	Q(32n + 5) INT	Q(32n + 4) INT	Q(32n + 3) INT	Q(32n + 2) INT	Q(32n + 1) INT	Q(32n) INT

Register	QUEINTREG(0 <= n <=1)			
Bits	Name	Description	Reset Value	Access
k	Interrupt	(0 <= k <= 31) Queue (32n+k) Interrupt.	0	RW

20.6.11 Queue Configuration Words 0 - 63

The 64 queue configuration words are located in internal SRAM and require initialization before AQM usage. The read and write pointers must be cleared on initialization, because this reflects an empty queue. A system reset sets the status registers to reflect empty queues, but until the queue configuration words have been set, this state is somewhat inconsistent. On a write access to any of the queue configuration words 0-31, the corresponding queue status is communicated on the flag bus. When the AQM is fully configured, use these configuration words for read-only purposes, to monitor queue pointers. Resetting a queue requires two operations. First the appropriate queue status flags is configured, then the queue configuration word is set. The data format for the queue configuration word is shown in the QUECONFIG register.

If the queue configuration word has not been initialized, reading or writing to a queue produces undefined operations, and can cause spurious parity errors and/or data corruption of another queue.

Register Name:	QUECONFIG (0 <= n <=63)				
Block Base Address:	Queue #n 0x2000	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Queue #n configuration word located in SRAM.			Access:	(See below.)

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
Queue (n) NF Watermark			Queue (n) NE Watermark			Q (n) Buffer Size		Q (n) Entry Size		Queue (n) Base Address								Queue (n) Read Pointer							Queue (n) Write Pointer						

Register	QUECONFIG (0 <= n <=63)			
Bits	Name	Description	Reset Value	Access
31:29	Nearly Full Watermark	These bits configure the Nearly Full Watermark, to 0 (“000”), 1 (“001”), 2 (“010”), 04(“011”), 8 (“100”), 16 (“101”), 32 (“110”), or 64 (“111”) entries from the top of the queue. The usable range of Nearly Full Watermark selection is less than the buffer size.	0	RW
28:26	Nearly Empty Watermark	These bits configure the Nearly Empty Watermark, to 0 (“000”), 1 (“001”), 2 (“010”), 04(“011”), 8 (“100”), 16 (“101”), 32 (“110”), or 64 (“111”) entries from the bottom of the queue. The usable range of Nearly Empty Watermark selections is less than the buffer size.	0	RW
25:24	Buffer Size	These bits configure the queue buffer size. The buffer size is configured for 16 (“00”), 32 (“01”), 64 (“10”) or 128 (“11”) words.	0	RW
23:22	Entry Size	These bits configure the queue entry size of the queue. The entry size is set at 1 (“00”) or 2 (“01”) or 4 (“10”) words. An input of “11” sets the entry size to 1.	0	RW
21:14	Base Address	This field configures the starting base address of the queue. The read and write pointer are offset addresses from the base address. This base address is a SRAM 16 word address. (In other words, this value is multiplied by 16 to get the SRAM address. Also, this value is a word address, not a byte address as the queue has no notion of bytes.) Base addresses, 00 to 03, are reserved for the queue configuration words. The most significant bit of the base address is reserved for growth to a 16 KB SRAM to provide 8 KB of additional queue buffer space. BIt 21 is implemented that is, is read and written but causes an address wrap (For example, 0x00 and 0x80 point to the same region in SRAM).	0	RW
13:7	Read Pointer	Pointer to the next entry to read from the queue. The pointer is the AQM’s internal SRAM word address. This pointer should be initialized to zero, and should be written only for diagnostic or test purposes.	0	RW
6:0	Write Pointer	Pointer to the next entry to write to the queue. The pointer is the AQM’s internal SRAM word address. This pointer should be initialized to zero, and should be written only for diagnostic or test purposes.	0	RW

20.6.12 Queue 32 to 63 Event 'A' Enable Register

This register contains a bit map that enables the chosen class of events (via the Event Source Select Register) through the Event 'A' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'A' event.

Register Name:	QUEUPPEVA				
Block Base Address:	0x0448	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Queue Event output 'A' enable register.			Access:	(See below.)

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
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register	QUEUPPEVA			
Bits	Name	Description	Reset Value	Access
k	Enable Empty 'A'	(0 <= k <= 31) Queue (32+k) Event Enable for 'A' Event.	0	RW

20.6.13 Queue 32 to 63 Event 'B' Enable Register

This register contains a bit map that enables the chosen class of events (via the Event Source Select Register) through the Event 'B' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'B' event.

Register Name:	QUEUPPEVB				
Block Base Address:	0x044C	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Queue Event output 'B' enable register.			Access:	(See below.)

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
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register	QUEUPPEVB			
Bits	Name	Description	Reset Value	Access
k	Enable Empty 'B'	(0 <= k <= 31) Queue (32+k) Event Enable for 'B' Event.	0	RW

20.6.14 Queue 32 to 63 Event 'C' Enable Register

This register contains a bit map that enables the chosen class of events (via the Event Source Select Register) through the Event 'C' funnel. The resulting event is the field OR of the selected event (Empty, Nearly empty, Nearly full, or Full) field ANDed with the enable bits. Any, none, or all of the bits can form the 'C' event.

Register Name:	QUEUPPEVC				
Block Base Address:	0x0450	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Queue Event output 'C' enable register.			Access:	(See below.)

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
Q63 EN	Q62 EN	Q61 EN	Q60 EN	Q59 EN	Q58 EN	Q57 EN	Q56 EN	Q55 EN	Q54 EN	Q53 EN	Q52 EN	Q51 EN	Q50 EN	Q49 EN	Q48 EN	Q47 EN	Q46 EN	Q45 EN	Q44 EN	Q43 EN	Q42 EN	Q41 EN	Q40 EN	Q39 EN	Q38 EN	Q37 EN	Q36 EN	Q35 EN	Q34 EN	Q33 EN	Q32 EN

Register	QUEUPPEVC			
Bits	Name	Description	Reset Value	Access
k	Enable Empty 'C'	(0 <= k <= 31) Queue (32+k) Event Enable for 'C' event.	0	RW

20.6.15 Event Source Select

For each event, this register allows one to select the class of flags that applies to an event. The chosen flags are ANDed with the event enabling to form the actual event.

Register Name:	QUEUPSOUSEL				
Block Base Address:	0x0458	Offset Address	+ 4n	Reset Value	0x000
Register Description:	Event Source Select Register.			Access:	(See below.)

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
(Reserved)																										Event C Source Select		Event B Source Select			Event A Source Select	

Register	QUEUPSOUSEL			
Bits	Name	Description	Reset Value	Access
5:4	Event Source Select C	Event type source select for event 'C' 'b00 => Empty 'b01 => Nearly Empty 'b10 => Nearly Full 'b11 => Full	'b01	RW
3:2	Source Select B	Event type source select for event 'B'	'b01	RW
1:0	Source Select A	Event type source select for event 'A'	'b01	RW

20.6.16 Queue 0 to 31 Status Selection Map Register

The tables below describe the Queue 0 to 31 Status Selection Map Register.

Register Name:	QUELOSTATMAP				
Block Base Address:	0x045C	Offset Address	+ 4n	Reset Value	Depends on AQM instantiation
Register Description:	Queue Status Selection register.			Access:	(See below.)

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
Resv'd	NPE Sel				Flag Bus Map for Queues 24-31			Resv'd	NPE Sel				Flag Bus Map for Queues 16-23			Resv'd	NPE Sel				Flag Bus Map for Queues 8-15			Resv'd	NPE Sel			Flag Bus Map for Queues 0-7			

Register	QUEUPPSTAT0			
Bits	**Name**	**Description**	**Reset Value**	**Access**
30:27	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP receives this bank at that slot, and if no bits are active, no CCP receives this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 24-31.	See Note	RW
26:24	Map for Queues 24-31	The three bits program that the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 24-31, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 26:24 are copied to aqm_flag_id[5:3].	'o3	RW
22:19	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP receives this bank at that slot, and if no bits are active, no CCP receives this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 16-23	See Note	RW
18:16	Map for Queues 16-23	The three bits program that the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 16-23, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 18:16 are copied to aqm_flag_id[5:3].	'o2	RW
14:11	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP receives this bank at that slot, and if no bits are active, no CCP receives this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_flag_strb[3:0] for queues 8-15	See Note	RW
10:8	Map for Queues 8-15	The three bits program that the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 8-15, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 10:8 are copied to aqm_flag_id[5:3].	'o1	RW
6:3	NPE Select	Each bit corresponds to the aqm_flag_strb of each CCP. If more than one bit is set, more than one CCP receives this bank at that slot, and if no bits are active, no CCP receives this bank in any slot. Bits [3:0] of this field correspond directly to the value of aqm_lag_strb[3:0] for queues 0-7.	See Note	RW
2:0	Map for Queues 0-7	The three bits program that the 8 banks of 8 queues to target in the CCP's limit of 64 queues. When addressing queues 0-7, aqm_flag_id[5:0] is composed of this three bit field and the lower three bits of the queue number. Here, bits 2:0 are copied to aqm_flag_id[5:3].	'o0	RW

20.6.17 Queue SRAM Error Data Register

The tables below describe the Queue SRAM Error Data Register:

Register Name:	QUEDATAERR				
Block Base Address:	0x0464	Offset Address	+ 4n	Reset Value	Not Applicable
Register Description:	Queue SRAM Parity Error Data Register.			Access:	(See below.)

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
Data Read																															

Register	QUEDATAERR			
Bits	Name	Description	Reset Value	Access
31:0	Data Read	The value of the data read from the SRAM when the parity error occurred	U	RO

20.6.18 Queue SRAM Error Address/Control Register

The only field to be implemented in this register is the error 'E', field. The other fields are reserved.

Register Name:	QUEADDERR				
Block Base Address:	0x0460	Offset Address	+ 4n	Reset Value	0x00000000
Register Description:	Queue SRAM Parity Error Address and Control Register.			Access:	(See below.)

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
(Reserved)															E	(Reserved)															

Register	QUEADDERR			
Bits	Name	Description	Reset Value	Access
31:17	(Reserved)		0	
16	Error	When a parity error occurs, the AQM asserts its parity output, capture the operation that caused the parity error and set this bit. Writing this bit to a zero via software clears the condition. The value of the address, parity and data fields are stable so long as this bit remains set. This bit is connected to the aqm_parity_error port.	0	RW
15:0	(Reserved)		0	

20.7 Error/Abnormal Conditions

Parity errors are abnormal once the queues have been initialized and are fatal. Parity errors may occur because of uninitialized operation, or soft errors. Errors due to uninitialized operation are due to an incorrect software initialization flow. Soft errors are an unavoidable consequence of cosmic rays and background radiation and are extremely infrequent, but a single event upset is fatal to the temporary operation of the AQM and is not recoverable in the general case.

§ §

21.0 Error Handling

This section explains the types of system level error conditions that may occur in the Intel® IXP43X Product Line of Network Processors. It also describes the effect of each of these errors.

21.1 Errors

21.1.1 Sources of AHB Bus Errors

The AHB Bus errors occur during the following situations:

- Attempt to access memory locations that are unimplemented or reserved
- Attempt to access units with types of accesses the units are not designed to accept such as INCR 16 in many network processors of the Intel® IXP43X Product Line
- Data Corruption in the target, indicated by parity errors or uncorrectable ECC errors in memory.

21.1.1.1 Accesses to Reserved or Unimplemented Addresses

Accesses to reserved or unimplemented memory results in an AHB error response being asserted by the bus arbiter.

21.1.1.2 Illegal-Access Types

Any unit that receives an access type that it cannot respond properly to, responds with an AHB error response. See individual unit descriptions to see what types of transactions each unit is capable of responding to.

21.1.1.3 AQM Parity Error

When the AQM receives a parity error from its internal memory, it returns AHB error to the requesting master. In addition, it sends an interrupt signal to the interrupt controller. As a result, the operating system assumes the entire queue is corrupt, and it should reset the unit, clearing all data entries in the queue.

21.1.1.4 Memory Controller Unit (MCU), Multiple-Bit, ECC Error

When the MCU unit receives a multiple-bit ECC error, it returns AHB error to the requesting AHB master. In addition, it sends an interrupt signal to the interrupt controller, and log the address of the access that failed.

21.2 Responses to Errors

Most memories internal or external to the IXP43X network processors have some variety of data corruption detection capability. The Expansion Bus, AQM, and MCU (also known as DDR SDRAM Controller) sends interrupt signals directly to the interrupt controller indicating data corruption detected by those units. The NPEs halts as a result of internal data corruption, and in turn send interrupt signals.

The interrupt unit has a special mechanism that allows software to set these sorts of interruptions as highest priority interruptions.

21.2.1 PCI Responses to Errors

The PCI Unit interprets AHB Error as AHB retry. As a result, it retries the errored transaction. In the event that the error was caused by a one-time transient condition, this retry receives good data the second time, and continue operating normally. In the event that this is a continuing system issue, the PCI unit does not receive a valid data response, and continues this retry until the condition is repaired or the PCI unit is reset.

In the event of such an error, although the PCI responds as if a retry had occurred, the unit that asserted AHB error sends an interrupt to the Interrupt Controller. This allows the Intel XScale® Processor to know that an error condition exists, and attempt to solve in whatever manner determined appropriate for the given system.

External PCI errors is not reflected back on the AHB bus.

21.2.2 NPE Responses to Errors

Three types of NPE soft errors are:

- NPE IMEM parity error
- NPE DMEM parity error
- NPE coprocessor errors

Multiple concurrent error handling and recovery is not supported in the IXP43X network processors. The only type of NPE Coprocessor error in the IXP43X network processors is the AHB error.

AHB error is valid on all the NPEs. The causes of AHB error include:

- AHB slave access error
- Read/Write to an invalid AHB address access
- An illegal transfer size access on the AHB Bus.

The AHB slave access error is caused by one of three sources:

- an uncorrectable ECC error on DDR read access
- parity error on a Queue Manager read Access
- an error during a PCI unit access

Refer to the PCI unit specification for the cause of errors during a PCI unit access.

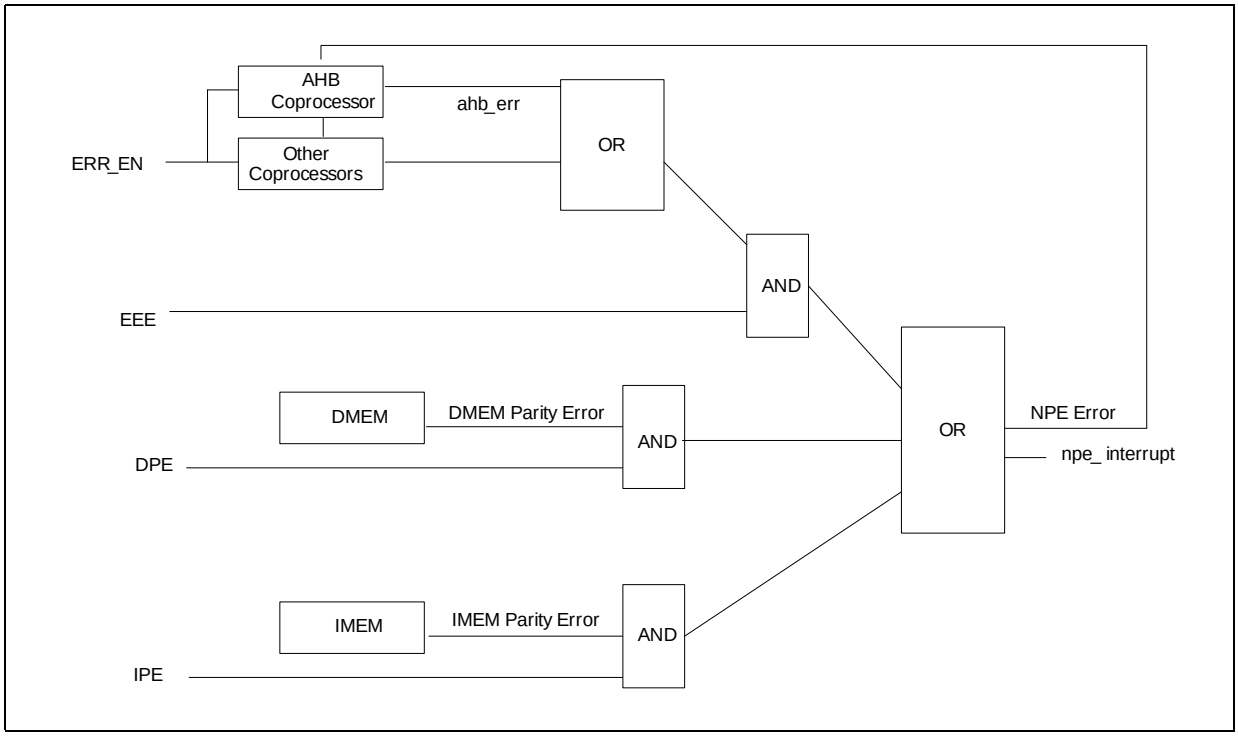
Table 257 displays the types of coprocessor errors that are generated to each NPE contained within the IXP43X network processors.

Table 257. NPE Coprocessor Error

NPE	Defined Coprocessor Errors
NPE A	AHB error
NPE C	AHB error

Figure 163 shows the various types of errors that generate interrupt to the Intel XScale processor and how the errors are communicated back to the coprocessors. Each of these errors, NPE IMEM parity error, NPE DMEM parity error and NPE Coprocessor error are individually masked through control bits as described below. The response of the coprocessors on an occurrence of an error are controlled by ERR_EN control bit as described below.

Figure 163. NPE Error Handling Illustration



EEE - Control bit to enable stop and interrupt on NPE Coprocessor error. This bit defaults to zero on reset. It is bit 18 of the NPE Core Configuration Bus Control Register.

DPE - Control bit to enable the NPE Core to stop and interrupt on NPE DMEM parity error. It defaults to zero on reset. It is bit 19 of the NPE Core Configuration Bus Control Register.

IPE - Control bit to enable the NPE Core to stop and interrupt on NPE IMEM parity error. It defaults to zero on reset. It is bit 20 of the NPE Core Configuration Bus Control Register.

ERR_EN - Control bit for Coprocessor error response enable. This is Bits 14:12 of configuration register 1 defined in Chapter 12.0, "Expansion Bus Controller." It is a software global control bit per NPE and NOT per coprocessor. These control bits are

independent of the first three control bits described above in that it dictates the response of each coprocessor as shown in Table 258 on page 754. These bits default to zero on reset.

The following scenario describes a typical error handling and recovery mechanism. In all error handling scenarios, all three control bits (IPE, DPE, and EEE) is enabled.

- If there is IMEM, DMEM, or NPE coprocessor error and the corresponding control bit is enabled, then the NPE core stops execution. The NPE also asserts an interrupt to the Intel XScale processor. In addition, the NPE core indicates to the coprocessors that an error happened (NPE error). The error indication to the coprocessors is deasserted only on resetting the NPE core.

  The software reads the Configuration bus status register to determine if the error that occurred is caused by an NPE IMEM parity error or a NPE DMEM parity error. If both the NPE IMEM parity error and NPE DMEM parity error bits are not set, then the interrupt is a result of an AHB error. The causes of AHB error include:

  - AHB slave access error
  - Read/write to an invalid AHB address access
  - An illegal transfer size access on the AHB Bus

  The AHB slave access error is caused by an uncorrectable ECC error on DDR read access, a parity error on a Queue Manager read Access, or an error during a PCI unit access. Refer to the PCI unit specification for the cause of errors during a PCI unit access. The software infers an AHB slave access error if there is an interrupt from the AHB slave in addition to the NPE interrupt. Else, the software infers an invalid AHB address access or illegal transfer size access on the AHB.

- Each of the coprocessors behaves uniquely on occurrence of the error. The behavior of the coprocessors is shown in Table 258.

Table 258. NPE Coprocessor Response (Sheet 1 of 2)

NPE Coprocessor	NPE Error	ERR_EN	NPE Coprocessor Response
AHB Coprocessor			
	Asserted	1	The AHB coprocessor locks-up and does not accept new transfers.
	Asserted	0	Error happened but the coprocessor continues Normal operation as if no error happened
	Deasserted	0	No error, continues Normal operation
Ethernet Coprocessor			
	Asserted	1	The transmit enable signal (ecp_tx_en) is forced to inactive state effectively ending all transmission immediately
	Asserted	0	Error happened but the coprocessor continues normal operation as if no error happened.
	Deasserted	0	No error, continues normal operation
UTOPIA Coprocessor			
	Asserted	1	The transmit Enable signal (UTP_OP_FCO) is forced to inactive state effectively ending all transmission immediately
	Asserted	0	Error happened but the coprocessor continues normal operation as if no error happened

Table 258. NPE Coprocessor Response (Sheet 2 of 2)

NPE Coprocessor	NPE Error	ERR_EN	NPE Coprocessor Response
	Deasserted	0	No error, continues normal operation
Other Coprocessors			
	X	X	For a parity error, continue operation without locking up.

- Software resets the errored NPE by writing to Bits 13:11 of the EXP_UNIT_FUSE_REG register defined in the Expansion Bus chapter. Note that resetting the NPE resets the core and all the coprocessors associated with that NPE.
  - The Expansion bus unit hardware handshakes with the NPE hardware to indicate that the AHB Bus transactions to be terminated cleanly (and the AHB Bus itself doesn't hang).
  - The NPE hardware indicates to the Expansion Bus unit hardware that is it ready to be reset.
  - The NPE then goes into reset state.
  - The coprocessors on the NPE are also reset. The state of the coprocessors and the state of the pins are described below:

Table 259. NPE Reset State (Sheet 1 of 2)

Coprocessor	State
AES	Reset state
AHB	Reset state
Condition	Reset state
DES	Reset state
Ethernet	The Ethernet Coprocessor is in reset state. The transmit enable and transmit data pin drives '0'. The transmit sync and the transmit clock pin toggles. The PHY may not recognize the fact that the Ethernet coprocessor is being soft-reset. The assumption is that there is an Upper Layer Protocol recovery in this scenario. MDIO under NPE C is not be reset even though NPE C and/or it's associated Ethernet coprocessor is being reset. The reason is if NPE C fuse out and left with NPE A, then MDIO on NPE C can still be used to set configurations for NPE A
FIFO	Reset state
Hash	Reset state
HSS	The HSS Coprocessor is in reset state. The HSS pins is tri-stated. So the framer potentially loses sync with the IXP43X network processors. The recommendation is that the HSS framer is also reset at the same time the NPE is soft reset. This can be accomplished by utilizing the GPIO pins on the IXP43X network processors or through other system mechanisms. The assumption made here is that there is an Upper Layer Protocol recovery in this scenario
NPE Core	The NPE Core is in reset state and the program counter is initialized to location 0.

Table 259. NPE Reset State (Sheet 2 of 2)

Coprocessor	State
NPE Data Memory (DMEM)	The DMEM is initialized to restart the NPE.
NPE Instruction Memory (IMEM)	The NPE code is downloaded to the IMEM to restart the NPE.
UTOPIA Coprocessor	The UTOPIA Coprocessor is in reset state. The UTOPIA pins is tri-stated. The recommendation is that the UTOPIA PHY is also reset at the same time the NPE is soft reset. This could be accomplished by utilizing the GPIO pins on the IXP43X network processors or through other system mechanisms. The assumption made here is that there is an Upper Layer Protocol recovery in this scenario. If it is not done, then the recommendation is that we have pull-ups or pull-downs on the board for the transmit enable pin (UTP_OP_FCO) and the Address pins (UTP_OP_ADDR). Multiple PHYs may potentially drive some pins if the pull-ups or pull-downs are not used. Refer to the *Intel® IXP43X Product Line of Network Processors Datasheet* and *Intel® IXP43X Product Line of Network Processors Hardware Design Guidelines* for details on the pull-up/pull-downs.

- The reset to the NPE is then deasserted by writing to bits 13:11 of the EXP_UNIT_FUSE_REG register.
- The instruction firmware for the NPE is downloaded.
- The data firmware for the NPE is downloaded.
- The NPE is restarted to resume normal operation by writing to the NPE core Execution Control Register.
- The system then deasserts reset to the HSS framer/UTOPIA device, if applicable.

21.2.3 Intel XScale® Processor Response to Errors

When the Intel XScale processor receives an error on the AHB bus or on its private MCU interface, it transitions to its fault handler. A second such error irretrievably loses the program counter, causing unpredictable system results. This is the inherent nature of the ARM* architecture and is not altered by the IXP43X network processors.

All other errors reported by the system is seen by the host processor via interrupt signals to the interrupt controller. As a result of such information, the host processor may attempt to clean up the system on its own, rather than reboot.

If the Intel XScale processor receives an indication that an NPE has locked up due to errors in internal memories or AHB errors to the AHB coprocessor, the IXP43X network processors may attempt to reset the afflicted NPE, reload its software, and restart. To ensure that the north AHB bus is not rendered unusable, a handshake signal between the AHB coprocessor and the reset mechanism in the expansion bus controller is implemented to ensure that the AHB coprocessor is idle before allowing reset to propagate to that unit.

Note: An important functional note is that parity errors are possible as a result of reads to the DMEM or IMEM from the APB. In this event, the only indication of an error condition to the source of the request is an interrupt to the Intel XScale processor. Due to this, error handling is only supported for accesses initiated by the Intel XScale processor. Other masters, such as the PCI controller or expansion bus controller, are not supported.

21.2.4 AHB-AHB Bridge Response to Errors

When the AHB-AHB bridge receives an AHB error on the south AHB, it responds with an AHB error on the north AHB to the originating master.

21.3 Multiple Error Conditions

It is possible that multiple error conditions might occur within the response period of the Intel XScale processor. For example, parity errors in the IMEM or DMEM may result not only in an indication of their condition, but the AHB coprocessor errors as a result of the NPE beginning its shutdown process. Similarly, an error in the Queue Manager may result not only in an indication of error to the Interrupt Controller, but a resulting error returned to the NPE that is accessing the Queue Manager may result in an error condition in the NPE.

No attempt to validate multiple error scenarios is implemented as a function in hardware. It is the responsibility of the system software to determine an appropriate response to a multiple error scenario.

§ §