Intel® 80331 I/O Processor

Developer's Manual

October 2003

Document Number: 273942-002

Contents

Figures

Tables

Revision History

Date	Revision	Description
October 2003	002	Revised MCU Chapter.
September 2003	001	Initial Release.

This Page Intentionally Left Blank

Introduction 1

1.1 About This Document

This document is the authoritative and definitive reference for the external architecture of the Intel® 80331 I/O processor (80331).

Intel Corporation assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice. In particular, descriptions of features, timings, packaging, and pin-outs does not imply a commitment to implement them. In fact, this specification does not imply a commitment by Intel to design, manufacture, or sell the product described herein.

1.1.1 How To Read This Document

This document describes the product-specific features of the 80331. Each chapter describes a different feature and starts with an overview followed by the theory of operation.

The reader should have a working understanding of the *PCI Local Bus Specification*, Revision 2.3.

1.1.2 Other Relevant Documents

1. *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual* (Order Number: 273411), Intel Corporation
2. *Intel® XScale™ Microarchitecture Programmer's Reference Manual* (Order Number: 273436), Intel Corporation
3. *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E.
4. *Intel® 80321 I/O Processor Developer's Manual* (Order Number: 273517), Intel Corporation
5. *PCI Local Bus Specification*, Revision 2.3 - PCI Special Interest Group
6. *PCI-to-PCI Bridge Architecture Specification*, Revision 1.1 - PCI Special Interest Group
7. *PCI Bus Power Management Interface Specification*, Revision 1.1 - PCI Special Interest Group
8. *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a - PCI Special Interest Group
9. *IEEE Standard Test Access Port and Boundary Scan Architecture 1149.1a* - IEEE

1.2 About the Intel® 80331 I/O Processor

The 80331 is a multi- function device that integrates the Intel® XScale™ core (ARM* architecture compliant) with intelligent peripherals and PCI-to-PCI Bridge. The 80331 consolidates, into a single system:

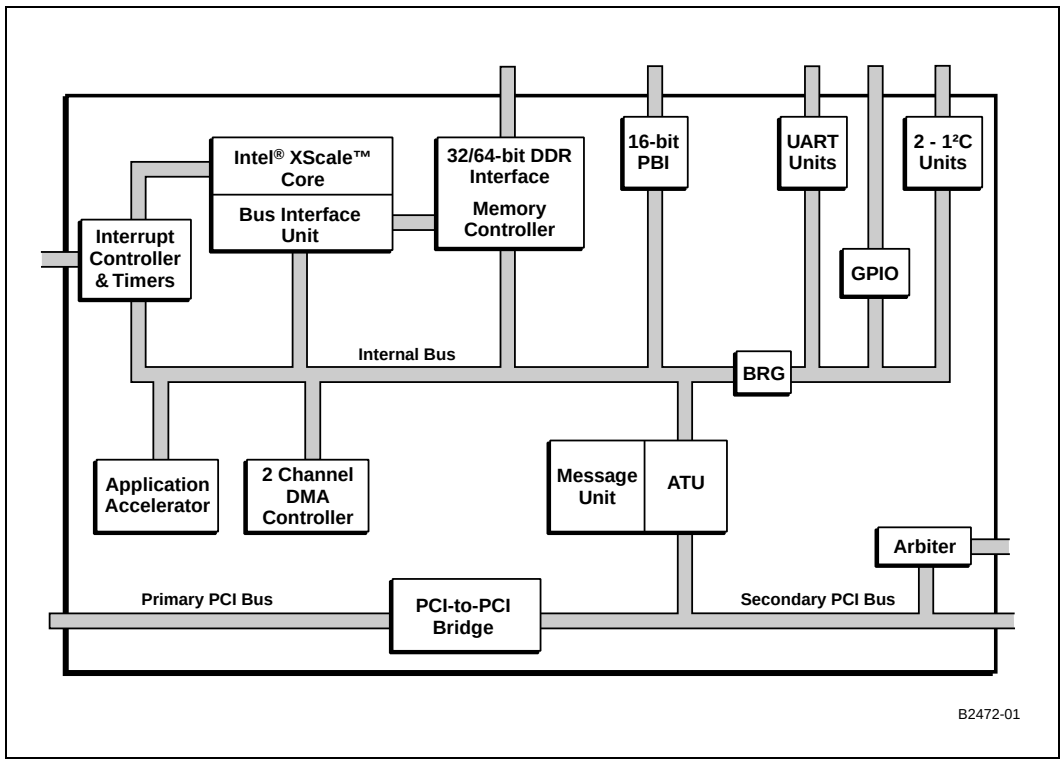
- Intel® XScale™ core.
- PCI-to-PCI Bridge supporting PCI-X interfaces on the Primary and Secondary bus.
- Address Translation Unit (PCI-to-Internal Bus Application Bridge) interfaced to the Secondary Bus.
- High-Performance Memory Controller.
- Interrupt Controller with 13 external interrupt inputs.
- Two Direct Memory Access (DMA) Controller.
- Application Accelerator.
- Messaging Unit.
- Peripheral Bus Interface Unit.
- Performance Monitor.
- Two I^2C Bus Interface Units.
- Two 16550 compatible UARTs with flow control (4 pins).
- Eight General Purpose Input Output (GPIO) ports.

It is an integrated processor that addresses the needs of intelligent I/O applications and helps reduce intelligent I/O system costs.

The 80331 integrates a PCI-to-PCI Bridge with the ATU as an integrated secondary PCI device. The Primary Address Translation Unit is compliant with the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a definitions of an 'application bridge'.

Figure 1 is a block diagram of the 80331.

Figure 1. Intel® 80331 I/O Processor Functional Block Diagram



1.3 Features

The 80331 combines the Intel® XScale™ core with powerful new features to create an intelligent I/O processor. This multi-device I/O Processor is fully compliant with the *PCI Local Bus Specification*, Revision 2.3 and the *PCI-to-PCI Bridge Architecture Specification*, Revision 1.1. 80331-specific features include:

- Intel® XScale™ Core
- Primary PCI-X 133MHz Bus Interface
- Application Accelerator Unit
- Address Translation Units
- Memory Controller
- Peripheral Bus Interface
- Two I^2C Bus Interface Units
- Two DMA Controllers
- Messaging Unit
- Internal Bus
- Two UARTs
- Interrupt Controller and GPIOs
- PCI-to-PCI Bridge to secondary PCI-X 133MHz Bus interface

The subsections that follow briefly overview each feature. Refer to the appropriate chapter for full technical descriptions.

1.3.1 Intel® XScale™ Core

The 80331 core processor is based upon the Intel® XScale™ core. The core processor operates at a maximum frequency of 800 MHz. The instruction cache is 32 Kbytes in size and is 32-way set associative. Also, the core processor includes a data cache that is 32 Kbytes and is 32-way set associative and a mini data cache that is 2 Kbytes and is 2-way set associative.

1.3.2 PCI-to-PCI Bridge Unit

80331 provides a PCI-to-PCI Bridge unit. The bridge Primary and Secondary PCI-X support 64-bit 133MHz interfaces compliant to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

1.3.3 Address Translation Units

An Address Translation Unit (ATU) allows PCI transactions direct access to the 80331 local memory. The Address Translation Unit supports transactions between PCI address space and 80331 address space. Address translation for the ATU is controlled through programmable registers accessible from both the PCI interface and the Intel® XScale™ core. The PCI interface of the ATU is connected to the 80331 Secondary PCI interface of the bridge. Upstream access to the Primary PCI interface is controlled by inverse decode with the address windows of the bridge. Dual access to registers allows flexibility in mapping the two address spaces. The ATU also supports the power management extended capability configuration header that as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

1.3.4 Memory Controller

The Memory Controller allows direct control of a DDR SDRAM memory subsystem. It features programmable chip selects and support for error correction codes (ECC). The memory controller can be configured for DDR SDRAM at 333 MHz and DDR-II at 400 MHz. The memory controller is dual-ported, with a dedicated interface for the Intel® XScale™ core Bus Interface Unit and a second interface to the Internal Bus. The memory controller supports pipelined access and arbitration control to maximize performance. The memory controller interface configuration support includes Unbuffered DIMMs, Registered DIMMs, and discrete DDR SDRAM devices.

External memory can be configured as host addressable memory or private 80331 memory utilizing the Address Translation Unit and Bridge.

1.3.5 Application Accelerator Unit

The Application Accelerator Unit (AA) provides low-latency, high-throughput data transfer capability between the AA unit, the 80331 local memory and the Primary PCI bus. It executes data transfers from and to the 80331 local memory, from the Primary PCI bus to the 80331 local memory, or from the 80331 local memory to the Primary PCI bus. The AA unit performs XOR operations, computes parity, generates and verifies an eight byte Data Integrity field which includes a 16-/32- bit Data Guard, performs memory block fills, and provides the necessary programming interface.

1.3.6 Peripheral Bus Interface

The Peripheral Bus Interface Unit is a data communication path to the flash memory components or other peripherals of a 80331 hardware system. The PBI includes support for either 8-/16-bit devices. To perform these tasks at high bandwidth, the bus features a burst transfer capability which allows successive 8-/16-bit data transfers.

1.3.7 DMA Controller

The DMA Controller allows low-latency, high-throughput data transfers between PCI bus agents and the local memory. Two separate DMA channels accommodate data transfers to the PCI bus. Both channels include a local memory to local memory transfer mode. The DMA Controller supports chaining and unaligned data transfers. It is programmable through the Intel® XScale™ core only.

1.3.8 I²C Bus Interface Unit

The I²C (Inter-Integrated Circuit) Bus Interface Unit allows the Intel® XScale™ core to serve as a master and slave device residing on the I²C bus. The I²C unit uses a serial bus developed by Philips Semiconductor consisting of a two-pin interface. The bus allows the 80331 to interface to other I²C peripherals and microcontrollers for system management functions. It requires a minimum of hardware for an economical system to relay status and reliability information on the I/O subsystem to an external device. Also refer to I²C *Peripherals for Microcontrollers* (Philips Semiconductor).

The 80331 includes two I²C bus interface units.

1.3.9 Messaging Unit

The Messaging Unit (MU) provides data transfer between the 80331 and PCI system. It uses interrupts to notify each system when new data arrives. The MU has four messaging mechanisms: Message Registers, Doorbell Registers, Circular Queues and Index Registers. Each allows a host processor or external PCI device and the 80331 to communicate through message passing and interrupt generation.

1.3.10 Internal Bus

The Internal Bus is a high-speed interconnect between internal units and Intel® XScale™ core processor. The Internal Bus operates at 266 MHz and is 64 bits wide.

1.3.11 UART Unit

The 80331 includes two UART units. The UART Unit allows the Intel® XScale™ core to serve as a master and slave device residing on the UART bus. The UART unit uses a serial bus consisting of a two-pin interface. The bus allows the 80331 to interface to other peripherals and microcontrollers. Also refer to *16550 Device Specification* (National Semiconductor*).

1.3.12 Interrupt Controller Unit

The Interrupt Controller Unit (ICU) aggregates interrupt sources both external and internal of 80331 to the Intel® XScale™ core processor. The ICU supports high performance interrupt processing with direct interrupt service routine vector generation on a per source basis. Each source has programmability for masking, core processor interrupt input, and priority.

1.3.13 GPIO

The 80331 includes 8 General Purpose I/O (GPIO) pins.

1.4 Terminology and Conventions

1.4.1 Representing Numbers

All numbers in this document can be assumed to be Base10 unless designated otherwise. In text, numbers in Base16 are represented as "nnnH", where the "H" signifies hexadecimal. In pseudo code descriptions, hexadecimal numbers are represented in the form 0x1234ABCD. Binary numbers are not explicitly identified but are assumed when bit operations or bit ranges are used.

1.4.2 Fields

A *reserved* field is a read-only field that may be used by a future implementation. Software should not modify or depend on any values in reserved fields.

A *preserved* field is a read-write field that may be used by a future implementation. Software should not modify or depend on any values in preserved fields.

A *read/write* field can written to a new value following initialization. This field can always be read to return the current value.

A *read only* field can be read to return the current value. Writes to *read only* fields are treated as no-op operations and does not change the current value nor result in an error condition.

A *read/clear* field can also be read to return the current value. A write to a *read/clear* field with the data value of 0 causes no change to the field. A write to a *read/clear* field with a data value of 1 causes the field to be cleared (reset to the value of 0). For example, when a *read/clear* field has a value of F0H, and a data value of 55H is written, the resultant field is A0H.

A *read/set* field can also be read to return the current value. A write to a *read/set* field with the data value of 0 causes no change to the field. A write to a *read/set* field with a data value of 1 causes the field to be set (set to the value of 1). For example, when a *read/set* field has a value of F0H, and a data value of 55H is written, the resultant field is F5H.

A *writeonce/readonly* field can be written to a new value **once** following initialization. After the this write has occurred, the *writeonce/readonly* field treats all subsequent writes as no-op operations and does not change the current value or result in an error condition. The field can always be read to return the current value.

1.4.3 Specifying Bit and Signal Values

The terms *set* and *clear* in this specification refer to bit values in register and data structures. When a bit is set, its value is 1; when the bit is clear, its value is 0. Likewise, *setting* a bit means giving it a value of 1 and *clearing* a bit means giving it a value of 0.

The terms *assert* and *deassert* refer to the logically active or inactive value of a signal or bit, respectively.

1.4.4 Signal Name Conventions

All signal names use the signal name convention of using the "#" symbol at the end of a signal name to indicate that the signal's active state occurs when it is at a low voltage. The absence of the "#" symbol indicates that the signal's active state occurs when it is at a high voltage.

1.4.5 Terminology

To aid the discussion of the 80331 architecture, the following terminology is used:

Table 1. Terminology

Term	Description
Core processor	Intel® XScale™ core within the 80331.
Downstream	At or toward the Secondary PCI interface from the Primary PCI interface.
DWORD	32-bit data quantity
Host processor	Processor located upstream from the 80331.
Inbound	At or toward the Internal Bus of the 80331 from the PCI interface of the ATU.
Local bus	80331 Internal Bus.
Local memory	Memory subsystem on the Intel® XScale™ core DDR SDRAM or Peripheral Bus Interface busses.
Local processor	Intel® XScale™ core within the 80331.
Outbound	At or toward the PCI interface of the 80331 ATU from the Internal Bus.
QWORD	64-bit data quantity.
Short	16-bit quantity.
Upstream	At or toward the Primary PCI interface from the Secondary PCI interface.
word	16-bit quantity.

PCI-to-PCI Bridge 2

2.1 Introduction

2.1.1 Product Overview

The Intel® 80331 I/O processor (80331) PCI-to-PCI bridge (PCIB) functions as a highly concurrent, low latency transparent bridge between two PCI busses. The bridge is capable of operating as a PCI-to-PCI bridge in the following configurations:

Table 2. PCI-to-PCI Bridge Configurations

Primary Bus Interface	Secondary Bus Interface
PCI Local Bus Specification, Revision 2.3	*PCI Local Bus Specification*, Revision 2.3
PCI Local Bus Specification, Revision 2.3	*PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a
PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0a	*PCI Local Bus Specification*, Revision 2.3
PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0a	*PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a

The bridge is used on motherboards as a means to provide additional I/O expansion slots. It is also used on PCI add-in cards as a means of mitigating the restrictive electrical loading constraints imposed on an expansion slot, enabling multiple Conventional PCI or multiple PCI-X devices to reside on a single PCI I/O adapter.

80331 supports any combination of 32- and 64-bit data transfers on its primary and secondary bus interfaces. The bridge is 33/66 MHz capable in Conventional PCI mode, and can run at 66 MHz, 100 MHz, or 133 MHz when operating in PCI-X mode depending upon its surrounding environment.

2.1.2 Features List

Table 3. Features List

- PCI Bus Interfaces (2)
  - — *PCI Local Bus Specification*, Revision 2.3 compliant
  - — *PCI-to-PCI Bridge Architecture Specification*, Revision 1.1 compliant
  - — *PCI Bus Power Management Interface Specification*, Revision 1.1 compliant
  - — *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a compliant
  - — Message Signal Interrupt (MSI) Support64-bit Initiator/Target Capable
  - — 64-bit addressing
- Secondary Bus Arbitration
  - — Secondary Bus Arbiter Supports six agents in addition to bridge
  - — Optimized for PCI-X mode
  - — Bus parking on bridge or last master
- Improved Buffer Architecture
  - — 8 K Bytes Data Buffers in each direction
  - — Improved level of Concurrency
    Up to nine outstanding transactions on each bus simultaneously
- Scalability/ Flexibility
  - — PCI-Rev 2.3 32/64-bit 33/66 MHz, 3.3 V
  - — PCI-X 32/64-bit 66/100/133 MHz, 3.3 V

2.1.3 Related External Specifications

- *PCI Local Bus Specification*, Revision 2.3
- *PCI-to-PCI Bridge Architecture Specification*, Revision 1.1
- *PCI Bus Power Management Interface Specification*, Revision 1.1
- *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a

2.2 Bus Operation

This section details the operation of the Bus interfaces. Topics described in this section include:

- RESET
- Pre-Boot Component Initialization
- Clock Domains
- Bus Transactions
- Data Flow
- PCI operation[a]
- PCI-X operation

a. *PCI Local Bus Specification*, Revision 2.3

2.2.1 RESET

The bridge receives a primary bus reset input, **P_RST#**, and generates a secondary reset output, S_RST#. The bridge also supports a software controlled secondary bus reset mechanism defined by the PCI standard Secondary Bus Reset bit (Bridge Control Register, offset 3Eh).

The bridge internal state is completely initialized to its default state whenever **P_RST#** is asserted. The bridge primary PCI bus data and control signals are tri-stated as long as **P_RST#** is asserted.

The bridge secondary bus reset output, S_RST#, is asserted and remains asserted whenever any of the following conditions are true:

- **P_RST#** is asserted.
- A configuration write sets the Secondary Reset bit to a 1b.

Assertion of S_RST# by setting the Bridge Control Register Secondary Reset Bit (BCR.6) does not cause all of bridge internal state logic to be reset. However, all of bridge data queues, request queues, and associated control logic are re-initialized to their default state. The primary bus interface and all configuration space registers are not affected by setting this bit.

Note: A second configuration write is required to clear the Secondary Reset bit.

Table 4 outlines bridge reset mechanisms along with corresponding impact of each on bridge. S_RST# is asserted for both reset mechanisms, but factors governing de-assertion of S_RST# and whether bridge experiences an internal state reset, (and to what extent), varies between mechanisms.

Table 4. Reset Behavior Summary

Reset Mechanism	Full Component Reset?	Deassertion of S_RST#
P_RST#	Yes	Following **P_RST#** deassertion.
Secondary Reset Bit set to 1b	No. Reset the secondary bus interface, data/request queues and associated state logic only.	On clearing of Secondary Reset bit via configuration write cycle to the bridge control register bit(6)

Note: A bridge transition from the D3 hot state to the D0 device PM state could be viewed as an internal reset however, since when in D3 hot the main power rails have never been switched off, there is NO internal re-initialization of any kind performed, and S_RST# is not asserted to the secondary bus.

2.2.2 Pre-Boot Component Initialization

Through the use of pin strappings several of bridge features can be configured prior to host software initialization of the bridge. Key bridge secondary bus attributes that are configured prior to host involvement include:

- Maximum Allowable Secondary Bus Operating Frequency
- Private Device
- Private Memory Address Space

2.2.3 Pin Strap Configuration

The bridge provides a number of inputs that must be strapped either high or low to configure the above features prior to host initialization of the bridge. All strapping options are sampled on the low-to-high, trailing edge of **P_RST#**.

2.2.3.1 Secondary Bus Maximum Allowable Frequency

The bridge supports secondary bus operation in either Conventional PCI or PCI-X modes. As per the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a the secondary bus must be configured for mode as well as for operating frequency when coming out of S_RST#. The bridge supports the strapping of a pin (S_133EN) as a means of enabling designer input of additional information relating to the system; information necessary in addition to reported device capabilities in order to choose the correct operating speed for the bus.

Refer to Section 2.2.4.2, "Secondary Bus Mode and Frequency Initialization" on page 47 for additional detail.

2.2.3.2 Bridge Disable

The bridge can be disabled to configure 80331 as a single PCI interface I/O processor. When **BRG_EN** is pulled low, the bridge is disabled, the primary PCI interface pins are pulled high, and the secondary PCI interface operates as the primary PCI interface for 80331 operating in this 'no bridge' mode. With the bridge disabled, the internal PCI bus arbiter can be enabled or disabled through the **ARB_EN** reset strap.

2.2.3.3 Arbiter and Central Resource

When operating with the bridge disabled (**BRG_EN** low), the **ARB_EN** reset strap can be pulled low to disable the integrated arbiter and central resource function. By default, **ARB_EN** is high, enabling the integrated arbiter and central resource function for embedded applications. When **ARB_EN** is pulled low, and the secondary PCI interface can be used as a primary PCI interface in an adapter card solution, or embedded solution with an external arbiter.

2.2.4 Bus Mode and Frequency Initialization

Both of bridge bus interfaces are capable of operating at a variety of frequencies, and in either Conventional PCI mode, or in PCI-X mode. The buses mode and frequency are established when coming out of their corresponding bus segment reset sequences. The resultant mode and frequency is dependent upon the device capabilities reported as well as any system specific loading information.

2.2.4.1 Primary Bus Mode and Frequency Initialization

The bridge reports its primary bus operating capabilities to the primary bus segments originating device (typically the host bridge). The bridge indicates to the primary bus segments originating device that its primary interface is PCI-X capable at frequencies of up to 133 MHz inclusive. It also indicates that bridge is capable of running at 66 MHz when operating in Conventional PCI mode.

2.2.4.2 Secondary Bus Mode and Frequency Initialization

The bridge is the originating device for its secondary bus, and as such sets the bus mode and frequency when exiting out of the secondary bus reset sequence. The two key components that factor into the resultant secondary bus mode and frequency are the PCI-X standard sampling of downstream device capabilities, and the system specific physical bus loading characteristics for which the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a does not provide any standard means of reporting.

Downstream device capabilities are indicated by the values of **S_M66EN**, and **S_PCIXCAP** during S_RST# assertion.

Table 5. Device Mode/Frequency Capability Reporting

M66EN	PCIXCAP	Conventional PCI Device Frequency Capability	PCI-X Device Frequency Capability
Ground	Ground	33 MHz	Not capable
Not connected	Ground	66 MHz	Not capable
Ground	Pull-down	33 MHz	PCI-X 66 MHz
Not connected	Pull-down	66 MHz	PCI-X 66 MHz
Ground	Not connected	33 MHz	PCI-X 133 MHz
Not connected	Not connected	66 MHz	PCI-X 133 MHz

Note: Knowledge of the device capabilities alone is insufficient information to robustly select the bus frequency. In order to be sure of what the bus operating frequency should be set to, knowledge of the bus layout (e.g., number of slots), is necessary.

When, for example, a 133 MHz PCI-X capable adapter was the sole occupant of a two slot segment, then it would be necessary to slow the bus to 100 MHz, even though the card reported it could operate at 133 MHz due to the additional electrical loading imposed by the two slot board and connector layout.

The bridge provides a strapping approach for reporting system specific secondary bus loading information that is used in determining the maximum operating frequency of the secondary bus. The bridge considers this strap along with the device capabilities reported during S_RST# to determine the secondary bus's mode and frequency when emerging from S_RST#.

This strap, entitled Secondary PCI-X Bus 133 MHz Enable, is sampled on S_133EN, indicating to bridge what to set the bus frequency to, given any of the possible device capabilities that it samples. The value of this field is determined by the system designer, after having assessed the characteristics of the secondary bus system/adapter implementation.

Table 6 details the secondary bus frequency initialization as a function of the Secondary Bus S_133EN reset strap, and the sampled secondary device capabilities when operating in PCI-X mode.

Table 6. Secondary Bus Frequency Initialization

M133EN[a]	M66EN	PCIXCAP	PCI Bus Mode	PCI Frequency
N/A	Ground	Ground	PCI Conventional	33 MHz
N/A	Not Connected	Ground	PCI Conventional	66 MHz
N/A	N/A	Pull-down	PCI-X	66 MHz
Ground	N/A	Not Connected	PCI-X	100 MHz
Not Connected	N/A	Not Connected	PCI-X	133 MHz

a. This pin is used by the motherboard designer to run the PCI-X bus at 100 MHz even when all the PCI-X cards on the bus are 133MHz capable, so as to accommodate the board routing limitation on frequency. Note that to accommodate running a PCI-X bus at 66MHz, when all cards are capable of 133 MHz, the motherboard has to drive the PCIXCAP pin to VCC/2.

Note: The Secondary Bus PCI-X 133 MHz Enable strapping feature enables implementations to force the Secondary bus of 80331 to operate at 100 MHz even with no standard provisions in the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a for reporting device capability of 100 MHz operation.

When a card is plugged into a four slot secondary bus, a **S_PCIXCAP** pull-down strapping ensures that the bus runs at no greater than 66 MHz in PCI-X mode, and grounding **S_M66EN** ensures that the bus runs at no greater than 33 MHz in PCI, regardless of the reported downstream device capabilities.

When a card is plugged into a two slot secondary bus, the S_133EN pull-down strapping ensures that the bus runs at no greater than 100 MHz in PCI-X mode regardless of the reported downstream device capabilities.

Finally, when a card is plugged into a single slot secondary (i.e., a segment that should be able to run at 133 MHz), by strapping the S_133EN to 0b (as though it were a two slot configuration), the bus operates at 100 MHz maximum[1].

Table 7 describes the bus mode and frequency initialization pattern that bridge signals on its secondary bus when coming out of S_RST#, after having evaluated the above information.

Table 7. PCI-X Initialization Pattern

DEVSEL#	STOP#	TRDY#	Mode	Clock Period (Ns)		Clock Frequency (MHz)	
				Maximum	Minimum	Minimum	Maximum
Deasserted	Deasserted	Deasserted	PCI 33	60	30	16	33
			PCI 66	30	15	33	66
Deasserted	Deasserted	Asserted	PCI-X	20	15	50	66
Deasserted	Asserted	Deasserted	PCI-X	15	10	66	100
Deasserted	Asserted	Asserted	PCI-X	10	7.5	100	133
Asserted	Deasserted	Deasserted	PCI-X	Reserved			
Asserted	Deasserted	Asserted	PCI-X				
Asserted	Asserted	Deasserted	PCI-X				
Asserted	Asserted	Asserted	PCI-X				

1. Adapters that report 133MHz PCI-X device capability with this MaxFreq setting is limited to 100MHz operation.

2.2.5 Private Devices on the Secondary Interface

Private devices are hidden from PCI configuration software but are accessible from the Address Translation Unit. Private devices are configured through the 80331 ATU. The control mechanism for enabling a private device and private memory address range for the private devices resides in the PCI-to-PCI bridge.

2.2.5.1 Private Type 0 Commands on Secondary Interface

Type 0 configuration reads and write commands can be generated by the Address Translation Unit of the 80331. These Type 0 configuration commands are required to configure private PCI devices on the Secondary bus which are in private PCI address space. These commands are initiated by the Address Translation Unit and not by Type 1 commands on the Primary bus. Any device mapped into this private address space *is not* part of the standard Secondary PCI address space and therefore is not configured by the system host processor. These devices are hidden from PCI configuration software but are accessible from the Address Translation Unit. See Chapter 3, "Address Translation Unit" for a complete description of the private PCI address space implementation.

In Type 0 commands on the Secondary interface, **S_AD[31:11]** are used to select the **IDSEL** input of the target device. In Type 1 to Type 0 conversions, **P_AD[15:11]** are decoded to assert a unique address line from **S_AD[31:16]** on the Secondary interface.

The Secondary IDSEL Select Register (SISR) can be programmed to block 10 address lines during Type 1 to Type 0 conversions from the Primary interface. Secondary address bits **S_AD[25:16]** are the address lines that can be masked by the SISR register. By setting bits 0 through 9 (corresponding to **S_AD[16] - S_AD[25]**) in the SISR, the associated address line can be forced to remain deasserted for the **P_AD[15:11]** encodings of 00000_2 - 01001_2 and therefore are free to be used as an **IDSEL** select line for private Secondary PCI devices. Table 8 shows the possible configurations of **S_AD[31:11]** for public/private Type 0 commands on the Secondary interface. For example, when SISR Bit 0 is set, **S_AD[16]** is not asserted during a Type 1 to Type 0 conversion from the Primary PCI bus. It can only be asserted by the Address Translation Unit.

When Primary interface receives a Type 1 command that intends to use one of the **S_AD** address lines reserved for private PCI devices, the bridge performs the Type 1 to Type 0 conversion but not assert the reserved **S_AD** address line. The Type 0 command is then ignored on the Secondary PCI bus.

By using the SISR register, a total of 10 **IDSEL** signals are available for private PCI devices. The remaining **S_AD** lines (**S_AD[31:26]**) are used for devices that are always public and accessible by Type 1 commands on the Primary bus. The ATU IDSEL input is internally wired to S_AD30 and therefore, S_AD30 is not to be used for external PCI device IDSEL input.

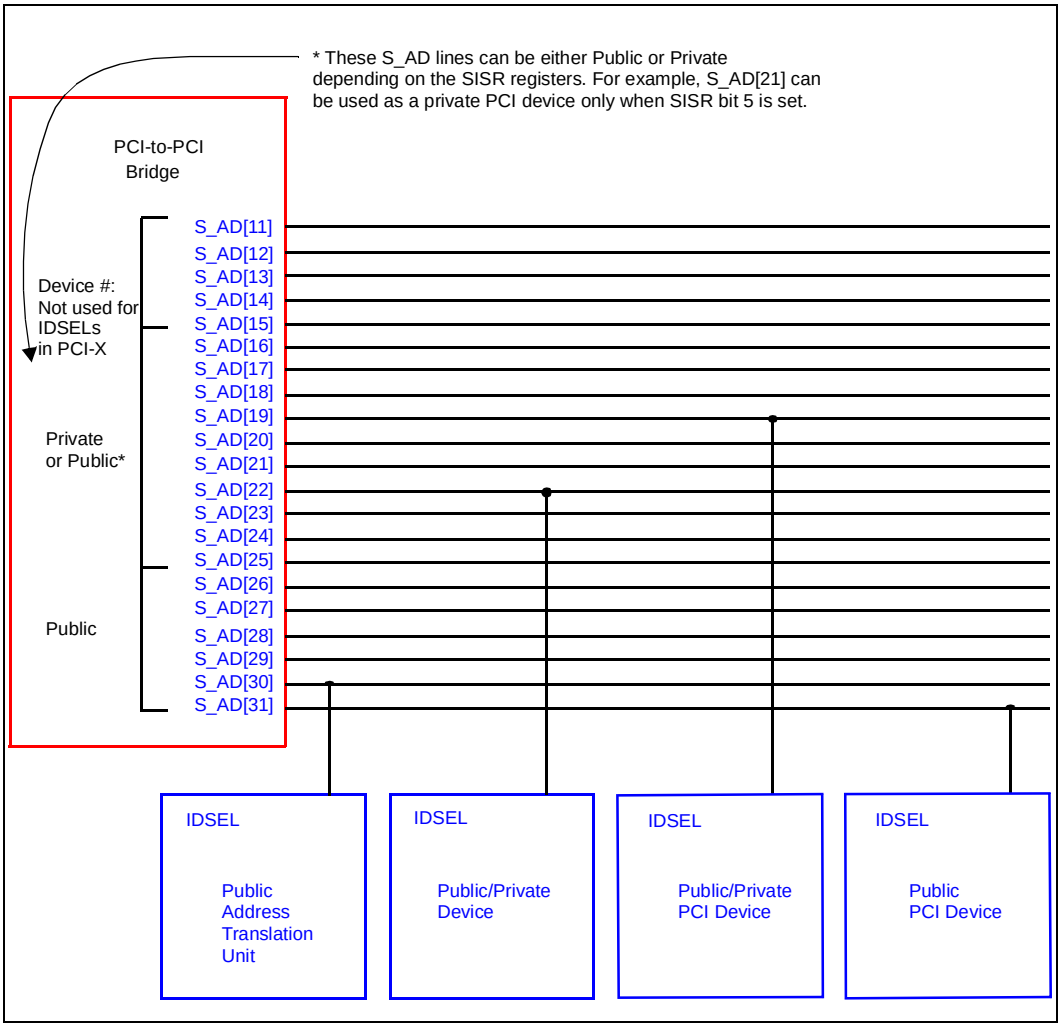
Table 8. Public/Private PCI Memory IDSEL Select Configurations

Primary Address P_AD[15:11]	Secondary Addresses S_AD[31:11] with All SISR Bits = 0	Secondary IDSEL Select Register Bits 9-0	Secondary Addresses S_AD[31:11] with SISR Bits Programmed
00000_2	0000 0000 0000 0001 0000 0_2	XXXXXXXX1$_2$	0000 0000 0000 0000 0000 0_2
00001_2	0000 0000 0000 0010 0000 0_2	XXXXXXX1X$_2$	0000 0000 0000 0000 0000 0_2
00010_2	0000 0000 0000 0100 0000 0_2	XXXXXX1XX$_2$	0000 0000 0000 0000 0000 0_2
00011_2	0000 0000 0000 1000 0000 0_2	XXXXX1XXX$_2$	0000 0000 0000 0000 0000 0_2
00100_2	0000 0000 0001 0000 0000 0_2	XXXX1XXXX$_2$	0000 0000 0000 0000 0000 0_2
00101_2	0000 0000 0010 0000 0000 0_2	XXX1XXXXX$_2$	0000 0000 0000 0000 0000 0_2
00110_2	0000 0000 0100 0000 0000 0_2	XX1XXXXXX$_2$	0000 0000 0000 0000 0000 0_2
00111_2	0000 0000 1000 0000 0000 0_2	X1XXXXXXX$_2$	0000 0000 0000 0000 0000 0_2
01000_2	0000 0001 0000 0000 0000 0_2	1XXXXXXXX$_2$	0000 0000 0000 0000 0000 0_2
01001_2	0000 0010 0000 0000 0000 0_2		0000 0000 0000 0000 0000 0_2

X = Don't Care

Figure 2 shows an example of connecting **S_AD** lines to **IDSEL** inputs of PCI devices and private PCI devices. The default value for all the SISR bits is controlled by the **PRIVDEV** reset strap, which enables private devices at reset.

Figure 2. Secondary IDSEL Example



2.2.5.2 Private Memory Space

When Private Memory Space Enable (bit 2) in the SDER (see Section 2.5.2.10, "Secondary Decode Enable Register - SDER" on page 116) is set, bridge overrides its secondary inverse decode logic and not forward upstream any secondary bus initiated DAC Memory transactions with address in the upper half of 64-bit address space (i.e. AD(63)=1b). This establishes a private memory space for secondary bus usage, independent of bridge downstream forwarding configuration. This private memory space can be used for private devices on the secondary bus addressing a private memory space of 80331 through the Address Translation Unit. The default value for the Private Memory Space Enable bit is controlled by the **PRIVMEM** reset strap.

2.2.6 Device Select Timing

Targets are required to claim transactions by asserting DEVSEL# as shown in Table 9. The bridge responds as a type A target (PCI-X), or Medium (Conventional PCI).

Table 9. DEVSEL# Timing

Decode Speed	Conventional PCI	PCI-X
1 clock after address phase(s)	Fast	Not Supported
2 clocks after address phase(s)	Medium	Decode A
3 clocks after address phase(s)	Slow	Decode B
4 clocks after address phase(s)	Subtractive	Decode C
5 clocks after address phase(s)	N/A	N/A
6 clocks after address phase(s)	N/A	Subtractive

2.2.7 64-Bit Operation

The bridge is 64-bit capable on both of its bus interfaces. Primary and secondary bus interfaces can operate in any combination of 32- or 64-bit operation at any given time.

As an initiator bridge always asserts **REQ64#** with the following exceptions:

- A Memory Read that has been reissued because the byte count has not been satisfied and whose starting address falls within four QWORDs of an ADB.
- Any DWORD transaction.
- When starting address is not QWORD aligned (operating in Conventional PCI mode only).
- When the total transaction length is less than four QWORDs.

As a target, bridge asserts **ACK64#** in response to the initiators assertion of **REQ64#** with the following exceptions:

- Upstream Memory Read (i.e., the originating bus is the secondary bus).
- Downstream MRL, MRM, or Memory Read to a non-prefetchable address space. (When the CLS is cleared to zero, or a non-supported value any read transaction is treated as non-prefetchable)
- Memory Read (MR, MRL, MRM) reconnection (because delayed read) which executed as a DWORD transaction on the destination bus.
- Any DWORD transaction.
- When the bus is configured to 32-bit mode during reset.
- When responding with a Split Response.

2.2.8 PCI Power Management Support

The bridge is compliant with *PCI Bus Power Management Interface Specification*, Revision 1.1.

Following are some bridge PCI-PM specifics:

- PMCSR[1:0] PowerState field value dictates bridge ability to respond to and initiate transactions:

  00 - (D0 state) bridge responds to, and initiates transactions normally.

  01, 10, 11 - bridge responds ONLY to type 0 configuration accesses. Does not master any transactions.
- When the PMCSR[1:0] is written to 00 from 11, an internal reset is performed transitioning bridge to the D0uninitialized state.

2.2.9 Overview of Bus Transactions

The following sub-sections details bridge behavior when operating in Conventional PCI/X modes.

For purposes of the following discussion the terms in define the buses being described. Split Completions are the only transactions that arbitrate for and start on the Destination Bus.

Table 10. Bus Terms Description

Term	Description
Origination Bus	The bus which initiates a request. Bus closest to the Requester.
Destination Bus	The bus where a request is completed. Bus closest to the Completer. All Split Completion originate from this bus in response to a request that had been made on the Origination Bus.
PMW	Posted Memory Write
NPMW	NOT (PMW)

Transaction crossing bridge assume one of the following configurations:

- PCI to PCI
- PCI-X to PCI
- PCI to PCI-X
- PCI-X to PCI-X

Operational type is dependent on the operating mode of each bus. PCI-X uses block transfers based on ADQ and ADB concepts. While Conventional PCI interfaces do not have the concept of ADQ and ADB, they observe the ADQ and ADB structures.

Each interface operates and places data into the queues and then dispatches the request to the other interface. A "valid" indication is used to determine when a request has been enqueued and is ready to be scheduled for being played out on the destination bus.

Other than for PCI Configuration space transactions the two buses are symmetrical. Each operates independently of the other bus, sending or receiving data from the queues.

2.2.9.1 PCI-to-PCI

PCI to PCI transactions use the ADQ Queue for all block transactions. Transactions can start and stop anywhere within an ADQ.

Delayed transactions are supported using the same data and request queue structures that are used when operating in PCI-X mode. PCI to PCI-X.

2.2.9.2 PCI to PCI-X

When going from PCI to PCI-X, a transaction bases the setting of valid for a given transaction on crossing at least one ADB, or upon transaction completion by the requestor on the initiating bus. Multiple ADB crossings would allow for larger transfers that have lower overhead on both buses.

Transactions originating from a Conventional PCI initiator do not contain any of the PCI-X additional attributes such as byte count, and so therefore bridge creates the attributes for the Conventional PCI master before presenting the transaction onto the PCI-X bus. Write byte count, for example, is created based on what has been received by the bridge, and conversely, read byte counts are created based on prefetchable, and non prefetchable read policies implemented by bridge.

2.2.9.3 PCI-X to PCI

Transactions to PCI can start as soon as the transaction is enqueued.

2.2.9.4 PCI-X to PCI-X

For the PCI-X to PCI-X case all transactions with the exception of immediate transactions play as unidirectional transactions. Each transaction plays until it is completed in one direction. No link is present to any transaction going in the opposite direction. Requests may result in multiple responses which each become an independent transaction.

2.2.10 Bus Interface Data Flow

Each bus interface of the bridge operates in one of four modes for a transaction in which it is involved. The following subsections describe the operation of each of the modes of operation.

- PCI as a target
- PCI as a master
- PCI-X as a target
- PCI-X as a master

2.2.10.1 Target Operation

The bridge checks each operation to determine whether or not to assert DEVSEL# thereby claiming the transaction. The bridge supports the standard set of bridge forwarding registers. When a transaction on the Origination Bus is within the standard set of forwarding ranges defined it is then forwarded to the Destination Bus.

Primary bus to Secondary bus "Base and Limit registers" define the ranges for transactions that are claimed and subsequently forwarded to the secondary bus interface. The bridge, being a transparent bridge, utilizes inverse decoding for the purpose of claiming and forwarding transactions that are to be forwarded from the secondary bus to the primary bus.

2.2.10.1.1 As the PCI Target

As a PCI target, bridge enqueues requests and responds with a Retry while tracking a delayed transaction. After a request is enqueued it is ready to be forwarded to the Destination Bus (in accordance with ordering rules). A link between initial request and data operation is used to track data returning from Destination Bus.

Transactions start from Origination Bus and play on Destination Bus. After a transaction is done on Destination Bus, operation completes when initial master comes back one more time for the response.

The bridge supports linear increment address mode only (AD[1:0]=00b), and disconnects Conventional PCI memory transactions whose least significant two address bits are not 00b after a single DWORD.

For PMW operations, data is received before transaction is enqueued. At that point the byte count is known and the transaction can start on the Destination Bus. Using standard PCI protocol, data is received by bridge without insertion of wait states. The bridge disconnects a transaction whenever Queue space is not available.

For operations with a delayed response, data is transferred on the Destination Bus and held until the transaction is successfully completed by the Originating Bus Master. Data is sent to the Originating Master then using standard PCI protocol.

Delayed read completion transactions that are disconnected by the Master while prefetch data is still available are cleared from bridge queues.

2.2.10.1.2 As the PCI-X Target

All transactions might have a Retry response when space is not available for either the data or the request itself. .

2.2.10.1.3 PCI-X Receiving Data

The bridge receives data whenever it is the target of a Split Read Completion, PMW or, in the case of an immediate read data transfer on the bus, when bridge is the Master.

2.2.10.2 Master Operation

2.2.10.2.1 As the PCI Master

As PCI Master, bridge issues requests to a target, but does not insert wait states on PCI interface.

For PMW transactions, data is sent on Destination Bus until either byte count is satisfied, MLT expires, or Target disconnects.

For Prefetch operations, data is received until the transaction disconnects. Prefetch Disconnects from target are not retried. On Originating Bus, bridge returns only amount of data transferred before the Disconnect occurred. When the Originating Bus is operating in PCI-X mode, bridge fetches to the extent defined by the original requested byte count, and creates a single split completion sequence.

All other delayed transactions Retry until they complete, or a time-out error occurs. Status from the Destination bus operation is sent back to the Origination Bus.

Before a delayed transaction requests bus, space must be available for a Split Completion entry going from Destination to Origination Bus. When transaction is a read, bridge checks for available ADQ of space in D2O (Destination to Origination) queue.

2.2.10.2.2 As the PCI-X Master

The bridge tracks requests that is has issued as a PCI-X master until the full byte count of the transaction has been completed. This is done whether the operation has an immediate response or a Split Response.

Transactions do not request the Destination Bus until a space is available in the D2O request queue. This space is required to accommodate an immediate response. Additionally, Block Read requests must have data space for at least two D2O ADQ blocks.

Master operation might be repeated multiple times for a single Origination Bus request. Each time a disconnect occurs, or partial immediate response is received from Destination Bus target, address and byte count are updated. A new request is then generated based on the updated address and byte count.

This repeats until the full byte count has been transferred.

Transmittal of PMW data, Split Requests, and Split Completion data/status is performed by the master portion of the bus interface.

2.2.11 Exclusive Access

Lock is defeatured in 80331.

2.2.12 Conventional PCI Mode

The central resource for a given PCI bus segment broadcasts that the operating mode for that bus is Conventional PCI mode following reset when any device on the bus reports that it is incapable of running in PCI-X mode. On the bridge Primary Bus, whenever the bus mode is indicated to be Conventional PCI by the upstream bridge, bridge enters Conventional PCI mode. On the Secondary Bus bridge enters Conventional PCI mode following secondary reset when it has determined that at least one agent on the secondary bus reports that it is incapable of operating in PCI-X mode.

2.2.12.1 Posted Memory Write Transactions

The bridge posts all Memory Write and Memory Write and Invalidate (MWI) transactions that are to be forwarded from one interface to the other. The bridge accepts write data into its buffers without wait states. A posted memory write transaction is terminated when any of the following occur:

- the originator ends the transaction.
- insufficient buffer space exists to buffer another ADQ.

In the second case, target disconnect is returned to the originator.

The bridge does not initiate a memory write transaction on the Destination bus until two ADQs of data have been captured during the transaction, or until the transaction is terminated by the originator. The bridge does not insert master wait states when it acts as a bus master for the playing out of posted memory writes.

The bridge ends the transaction when all of the posted data for this queue entry has been played out and accepted by the target on the Destination Bus.

When bridge receives 2^{24} consecutive target retries from the target when attempting to deliver posted write data, it discards the posted write transaction and conditionally asserts **P_SERR#** (See Section 2.4, "Error Detection and Reporting" on page 73). This retry counter may be disabled by setting the Watchdog Timer Disable bit in the Bridge Control Register 1 - BCR1. The bridge also conditionally asserts **P_SERR#** when a target abort or master abort is detected on the target bus in response to the posted write.

2.2.12.2 Fast Back-to-Back Transactions

The bridge accepts fast back-to-back transactions as a target, but does not generate them as an initiator.

2.2.12.3 Write Flow-Through

Write flow-through is supported opportunistically.

2.2.12.4 Delayed Write Transactions

The bridge executes delayed transactions when forwarding I/O and configuration writes from the Conventional PCI interface to the other bus.

When an I/O or Configuration write transaction targeting the other PCI bus is first initiated, bridge returns a target retry. The bridge latches the transaction information, including address, bus command, write data, byte enables, **PAR** and **REQ64#**, and enqueues a delayed write request (DWR) when all of the following conditions are true at that time:

- space is available in the request queue structure
- a transaction having the same address, bus command, write data, and byte enables does not already exist in the request queue structure.

The bridge requests the destination bus and, once bus ownership is obtained, initiates the corresponding delayed write transaction in accordance with PCI ordering rules.

The bridge always executes delayed write transactions as single data phase DWORD writes. Once bridge completes the destination bus delayed write transaction it adds the transaction completion status to origination side request queue status structure. Completion status returns termination mode (TRDY#, target abort, master abort) and whether PERR# assertion was detected during delayed write transaction. This phase of delayed transaction is called delayed write completion (DWC).

When originator repeats transaction using same transaction information, bridge either retries the originator indicating that the delayed write transaction has not yet completed, or target disconnects the transaction with either TRDY# (normal completion) or Target Abort (error indication).

2.2.12.4.1 Delayed Write Transaction Time-out Errors on the Destination Bus

The bridge may be configured to discard delayed write requests when 2^{24} consecutive target retries occur on the Destination Bus to the same transaction. The Watchdog Timers Disable bit in the Bridge Control Register 1 - BCR1 enables the discard.

The bridge may also be configured to assert **P_SERR#** (**SERR#** on the primary bus) when the discard timer expires. This is done by setting the Delayed Write Watchdog Timer Expired bit of the P_SERR# Assertion Control - SERR_CTL.

When the timer expires bridge sends status back to the Origination Bus interface to indicate that the timer expired. When the Originating Master again tries to write this location a target abort response is given instead of a Retry. When the target abort occurs the transaction is deleted from the request queuing structure and **SERR#** is asserted when it is enabled.

2.2.12.4.2 Delayed Write Transaction Time-out Errors on the Origination Bus

When bridge has delayed write transaction completion status for an originator and the originator does not repeat the initial transaction before the Discard Timer for that interface expires, (see Section 2.5.1.20, "Bridge Control Register - BCR" on page 98 and Section 2.5.2.3, "Bridge Control Register 1 - BCR1" on page 104), bridge discards the delayed transaction freeing up both request and completion queue space for re-allocation to a future delayed transaction. When configured to do so via the Discard Timer SERR# Enable bit of the Bridge Control Register, bridge also asserts **SERR#** on its primary bus in the event that the delayed transaction had timed out and had been discarded by bridge.

The timer starts when Completion Status is available. The Discard Timer is reset when the transaction which started the timer completes.

2.2.12.5 Read Transactions

2.2.12.5.1 Delayed Read Transactions

Delayed read transaction protocol is similar to that of delayed write transactions, with the exception that transfers longer than a DWORD may also utilize delayed read transactions, specifically delayed memory read transactions. When a read to I/O, configuration, or memory intended for the other PCI bus is first initiated, bridge returns a target retry. The bridge enqueues the transaction information, including address, bus command, byte enables for non-prefetchable reads, and **REQ64#** when all of the following conditions are true:

- space is available in the request queue structure.
- a delayed request having the same address, bus width, and bus command does not already exist in the request queue structure.

This phase of the delayed transaction is called a delayed read request (DRR).

The bridge then requests the destination bus and, once it obtains bus ownership, initiates the delayed read request (read transaction) in accordance with PCI ordering rules. When the transaction is a non-prefetchable read, then bridge requests only a single DWORD of data. When the transaction is a memory read to a pre-fetchable memory space, then the pre-fetchs a programmable amount of data. Refer to Section 2.2.12.5.4, "Prefetching" on page 60 for details.

The bridge completes the transaction on the destination bus[1], and stores response to the appropriate queues. This phase of the delayed transaction is called the delayed read completion (DRC).

- Delayed Read Transaction Time-out Errors on the Destination Bus.

  When on destination bus, bridge attempts to execute the delayed read transaction are met with 2^{24} consecutive target retries, then bridge has a configuration option that enables it to discard the delayed read request and to conditionally assert **P_SERR#**. The Watchdog Timers Disable bit is enabled/disabled in BCR1, and optional **P_SERR#** assertion. When a read transaction on the Origination Bus matches a delayed read transaction having a DRC, bridge returns the read data, data parity bit(s), and appropriate target termination in accordance with PCI ordering rules. For all prefetchable memory read transactions, bridge aliases the memory read, memory read line, and memory read multiple commands when comparing a transaction in the delayed transaction queue to one initiated on the Origination PCI bus. Therefore, regardless of exact command used, when the address matches, and both commands are any type of memory read, bridge considers it a match. When there is no match or, the ordering rules prevent returning the completion data at that point, bridge returns target retry. The target terminations returned are as follows:

Table 11. Target Termination Returns

Destination Bus Target Response to Bridge	Bridge response to Originator
TRDY#	TRDY# (with STOP# when returning last data phase)
Target Abort	Target Abort
Master Abort	• TRDY# and FFFF FFFFh (when Master Abort Mode bit = 0)· • Target Abort (when Master Abort Mode bit = 1)

- Delayed Read Transaction Time-out Errors on the Origination Bus.

  When the bridge has read completion data for an originator and originator does not repeat initial transaction before the Discard Timer interface expires (when enabled), (Section 2.5.1.20, "Bridge Control Register - BCR" on page 98 and Section 2.5.2.3, "Bridge Control Register 1 - BCR1" on page 104), bridge discards the delayed transaction. When configured to do so via the Discard Timer SERR# Enable bit of the BCR, bridge also asserts **SERR#** on its primary bus in the event that the delayed transaction had timed out and been discarded.

1. Which might execute as another delayed read transaction, ase when target is another bridge device also executing a delayed read transaction.

2.2.12.5.2 Non-prefetchable Reads

The following transactions are considered by bridge to be non-prefetchable:

- I/O transactions.
- configuration transactions.
- transactions using the Memory Read, Memory Read Line, or Memory Read Multiple commands that address the non-prefetchable memory range.
- Single DWORD Memory Read, Memory Read Line, or Memory Read Multiple transactions.

When originating a non-prefetchable read, bridge requests only a single DWORD of read data from the target. The bridge uses the same byte enables driven by the originator of the transaction[1].

When bridge returns the read data to the originator, it asserts STOP# with TRDY# to enforce a single DWORD transaction.

2.2.12.5.3 Read Flow Through

Read Flow Through can be either fine grained or large grained. For the fine grained case data is tracked on each cache line boundary using an additional tracking mechanism. Large grained tracking is done at the ADB level passing data between the interfaces when a full ADQ is available for the transaction.

Flow-through is achieved when data is transferring on both the Origination Bus and Destination Bus at the same time. The bridge stops issuing Retry when a valid DRC from the Destination Bus is indicated to the Origination Bus.

- Fine Grained Tracking.

  Fine grained tracking provides the greatest flexibility. Fine grain data tracking occurs at the cache line level allowing each interface to use either a 32- or 64-bit width for a transfer.

  Using a fine grain tracking mechanism is particularly useful for PCI-to-PCI tracking, allowing each data transfer to proceed across the bridge as soon as it becomes available and allowing wait state insertion on a data phase basis. Fine grain tracking is employed whenever forwarding data to a bus of less than or equal bandwidth.

- The insertion of wait states on each data phase is attractive because of the eight clock rule on PCI for consecutive data phases. Large grain tracking has the sum of the delays of all the data phases in an ADQ. Large grain tracking must wait longer to start transferring, and is more likely to disconnect due to eight or more clocks of accumulated delay.Large Grained Tracking.

  Large grained tracking is done at the ADQ level. Each time a transfer on the Destination Bus crosses an ADB, that ADQ may be transferred onto the Origination Bus. This is the policy that is executed whenever the transaction is a PMW or the Originating Bus is operating in PCI-X mode.

1. This is the case where FRAME# is de-asserted for the first data phase, and **REQ64#** was never asserted.

2.2.12.5.4 Prefetching

The bridge supports a staged pre-fetch mechanism that enables fine tuning of how much data to pre-fetch. The feature remembers, from past experience, whether or not the needs of the initiator were sufficiently met, or whether even more data should be pre-fetched for a given transaction.

Prefetching is performed for memory read transactions when targeting prefetchable memory. The amount of data prefetched by bridge is programmable. When the prefetch size from one of the following tables is greater than 128 bytes, bridge disconnects at the last ADB it crosses.

Prefetch management uses a multiplication factor to determine the amount of data to be prefetched. Two sets of Prefetch factors are used, one for the Primary Bus and another for the Secondary Bus. The Secondary Bus has four control bits that select when a prefetch factor is applied to operations based on the **REQ#**/GNT# pairs. The first three **REQ#**/GNT#[2:0] pairs have individual enables for the prefetch factor, while the remaining **REQ#**/GNT# pairs [5:3] are controlled by a single control bit.

Disabling the prefetch factor results in the use of a prefetch factor equal to zero during all prefetch operations.

The bridge uses the 64-bit extension signals to initiate and complete prefetchable read transactions as a master. As a target bridge asserts **ACK64#** in response to an originator's assertion of **REQ64#** for read transactions to prefetchable memory[1].

Prefetch factors are divided into two types, FirstRead and ReRead Factors. These factors are stored in the PF_POLICY register; (see Section 2.5.2.7, "Read Prefetch Policy Register - RPPR" on page 110), and are used to compute the amount of data prefetched as shown in Table 12. The "(Factor+1)" value of the same table adds 1 to either the FirstRead or ReRead factors, depending upon whether the request is the first, or a continuing sequence.

The bridge determines whether a transaction might continue by looking for the following combination of events when the Master completes a prefetched transaction on the Origination Bus.

- bridge disconnects the transaction because data is exhausted.
- The Master still has FRAME# asserted at the disconnect.

This results in bridge storing the beginning address of cacheline. Up to eight beginning addresses are stored by bridge for prefetch ReRead transactions.

Every MR, MRL, and MRM transaction is compared to these stored values. When a match is detected the byte count is set based on the ReRead value. When a match does not occur the FirstRead value determines the byte count.

Prefetchable amounts are not guaranteed; when the queue fills before the full amount is prefetched then bridge disconnects the target and only returns the amount of data buffered. Prefetching is limited to not cross a 4KB boundary and only data up to the 4KB boundary is returned to the initiator.

Table 12. Prefetch Data Length

Memory Operation	Alignment	Read Size
Read	4 * DWORD	(Factor +1) * 4 * DWORD
Read Line	cacheline	(Factor +1) * cacheline
Read Multiple	2 cachelines	(Factor +1) *2* cachelines

1. Refer to Section 2.2.7, "64-Bit Operation" on page 51.

Prefetch size, in bytes, for Read, ReadLine Read Multiple transactions are shown in Table 13 through, Table 15 respectively[1].

Table 13. Read Command Prefetch Size

Factor	Bytes
0	16
1	32
2	48
3	64
4	80
5	96
6	112
7	128

Table 14. Read Line Command Prefetch Size

Factor	Cache Line Size	
	32	64
0	32	64
1	64	128
2	96	192
3	128	256
4	160	320
5	192	384
6	224	448
7	256	512

Table 15. Read Multiple Prefetch Size

Factor	Cache Line Size	
	32	64
0	64	128
1	128	256
2	192	384
3	256	512
4	320	640
5	384	768
6	448	896
7	512	1024

- FirstRead

  The FirstRead value determines how much data is read on the FirstRead transaction for a given originator. Each type of read command (e.g., MEM_Rd MRM, MRL) prefetches the amount of data specified in the three preceding Tables.

  Requests not starting on cache line boundary are aligned to cache line at end of FirstRead transaction.

- ReRead (repeated until time-out)

  The ReRead value determines how much data is read on the ReRead transaction for a given originator. Each type of read command (e.g., MEM_Rd MRM, MRL) prefetches the amount of data specified in Table 13 through, Table 15.

1. The cache line size is a power of two. Masters should only start on a cache line boundary.

2.2.12.6 Transaction Ordering

Transaction ordering for all data traveling in the same direction across the bridge adheres to the Conventional PCI set of ordering rules. Note that all "Don't Care" (i.e., yes/no) entries are implemented as "yes" providing the maximum opportunity for making forward progress. There are no ordering requirements for transactions traveling in opposite directions.

Table 16. Conventional PCI Ordering Rules

Row Pass Column?	PMW (Column 2)	DRR (Column 3)	DWR (Column 4)	DRC (Column 5)	DWC (Column 6)
PMW (Row 1)	No	Yes	Yes	Yes	Yes
DRR (Row 2)	No	Yes	Yes	Yes	Yes
DWR (Row 3)	No	Yes	Yes	Yes	Yes
DRC (Row 4)	No	Yes	Yes	Yes	Yes
DWC (Row 5)	Yes	Yes	Yes	Yes	Yes

2.2.13 PCI-X Bus Mode

This portion of the *Intel® 80331 I/O Processor Component Specification* describes bridge behavior when operating in PCI-X mode. The bridge behavior when dealing with areas of PCI-X operation that are open to interpretation is also highlighted. Please see the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a for all details related to PCI-X operation.

Unless otherwise noted in this section, bridge follows all rules of the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

2.2.13.1 Attributes

Table 17 describes how bridge handles a few of the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a less crisply defined attribute fields.

Table 17. Bridge Implementation of Requester Attribute Fields

Attribute	Function
No Snoop (NS)	As a target, this bit is forwarded with the transaction to allow a north bridge to not snoop the transaction. The bridge takes no action on this bit.
Relaxed Ordering (RO)	This bit allows relaxed ordering of transactions, which bridge does not permit. This bit is simply forwarded by bridge, and is never initially set by bridge.
Tag	The bridge stores and forwards tag information presented to it by the original requester. The only exception to this behavior is when bridge must forward a request on behalf of a Conventional PCI requester. In this case there is not any tag (or other Sequence ID attributes) available, and bridge creates one for the sequence.

2.2.13.2 Special Notes for Burst Transactions

The *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a allows burst transactions to cross the minimum memory map boundary (in bridge case, this is 1 M) and 4 GB address boundaries. The bridge never does this as an originator or as a target. It always ends the transaction at a 1M boundary with an exception for Block Read Transactions which are handled as per the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a. As a target, it disconnects transactions on these boundaries.

The bridge never issues an immediate response as a target for a burst read command, but it must be ready with 256 bytes of data space (two ADQs) when originating one. When it does not have this space available, it does not issue the transaction and wait for the internal congestion to resolve naturally.

2.2.13.3 Split Transactions

2.2.13.3.1 Completer Attributes

Completer attributes are normally passed through bridge without modification. However, certain error or other conditions may cause bridge to modify these bits.

Additionally, whenever an immediate response is received bridge becomes the completer and must report any error conditions it encounters in handling the transaction.

Table 18. Bridge Implementation Completer Attribute Fields

Attribute	Function
Byte Count Modified (BCM)	This bit is used for diagnostic purposes. The bridge forwards this bit, but might also generate it as specified by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.
Split Completion Error (SCE)	The bridge sets this bit in accordance with the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.
Split Completion Message (SCM)	The bridge returns an SCM in the event that an error, corresponding to the current sequence has been detected. The bridge also returns an SCM whenever completing a split write transaction.

2.2.13.3.2 Requirements for Accepting Split Completions

The bridge accepts (i.e., assert DEVSEL#) Split Completions targeting resources on its other side provided the following conditions are met:

- At least two ADQ of buffer space is available (otherwise the Split Completion is retried).
- The Sequence ID matches an outstanding Split Request that had been issued by bridge**.**
- There are less than two Split Completions of the same sequence ID currently staged within bridge internal buffers[1].

2.2.13.3.3 Split Completion Messages

Split Completion messages provide a means of reporting, to the requestor, status information relating to the current completion transaction. An SCM received by bridge is passed to the other bus unaltered.

The bridge generates a Split Completion Message (SCM) when it has encountered an error with a transaction on the Destination Bus. SCMs are issued for error conditions including Master, or Target Abort encountered on the Destination Bus, as well as for error conditions relating to corrupted completion transactions. In such cases bridge forwards an SCM reporting the error condition to the original requestor.

Additionally bridge generates an SCM for split write transactions that are completed on the destination bus as an immediate transaction.

Refer to Section 2.4, "Error Detection and Reporting" on page 73 for a complete, detailed description of error handling.

1. The bridge allows up to two SCs of the same sequence ID on the bridge at any given time. When a third Split Completion of the same sequence ID targets bridge, it is Retried.

2.2.13.4 Transaction Ordering

Transaction ordering for all data traveling in the same direction across the bridge adheres to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a ordering rules for bridges. Note that all "Don't Care" choices (i.e., yes/no entries) have been chosen to align completely with the ordering policy for PCI Conventional Mode. By making the same policy decisions regarding the ordering of transactions flowing out of the bridge, the logic for controlling transaction queuing is greatly simplified. There are no ordering requirements for transactions traveling in opposite directions.

Table 19. PCI-X Ordering Rules

Row Pass Column?	Memory Write (Column 2)	Split Read Request (Column 3)	Split Write Request (Column 4)	Split Read Completion (Column 5)	Split Write Completion (Column 6)
Memory Write (Row A)	No[a]	Yes	Yes	Yes	Yes
Split Read Request (Row B)	No	Yes	Yes	Yes	Yes
Split Write Request (Row C)	No	Yes	Yes	Yes	Yes
Split Read Completion (Row D)	No[b]	Yes	Yes	Yes[c]	Yes
Split Write Completion (Row E)	Yes	Yes	Yes	Yes	Yes

a. The bridge ignores the Relaxed Ordering bit, and so therefore does not allow a PMW to pass a previously posted PMW.
b. The bridge ignores the Relaxed Ordering bit, and so therefore does not allow a Split Read Completion to pass a previously posted PMW.
c. The bridge allows Split Completions possessing different Sequence IDs to pass each other. Split Completions possessing the same Sequence ID must play out in the order that they were received. The bridge accepts two such Split Completions on chip at any given time.

2.2.13.5 Transaction Termination as a PCI-X Target

2.2.13.5.1 Retry

The bridge retries a transaction when any of the following conditions is true.

- Target queue (Request queue) is full.
- When receiving a third Split Completion of the same sequence (bridge allows a maximum of two on chip at any time).

The bridge stores no state from a transaction that it has target terminated with retry.

Retries due to buffer full conditions should be infrequent as bridge supports a deep request queue for pending operations. Additionally, a Split Queue keeps transactions that are in process, between a Split Response and the final Split Completion for that sequence.

2.2.13.5.2 Split Response

The bridge responds to all "splitable" transactions[1] with a Split Response, when not retried due to insufficient buffer availability. One exception to this is for PCI Configuration read cycles targeting bridge internal configuration registers.

2.2.13.5.3 Split Completion Errors

During the address phase bridge looks for an exact Sequence ID match between a split completion transaction on the bus with any pending completion enqueued within its corresponding Split Queue. Unless there is an exact match, a split completion transaction that would appear to be targeting an agent on the opposite side of bridge is ignored, causing the Completer to Master Abort.

When a problem with the byte count is detected within the completion attribute phase, bridge target aborts the completer.

Refer to Section 2.4, "Error Detection and Reporting" on page 73 for a complete, detailed description of error handling.

2.2.13.6 Bridge Buffer Requirements

The bridge always has at least 256 bytes (2 ADQ) in use by a PMW or reserved for incoming PMW transactions. The bridge does not implement the split completion buffer pre-allocation algorithm as proposed by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

1. All transactions except for Split Completion, Memory Write, and Memory Write Block transactions which are immediate transactions, and so cannot be converted to split transactions.

2.3 Arbitration

The bridge provides an integrated secondary bus arbiter.

The internal arbiter is able to make more intelligent arbitration decisions based on information available to it internally; information not typically available external to the bridge.

2.3.1 Arbitration Events

An arbitration event is defined as the act of evaluating all bus requests, applying arbitration policy algorithms, and assigning a new bus grant. Within a busy environment PCI arbitration events occur in a "hidden" fashion, where the next bus owner is chosen during the current bus transaction thereby minimizing bus latency in turning the bus over to its new owner.

Aside from performing arbitration events within an active environment, two other cases exist:

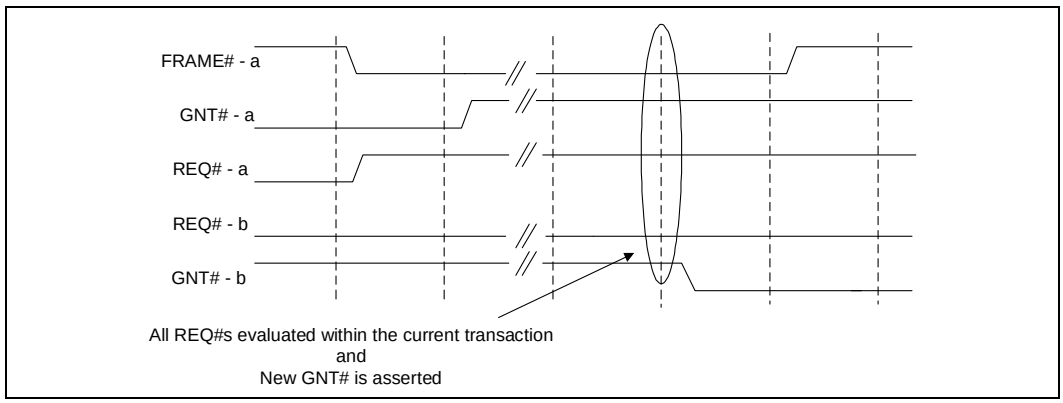
- Idle bus with no pending requests (bus parking).
- Granted agent is non-responsive (Grant Time-out situation).

2.3.1.1 Arbitrating in Traffic

The bridge secondary bus arbiter always performs arbitration events for an active secondary bus during execution of the current transaction, effectively "hiding" any additional latencies when turning the bus over to it's next owner. Independent of the secondary bus mode of operation (Conventional PCI or PCI-X), bridge always performs an arbitration event as follows:

1. De-assert the current GNT# as soon as bridge has observed that the new bus owner has begun the next bus transaction (i.e., FRAME# assertion sampled).
2. Within the current transaction bridge evaluates all outstanding bus requests and determines which agent is granted the bus next.
3. GNT# is asserted to the next bus owner while the current transaction is executing.

Figure 3. Arbitration Events



2.3.1.2 Bus Parking

When the secondary bus is idle with no pending bus requests, the bridge arbiter parks bus ownership with either the last granted agent or itself depending on the programming of the Secondary Bus Arbiter Control register.

When emerging from S_RST# the secondary bus is always internally parked at bridge.

Refer to Section 2.5.2.1, "Secondary Arbiter Control/Status Register - SACSR" on page 102 for details on the SACSR device specific register (offset 41h).

2.3.1.3 Grant Time-Out

Once a grant has been asserted on the bus, and the bus has gone idle, the granted agent is required to begin its initial transaction, indicated by the assertion of FRAME# within a maximum number of clocks. When the bridge secondary bus is operating in PCI Conventional mode the maximum permissible delay is 16 clocks. When the bridge secondary bus is operating in PCI-X mode the maximum permissible delay is 6 clocks. This requirement intends to ensure that PCI agents do not request the bus unless they are prepared to use it.

The bridge employs a Grant Time-out counter mechanism that begins counting from the time the bus goes to the idle state with a bus grant asserted. When FRAME# is not sampled in the asserted state by the granted agent before the time-out counter expires, the agent's grant is de-asserted, and a new arbitration event occurs.

Note: In an idle, bus parked situation, the Grant Time-out counter does not apply.

Agents that are slow to assert FRAME# may, when operating on a busy bus, be starved as a result of their PCI non-compliant behavior.

For diagnostic purposes, status information detailing the detection of a Grant Time-out along with information as to which secondary agent was in violation of the *PCI Local Bus Specification*, Revision 2.3, is available in the SACSR register.

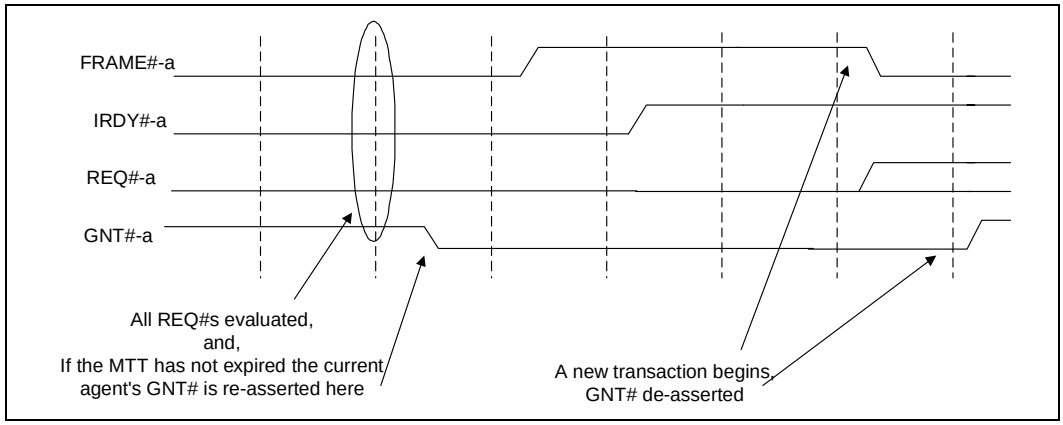
Refer to Section 2.5.2.1, "Secondary Arbiter Control/Status Register - SACSR" on page 102 for details on the SACSR device specific register (offset 41h).

2.3.2 Multi-Transaction Timer (MTT)

The bridge incorporates a multi-transaction timer that, in conjunction with the granted agent MLT and the Grant Time-out mechanism, enables the grantee to perform multiple transactions within a single arbitration cycle. Whereas an agent MLT governs the maximum duration of a single PCI transaction, the MTT residing in the secondary arbiter block, enables multiple PCI transactions to proceed by effectively establishing a minimum grant time period.

One MTT counter/value supports all secondary bus agents. Mask bits individually enable, or disable the MTT for each agent on the secondary bus.

Figure 4. Multi-Transaction Timer Operation



For MTT programming model details refer to Section 2.5.2.6, "Bridge Multi-Transaction Timer Register - BMTTR" on page 109.

2.3.2.1 MTT Rules

- The MTT counter is loaded with its programmed value each time, as a result of an arbitration event, a GNT# is asserted to an agent other than the last bus owner with an exception case for a bus parked situation. When the bus is in an idle, parked GNT# situation the MTT programmed value is re-loaded each time the park agent's GNT# is re-asserted.
- The MTT countdown begins when the granted agent's initial assertion of FRAME# is sampled by bridge on the bus.
- An agent wishing to utilize the MTT is required to continually assert its **REQ#** signal throughout the current transaction in its entirety. When at any time during the current transaction the agent de-asserts it's **REQ#** signal, MTT usage is terminated for the remainder of the current arbitration cycle.
- No greater than 4 transactions is re-granted by bridge during any single arbitration cycle, regardless of whether or not the MTT had expired.

When the following three conditions are satisfied at the next arbitration event then the bridge arbiter re-asserts the current owner's GNT# enabling a new transaction to begin from the same secondary bus agent.

- The current bus owner **REQ#** is sampled asserted, having never been de-asserted throughout the current transaction.
- The MTT count has not yet expired.
- The number of consecutive transactions already executed by this same bus agent is three or less.

For each subsequent transaction within the same arbitration cycle the grantee's MLT is re-initialized to its programmed value, while the MTT continues to count down. When during a transaction the grantee's MLT expires, the grantee should self evict, and de-assert its **REQ#** thereby ending its current ownership of the bus.

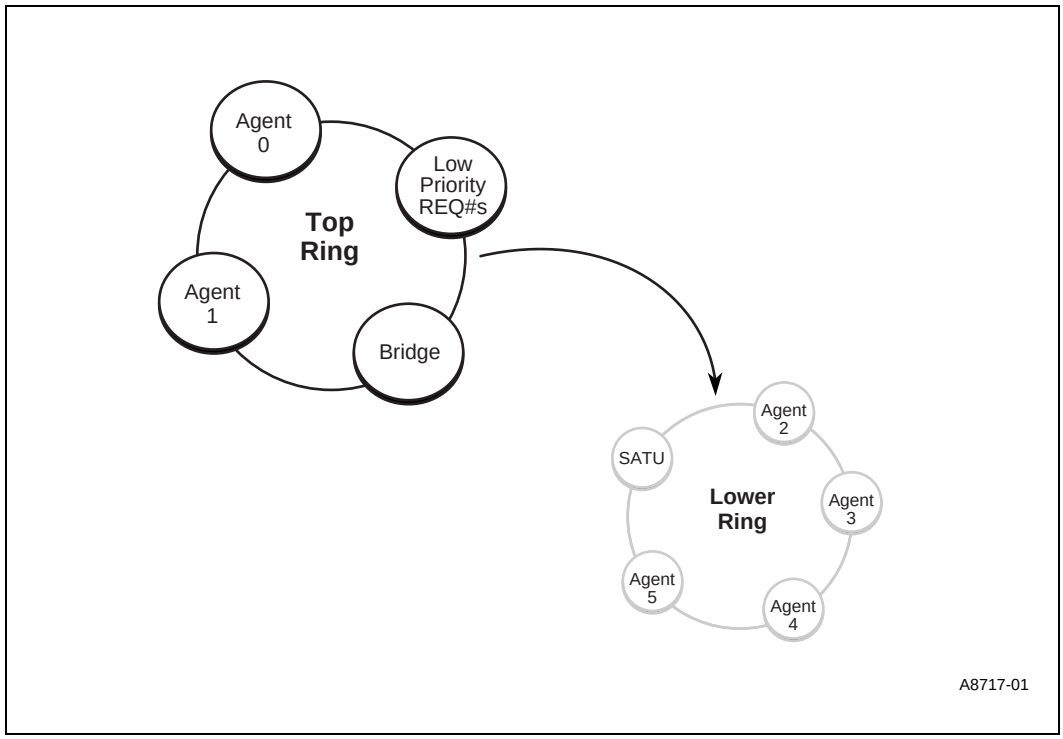
2.3.3 PCI Conventional Mode Arbitration

When operating in PCI Conventional mode the bridge conducts arbitration using a two tiered "round robin" rotating scheme (Figure 5).

When bridge owns the bus it plays out any newly enqueued delayed transaction requests, or PMW data in accordance with PCI ordering rules.

Bus agents can be programmed to reside in the high (top), or lower priority ring of arbitration. Refer to the SACSR register (offset 41h).

Figure 5. Two-Tiered Round Robin Rotating Scheme



Refer to Section 2.5.2.1, "Secondary Arbiter Control/Status Register - SACSR" on page 102 for detailed programming interface information.

2.3.4 PCI-X Arbitration

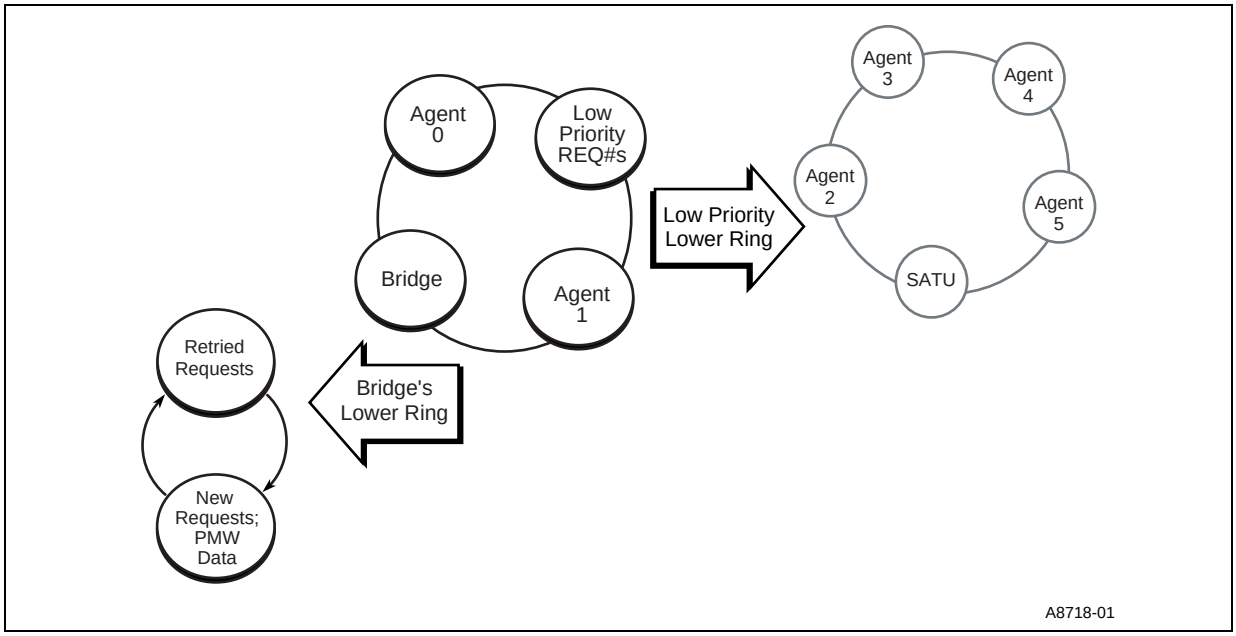
The bridge secondary arbitration scheme for PCI-X mode is nearly identical to that of its PCI Conventional arbitration scheme. The only real difference is that when the secondary bus is running in PCI-X mode bridge requests the bus for playing out split completions, as well as for playing out new split requests and PMW immediate data.

2.3.4.1 Fair Internal Arbitration

Split Completions, new internal requests, and posted Memory Write data are rotated fairly when determining the next transaction to be played out of bridge on its next bus ownership. Due to the effect of the MLT, bridge interleaves other valid split completions in an attempt to promote more uniform forward progress for all agents through the bridge.

Figure 6 below illustrates the PCI-X arbitration scheme.

Figure 6. PCI-X Priority Rings



2.3.4.2 Retry or Disconnected Request

When bridge mastered transaction receives either a Target Retry or other Target Disconnect response, bridge internal transaction flow arbiter[1] ensures that this particular internal request is masked until all other currently outstanding requests have been honored. In a sense this could be viewed as a third level lower arbitration ring for bridge that is underneath the "New Internal Requests" ring.

Request masking is only done for internal requests that have been target terminated. The bridge does not block grants to external agents based on Retry or a Disconnect. External agents may have multiple functions with request streams active that are not related to the request that was retried. External agents simply wait for their next slot in the normal arbitration queue.

1. Arbiter/control logic that determines which of the valid internal transactions plays out next. Not to be confused with the PCI bus arbiter.

2.4 Error Detection and Reporting

Error detection and reporting are separate items within bridge. Error detection is a continuous on going operation. Every detected error is logged in the status bits for reporting errors at all times.

When error reporting is enabled for an interface; PER set in the Command and Bridge Control registers for the primary and secondary busses respectively allows bridge to respond to error detection. Further, errors can be reported to the system by the assertion of **P_SERR#**, which is enable by the SERR# Enable bit in the Command register.

During normal operation all transactions are passed from one interface to the other without modification, unless they require special action by bridge. All phases of a transaction are passed from one bus to the other without modification including address and attributes.

2.4.1 Normal Operation

Address, attributes and data are accepted on the Origination Bus and passed without modification for most types of transactions. Parity for each phase of a transaction is also passed though the bridge without modification. Parity is generated by bridge for the following types of transactions:

- Type 1 configuration to Special cycle conversion.
- Type 1 configuration to Type 0 conversion.
- The bridge generated Split Completions.

Detection of errors always results in the error detection state being set. This is true for all types of errors detected by bridge.

2.4.2 Response Enabled

Each bus has a separate PER control bit to enable bridge responding to errors. When the PER bit is set bridge takes action on each transaction depending on the type of error. Types of errors on which action is taken are:

Table 20. Detected Error Types and Action Taken

Error Type	Action Taken
address or attribute parity	For all transactions in error it sets the Detected Parity Error bit in the associated buses Status register. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.
	Split Completion: The bridge target aborts (when requestor ID matches) or master abort. There is no SCM error message generated.
	Other transaction: Generates a SCM response to all transactions needing a Split Completion response. Discards transaction and associated data.
data parity	Sets Detected Parity Error bit in the Status register. Asserts x_PERR# on the associated bus.
	PCI-X transaction: Store and forward all data including parity bit.
	Delayed transaction: Discard transaction and purge data from buffers.
	Type 0 Configuration Write Transaction (Conventional Mode or PCI-X Mode) Bridge configuration register is written with data.
x_PERR# assertion by external agent	Set Master Data Parity Error bit in the Status register for the associated bus
Master Abort	Set Received Master Abort bit in the associated Status register. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.
	PCI and read: Send FFFF FFFFh as read data on the Origination bus.
	PCI-X: Generate a SCM message to all transactions that received a split response.
Target Abort	Set Received Target Abort bit in the associated Status register.
	PMW: Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.
S_SERR#	Set the Received System Error bit in the Secondary Status Register. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **P_SERR#**.
Discard Timer	Set the Discard Timer bit in the associated Status register. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.
Retry Timer	After 2^{24} retries on the destination bus discard the transaction. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.
Unexpected Split Completion	When a Split Completion does not match byte count, or address field is incorrect. Conditionally sets Signaled System Error bit in the Status register when it causes an assertion of **SERR#**.

2.4.3 P_SERR# Assertion

Asserted when an error capable of **SERR#** assertion occurs, the SERR# enable bit in the Command register is set, and that particular error condition has been enabled to assert **P_SERR#** in the P_ERR# Assertion Control Register. In all cases the assertion of **SERR#** results in setting of the Signaled System Error bit in the Status register.

2.5 Programming Interface

The bridge programming interface is provided only in the configuration register address space, which is accessible from the primary bus interface. The bridge registers are not accessible by the Intel® XScale core. The space is divided into four groups with the ranges indicated in Table 21.

Table 21. Configuration Register Address Space Groupings and Ranges

Register Group	Configuration Offset
Standard PCI Configuration	00-3Fh
Device Specific Registers	40-A7h
Reserved	A8-CBh
Enhanced Capability List	CC-FFh

Details of three of these four groups are given in the following subsections. The Reserved section needs no further explanation.

The following sections define bridge programming interface. In addition to the minimum standard set of PCI-to-PCI bridge features, bridge also supports several device specific features as well as other PCI standard enhanced capabilities. The PCI enhanced capabilities supported consist of:

Table 22. PCI Enhanced Capabilities Supported

Enhanced Capability	Configuration Offset
PCI Bus Power Management Interface Specification, Revision 1.1	DCh
PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0a	F0h

PCI enhanced capabilities are enumerated, and managed via the PCI standard Capabilities linked list infrastructure. System software discovers the existence of enhanced capabilities by sampling the state of the PCI Status Register (Offset 06h). Bit(4) of the PCI Status Register sampled as a 1b indicates that a capabilities linked list for at least one enhanced capability exits, beginning at the address offset contained in the Capabilities Pointer register, located at offset 34h.

Table 23, "Standard PCI Type 1 Configuration Space Address Map" on page 76, Table 44, "Bridge Device-Specific Configuration Address Map" on page 101 and Table 56, "Enhanced Capabilities Register File" on page 119provide an overview of bridge Configuration Space Registers. Detailed register definitions follow each table.

Note: Some register bits have defined behavior that differs depending upon the mode of the subject interface. (i.e., Conventional PCI or PCI-X). In such cases the behavior is explicitly spelled out for each mode. In all other cases the stated definition applies to both modes of operation.

2.5.1 Standard PCI Configuration Header Registers (Offset 00H-3FH)

Table 23. Standard PCI Type 1 Configuration Space Address Map

Byte 3	Byte 2	Byte 1	Byte 0	Configu-ration Byte Offset
Device ID		Vendor ID		00h
Primary Status		Primary Command		04h
Class Code			RevID	08h
(reserved)	Header Type	Primary MLT	Primary CLS	0Ch
Reserved				10h
Reserved				14h
Secondary MLT	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Non-prefetchable Memory Limit Address		Non-prefetchable Memory Base Address		20h
Prefetchable Memory Limit Address		Prefetchable Memory Base Address		24h
Prefetchable Memory Base Address Upper 32 Bits				28h
Prefetchable Memory Limit Address Upper 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16		30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Bridge Control		Primary Interrupt Pin	Primary Interrupt Line	3Ch

The following subsections detail the bit assignments, and corresponding definitions for each of the bridge standard PCI configuration Type 1 header registers.

Register bits of the type "RC" bits are readable, and writable only to the extent that they are cleared by writing a 1b to them. Writing 0b to a RC bit has no affect on the current state of the bit.

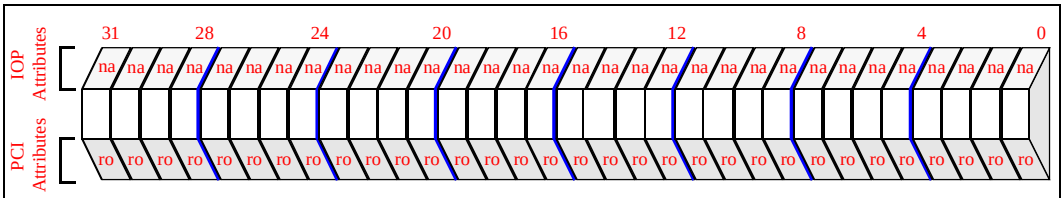
Register bits of the type "Reserved" must always return 0b when read, and writes to them must have no affect.

Default power on reset states are provided for all but those fields that are device dependent. Device specific defaults are labeled "DS".

2.5.1.1 Identifiers - ID

Contains the vendor and device identifiers.

Table 24. Identifiers - ID



Internal Bus Address: NA

PCI Configuration Address Offset: 00h

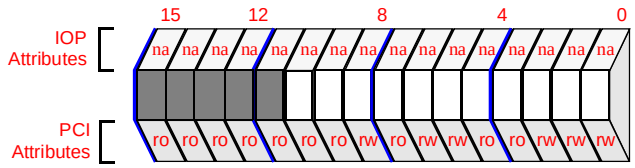
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	0335h	Device ID (DID): Indicates the unique device ID that is assigned to bridge by the PCI SIG. ID is unique per product speed as indicated.
15:00	8086h	Vendor ID (VID): 16-bit field which indicates that Intel is the vendor.

2.5.1.2 Primary Command Register - PCR

Controls how bridge behaves on its primary interface, and is the same as all other devices, with the exception of the VGA Palette Snoop bit. As this component is a bridge, additional command information is located in a separate register called "Bridge Control" located at offset 3Eh.

Table 25. Primary Command Register - PCR (Sheet 1 of 2)



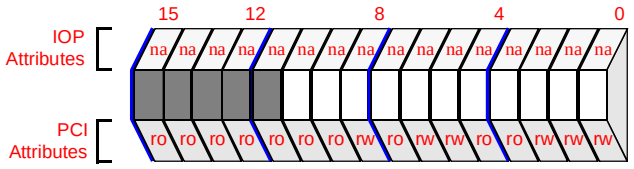
Internal Bus Address: NA

PCI Configuration Address Offset: 04h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:11	00h	Reserved
10	0b	Interrupt Disable: Disables/Enables the generation of Interrupts on the primary bus. The bridge does not support interrupts.
09	0b	FB2B Enable: Enables/Disables the generation of fast back to back transactions on the primary bus. The bridge does not generate fast back to back transactions on the primary bus.
08	0b	SERR# Enable (SEE): Enables primary bus **SERR#** assertions. 0b = The bridge does not assert **P_SERR#**. 1b = The bridge may assert **P_SERR#**, subject to other programmable criteria.
07	0b	Wait Cycle Control (WCC): Always returns 0bzero indicating that bridge does not perform address or data stepping,
06	0b	Parity Error Response (PER): Controls bridge response to a detected primary bus parity error. 0b = When a data parity error is detected bridge does not assert S_PERR#. Also bridge does not assert **P_SERR#** in response to a detected address or attribute parity error. 1b = When a data parity error is detected bridge asserts S_PERR#. The bridge also asserts **P_SERR#** (when enabled globally via bit(8) of this register) in response to a detected address or attribute parity error.
05	0b	VGA Palette Snoop Enable (VGA_PSE): Controls bridge response to VGA-compatible palette write transactions. VGA palette write transactions are I/O transactions whose address bits are: • P_AD[9:0] equal to 3C6h, 3C8h or 3C9h· • P_AD[15:10] are not decoded (i.e. aliases are claimed), or are fully decoding (i.e., must be all 0's depending upon the VGA aliasing bit in the Bridge Control Register, offset 3Eh.· • P_AD[31:16] equal to 0000h 0 = The bridge ignores VGA palette write transactions, unless decoded by the standard I/O address range window. 1 = The bridge responds to VGA palette write transactions with medium DEVSEL# timing and forwards them to the secondary bus.

Table 25. Primary Command Register - PCR (Sheet 2 of 2)



Internal Bus Address: NA

PCI Configuration Address Offset: 04h

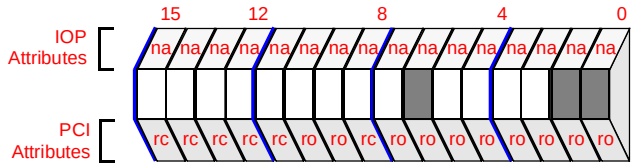
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
04	0b	Memory Write and Invalidate Enable (MWIE): The bridge does not promote MW transactions to MWI transactions. MWI transactions targeting resources on the opposite side of the bridge, however, are forwarded as MWI transactions.
03	0b	Special Cycle Enable (SCE): The bridge ignores special cycle transactions. This bit is read only and always returns 0 when read
02	0b	Bus Master Enable (BME): Enables bridge to initiate memory and I/O transactions on the primary interface. Initiation of configuration transactions is not affected by the state of this bit. 0 = The bridge does not initiate memory or I/O transactions on the primary interface. 1 = The bridge is enabled to function as an initiator on the primary interface.
01	0b	Memory Space Enable (MSE): Controls target response to memory transactions on the primary interface. 0 = The bridge target response to memory transactions on the primary interface is disabled. 1 = The bridge target response to memory transactions on the primary interface is enabled.
00	0b	I/O Space Enable (IOSE): Controls target response to I/O transactions on the primary interface. 0 = The bridge target response to I/O transactions on the primary interface is disabled. 1 = The bridge target response to I/O transactions on the primary interface is enabled.

2.5.1.3 Primary Status Register - PSR

Bits in this register are either Read Only (RO) or Read Clear (RC).

Table 26. Primary Status Register - PSR (Sheet 1 of 2)



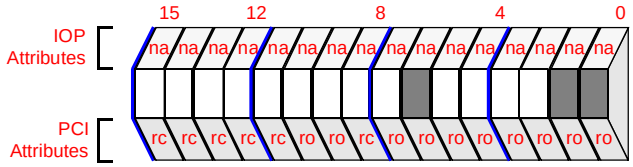
Internal Bus Address: NA

PCI Configuration Address Offset: 06h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0b	Detected Parity Error: The bridge sets this bit to a 1b whenever it detects an address, attribute or data parity error. This bit is set regardless of the state of the PER bit in the command register.
14	0b	Signaled System Error: The bridge sets this bit to a 1b whenever it asserts **SERR#** on the primary bus.
13	0b	Received Master Abort: The bridge sets this bit to a 1b when, acting as the initiator on the primary bus, its transaction (with the exception of special cycles) has been terminated with a Master Abort.
12	0b	Received Target Abort: The bridge sets this bit to a 1b when, acting as the initiator on the primary bus, its transaction has been terminated with a Target Abort.
11	0b	Signaled Target Abort: The bridge sets this bit to a 1b when it, as the target of a transaction, terminates it with a Target Abort. In PCI-X mode this bit is also set when it forwards a SCM with a target abort error code.
10:09	01b	DEVSEL# Timing: Indicates slowest response to a non-configuration command on the primary interface. Returns "01b" when read, indicating that bridge responds no slower than with medium timing.
08	0b	Master Data Parity Error: The bridge sets this bit to a 1b when all of the following conditions are true: • The bridge is the current master on the primary bus· • S_PERR# is detected asserted or is asserted by bridge· • The Parity Error Response bit is set in the Command register
07	1b	Fast Back to Back Capable: Returns a 1b when read indicating that bridge is able to respond to fast back to back transactions on its primary interface.
06	0b	Reserved
05	1b	66 MHz Capable Indication: Returns a 1b when read indicating that bridge primary interface is 66 MHz capable. 1 =

Table 26. Primary Status Register - PSR (Sheet 2 of 2)



Internal Bus Address: NA

PCI Configuration Address Offset: 06h

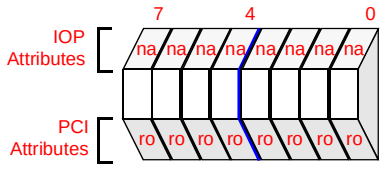
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
04	1b	Capabilities List Enable: Returns 1b when read indicating that bridge supports PCI standard enhanced capabilities. Offset 34h (Capability Pointer register) provides the offset for the first entry in the linked list of enhanced capabilities.
03	0b	Interrupt Status: Reflects the state of the interrupt in the device/function. The bridge does not support interrupts.
02:00	0h	Reserved

2.5.1.4 Revision ID Register - RID

Revision ID Register.

Table 27. Revision ID Register - RID



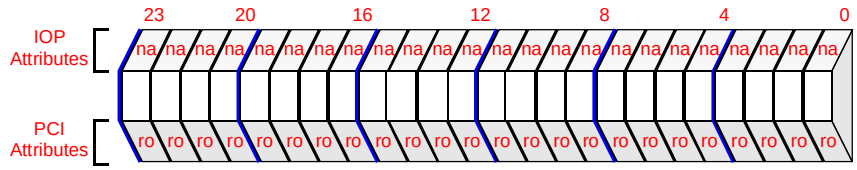
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	08h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07:00	00h	Revision ID (RID): '00h' indicating bridge A-0 stepping.

2.5.1.5 Class Code Register - CCR

This contains the class code, sub class code, and programming interface for the device.

Table 28. Class Code Register - CCR



Internal Bus Address: NA

PCI Configuration Address Offset: 09h

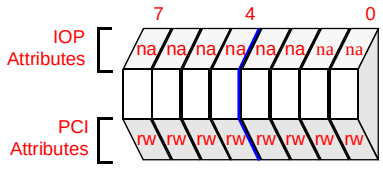
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
23:16	06h	Base Class Code (BCC): Indicates that this is a bridge device.
15:08	04h	Sub Class Code (SCC): Indicates this is of type PCI-to-PCI bridge.
07:00	00h	Programming Interface (PIF): Indicates that this is standard (non-subtractive) PCI-PCI bridge.

2.5.1.6 Cacheline Size Register - CLSR

This indicates the cache line size of the host system.

Table 29. Cacheline Size Register - CLSR



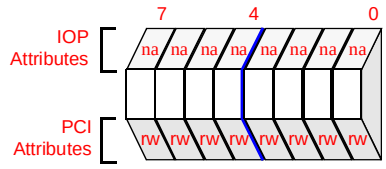
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	0Ch	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07:00	00h	Cache Line Size (CLS): Designates the cache line size in 32-bit dword units. The contents of this register are factored into internal policy decisions associated with memory read prefetching, and the promotion of Memory Write transactions to MWI transactions. Valid cache line sizes are 8 and 16 dwords. When the cache line size is set to an invalid value, bridge behaves as though the cache line size was set to 00h.

2.5.1.7 Primary Latency Timer Register - PLTR

Primary bus MLT.

Table 30. Primary Latency Timer Register - PLTR



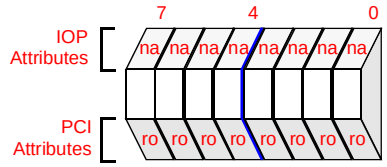
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	0Dh	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07:00	00h (Conven-tional PCI) -or- 40h (PCI-X)	Primary Latency Timer (PTV): Conventional PCI Mode: Primary bus Master latency timer. Indicates the number of PCI clock cycles, referenced from the assertion of FRAME# to the expiration of the timer, when bridge may continue as master of the current transaction. All bits are writable, resulting in a granularity of 1 PCI clock cycle. When the timer expires (i.e., equals 00h) bridge relinquishes the bus after the first data transfer when its PCI bus grant has been deasserted. PCI-X Mode: Primary bus Master latency timer. Indicates the number of PCI clock cycles, referenced from the assertion of FRAME# to the expiration of the timer, when bridge may continue as master of the current transaction. All bits are writable, resulting in a granularity of 1 PCI clock cycle. When the timer expires (i.e., equals 00h) bridge relinquishes the bus at the next ADB. (Except in the case where MLT expires within 3 data phases of an ADB. In this case bridge continues on until it reaches the next ADB before relinquishing the bus)

2.5.1.8 Header Type Register - HTR

This register indicates bridge PCI Configuration Header Type. The bridge reports that it is a single function (i.e., not a multi-function device), and that its PCI header type conforms to the PCI standard Type 1 header layout.

Table 31. Header Type Register - HTR



Internal Bus Address: NA

PCI Configuration Address Offset: 0Eh

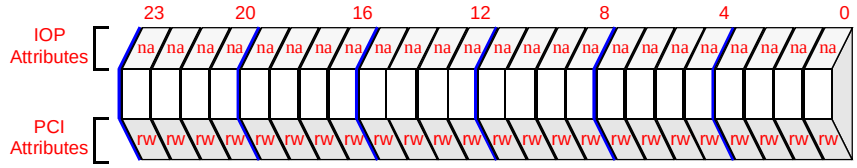
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07	0	Multi-function device (MVD): 80331 is a single-function device.
06:00	01h	Header Type (HTYPE): Defines the layout of addresses 10h through 3Fh in configuration space. Returns "01h" when read indicating that the register layout conforms to the standard PCI-to-PCI bridge layout.

2.5.1.9 Bus Number Register - BNR

This register is used to configure bridge primary, secondary, and maximum subordinate bus numbers.

Table 32. Bus Number Register - BNR



Internal Bus Address: NA

PCI Configuration Address Offset: 18h

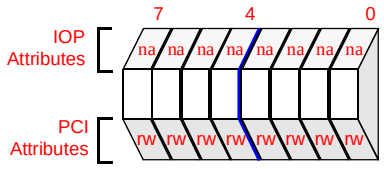
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
23:16	00h	Subordinate Bus Number (SBBN): Indicates the highest PCI bus number below this bridge. Any Type 1 configuration cycle on the primary bus whose bus number is greater than the secondary bus number, and less than or equal to the subordinate bus number is forwarded unaltered as a Type 1 configuration cycle on the secondary PCI bus.
15:08	00h	Secondary Bus Number (SCBN): Indicates the bus number of PCI to which the secondary interface is connected. Any Type 1 configuration cycle matching this bus number is translated to a Type 0 configuration cycle (or a Special Cycle) before being executed on bridge's secondary PCI bus.
07:00	00h	Primary Bus Number (PBN): Indicates bridge primary bus number. Any Type 1 configuration cycle on the primary interface with a bus number that is less than the contents of this register field does not be claimed by bridge.

2.5.1.10 Secondary Latency Timer Register - SLTR

Secondary MLT.

Table 33. Secondary Latency Timer Register - SLTR



Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	1Bh	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

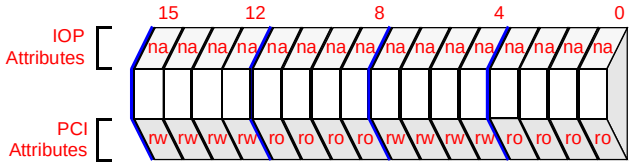
Bit	Default	Description
07:00	00h (Conventional PCI) -or- 40h (PCI-X)	Secondary Latency Timer (STV): Conventional PCI Mode: Secondary bus Master latency timer. Indicates the number of PCI clock cycles, referenced from the assertion of FRAME# to the expiration of the timer, when bridge may continue as master of the current transaction. All bits are writable, resulting in a granularity of 1 PCI clock cycle. When the timer expires (i.e., equals 00h) bridge relinquishes the bus after the first data transfer when its PCI bus grant has been deasserted. PCI-X Mode: Secondary bus Master latency timer. Indicates the number of PCI clock cycles, referenced from the assertion of FRAME# to the expiration of the timer, when bridge may continue as master of the current transaction. All bits are writable, resulting in a granularity of 1 PCI clock cycle. When the timer expires (i.e., equals 00h) bridge relinquishes the bus at the next ADB. (Except in the case where MLT expires within 3 data phases of an ADB. In this case bridge continues on until it reaches the next ADB before relinquishing the bus)

2.5.1.11 I/O Base and Limit Register - IOBL

This register defines, by way of base and limit fields, a 4K Byte aligned address range. I/O transactions originating on bridge primary bus that address an I/O location falling within the range specified in this register are subsequently forwarded to bridge secondary PCI bus[1].

I/O transactions on the primary bus that address an I/O location that is outside of the range specified by the contents of this register is not forwarded by bridge[2].

Table 34. I/O Base and Limit Register - IOBL



Internal Bus Address: NA

PCI Configuration Address Offset: 1Ch

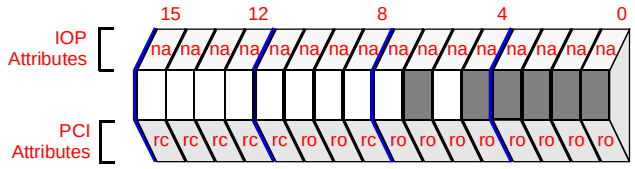
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:12	0h	I/O Limit Address Bits [15:12]: Defines the top address of an address range to determine when to forward I/O transactions from one interface to the other. These bits correspond to address lines 15:12 for 4KB alignment. Bits 11:0 are assumed to be FFFh.
11:08	1h	I/O Limit Addressing Capability: This field is hard-wired to 1h, indicating support 32-bit I/O addressing.
07:04	0h	I/O Base Address Bits [15:12]: Defines the bottom address of an address range to determine when to forward I/O transactions from one interface to the other. These bits correspond to address lines 15:12 for 4KB alignment. Bits 11:0 are assumed to be 000h.
03:00	1h	I/O Base Addressing Capability: This is hard-wired to 1h, indicating support for 32-bit I/O addressing.

1. Provided that the I/O Space Enable bit of the PCI Command register is set.
2. With the exception of legacy I/O transactions when enabled (e.g., VGA palette snoops).

2.5.1.12 Secondary Status Register - SSR

Table 35. Secondary Status Register - SSR



Internal Bus Address: NA

PCI Configuration Address Offset: 1Eh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

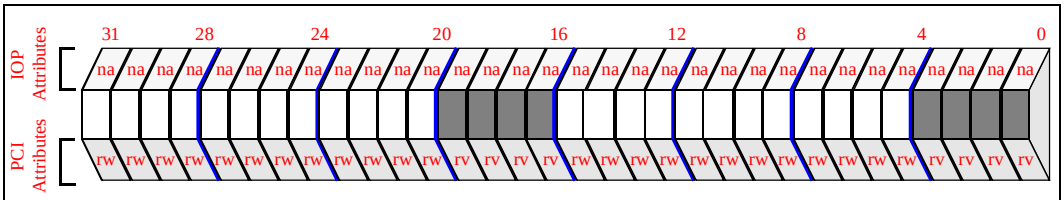
Bit	Default	Description
15	0b	Detected Parity Error: The bridge sets this bit to a 1b whenever it detects an address, attribute or data parity error on its secondary interface.
14	0b	Received System Error: The bridge sets this bit when it samples **SERR#** asserted on its secondary bus interface.
13	0b	Received Master Abort: The bridge sets this bit to a 1b when, acting as the initiator on the secondary bus, it's transaction (with the exception of special cycles) has been terminated with a Master Abort.
12	0b	Received Target Abort: The bridge sets this bit to a 1b when, acting as the initiator on the secondary bus, it's transaction has been terminated with a Target Abort.
11	0b	Signaled Target Abort: The bridge sets this bit to a 1b when it, as the target of a transaction, terminates it with a Target Abort. In PCI-X mode this bit is also set when it forwards a SCM with a target abort error code.
10:09	01b	DEVSEL# Timing: Indicates slowest response to a non-configuration command on the secondary interface. Returns "01b" when read, indicating that bridge responds no slower than with medium timing.
08	0b	Master Data Parity Error: The bridge sets this bit to a 1b when all of the following conditions are true: • The bridge is the current master on the secondary bus· • S_PERR# is detected asserted or is asserted by bridge· • The Parity Error Response bit is set in the Command register
07	1b	Fast Back-to-Back Capable (FBC): Indicates that the secondary interface of bridge can receive fast back-to-back cycles.
06	0b	Reserved
05	1b	66 MHz Capable (C66): Indicates the secondary interface of the bridge is 66 MHz capable. 1 =
04:00	00h	Reserved.

2.5.1.13 Memory Base and Limit Register - MBL

This register defines, by way of base and limit fields, a 1M Byte aligned address range.

A memory transaction, whose address is within the address range specified by this register is treated as a non-prefetchable transaction. The non-prefetchable memory address window is typically used for accessing memory mapped I/O locations where read side-affects may occur.

Table 36. Memory Base and Limit Register - MBL



Internal Bus Address: NA

PCI Configuration Address Offset: 20h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000h	Memory Limit: These 12 bits are compared with P_AD[31:20] of the incoming address to determine the upper 1MB aligned value (exclusive) of the range. The incoming address must be less than or equal to this value. For the purposes of address decoding the lower 20 address bits (P_AD[19:0] are assumed to be F FFFFh.
19:16	0h	Reserved.
15:04	000h	Memory Base: These 12 bits are compared with bits P_AD[31:20] of the incoming address to determine the lower 1MB aligned value (inclusive) of the range. The incoming address must be greater than or equal to this value. For the purposes of address decoding the lower 20 address bits (P_AD[19:0]) are assumed to be 0 0000h.
03:00	0h	Reserved.

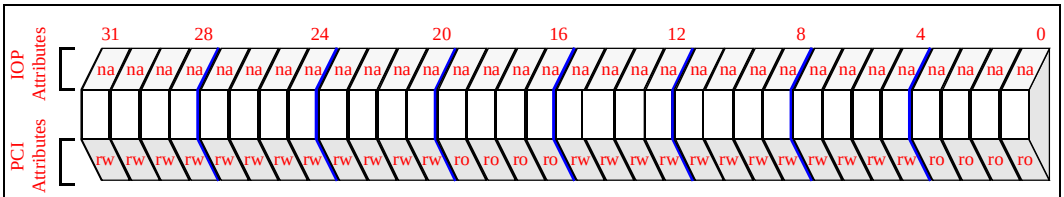
2.5.1.14 Prefetchable Memory Base and Limit Register - PMBL

This register defines, by way of base and limit fields, a 1M Byte aligned address range. Memory transactions originating on the bridge primary bus that address a memory location falling within the range specified by this register are subsequently forwarded to bridge secondary PCI bus[1].

A memory transaction, whose address is within the address range specified by this register is treated as a prefetchable transaction. The prefetchable memory address window must only be used for accessing memory locations where there are no read side-affects.

The guarantee of no read side affects enables bridge to speculatively prefetch additional read data in anticipation of its being requested shortly thereafter.

Table 37. Prefetchable Memory Base and Limit Register - PMBL



Internal Bus Address: NA

PCI Configuration Address Offset: 24h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

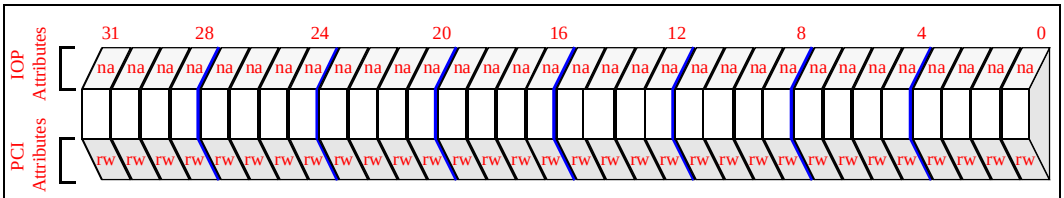
Bit	Default	Description
31:20	000h	Prefetchable Memory Limit: These 12 bits are compared with P_AD[31:20] of the incoming address to determine the upper 1MB aligned value (exclusive) of the range. The incoming address must be less than or equal to this value. For the purposes of address decoding the lower 20 address bits (P_AD[19:0] are assumed to be F FFFFh.
19:16	1h	64-bit Indicator: Indicates that 64-bit addressing is supported.
15:04	000h	Prefetchable Memory Base: These 12 bits are compared with bits P_AD[31:20] of the incoming address to determine the lower 1MB aligned value (inclusive) of the range. The incoming address must be greater than or equal to this value. For the purposes of address decoding the lower 20 address bits (P_AD[19:0]) are assumed to be 0 0000h.
03:00	1h	64-bit Indicator: Indicates that 64-bit addressing is supported.

1. .Provided that the Memory Space Enable bit of the PCI Command register is set.

2.5.1.15 Prefetchable Memory Base Upper 32 Bits - PMBU32

This defines the upper 32 bits of the prefetchable address base register.

Table 38. Prefetchable Memory Base Upper 32 Bits - PMBU32



Internal Bus Address: NA

PCI Configuration Address Offset: 28h

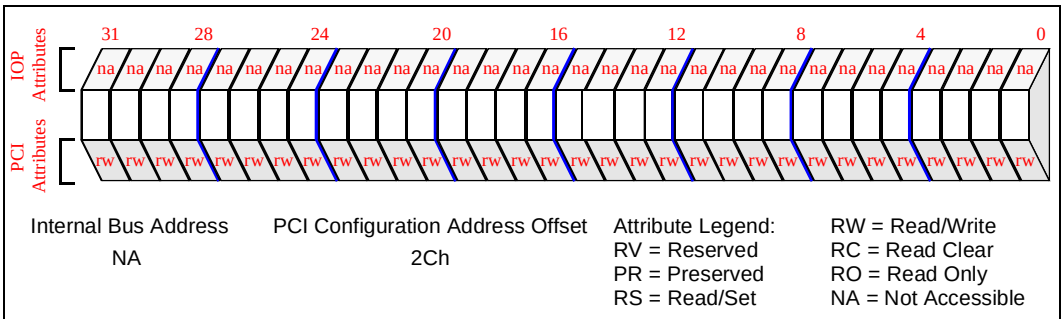
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000h	Prefetchable Memory Base Upper Portion: All bits are read/writable - bridge supports full 64-bit addressing.

2.5.1.16 Prefetchable Memory Limit Upper 32 Bits - PMLU32

This defines the upper 32 bits of the prefetchable address base register.

Table 39. Prefetchable Memory Limit Upper 32 Bits - PMLU32



Internal Bus Address: NA

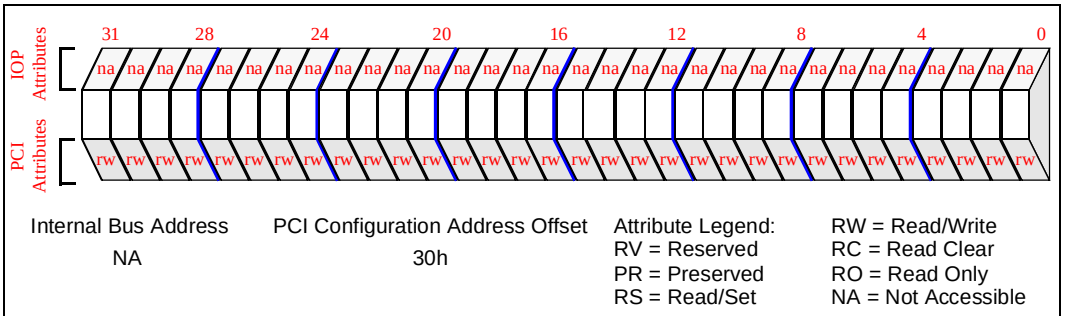
PCI Configuration Address Offset: 2Ch

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000h	Prefetchable Memory Limit Upper Portion: All bits are read/writable - bridge supports full 64-bit addressing.

2.5.1.17 I/O Base and Limit Upper 16 Bits - IOBLU16

Table 40. I/O Base and Limit Upper 16 Bits - IOBLU16



Internal Bus Address: NA

PCI Configuration Address Offset: 30h

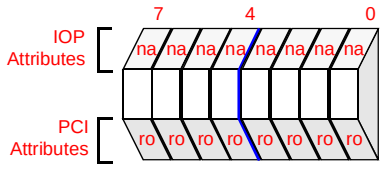
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	0000h	I/O Limit High 16 Bits: Upper 16 address lines (AD(31:16) used in conjunction with the I/O Limit Register (offset 1Dh) to construct the 32-bit I/O limit address.
15:00	0000h	I/O Base High 16 Bits: Upper 16 address lines (AD(31:16) used in conjunction with the I/O Base Register (offset 1Ch) to construct the 32-bit I/O base address.

2.5.1.18 Capabilities Pointer Register - Cap_Ptr

Contains the pointer to the first entry in the capabilities list.

Table 41. Capabilities Pointer Register - Cap_Ptr



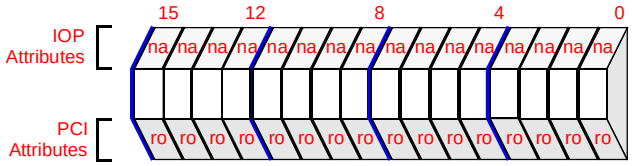
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	34h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07:00	DCh	Capabilities Pointer: Pointer to the first CAP ID entry in the capabilities list is at DCh in PCI configuration space. (Power Management Capability Registers)

2.5.1.19 Interrupt Information - INTR

This register contains information on interrupts and while required is not utilized by PCI-to-PCI bridges.

Table 42. Interrupt Information - INTR



Internal Bus Address: NA

PCI Configuration Address Offset: 3Ch

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

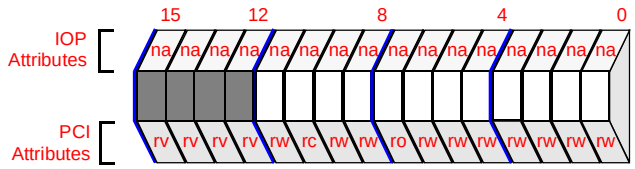
Bit	Default	Description
15:08	00h	Interrupt Pin (PIN): Bridges do not support the generation of interrupts.
07:00	00h	Interrupt Line (LINE): The bridge does not generate interrupts, so this is reserved as '00h'.

2.5.1.20 Bridge Control Register - BCR

This register provides extensions to the Command register that are specific to a PCI-to-PCI bridge. The Bridge Control register provides many of the same controls for the secondary interface that are provided by the Command register for the primary interface.

Some bits affect operation of both interfaces of the bridge.

Table 43. Bridge Control Register - BCR (Sheet 1 of 3)



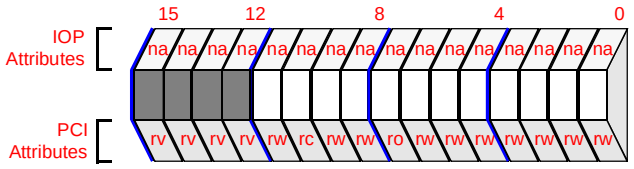
Internal Bus Address: NA

PCI Configuration Address Offset: 3Eh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:12	0h	Reserved
11	0b	Discard Timer SERR# Enable: Controls the generation of **SERR#** on the primary interface (**P_SERR#**) in response to a timer discard on either the primary or secondary interface. 0b = **SERR#** is not asserted. 1b = **SERR#** is asserted.
10	0b	Discard Timer Status (DTS): This bit is set to a '1b' when either the primary or secondary discard timer expires. The delayed completion is then discarded.
09	0b	Secondary Discard Timer (SDT): Sets the maximum number of PCI clock cycles that bridge waits for an initiator on the secondary bus to repeat a delayed transaction request. The counter starts when the delayed transaction completion is ready to be returned to the initiator. When the initiator has not repeated the transaction at least once before the counter expires, bridge discards the delayed transaction from its queues. 0b = The secondary master time-out counter is 2^{15} PCI clock cycles. 1b = The secondary master time-out counter is 2^{10} PCI clock cycles.
08	0b	Primary Discard Timer (PDT): Sets the maximum number of PCI clock cycles that bridge waits for an initiator on the primary bus to repeat a delayed transaction request. The counter starts when the delayed transaction completion is ready to be returned to the initiator. When the initiator has not repeated the transaction at least once before the counter expires, bridge discards the delayed transaction from its queues. 0b = The primary master time-out counter is 2^{15} PCI clock cycles. 1b = The primary master time-out counter is 2^{10} PCI clock cycles.
07	0b	Fast Back-to-Back Enable (FBE): The bridge does not initiate back to back transactions.
06	0b	Secondary Bus Reset (SBR): When cleared to 0b: The bridge deasserts S_RST#, when it had been asserted by writing this bit to a 1b. When set to 1b: The bridge asserts S_RST#.

Table 43. Bridge Control Register - BCR (Sheet 2 of 3)



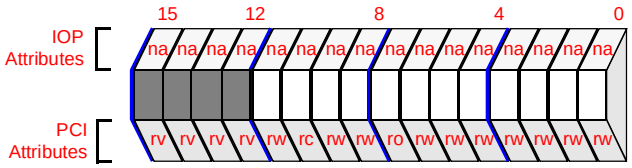
Internal Bus Address: NA

PCI Configuration Address Offset: 3Eh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
05	0b	Master Abort Mode (MAM): Dictates bridge behavior on the initiator bus when a master abort termination occurs in response to a delayed transaction initiated by bridge on the target bus. 0b = The bridge asserts TRDY# in response to a non-locked delayed transaction, and returns FFFF FFFFh when a read. 1b = When the transaction had not yet been completed on the initiator bus (e.g., delayed reads, or non-posted writes), then bridge returns a Target Abort in response to the original requester when it returns looking for its delayed completion on the initiator bus. When the transaction had completed on the initiator bus (e.g., a PMW), then bridge asserts **P_SERR#** (when enabled). For PCI-X transactions this bit is an enable for the assertion of **P_SERR#** due to a master abort while attempting to deliver a posted memory write on the destination bus.
04	0b	VGA Alias Filter Enable: This bit dictates bridge behavior in conjunction with the VGA enable bit (also of this register), and the VGA Palette Snoop Enable bit (Command Register). When the VGA enable, or VGA Palette Snoop enable bits are on (i.e., 1b) the VGA Aliasing bit for the corresponding enabled functionality,: 0b = Ignores address bits AD[15:10] when decoding VGA I/O addresses. 1b = Ensures that address bits AD[15:10] equal 000000b when decoding VGA I/O addresses. When all VGA cycle forwarding is disabled, (i.e., VGA Enable bit =0b and VGA Palette Snoop bit =0b), then this bit has no impact on bridge behavior.
03	0b	VGA Enable: Setting this bit enables address decoding and transaction forwarding of the following VGA transactions from the primary bus to the secondary bus: • frame buffer memory addresses 000A0000h:000BFFFFh· • VGA I/O addresses 3B0:3BBh and 3C0h:3DFh, where AD[31:16] = "0000h" and AD[15:10] are either not decoded (i.e., don't cares), or must be "000000b" depending upon the state of the VGA Alias Filter Enable bit. (bit(4) of this register) I/O and Memory Enable bits must be set in the Command register to enable forwarding of VGA cycles.

Table 43. Bridge Control Register - BCR (Sheet 3 of 3)



Internal Bus Address: NA

PCI Configuration Address Offset: 3Eh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
02	0b	ISA Enable: Setting this bit enables special handling for the forwarding of ISA I/O transactions that fall within the address range specified by the I/O Base and Limit registers, and are within the lowest 64Kbyte of the I/O address map (i.e., 0000 0000h - 0000 FFFFh). 0b = All I/O transactions that fall within the I/O Base and Limit registers' specified range are forwarded from primary to secondary unfiltered. 1b = Blocks the forwarding from primary to secondary of the top 768 bytes of each 1Kbyte alias. On the secondary the top 768 bytes of each 1K alias are inversely decoded and forwarded from secondary to primary.
01	0b	SERR# Forward Enable: 0b = The bridge does not assert **P_SERR#** as a result of an S_SERR# assertion. 1b = The bridge asserts **P_SERR#** whenever S_SERR# is detected asserted provided the SERR# Enable bit is set (PCI Command Register bit(8)=1b).
00	0b	Parity Error Response: This bit controls bridge response to a parity error that is detected on its secondary interface. 0b = When a data parity error is detected bridge does not assert S_PERR#. Also bridge does not assert **P_SERR#** in response to a detected address or attribute parity error. 1b = When a data parity error is detected bridge asserts S_PERR#. The bridge also asserts **P_SERR#** (when enabled globally via bit(8) of the Command register) in response to a detected address or attribute parity error.

2.5.2 Device Specific Registers

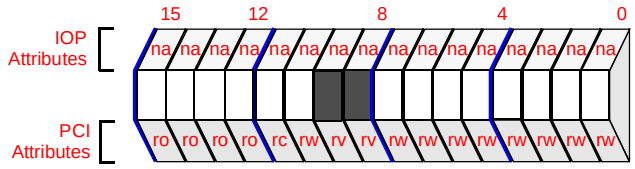
The following sections detail the programming interface for bridge device specific registers. These registers reside in bridge PCI configuration address space with an offset range of 40h - CBh.

Table 44. Bridge Device-Specific Configuration Address Map

Byte 3	Byte 2	Byte 1	Byte 0	Configuration Byte Offset
Bridge Control 0	Arbiter Control/Status		Reserved	40h
Bridge Control 2		Bridge Control 1		44h
Reserved		Bridge Status		48h
Reserved				4Ch
Prefetch Policy		Multi-Transaction Timer		50h
Reserved	Pre-boot Status	P_SERR# Assertion Control		54h
Reserved	Reserved	Secondary Decode Enable		58h
Reserved		Secondary IDSEL		5Ch
Reserved				5Ch
Reserved				68h:CBh

2.5.2.1 Secondary Arbiter Control/Status Register - SACSR

Table 45. Secondary Arbiter Control/Status Register - SACSR



Internal Bus Address: NA

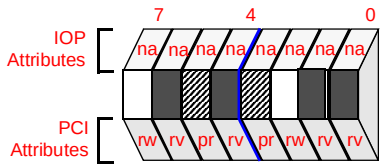
PCI Configuration Address Offset: 41h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:12	1111b	Grant Time-out Violator: This field indicates the agent that violated the Grant Time-out rule (PCI = 16 clocks, PCI-X = 6 clocks). Note that this field is only meaningful when: • Bit[11] of this register is set to 1b, indicating that a Grant Time-out violation had occurred. • bridge internal arbiter is enabled. Bits[15:12] Violating Agent (REQ#/GNT# pair number) 0000b REQ#/GNT#[0] 0001b REQ#/GNT#[1] 0010b REQ#/GNT#[2] 0011b REQ#/GNT#[3] 1111bDefault Value (no violation detected) When bit[11] is cleared by software, this field reverts back to its default value. All other values are Reserved
11	0b	Grant Time-out Occurred: When set to 1b, this indicates that a Grant Time-out error had occurred involving one of the secondary bus agents. Software clears this bit by writing a 1b to it.
10	0b	Bus Parking Control: 0 = During bus idle, bridge parks the bus on the last master to use the bus. 1 = During bus idle, bridge parks the bus on itself. The bus grant is removed from the last master and internally asserted to bridge.
09:08	00b	Reserved
07:00	0000 0000b	Secondary Bus Arbiter Priority Configuration: The bridge secondary arbiter provides two rings of arbitration priority. Each bit of this field assigns its corresponding secondary bus master to either the high priority arbiter ring (1b) or to the low priority arbiter ring (0b). Bits [3:0] correspond to request inputs S_REQ#[3:0], respectively. Bit [6] corresponds to the bridge internal secondary bus request while Bit [7] corresponds to the SATU secondary bus request. Bits [5:4] are unused. 0b = Indicates that the master belongs to the low priority group. 1b = Indicates that the master belongs to the high priority group

2.5.2.2 Bridge Control Register 0 - BCR0

Table 46. Bridge Control Register 0 - BCR0

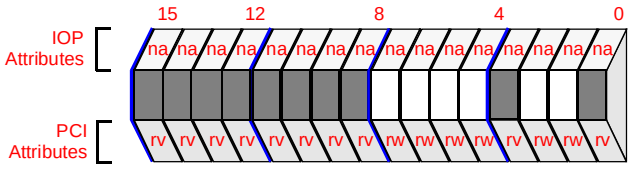


Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	43h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07	0b	Fully Dynamic Queue Mode: 0 = The number of Posted write transactions is limited to eight and the Posted Write data is limited to 4KB. 1 = Operation in fully dynamic queue mode. The bridge enqueues up to 14 Posted Memory Write transactions and 8KB of posted write data.
06:03	0H	Reserved.
02	0b	Upstream Prefetch Disable: This bit disables bridge ability to perform upstream prefetch operations for Memory Read requests received on its secondary interface. This bit also controls the bridge's ability to generate advanced read commands when forwarding a Memory Read Block transaction request upstream from a PCI-X bus to a Conventional PCI bus. 0b = bridge treats all upstream Memory Read requests as though they target prefetchable memory. The use of Memory Read Line and Memory Read Multiple is enabled when forwarding a PCI-X Memory Read Block request to an upstream bus operating in Conventional PCI mode. 1b = bridge treats upstream PCI Memory Read requests as though they target non-prefetchable memory and forwards upstream PCI-X Memory Read Block commands as Memory Read when the primary bus is operating in Conventional PCI mode. **NOTE:** This bit does not affect bridge ability to perform read prefetching when the received command is Memory Read Line or Memory Read Multiple.
01:00	0b	Reserved.

2.5.2.3 Bridge Control Register 1 - BCR1

Table 47. Bridge Control Register 1 - BCR1 (Sheet 1 of 2)



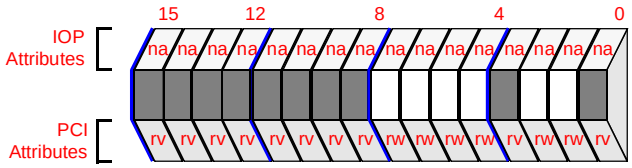
Internal Bus Address: NA

PCI Configuration Address Offset: 44h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:08	0000000b	Reserved
07:06	00b	Alias Command Mapping: This two bit field determines how bridge handles PCI-X "Alias" commands, specifically the Alias to Memory Read Block and Alias to Memory Write Block commands. The three options for handling these alias commands are to either pass it as is, re-map to the actual block memory read/write command encoding, or ignore the transaction forcing a Master Abort to occur on the Origination Bus. Bit (7:6) — Handling of command 0 0 — Re-map to Memory Read/Write Block before forwarding 0 1 — Enqueue and forward the alias command code unaltered 1 0 — Ignore the transaction, forcing Master Abort 1 1 — Reserved
05	1b	Watchdog Timers Disable: Disables or enables all 2^{24} Watchdog Timers in both directions. The watchdog timers are used to detect prohibitively long latencies in the system. The watchdog timer expires when any Posted Memory Write (PMW), Delayed Request, or Split Requests (PCI-X mode) is not completed within 2^{24} events ("events" are defined as PCI Clocks when operating in PCI-X mode, and as the number of times being retried when operating in Conventional PCI mode) 0b = All 2^{24} watchdog timers are enabled. 1b = All 2^{24} watchdog timers are disabled and there is no limits to the number of attempts bridge makes when initiating a PMW, transacting a Delayed Transaction, or how long it waits for a split completion corresponding to one of its requests.
04	0b	GRANT# time-out disable: This bit enables/disables the GNT# time-out mechanism. Grant time-out is 16 clocks for conventional PCI, and 6 clocks for PCI-X. 0b = The Secondary bus arbiter times out an agent that does not assert FRAME# within 16/6 clocks of receiving its grant, once the bus has gone idle. The time-out counter begins as soon as the bus goes idle with the new GNT# asserted. An infringing agent does not receive a subsequent GNT# until it de-asserts its REQ# for at least one clock cycle. 1b = GNT# time-out mechanism is disabled.
03	00b	Reserved.

Table 47. Bridge Control Register 1 - BCR1 (Sheet 2 of 2)



Internal Bus Address: NA

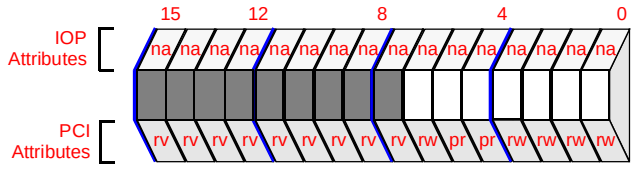
PCI Configuration Address Offset: 44h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
02	0b	Secondary Discard Timer Disable: This bit enables/disables bridge secondary delayed transaction discard mechanism. The time out mechanism is used to ensure that initiators of delayed transactions return for their delayed completion data/status within a reasonable amount of time after it is available from bridge. 0b = The secondary master time-out counter is enabled and uses the value specified by the Secondary Discard Timer bit (see Bridge Control Register). 1b = The secondary master time-out counter is disabled. The bridge waits indefinitely for a secondary bus master to repeat a delayed transaction.
01	0b	Primary Discard Timer Disable: This bit enables/disables bridge primary delayed transaction discard mechanism. The time out mechanism is used to ensure that initiators of delayed transactions return for their delayed completion data/status within a reasonable amount of time after it is available from bridge. 0b = The primary master time-out counter is enabled and uses the value specified by the Primary Discard Timer bit (see Bridge Control Register). 1b = The secondary master time-out counter is disabled. The bridge waits indefinitely for a secondary bus master to repeat a delayed transaction.
00	0b	Reserved

2.5.2.4 Bridge Control Register 2 - BCR2

Table 48. Bridge Control Register 2 - BCR2



Internal Bus Address: NA

PCI Configuration Address Offset: 46h

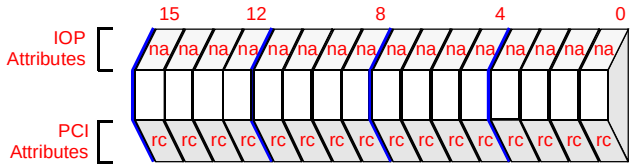
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:07	0000b	Reserved.
06	0b	Global Clock Out Disable (External Secondary Bus Clock Source Enable): This bit disables all of the secondary PCI clock outputs including the feedback clock **S_CLKOUT**. This means that the user is required to provide an S_CLKIN input source.
05:04	11 (66 MHz) 01 (100 MHz) 00 (133 MHz)	Preserved.
03:00	Fh (100 MHz & 66 MHz) 7h (133 MHz)	This 4 bit field provides individual enable/disable mask bits for each of bridge secondary PCI clock outputs. Some, or all secondary clock outputs (S_CLKO[3:0]) default to being enabled following the rising edge of P_RST#, depending on the frequency of the secondary bus clock: • Designs with 100 MHz (or lower) Secondary PCI clock power up with all four S_CLKOs enabled by default. (SCLKO[3:0])· • Designs with 133 MHz Secondary PCI clock power up with the lower order 3 S_CLKOs enabled by default. (S_CLKO[2:0]) Only those SCLKs that power up enabled by can be connected to downstream device clock inputs.

2.5.2.5 Bridge Status Register - BSR

Table 49. Bridge Status Register - BSR (Sheet 1 of 2)



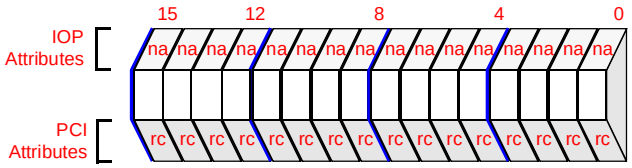
Internal Bus Address: NA

PCI Configuration Address Offset: 48h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0b	Upstream Delayed Transaction Discard Timer Expired: This bit is set to a 1b and **P_SERR#** is conditionally asserted when the secondary discard timer expires.
14	0b	Upstream Delayed/Split Read Watchdog Timer Expired: Conventional PCI Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards an upstream delayed read transaction request after 2^{24} retries following the initial retry. PCI-X Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards an upstream split read request after waiting in excess of 2^{24} clocks for the corresponding Split Completion to arrive.
13	0b	Upstream Delayed/Split Write Watchdog Timer Expired: Conventional PCI Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards an upstream delayed write transaction request after 2^{24} retries following the initial retry. PCI-X Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards an upstream split write request after waiting in excess of 2^{24} clocks for the corresponding Split Completion to arrive.
12	0b	Master Abort during Upstream Posted Write: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a Master Abort occurs as a result of an attempt, by bridge, to retire a PMW upstream.
11	0b	Target Abort during Upstream Posted Write: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a Target Abort occurs as a result of an attempt, by bridge, to retire a PMW upstream.
10	0b	Upstream Posted Write Data Discarded: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards an upstream PMW transaction after receiving 2^{24} target retries from the primary bus target
09	0b	Upstream Posted Write Data Parity Error: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a data parity error is detected by bridge while attempting to retire a PMW upstream
08	0b	Secondary Bus Address Parity Error: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge detects an address parity error on the secondary bus.

Table 49. Bridge Status Register - BSR (Sheet 2 of 2)



Internal Bus Address: NA

PCI Configuration Address Offset: 48h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07	0b	Downstream Delayed Transaction Discard Timer Expired: This bit is set to a 1b and **P_SERR#** is conditionally asserted when the primary bus discard timer expires.
06	0b	Downstream Delayed/Split Read Watchdog Timer Expired: Conventional PCI Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards a downstream delayed read transaction request after receiving 2^{24} target retries from the secondary bus target. PCI-X Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards a downstream split read request after waiting in excess of 2^{24} clocks for the corresponding Split Completion to arrive.
05	0b	Downstream Delayed Write/Split Watchdog Timer Expired: Conventional PCI Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards a downstream delayed write transaction request after receiving 2^{24} target retries from the secondary bus target. PCI-X Mode: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards a downstream split write request after waiting in excess of 2^{24} clocks for the corresponding Split Completion to arrive.
04	0b	Master Abort during Downstream Posted Write: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a Master Abort occurs as a result of an attempt, by bridge, to retire a PMW downstream.
03	0b	Target Abort during Downstream Posted Write: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a Target Abort occurs as a result of an attempt, by bridge, to retire a PMW downstream.
02	0b	Downstream Posted Write Data Discarded: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge discards a downstream PMW transaction after receiving 2^{24} target retries from the secondary bus target
01	0b	Downstream Posted Write Data Parity Error: This bit is set to a 1b and **P_SERR#** is conditionally asserted when a data parity error is detected by bridge while attempting to retire a PMW downstream.
00	0b	Primary Bus Address Parity Error: This bit is set to a 1b and **P_SERR#** is conditionally asserted when bridge detects an address parity error on the primary bus.

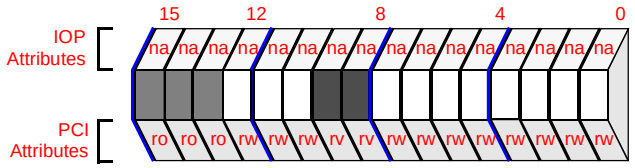
2.5.2.6 Bridge Multi-Transaction Timer Register - BMTTR

The bridge Multi-Transaction Timer mechanism (MTT) provides the bridge itself, and any secondary bus master a configurable extended grant duration when desired. The contents of this register have no affect on the operation of the bridge when the internal secondary arbiter is disabled.

The MTT register specifies a common value that establishes a total grant time that applies for all secondary masters. The default value for MTT is 64 PCI clocks.

Refer to Section 2.3.1, "Arbitration Events" on page 67, and Section 2.3.2, "Multi-Transaction Timer (MTT)" on page 69 for detailed information regarding the operation of, and rules governing, the Multi-Transaction Timer.

Table 50. Bridge Multi-Transaction Timer Register - BMTTR



Internal Bus Address: NA

PCI Configuration Address Offset: 50h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:13	000b	Reserved
12:10	000b	GRANT# Duration: This field specifies the count (in PCI clocks) that a secondary bus master has its grant maintained in order to enable multiple transactions to execute within the same arbitration cycle. Bit[02:00] GNT# Extended Duration 000 MTT Disabled (Default = no GNT# extension) 001 16 clocks 010 32 clocks 011 64 clocks 100 128 clocks 101 256 clocks 110 Invalid (treated as 000) 111 Invalid (treated as 000)
09:08	00b	Reserved
07:00	FFh	MTT Mask: This field enables/disables MTT usage for each REQ#/GNT# pair supported by bridge secondary arbiter. Bit(7) corresponds to SATU internal REQ#/GNT# pair, bit(6) corresponds to bridge internal REQ#/GNT# pair, bit 5 corresponds to REQ#/GNT#(5) pair, etc. When a given bit is set to 1b, its corresponding REQ#/GNT# pair is enabled for MTT functionality as determined by bits(12:10) of this register. When a given bit is cleared to 0b, its corresponding REQ#/GNT# pair is disabled from using the MTT.

2.5.2.7 Read Prefetch Policy Register - RPPR

The Prefetch Policy register is used when the initiating bus is operating in Conventional PCI mode to determine bridge read prefetch parameters.

The amount of data that is to be prefetched is based on the type of read command, i.e., MRL, MRM or a Memory Read targeting pre-fetchable memory space, coupled with the read data demand demonstrated by the PCI initiator.

This register configures a pre-fetchable read byte count based on a "FIRSTREAD" parameter, and a subsequent, larger read byte count based on the "REREAD" parameter that takes affect for a given initiator when its initial demands are not satisfied by the initial pre-fetch amount. (i.e., bridge, not the delayed read transaction initiator, is forced to disconnect the initial read transaction on the originating bus, leaving the initiator waiting for additional read data.)

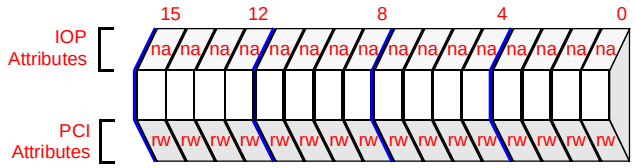
This "smart" pre-fetch policy may be disabled, in which case imposes a fixed pre-fetch policy, also based on read command type. The first three secondary **REQ#**/GNT# pairs have individual enable/disable bits. A single enable/disable bit governs smart pre-fetch operation for the remainder of the secondary **REQ#**/GNT# pairs.

The primary bus interface does not require an enable bit. The affect of leaving the Primary bus FirstRead, and ReRead parameters in their power on default states establishes the equivalent behavior.

The bridge supports separate smart pre-fetch parameters for the Primary and secondary bus interfaces.

Refer to Section 2.2.12.5.4, "Prefetching" on page 60 for detailed information regarding the operation of the Prefetch feature.

Table 51. Read Prefetch Policy Register - RPPR (Sheet 1 of 2)

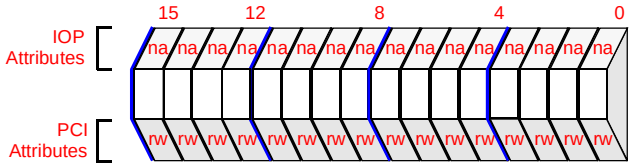


Internal Bus Address: NA
PCI Configuration Address Offset: 52h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:13	000b	ReRead_Primary Bus: 3-bit field indicating the multiplication factor to be used in calculating the number of bytes to prefetch from the secondary bus interface on subsequent PreFetch operations given that the read demands were not satisfied using the FirstRead parameter. The default value of 000b correlates to: Command Type — Hardwired pre-fetch amount Memory Read — 4 DWORDs Memory Read Line — 1 cache lines Memory Read Multiple — 2 cache lines

Table 51. Read Prefetch Policy Register - RPPR (Sheet 2 of 2)



Internal Bus Address: NA

PCI Configuration Address Offset: 52h

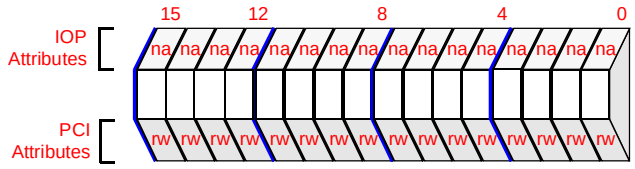
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
12:10	000b	FirstRead_Primary Bus: 3-bit field indicating the multiplication factor to be used in calculating the number of bytes to prefetch from the secondary bus interface on the initial PreFetch operation. The default value of 000b correlates to: Command Type — Hardwired pre-fetch amount Memory Read — 4 DWORDs Memory Read Line — 1 cache line Memory Read Multiple — 2 cache lines
09:07	010b	ReRead_Secondary Bus: 3-bit field indicating the multiplication factor to be used in calculating the number of bytes to prefetch from the primary bus interface on subsequent PreFetch operations given that the read demands were not satisfied using the FirstRead parameter. The default value of 010b correlates to: Command Type — Hardwired pre-fetch amount Memory Read — 3 cache lines Memory Read Line — 3 cache lines Memory Read Multiple — 6 cache lines
06:04	000b	FirstRead_Secondary Bus: 3-bit field indicating the multiplication factor to be used in calculating the number of bytes to prefetch from the primary bus interface on the initial PreFetch operation. The default value of 000b correlates to: Command Type — Hardwired pre-fetch amount Memory Read — 4 DWORDs Memory Read Line — 1 cache line Memory Read Multiple — 2 cache lines
03:00	1111b	Staged Prefetch Enable: This field enables/disables the FirstRead/ReRead pre-fetch algorithm for the secondary and the primary bus interfaces. Bit(3) is a ganged enable bit for REQ#/GNT#[7:3], and bits(2:0) provide individual enable bits for REQ#/GNT#[2:0]. (bit(2) is the enable bit for REQ#/GNT#[2], etc...) 1b: enables the staged pre-fetch feature 0b: disables staged pre-fetch, and hardwires read pre-fetch policy to the following for Memory Read, Memory Read Line, and Memory Read Multiple commands: Command Type Hardwired Pre-Fetch Amount Memory Read 4 DWORDs Memory Read Line 1 cache line Memory Read Multiple 2 cache lines **NOTE:** When the starting address is not cache line aligned, bridge pre-fetches Memory Read line commands only to the next higher cache line boundary. For non-cache line aligned Memory Read Multiple commands bridge pre-fetches only to the second cache line boundary encountered.

2.5.2.8 P_SERR# Assertion Control - SERR_CTL

This register specifies which events, when detected, cause bridge to assert **SERR#** on its primary bus. This register does not affect the behavior of the bridge unless the SERR# Enable bit (PCI command register) is set which enables, subject to the further conditions specified herein, the assertion of **SERR#** on the primary bus.

Table 52. P_SERR# Assertion Control - SERR_CTL (Sheet 1 of 3)



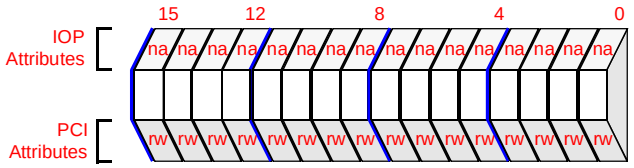
Internal Bus Address: NA

PCI Configuration Address Offset: 54h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0b	Upstream Delayed Transaction Discard Timer Expired: Dictates the bridge behavior in response to its discarding of a delayed transaction that was initiated from the primary bus. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
14	0b	Upstream Delayed/Split Read Watchdog Timer Expired: Dictates bridge behavior following expiration of the subject watchdog timer. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
13	0b	Upstream Delayed/Split Write Watchdog Timer Expired: Dictates bridge behavior following expiration of the subject watchdog timer. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
12	0b	Master Abort during Upstream Posted Write: Dictates bridge behavior following its having detected a Master Abort while attempting to retire one of its PMWs upstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
11	0b	Target Abort during Upstream Posted Write: Dictates bridge behavior following its having been terminated with Target Abort while attempting to retire one of its PMWs upstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
10	0b	Upstream Posted Write Data Discarded: Dictates bridge behavior in the event that it discards an upstream posted write transaction. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
09	0b	Upstream Posted Write Data Parity Error: Dictates bridge behavior when a data parity error is detected while attempting to retire on of its PMWs upstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**

Table 52. P_SERR# Assertion Control - SERR_CTL (Sheet 2 of 3)



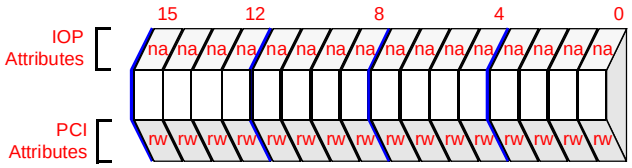
Internal Bus Address: NA

PCI Configuration Address Offset: 54h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
08	0b	Secondary Bus Address Parity Error: This bit dictates bridge behavior when it detects an address parity error on the secondary bus. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
07	0b	Downstream Delayed Transaction Discard Timer Expired: Dictates bridge behavior in response to its discarding of a delayed transaction that was initiated on the secondary bus. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
06	0b	Downstream Delayed/Split Read Watchdog Timer Expired: Dictates bridge behavior following expiration of the subject watchdog timer. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
05	0b	Downstream Delayed/Split Write Watchdog Timer Expired: Dictates bridge behavior following expiration of the subject watchdog timer. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
04	0b	Master Abort during Downstream Posted Write: Dictates bridge behavior following its having detected a Master Abort while attempting to retire one of its PMWs downstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
03	0b	Target Abort during Downstream Posted Write: Dictates bridge behavior following its having been terminated with Target Abort while attempting to retire one of its PMWs downstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**

Table 52. P_SERR# Assertion Control - SERR_CTL (Sheet 3 of 3)



Internal Bus Address: NA

PCI Configuration Address Offset: 54h

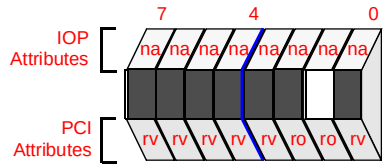
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
02	0b	Downstream Posted Write Data Discarded: Dictates bridge behavior in the event that it discards a downstream posted write transaction. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
01	0b	Downstream Posted Write Data Parity Error: Dictates bridge behavior when a data parity error is detected while attempting to retire on of its PMWs downstream. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**
00	0b	Primary Bus Address Parity Error: This bit dictates bridge behavior when it detects an address parity error on the primary bus. 0b = bridge asserts **P_SERR#**. 1b = bridge does not assert **P_SERR#**

2.5.2.9 Pre-Boot Status Register - PBSR

This register provides status for each of the strapable configuration settings. Status is updated, and is available immediately following power on reset.

Table 53. Pre-Boot Status Register - PBSR



Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	56h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

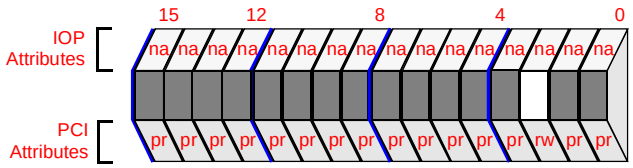
Bit	Default	Description
07	1	Reserved
06	-	Reserved - value indeterminate
05:02	0	Reserved
01	Varies with External State of **S_133EN** at PCI Bus Reset	Secondary Bus Max Frequency Setting: This bit reflect captured S_133EN strap, indicating the maximum secondary bus clock frequency when in PCI-X mode. Max Allowable Secondary Bus Frequency S_133EN PCI-X Mode 0 100 MHz 1 133 MH
00	0b	Reserved

2.5.2.10 Secondary Decode Enable Register - SDER

The Secondary Decode Enable Register controls the address decode functions on the Secondary PCI interface of the bridge unit.

The Private Memory Space Enable bit allows a private memory space to be created on the Secondary PCI bus. When this bit is set, bridge overrides its secondary inverse decode logic and not forward upstream any secondary bus initiated DAC Memory transactions with AD(63)=1b.

Table 54. Secondary Decode Enable Register - SDER



Internal Bus Address: NA

PCI Configuration Address Offset: 58h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:03	FFF1h	Preserved.
02	Varies with External State of **PRIVMEM** at PCI Bus Reset	Private Memory Space Enable - when set, bridge overrides its secondary inverse decode logic and not forward upstream any secondary bus initiated DAC Memory transactions with AD(63)=1b.This creates a private memory space on the Secondary PCI bus that allows peer-to-peer transactions.
01:00	10_2	Preserved.

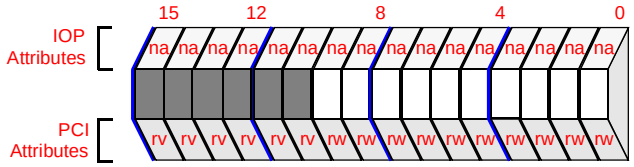
2.5.2.11 Secondary IDSEL Select Register - SISR

The Secondary IDSEL Select Register controls the usage of **S_AD[25:16]** in Type 1 to Type 0 conversions from the Primary to Secondary interface. In default operation, a unique encoding on Primary addresses **P_AD[15:11]** results in the assertion of one bit on the Secondary address bus **S_AD[31:16]** during a Type 1 to Type 0 conversion (See Section 2.2.5.1.). This is used for the assertion of **IDSEL** on the device being targeted by the Type 0 configuration command. This register allows Secondary address bits **S_AD[25:16]** to be used to configure private PCI devices by forcing Secondary address bits **S_AD[25:16]** to all zeros during Type 1 to Type 0 conversions, regardless of the state of Primary addresses **P_AD[15:11]** (device number in Type 1 configuration command).

When any address bit within **S_AD[25:16]** is to be used for private Secondary PCI devices, the user must guarantee that the corresponding bit in the SISR register is set before the host tries to configure the hierarchical PCI buses.

Private devices can be initially configured in 80331 through the PRIVMEM reset strap. Pulling this input high during reset causes the SISR bits to default to one, resulting in Device Numbers zero through nine (devices with **IDSEL** tied to **S_AD[25:16]**) to be hidden from the Primary PCI interface.

Table 55. Secondary IDSEL Select Register - SISR (Sheet 1 of 2)



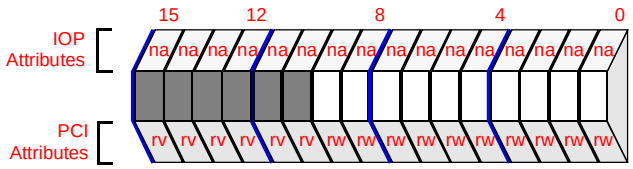
Internal Bus Address: NA

PCI Configuration Address Offset: 5Ch

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:10	000000_2	Reserved.
09	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD25**- **IDSEL** Disable - When this bit is set, **AD25** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD25** is asserted when Primary addresses **AD[15:11]** = 01001_2 during a Type 1 to Type 0 conversion.
08	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD24**- **IDSEL** Disable - When this bit is set, **AD24** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD24** is asserted when Primary addresses **AD[15:11]** = 01000_2 during a Type 1 to Type 0 conversion.
07	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD23** - **IDSEL** Disable - When this bit is set, **AD23** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD23** is asserted when Primary addresses **AD[15:11]** = 00111_2 during a Type 1 to Type 0 conversion.
06	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD22** - **IDSEL** Disable - When this bit is set, **AD22** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD22** is asserted when Primary addresses **AD[15:11]** = 00110_2 during a Type 1 to Type 0 conversion.

Table 55. Secondary IDSEL Select Register - SISR (Sheet 2 of 2)



Internal Bus Address: NA

PCI Configuration Address Offset: 5Ch

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
05	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD21** - **IDSEL** Disable - When this bit is set, **AD21** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD21** is asserted when Primary addresses **AD[15:11]** = 00101_2 during a Type 1 to Type 0 conversion.
04	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD20** - **IDSEL** Disable - When this bit is set, **AD20** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD20** is asserted when Primary addresses **AD[15:11]** = 00100_2 during a Type 1 to Type 0 conversion.
03	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD19** - **IDSEL** Disable - When this bit is set, **AD19** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD19** is asserted when Primary addresses **AD[15:11]** = 00011_2 during a Type 1 to Type 0 conversion.
02	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD18** - **IDSEL** Disable - When this bit is set, **AD18** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD18** is asserted when Primary addresses **AD[15:11]** = 00010_2 during a Type 1 to Type 0 conversion.
01	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD17** - **IDSEL** Disable - When this bit is set, **AD17** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD17** is asserted when Primary addresses **AD[15:11]** = 00001_2 during a Type 1 to Type 0 conversion.
00	Varies with External State of **PRIVDEV** at PCI Bus Reset	**AD16** - **IDSEL** Disable - When this bit is set, **AD16** is deasserted for any possible Type 1 to Type 0 conversion. When clear, **AD16** is asserted when Primary addresses **AD[15:11]** = 00000_2 during a Type 1 to Type 0 conversion.

2.5.3 PCI Extended Capabilities List

The following sections detail the programming interface for the bridge Capabilities List. This standard link list infrastructure lays out the programming model for the following extended capabilities:

- *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a compliant.
- *PCI Bus Power Management Interface Specification*, Revision 1.1 compliant.

Table 56 outlines the address map for the extended capabilities, followed by the individual sub sections where full interface details are provided.

Table 56. Enhanced Capabilities Register File

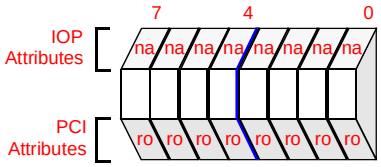
Byte 3	Byte 2	Byte 1	Byte 0	Configuration Byte Offset
Power Management Capabilities		Next Item Ptr	Capability ID	DCh
PM Data	PPB Support Extensions	Power Management CSR		E0h
Reserved		Reserved	Reserved	E4h
Reserved				E8h
Reserved	Reserved	Reserved	Reserved	ECh
PCI-X Secondary Status		Next Item Ptr	Capability ID	F0h
PCI-X Bridge Status				F4h
PCI-X Upstream Split Transaction Control				F8h
PCI-X Downstream Split Transaction Control				FCh

2.5.3.1 PCI Bus Power Management

2.5.3.1.1 Power Management Capabilities Identifier - PM_CAPID

Read by system software as 01h to indicate that this is the PCI Power Management data structure.

Table 57. Power Management Capabilities Identifier - PM_CAPID



Internal Bus Address: NA

PCI Configuration Address Offset: DCh

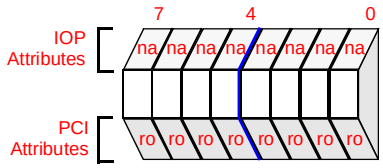
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	01h	Identifier (ID): PCI SIG assigned ID for PCI-PM register block

2.5.3.1.2 Next Item Pointer - PM_NXTP

This points to the next item in the function's capability list.

Table 58. Next Item Pointer - PM_NXTP



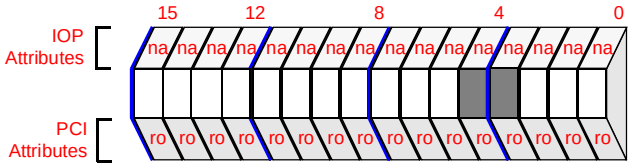
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	DDh	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07:00	F0H	Next Capabilities Pointer (PTR): The register defaults to F0H pointing to the PCI-X Extended Capability Header.

2.5.3.1.3 Power Management Capabilities Register - PMCR

This register reports bridge power management capabilities.

Table 59. Power Management Capabilities Register - PMCR



Internal Bus Address: NA

PCI Configuration Address Offset: DEh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

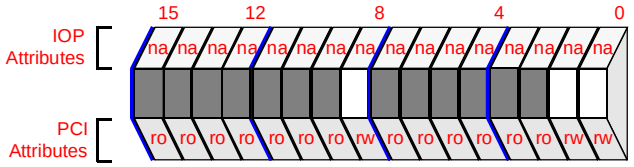
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:11	00h	PME Supported (PME): **PME#** cannot be asserted by bridge.
10	0h	State D2 Supported (D2): Indicates no support for state D2. No power management action in this state.
09	1h	State D1 Supported (D1): Indicates support for state D1. No power management action in this state.
08:06	0h	Auxiliary Current (AUXC): This 3 bit field reports the 3.3Vaux auxiliary current requirements for the PCI function. This returns 000b as **PME#** wake-up for bridge is not implemented.
05	0	Special Initialization Required (SINT): Special initialization is not required for bridge.
04:03	00	Reserved
02:00	010	Version (VS): Indicates that this supports *PCI Bus Power Management Interface Specification*, Revision 1.1.

2.5.3.1.4 Power Management Control / Status Register - PMCSR

This register is used to manage the bridge power management state as well as to enable/monitor PMEs.

Table 60. Power Management Control / Status - Register - PMCSR



Internal Bus Address: NA

PCI Configuration Address Offset: E0h

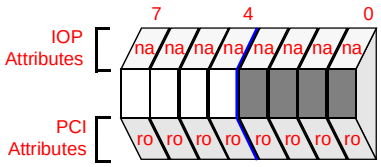
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:09	00h	Reserved
08	0b	PME_Enable: This bit, when set to 1b enables bridge to assert **PME#**. Note that bridge never has occasion to assert **PME#** and implements this dummy R/W bit only for the purpose of working around an OS PCI-PM bug.
07:02	00h	Reserved
01:00	00	Power State (PSTATE): This 2-bit field is used both to determine the current power state of a function and to set the Function into a new power state. 00 - D0 state 01 - D1 state 10 - D2 state 11 - $D3_{hot}$ state

2.5.3.1.5 Power Management Control / Status PCI-to-PCI Bridge Support - PMCSR_BSE

Indicates support for PCI bridge specific functionality.

Table 61. Power Management Control / Status PCI to PCI Bridge Support - PMCSR_BSE



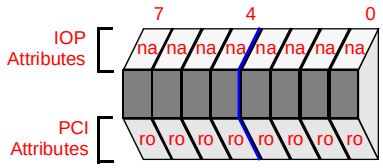
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	E2h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

Bit	Default	Description
07	0	Bus Power/Clock Control Enable (BPCC_En): Indicates that the bus power/clock control policies have been disabled.
06	0	B2/B3 support for D3 Hot (B2_B3#): The state of this bit determines the action that is to occur as a direct result of programming the function to D3 hot. This bit is only meaningful when bit 7 (BPCC_En) is a "1".
05:00	00h	Reserved

2.5.3.1.6 Power Management Data Register - PMDR

The data register is not supported.

Table 62. Power Management Data Register - PMDR



Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
NA	E3h	RV = Reserved PR = Preserved RS = Read/Set	RW = Read/Write RC = Read Clear RO = Read Only NA = Not Accessible

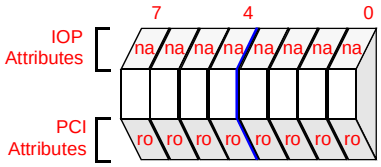
Bit	Default	Description
07:00	00h	Reserved

2.5.3.2 *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a

2.5.3.2.1 PCI-X Capabilities Identifier - PX_CAPID

Identifies this item in the Capabilities list as a PCI-X register set. It returns 07h when read.

Table 63. PCI-X Capabilities Identifier - PX_CAPID



Internal Bus Address: NA

PCI Configuration Address Offset: F0h

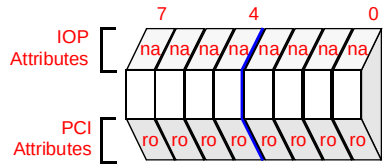
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	07h	Identifier (ID): Indicates this is a PCI-X capabilities list.

2.5.3.2.2 Next Item Pointer - PX_NXTP

Indicates where the next item in the capabilities list resides. The PCI-X capability register block is normally the end of the bridge Capabilities List, so "00h" is the power on reset default state.

Table 64. Next Item Pointer - PX_NXTP



Internal Bus Address: NA

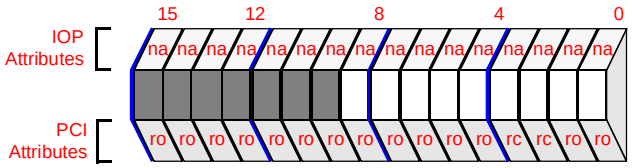
PCI Configuration Address Offset: F1h

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	00h	Next Item Pointer: Points to the next capability in the linked list The power on default value of this register is 00h indicating that this is the last entry in the linked list of capabilities.

2.5.3.2.3 PCI-X Secondary Status - PX_SSTS

Table 65. PCI-X Secondary Status - PX_SSTS



Internal Bus Address: NA

PCI Configuration Address Offset: F2h

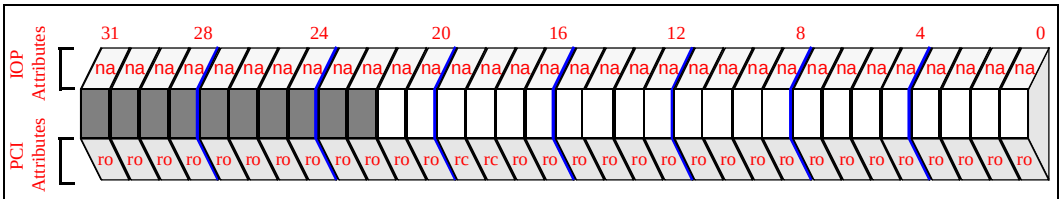
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:09	00h	Reserved
08:06	Xxx	Secondary Clock Frequency (SCF): This field is set with the frequency of the secondary bus. The values are: BitsMax FrequencyClock Period 000PCI ModeN/A 00166 15 01010010 0111337.5 1xxreservedreserved The default value for this register is the operating frequency of the secondary bus
05	0b	Split Request Delayed. (SRD): This bit is supposed to be set by a bridge when it cannot forward a transaction on the secondary bus to the primary bus because there is not enough room within the limit specified in the Split Transaction Commitment Limit field in the Downstream Split Transaction Control register. The bridge does not set this bit.
04	0b	Split Completion Overrun (SCO): This bit is supposed to be set when a bridge terminates a Split Completion on the secondary bus with retry or Disconnect at next ADB because its buffers are full. The bridge does not set this bit.
03	0b	Unexpected Split Completion (USC): This bit is set when an unexpected split completion with a requester ID equal to bridge secondary bus number, device number 00h, and function number 0 is received on the secondary interface. This bit is cleared by software writing a '1'.
02	0b	Split Completion Discarded (SCD): This bit is set when bridge discards a split completion moving toward the secondary bus because the requester would not accept it. This bit cleared by software writing a '1'.
01	1b	133 MHz Capable: Indicates that bridge is capable of running its secondary bus at 133 MHz
00	1b	64-bit Device (D64): Indicates the width of the secondary bus as 64-bits.

2.5.3.2.4 PCI-X Bridge Status - PX_BSTS

Identifies PCI-X capabilities and current operating mode of the bridge.

Table 66. PCI-X Bridge Status - PX_BSTS



Internal Bus Address: NA

PCI Configuration Address Offset: F4h

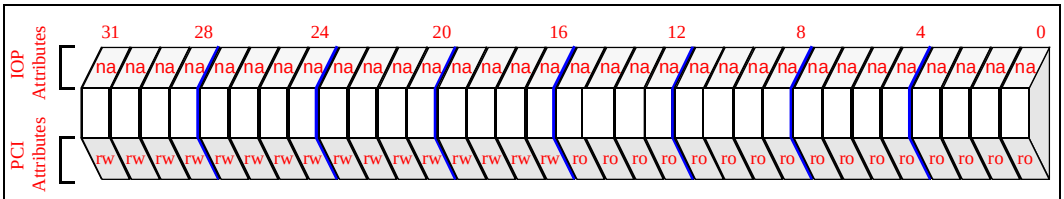
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:22	0	Reserved
21	0	Split Request Delayed (SRD): This bit does not be set by bridge.
20	0	Split Completion Overrun (SCO): This bit does not be set by bridge because bridge throttles traffic on the completion side.
19	0	Unexpected Split Completion (USC): The bridge sets this bit to 1b when it encounters a corrupted Split Completion, possibly with an inconsistent remaining byte count.Software clears this bit by writing a 1b to it.
18	0	Split Completion Discarded (SCD): The bridge sets this bit to 1b when it has discarded a Split Completion.Software clears this bit by writing a 1b to it.
17	1	133 MHz Capable: This bit indicates that the bridge primary interface is capable of 133 MHz operation in PCI-X mode. 0 = The maximum operating frequency is 66 MHz. 1 = The maximum operating frequency is 133 MHz.
16	Varies with the external state of **P_32BITPCI#** at PCI Bus Reset	64-bit Device (D64): Indicates bus width of the Primary PCI bus interface. 0 = Primary Interface is connected as a 32-bit PCI bus. 1 = Primary Interface is connected as a 64-bit PCI bus.
15:08	00h	Bus Number (BNUM): This field is simply an alias to the PBN field of the BNUM register at offset 18h. Apparently it was deemed necessary reflect it here for diagnostic purposes.
07:03	1fh	Device Number (DNUM): Indicates which IDSEL bridge consumes. May be updated whenever a PCI-X configuration write cycle that targets bridge scores a hit.
02:00	0h	Function Number (FNUM): The bridge Function #

2.5.3.2.5 PCI-X Upstream Split Transaction Control - PX_USTC

This register identifies controls behavior of bridge buffers for forwarding Split Transactions from the secondary bus to primary.

Table 67. PCI-X Upstream Split Transaction Control - PX_USTC



Internal Bus Address: FFFF.FFFF

PCI Configuration Address Offset: F8h

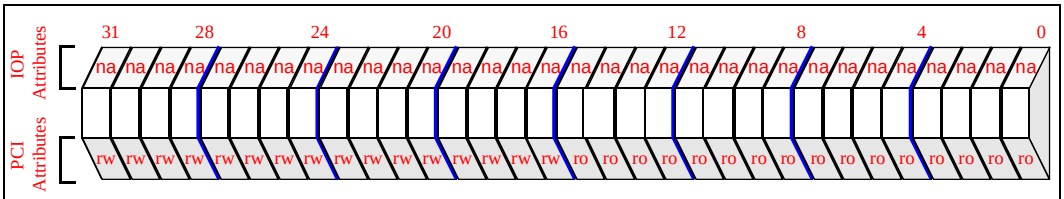
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	003Eh	Split Transaction Limit (STL): This register indicates the size of the commitment limit in units of ADQs. Software is permitted to program this register to any value greater than or equal to the contents of the Split Transaction Capacity register. A value less than the contents of the Split Transaction Capacity register causes unspecified results. A value of 003Eh or greater enables the bridge to forward all Split Requests of any size regardless of the amount of buffer space available.
15:00	003Eh	Split Transaction Capacity (STC): This read-only field indicates the size of the buffer (in number of ADQs) for storing split completions. This register controls behavior of the bridge buffers for forwarding Split Transactions from a primary bus requester to a secondary bus completer. The default value of 003Eh indicates there is available buffer space for 62 ADQs (7936 bytes).

2.5.3.2.6 PCI-X Downstream Split Transaction Control - PX_DSTC

This register controls behavior of bridge buffers for forwarding Split Transactions from primary to the secondary bus.

Table 68. PCI-X Downstream Split Transaction Control - PX_DSTC



Internal Bus Address
FFFF.FFFF

PCI Configuration Address Offset
FCh

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	003Eh	Split Transaction Limit (STL): This register indicates the size of the commitment limit in units of ADQs. Software is permitted to program this register to any value greater than or equal to the contents of the Split Transaction Capacity register. A value less than the contents of the Split Transaction Capacity register causes unspecified results. A value of 003Eh or greater enables the bridge to forward all Split Requests of any size regardless of the amount of buffer space available.
15:00	003Eh	Split Transaction Capacity (STC): This read-only field indicates the size of the buffer (in number of ADQs) for storing split completions. This register controls behavior of the bridge buffers for forwarding Split Transactions from a primary bus requester to a secondary bus completer. The default value of 003Eh indicates there is available buffer space for 62 ADQs (7936 bytes).

This Page Intentionally Left Blank

Address Translation Unit 3

This chapter describes the operation modes, setup, and implementation of the module which interfaces between the PCI bus and the Intel® 80331 I/O processor (80331) internal bus.

3.1 Overview

As indicated in Figure 7, the Address Translation Unit (ATU) — the interface between the PCI bus and the on-chip internal bus — consists of the Address Translation Unit (ATU), the Expansion ROM Unit and the Messaging Unit (MU) described in Chapter 4, "Messaging Unit"

The ATU supports both inbound and outbound address translation. The ATU provides access between the PCI bus and the 80331 internal bus. The ATU and the MU share PCI address space.

Transactions initiated on the PCI bus and targeted at the 80331 internal bus are referred to as *inbound transactions* (PCI to internal bus). Transactions initiated on the 80331 internal bus and targeted at the PCI bus are referred to as *outbound transactions* (internal bus to PCI). The ATU accepts multiple inbound or outbound transactions and processes them simultaneously.

During inbound transactions, the ATU converts PCI addresses (initiated by a PCI bus master) to internal bus addresses and initiates the data transfer on the 80331 internal bus. During outbound transactions, the ATU converts internal bus addresses to PCI addresses and initiates the data transfer on the PCI bus.

The Messaging Unit provides a mechanism for the system processor and the 80331 to transfer control information. The Messaging Unit occupies the first 4 Kbytes in Memory Window 0 of the ATU address space. PCI masters on the PCI interface of the 80331 access the MU by addressing the ATU anywhere in the first 4 KB offset from the ATU Base Address Register 0.

The Expansion ROM provides the PCI mechanism for downloading device/board driver code during system boot sequence. It consists of a separate inbound address range which accesses a Flash EPROM device connected through the 80331 memory controller. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details of Expansion ROM usage.

The Address Translation Unit, the Expansion ROM Translation Unit, and the Messaging Unit appear as a single PCI device on the PCI bus. That is, the 80331 is a single-function device.

The ATU supports the PCI 64-bit and 66 MHz extensions providing up to 532 Mbytes/sec of PCI bandwidth as well as the PCI-X 64-bit and 133 MHz extensions providing up to 1064 Mbytes/sec of PCI bandwidth. On the internal interface, the ATU implements the 80331 internal bus protocol which provides for a maximum of 1600 Mbytes/sec using 64-bit/200 MHz signaling.

The ATU includes three extended capability headers that implement Power Management capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1, MSI capability as defined by *PCI Local Bus Specification*, Revision 2.3, and PCI-X capability as defined by *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

The functionality of the ATU is described in the following sections. The ATU and the MU have a memory-mapped register interface that is visible from either the PCI interface, the internal bus interface, or both.

Figure 7. ATU Block Diagram

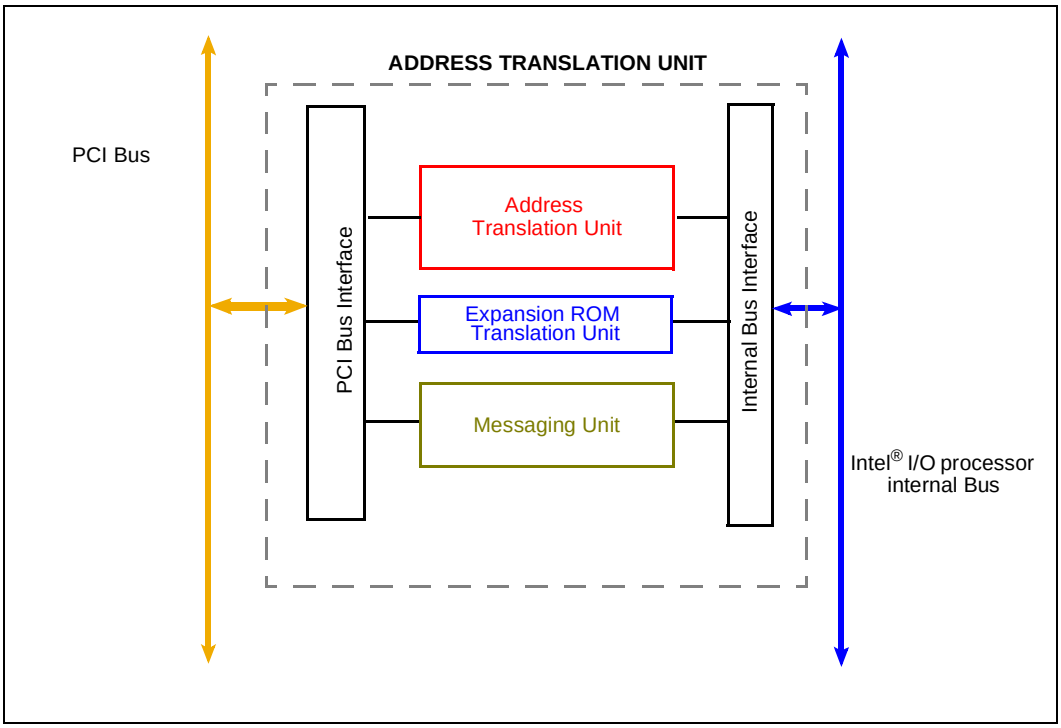
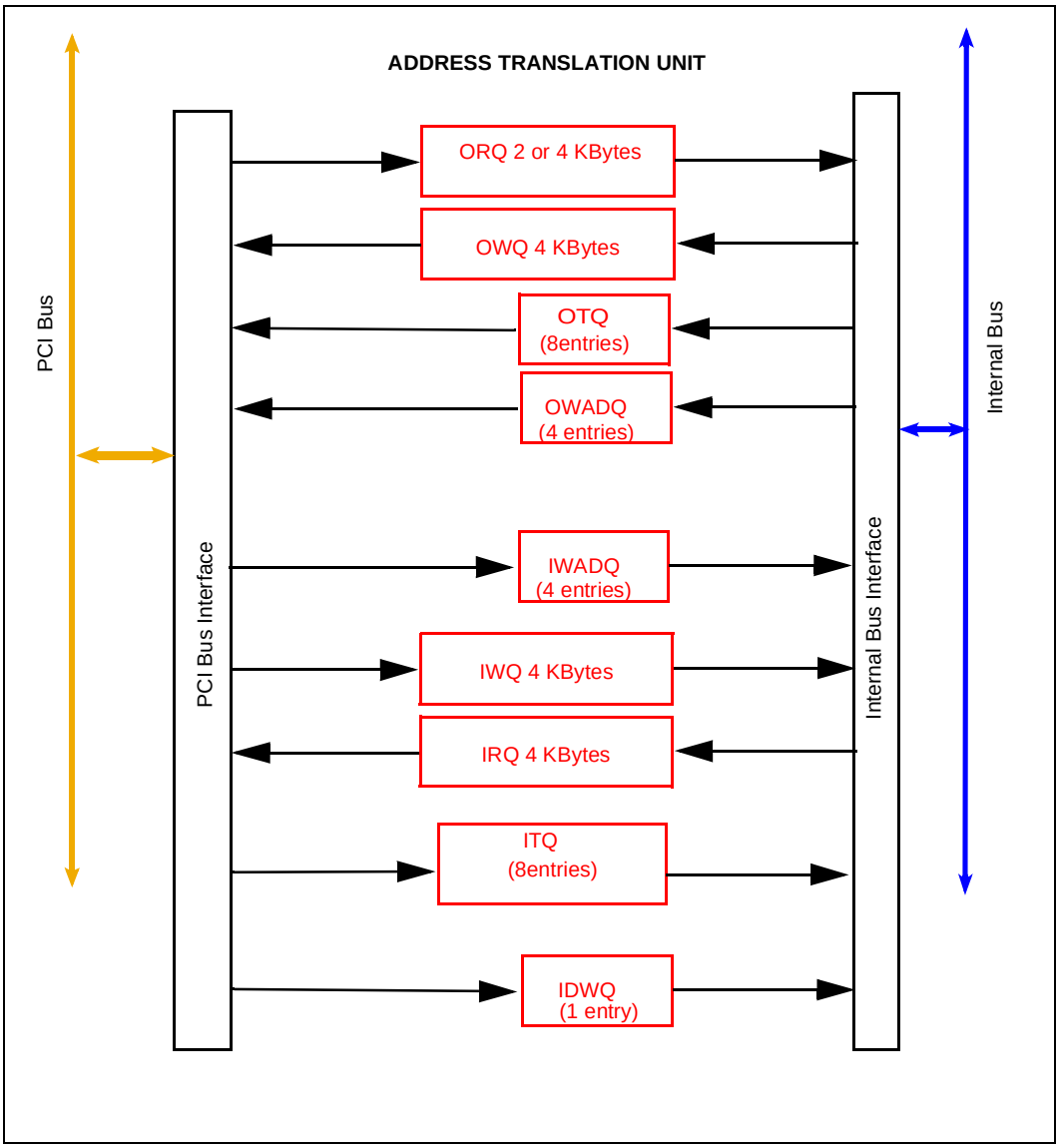


Figure 8. ATU Queue Architecture Block Diagram



3.2 ATU Address Translation

The ATU allows PCI masters on the PCI bus to initiate transactions to the 80331 internal bus and allows the Intel® XScale™ core (ARM* architecture compliant) to initiate transactions to the PCI bus.

The ATU implements an address windowing scheme to determine which addresses to claim and translate to the destination bus.

- The address windowing mechanism for inbound translation is described in Section 3.2.1.1, "Inbound Address Translation" on page 139
- The address windowing mechanism for outbound translation is described in Section 3.2.2, "Outbound Transactions- Single Address Cycle (SAC) Internal Bus Transactions" on page 149 and Section 3.2.3, "Outbound Write Transaction" on page 157

The ATU has the ability to accept up to eight inbound PCI read transactions and four inbound PCI write transactions simultaneously. Also, the ATU has the ability to accept up to eight outbound internal bus read transactions and four outbound internal bus write transactions simultaneously. Refer to Figure 8 and Section 3.5 for details of the ATU queue architecture.

The ATU unit allows for recognition and generation of multiple PCI cycle types. Table 69 shows the PCI and PCI-X commands supported for both inbound and outbound ATU transactions. The type of operation seen by the ATU on inbound transactions is determined by the PCI master who initiates the transaction. Claiming an inbound transaction depends on the address range programmed within the inbound translation window. The type of transaction used by the ATU on outbound transactions generated by the core processor is determined by the internal bus address and the fixed outbound windowing scheme.

ATU supports the 64-bit addressing specified by the *PCI Local Bus Specification*, Revision 2.3. This 64-bit addressing extension is supported for both inbound and outbound data transactions. This is in addition to the 64-bit data extensions supported by the 80331.

ATU does not support exclusive access using the PCI LOCK# signal. Also, the ATU does not guarantee atomicity for outbound transactions.

Table 69. ATU Command Support

PCI Command Encoding	PCI Command Type	PCI-X Command Type	Claimed on Inbound Transactions on PCI Bus	Generated by Outbound Transactions on PCI Bus	Valid Internal Bus Command
0000	Interrupt Acknowledge	Interrupt Acknowledge	No	No	Interrupt Acknowledge
0001	Special Cycle	Special Cycle	No	No	Special Cycle
0010	I/O Read	I/O Read	No	Yes	I/O Read
0011	I/O Write	I/O Write	No	Yes	I/O Write
0100	reserved	reserved	No	No	reserved
0101	reserved	reserved	No	No	reserved
0110	Memory Read	Memory Read DWORD	**Yes**	Yes	Memory Read DWORD
0111	Memory Write	Memory Write	**Yes**	Yes	Memory Write
1000	reserved	Alias to Memory Read Block	**Yes**	No	Alias to Memory Read Block
1001	reserved	Alias to Memory Write Block	**Yes**	No	Alias to Memory Write Block
1010	Configuration Read	Configuration Read	**Yes**	Yes	Configuration Read
1011	Configuration Write	Configuration Write	**Yes**	Yes	Configuration Write
1100	Memory Read Multiple	Split Completion	**Yes**	Yes	Split Completion
1101	Dual Address Cycle	Dual Address Cycle	**Yes**	Yes	Dual Address Cycle
1110	Memory Read Line	Memory Read Block	**Yes**	Yes[a]	Memory Read Block
1111	Memory Write and Invalidate	Memory Write Block	**Yes**	Yes	Memory Write Block

a. PCI-X mode only.

Inbound and outbound ATU transactions are best described by the data flows used on the PCI bus and the 80331 internal bus during read and write operations. The following sections describe read and write operations for inbound ATU transactions (PCI to internal bus) and outbound transactions (internal bus to PCI).

3.2.1 Inbound Transactions

Inbound transactions which target the ATU are translated and executed on the 80331 internal bus. As a PCI target, the ATU is capable of accepting all PCI memory read and write operations as either a 32-bit or a 64-bit PCI target. In the conventional PCI mode *Memory Write* and *Memory Write and Invalidate* operations are performed as posted operations and all memory read operations are performed as delayed reads. In the PCI-X mode *Memory Write*, *Memory Write Block*, and *Alias to Memory Write Block* operations are performed as posted operations and *Memory Read DWORD*, *Memory Read Block*, and *Alias to Memory Read Block* operations are executed as split transactions. The ATU is capable of accepting configuration read and write cycles. In the conventional PCI mode, *Configuration Writes* are performed as delayed memory write operations and *Configuration Reads* are performed as delayed read operations. In the PCI-X mode, both *Configuration Writes* and *Configuration Reads* are performed as split transactions.

Inbound memory write transactions have their addresses entered into the inbound write address queue (IWADQ) and data entered into the inbound write data queue (IWQ). The IWQ/IWADQ pair are capable of holding up to 4 write operations up to the size of the data queue. Inbound configuration writes use the inbound delayed write queue (IDWQ) for address and data. Refer to Section 3.5 for details of queue operation. Inbound read operations (memory and configuration) have their address entered into the inbound transaction queue (ITQ) and the data is returned to the PCI master in the inbound read queue (IRQ). The ITQ is capable of holding up to 8 delayed read requests (split read requests when operating in the PCI-X mode).

In the conventional PCI mode, for inbound transactions, the ATU is a slave on the PCI bus and is a requester on the internal bus. PCI slave operation is defined in the *PCI Local Bus Specification*, Revision 2.3. In the PCI-X mode, for inbound transactions, the ATU initially is a target on the PCI bus and becomes an initiator when performing split completion transactions, and is an initiator on the internal bus.

3.2.1.1 Inbound Address Translation

The ATU allows external PCI bus initiators to directly access the internal bus. These PCI bus initiators can read or write 80331 memory-mapped registers or 80331 local memory space. The process of inbound address translation involves two steps:

1. Address Detection.
   - Determine when the 32-bit PCI address (64-bit PCI address during DACs) is within the address windows defined for the inbound ATU.
   - Claim the PCI transaction with medium DEVSEL# timing in the conventional PCI mode and with Decode A DEVSEL# timing in the PCI-X mode.
2. Address Translation.
   - Translate the 32-bit PCI address (lower 32-bit PCI address during DACs) to a 32-bit 80331 internal bus address.

The ATU uses the following registers in inbound address window 0 translation:

- Inbound ATU Base Address Register 0
- Inbound ATU Limit Register 0
- Inbound ATU Translate Value Register 0

The ATU uses the following registers in inbound address window 2 translation:

- Inbound ATU Base Address Register 2
- Inbound ATU Limit Register 2
- Inbound ATU Translate Value Register 2

The ATU uses the following registers in inbound address window 3 translation:

- Inbound ATU Base Address Register 3
- Inbound ATU Limit Register 3
- Inbound ATU Translate Value Register 3

Note: Inbound Address window 1 is not a translate window. Instead, window 1 may be used to allocate host memory for Private Devices. Inbound Address window 3 does not reside in the standard section of the configuration header (offsets 00H - 3CH), thus the host BIOS does not configure window 3. Window 3 is intended to be used as a special window into local memory for private PCI agents controlled by the 80331 in conjunction with the Private Memory Space of the bridge. PCI-to-PCI Bridge in 80331 or

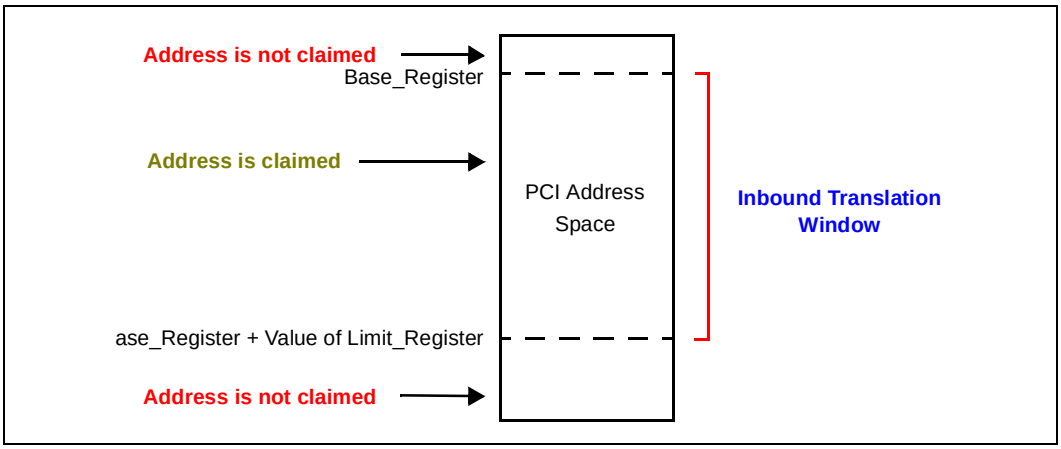
Inbound address detection is determined from the 32-bit PCI address, (64-bit PCI address during DACs) the base address register and the limit register. In the case of DACs none of the upper 32-bits of the address is masked during address comparison. The algorithm for detection is:

Equation 1. Inbound Address Detection

When PCI_Address [31:0] and Limit_Register[31:0] == Base_Register[31:0] and
PCI_Address [63:32] == Base_Register[63:32] (for DACs only) the PCI Address is claimed by the Inbound ATU.

Figure 9 shows an example of inbound address detection.

Figure 9. Inbound Address Detection



The incoming 32-bit PCI address (lower 32-bits of the address in case of DACs) is bitwise ANDed with the associated inbound limit register. When the result matches the base register (and upper base address matches upper PCI address in case of DACs), the inbound PCI address is detected as being within the inbound translation window and is claimed by the ATU.

Note: The first 4 Kbytes of the ATU inbound address translation window 0 are reserved for the Messaging Unit. See Section 3.3, "Messaging Unit" on page 161.

Once the transaction is claimed, the address must be translated from a PCI address to a 32-bit internal bus address. In case of DACs upper 32-bits of the address is simply discarded and only the lower 32-bits are used during address translation. The algorithm is:

Equation 2. Inbound Translation

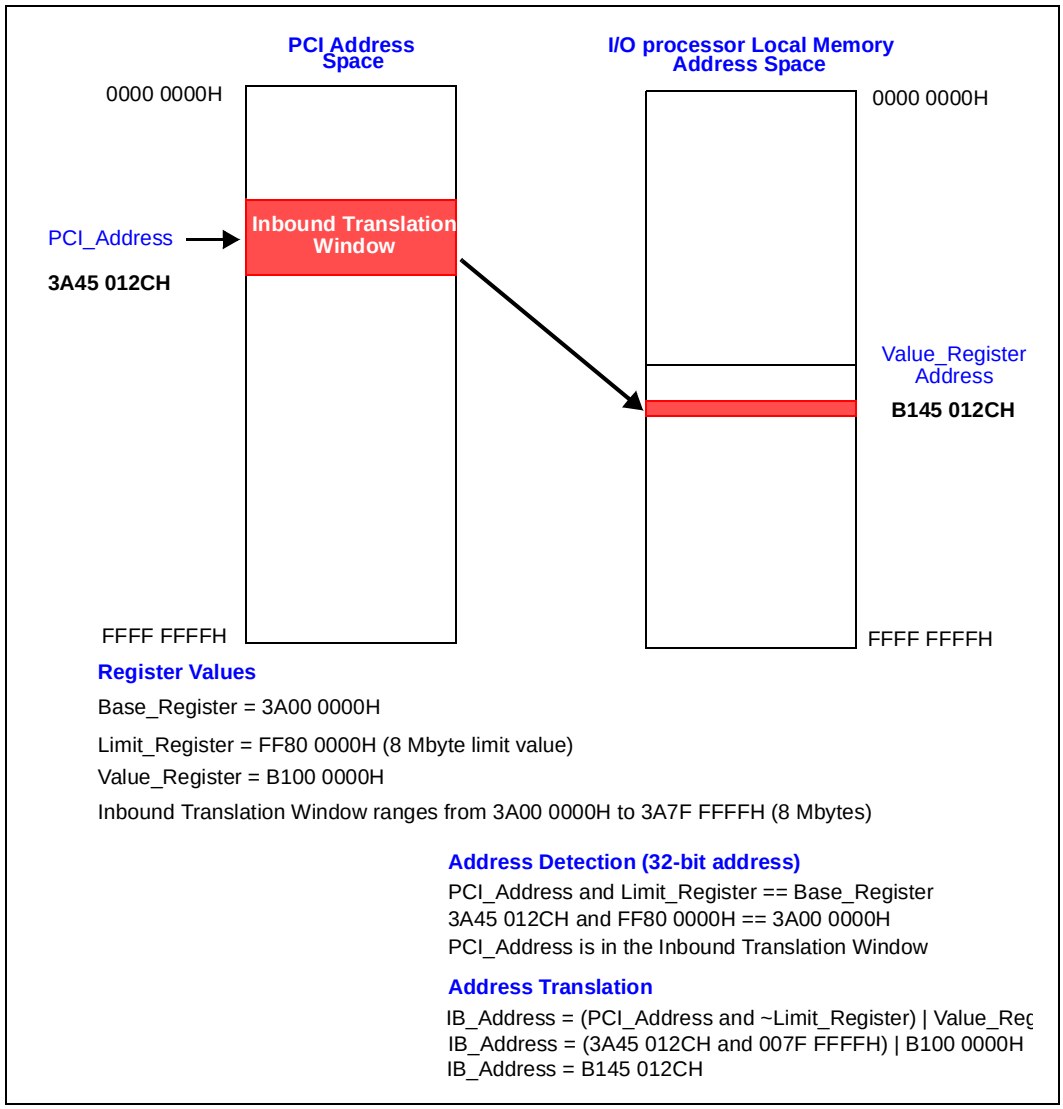
Intel® I/O processor Internal Bus Address = (PCI_Address[31:0] and ~Limit_Register[31:0]) | ATU_Translate_Value_Register[31:0].

The incoming 32-bit PCI address (lower 32-bits in case of DACs) is first bitwise ANDed with the bitwise inverse of the limit register. This result is bitwise ORed with the ATU Translate Value and the result is the internal bus address. This translation mechanism is used for all inbound memory read and write commands excluding inbound configuration read and writes. Inbound configuration cycle translation is described in Section 3.2.1.4, "Inbound Configuration Cycle Translation" on page 147.

In the PCI mode for inbound memory transactions, the only burst order supported is Linear Incrementing. For any other burst order, the ATU signals a Disconnect after the first data phase. The PCI-X supports linear incrementing only, and hence above situation is not encountered in the PCI-X mode.

Figure 10 shows an inbound translation example for 32-bit addressing. This example would hold true for an inbound transaction from PCI bus.

Figure 10. Inbound Translation Example



3.2.1.2 Inbound Write Transaction

An inbound write transaction is initiated by a PCI master and is targeted at either 80331 local memory or a 80331 memory-mapped register.

Data flow for an inbound write transaction on the PCI bus is summarized as:

- The ATU claims the PCI write transaction when the PCI address is within the inbound translation window defined by the ATU Inbound Base Address Register (and Inbound Upper Base Address Register during DACs) and Inbound Limit Register.
- When the IWADQ has at least one address entry available and the IWQ has at least one buffer available, the address is captured and the first data phase is accepted.
- The PCI interface continues to accept write data until one of the following is true:
  - The initiator performs a disconnect.
  - The transaction crosses a buffer boundary.
- When an address parity error is detected during the address phase of the transaction, the address parity error mechanisms are used. Refer to Section 3.7.1 for details of the address parity error response.
- When operating in the PCI-X mode when an attribute parity error is detected, the attribute parity error mechanism described in Section 3.7.1 is used.
- When a data parity error is detected while accepting data, the slave interface sets the appropriate bits based on PCI specifications. No other action is taken. Refer to Section 3.7.2.6 for details of the inbound write data parity error response.

Once the PCI interface places a PCI address in the IWADQ, when IWQ has received data sufficient to cross a buffer boundary or the master disconnects on the PCI bus, the ATUs internal bus interface becomes aware of the inbound write. When there are additional write transactions ahead in the IWQ/IWADQ, the current transaction remains posted until ordering and priority have been satisfied (Refer to Section 3.5.3) and the transaction is attempted on the internal bus by the ATU internal master interface. The ATU does not insert target wait states nor do data merging on the PCI interface, when operating in the PCI mode.

In the PCI-X mode memory writes are always executed as immediate transactions, while configuration write transactions are processed as split transactions. The ATU generates a Split Completion Message, (with Message class = 0h - Write Completion Class and Message index = 00h - Write Completion Message) once a configuration write is successfully executed.

Also, when operating in the PCI-X mode a write sequence may contain multiple write transactions. The ATU handles such transactions as independent transactions.

Data flow for the inbound write transaction on the internal bus is summarized as:

- The ATU internal bus master requests the internal bus when IWADQ has at least one entry with associated data in the IWQ.
- When the internal bus is granted, the internal bus master interface initiates the write transaction by driving the translated address onto the internal bus. For details on inbound address translation, see Section 3.2, "ATU Address Translation" on page 136.
- When **IB_DEVSEL#** is not returned, a master abort condition is signaled on the internal bus. The current transaction is flushed from the queue and **SERR#** may be asserted on the PCI interface.
- The ATU initiator interface asserts **IB_REQ64#** to attempt a 64-bit transfer. When **IB_ACK64#** is not returned, a 32-bit transfer is used. Transfers of less than 64-bits use the **IB_C/BE[7:0]#** to mask the bytes not written in the 64-bit data phase. Write data is transferred from the IWQ to the internal bus when data is available and the internal bus interface retains internal bus ownership.
- The internal bus interface stops transferring data from the current transaction to the internal bus when one of the following conditions becomes true:
  - The internal bus initiator interface loses bus ownership. The ATU internal initiator terminates the transfer (initiator disconnection) at the next ADB (for the internal bus ADB is defined as a naturally aligned 128-byte boundary) and attempt to reacquire the bus to complete the delivery of remaining data using the same sequence ID but with the modified starting address and byte count.
  - A Disconnect at Next ADB is signaled on the internal bus from the internal target. When the transaction in the IWQ completes at that ADB, the initiator returns to idle. When the transaction in the IWQ is not complete, the initiator attempts to reacquire the bus to complete the delivery of remaining data using the same sequence ID but with the modified starting address and byte count.
  - A Single Data Phase Disconnect is signaled on the internal bus from the internal target. When the transaction in the IWQ needs only a single data phase, the master returns to idle. When the transaction in the IWQ is not complete, the initiator attempts to reacquire the bus to complete the delivery of remaining data using the same sequence ID but with the modified starting address and byte count.
  - The data from the current transaction has completed (satisfaction of byte count). An initiator termination is performed and the bus returns to idle.
  - A Master Abort is signaled on the internal bus. **SERR#** may be asserted on the PCI bus. Data is flushed from the IWQ.

3.2.1.3 Inbound Read Transaction

An inbound read transaction is initiated by a PCI initiator and is targeted at either 80331 local memory or a 80331 memory-mapped register space. The read transaction is propagated through the inbound transaction queue (ITQ) and read data is returned through the inbound read queue (IRQ).

When operating in the conventional PCI mode, all inbound read transactions are processed as delayed read transactions. When operating in the PCI-X mode, all inbound read transactions are processed as split transactions. The ATUs PCI interface claims the read transaction and forwards the read request through to the internal bus and returns the read data to the PCI bus. Data flow for an inbound read transaction on the PCI bus is summarized in the following statements:

- The ATU claims the PCI read transaction when the PCI address is within the inbound translation window defined by ATU Inbound Base Address Register (and Inbound Upper Base Address Register during DACs) and Inbound Limit Register.
- When operating in the conventional PCI mode, when the ITQ is currently holding transaction information from a previous delayed read, the current transaction information is compared to the previous transaction information (based on the setting of the DRC Alias bit in Section 3.10.39, "ATU Configuration Register - ATUCR" on page 252). When there is a match and the data is in the IRQ, return the data to the master on the PCI bus. When there is a match and the data is not available, a Retry is signaled with no other action taken. When there is not a match and when the ITQ has less than eight entries, capture the transaction information, signal a Retry and initiate a delayed transaction. When there is not a match and when the ITQ is full, then signal a Retry with no other action taken.
  - When an address parity error is detected, the address parity response defined in Section 3.7 is used.
- When operating in the conventional PCI mode, once read data is driven onto the PCI bus from the IRQ, it continues until one of the following is true:
  - The initiator completes the PCI transaction. When there is data left unread in the IRQ, the data is flushed.
  - An internal bus Target Abort was detected. In this case, the QWORD associated with the Target Abort is never entered into the IRQ, and therefore is never returned.
  - Target Abort or a Disconnect with Data is returned in response to the Internal Bus Error.
  - The IRQ becomes empty. In this case, the PCI interface signals a Disconnect with data to the initiator on the last data word available.
- When operating in the PCI-X mode, when ITQ is not full, the PCI address, attribute and command are latched into the available ITQ and a Split Response Termination is signalled to the initiator.
- When operating in the PCI-X mode, when the transaction does not cross a 1024 byte aligned boundary, then the ATU waits until it receives the full byte count from the internal bus target before returning read data by generating the split completion transaction on the PCI-X bus. When the read requested crosses at least one 1024 byte boundary, then ATU completes the transfer by returning data in 1024 byte aligned chunks.

- When operating in the PCI-X mode, once a split completion transaction has started, it continues until one of the following is true:
  - The requester (now the target) generates a Retry Termination, or a Disconnection at Next ADB (when the requester is a bridge)
  - The byte count is satisfied.
  - An internal bus Target Abort was detected. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - target abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to Section 3.7.1.
  - An internal bus Master Abort was detected. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=80h - Master abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to Section 3.7.1
- When operating in the conventional PCI mode, when the master inserts wait states on the PCI bus, the ATU PCI slave interface waits with no premature disconnects.
- When a data parity error occurs signified by **PERR#** asserted from the initiator, no action is taken by the target interface. Refer to Section 3.7.2.5.
- When operating in the conventional PCI mode, when the read on the internal bus is target-aborted, either a target-abort or a disconnect with data is signaled to the initiator. This is based on the ATU ECC Target Abort Enable bit (bit 0 of the ATUIMR for ATU). When set, a target abort is used, when clear, a disconnect is used.
- When operating in the PCI-X mode (with the exception of the MU queue ports at offsets 40h and 44h), when the transaction on the internal bus resulted in a target abort, the ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - internal bus target abort) to inform the requester about the abnormal condition. For the MU queue ports, the ATU returns either a target abort or a single data phase disconnect depending on the ATU ECC Target Abort Enable bit (bit 0 of the ATUIMR for ATU). The ITQ for this transaction is flushed. Refer to Section 3.7.1.
- When operating in the conventional PCI mode, when the transaction on the internal bus resulted in a master abort, the ATU returns a target abort to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to Section 3.7.1
- When operating in the PCI-X mode, when the transaction on the internal bus resulted in a master abort, the ATU generates a Split Completion Message (message class=2h - completer error, and message index=80h - internal bus master abort) to inform the requester about the abnormal condition. The ITQ for this transaction is flushed. Refer to Section 3.7.1.
- When operating in the PCI-X mode, when the Split Completion transaction completes with either Master-Abort or Target-Abort, the requester is indicating a failure condition that prevents it from accepting the completion it requested. In this case, since the Split Request addresses a location that has no read side effects, the completer must discard the Split Completion and take no further action.

The data flow for an inbound read transaction on the internal bus is summarized in the following statements:

- The ATU internal bus master interface requests the internal bus when a PCI address appears in an ITQ and transaction ordering has been satisfied. When operating in the PCI-X mode the ATU does not use the information provided by the Relax Ordering Attribute bit. That is, ATU always uses conventional PCI ordering rules.
- Once the internal bus is granted, the internal bus master interface drives the translated address onto the bus and wait for **IB_DEVSEL#**. When a Retry is signaled, the request is repeated. When a master abort occurs, the transaction is considered complete and a target abort is loaded into the associated IRQ for return to the PCI initiator (transaction is flushed once the PCI master has been delivered the target abort).
- Once the translated address is on the bus and the transaction has been accepted, the internal bus target starts returning data with the assertion of **IB_TRDY#**. Read data is continuously received by the IRQ until one of the following is true:
  - The full byte count requested by the ATU read request is received. The ATU internal bus initiator interface performs a initiator completion in this case.
  - When operating in the conventional PCI mode, a Target Abort is received on the internal bus from the internal bus target. In this case, the transaction is aborted and the PCI side is informed.
  - When operating in the PCI-X mode, a Target Abort is received on the internal bus from the internal bus target. In this case, the transaction is aborted. The ATU generates a Split Completion Message (message class=2h - completer error, and message index=81h - target abort) on the PCI bus to inform the requester about the abnormal condition. The ITQ for this transaction is flushed.
  - When operating in the conventional PCI mode, a single data phase disconnection is received from the internal bus target. When the data has not been received up to the next QWORD boundary, the ATU internal bus master interface attempts to reacquire the bus. When not, the bus returns to idle.
  - When operating in the PCI-X mode, a single data phase disconnection is received from the internal bus target. The ATU IB initiator interface attempts to reacquire the bus to obtain remaining data.
  - When operating in the conventional PCI mode, a disconnection at Next ADB is received from the internal bus target. The bus returns to idle.
  - When operating in the PCI-X mode, a disconnection at Next ADB is received from the internal bus target. The ATU IB initiator interface attempts to reacquire the bus to obtain remaining data.

To support *PCI Local Bus Specification*, Revision 2.0 devices, the ATU can be programmed to ignore the memory read command (Memory Read, Memory Read Line, and Memory Read Multiple) when trying to match the current inbound read transaction with data in a DRC queue which was read previously (DRC on target bus). When the Read Command Alias Bit in the ATUCR register is set, the ATU does not distinguish the read commands on transactions. For example, the ATU enqueues a DRR with a Memory Read Multiple command and performs the read on the internal bus. Some time later, a PCI master attempts a Memory Read with the same address as the previous Memory Read Multiple. When the Read Command Bit is set, the ATU would return the read data from the DRC queue and consider the Delayed Read transaction complete. When the Read Command bit in the ATUCR was clear, the ATU would not return data since the PCI read commands did not match, only the address.

3.2.1.4 Inbound Configuration Cycle Translation

The ATU only accepts Type 0 configuration cycles with a function number of zero.

The ATU is configured through the PCI bus. When operating in the conventional PCI mode, all inbound configuration cycles are processed as delayed transactions. When operating in the PCI-X mode, all inbound configuration cycles are processed as split transactions.The translation mechanism for inbound configuration cycles is defined by the *PCI Local Bus Specification*, Revision 2.3.

The ATU configuration space is selected by a PCI configuration command and claims the access (by asserting DEVSEL#) when the IDSEL pin is asserted, the PCI command indicates a configuration read or write, and address bits **AD[1:0]** are 00_2 all during the address phase. The ATU interface ignores any configuration command (IDSEL active) where **AD[1:0]** are not 00_2 (e.g. Type 1 commands). During the configuration access address phase, the PCI address is divided into a number of fields to determine the actual configuration register access. These fields, in combination with the byte enables during the data phase create the unique encoding necessary to access the individual registers of the configuration address space:

- **AD[7:2] -** Register Number. Selects one of 64 DWORD registers in the ATU PCI configuration address space.
- **C/BE[3:0]# -** Used during the data phase. Selects which actual configuration register is used within the DWORD address. Creates byte addressability of the register space.
- **AD[10:8]** - Function Number. Used to select which function of a multi-function device is being accessed. The ATU is function 0 and therefore it only responds to 000_2 in this bit field and ignore all other bit combinations.

The ATU configuration address space starts at internal address FFFF.E100H. Therefore **AD[7:2]** equal to 000000_2 equates to address FFFF.E100H and **AD[7:2]** equal to 000001_2 results in address FFFF.E104H and so on.

For inbound configuration reads, the IRQ and ITQ are used in the same manner as inbound memory read operations. The internal bus cycle that results is a 32-bit transaction where **I_REQ64#** is not asserted.

For inbound configuration writes, the ATU adds a delayed write data queue, IDWQ, which holds data in the same manner as the IWQ. The transaction information from the configuration write operation on the PCI interface is captured into the IDWQ (when full, a Retry is signaled). The data from the delayed write (split write when operating in the PCI-X mode) request cycle is latched into the IDWQ and forwarded to the internal bus interface. Once transaction ordering and priority have been satisfied, the internal bus master interface requests the internal bus and deliver the write data to the target as defined in Section 3.2.1.2.

The status of the transaction on the internal bus is returned to the PCI initiator on the PCI bus. When operating in the conventional PCI mode, the retry cycle from the initiator is accepted once the write has been completed on the internal bus and the status has been captured for return to the PCI master. When operating in the PCI-X mode, a Split Completion Message (message class=0h and message index= 00h) is generated on the PCI bus, once the write has been completed on the internal bus and the status has been latched for return to the PCI master. Since Master Aborts and Target Aborts cannot occur during configuration cycles on the internal bus, normal completion status is returned. The data from PCI completion transaction is discarded.

3.2.1.5 Discard Timers

The ATU implements discard timers for inbound delayed transactions. These timers prevent deadlocks when the initiator of a retried delayed transaction fails to complete the transaction within 2^{10} or 2^{15} PCI clock cycles on the initiating bus when operating in the conventional PCI mode. The timer starts counting when the delayed request becomes a delayed completion by completing on the internal bus and all passing rules are satisfied. When the originating master on the PCI bus has not retried the transaction before the timer expires, the completion transaction is discarded.

Discard timer values are controlled by the ATU Configuration Register's Discard Timer Value bit. The ATU queues covered by discard timers are the IRQ and the IDWQ. After discarding a transaction, the ATU must set the Discard Timer Status bit in the ATU Configuration Register.The ATU does *not* assert the **SERR#** signal after discarding a transaction.

3.2.2 Outbound Transactions- Single Address Cycle (SAC) Internal Bus Transactions

Outbound transactions initiated by the 80331 core processor are directed to the PCI interface through the ATU. The core processor always generates Single Address Cycles on the internal bus. As a PCI master, the ATU is capable of PCI I/O transactions, PCI memory reads (in case of conventional PCI), memory read DWORD (in case of PCI-X), PCI memory writes, configuration reads and writes, and DAC cycles. Outbound memory transactions are always attempted as 64-bit PCI transactions. Outbound memory write operations are performed as posted operations and outbound memory read operations are all performed as split read operations.

Outbound transactions use a separate set of queues from inbound transactions. Outbound write operations have their address entered into the outbound write address queue (OWADQ) and their data into the outbound write queue (OWQ). Outbound read transactions, performed as split transactions, use the Outbound Transaction Queue (OTQ) to store address, and get data returned into the outbound read queue (ORQ). Refer to Section 3.5.2 for details of outbound queue architecture. Outbound configuration transactions use a special outbound port structure. Refer to Section 3.2.3 for details.

For outbound write transactions, the ATU is a target on the internal bus and initiator on the PCI bus. For outbound read transactions, the ATU is a completer on the internal bus (initially accepts the split read as a target and then provides read data by initiating a split completion).

3.2.2.1 Outbound Address Translation - Single Address Cycle (SAC) Internal Bus Transactions

In addition to providing the mechanism for inbound translation, the ATU translates Intel® XScale™ core-initiated cycles to the PCI bus. This is known as *outbound address translation*. Outbound transactions are processor or DMA transactions targeted at the PCI bus. The ATU internal bus target interface claims internal bus cycles and completes the cycle on the PCI bus on behalf of the Intel® XScale™ core or DMAs. The ATU supports two different outbound windowing modes:

- Address Translation Windowing
- Direct Address Windowing (No translation)

Figure 11 shows a 80331 memory map with all reserved address locations highlighted. The two 64 MByte outbound translation memory windows exist from 8000.0000H to 83FF.FFFFH and 8400.0000H to 87FF.FFFFH. The direct addressing window is from 0000.0040H to 7FFF.FFFFH. The 64KByte outbound I/O window is from 9000.0000H to 90000.FFFFH.

The response of the ATU to Outbound Transactions is controlled by bits in the ATU Command Register and the ATU Configuration Register. The Outbound ATUs behavior for the different combinations of these control bits is described in Table 70.

Table 70. Outbound Address Translation Control

Outbound Response	Bus Master Enable (ATUCMD - bit2)	Outbound Enable (ATUCR - bit1)
Master-Abort	0	0
Retry	0	1
Master-Abort	1	0
Claim[a]	1	1

a. The ATU may respond with a Retry in this case when the Outbound Transaction Queues are full.

3.2.2.2 Outbound Address Translation Windows- Single Address Cycle (SAC) Internal Bus Transactions

Inbound translation involves a programmable inbound translation window consisting of a base and limit register and a translate value register for PCI to internal bus translation. The outbound address translation windows use a similar methodology except that the outbound translation window base addresses and limit sizes are fixed in 80331 internal bus local address space; this removes the need for separate base and limit registers.

Figure 12 on page 154 illustrates the outbound address translation windows. The ATU has three windows: two are 64 Mbyte and one is 64 Kbyte. The two outbound memory translation windows range from 8000.0000H to 87FF.FFFFH. After these two windows, the outbound I/O window range from 9000.0000H to 9000.FFFFH.

The two Memory windows are 64 Mbytes and I/O window is 64 Kbytes. An internal bus cycle with an address within one of the outbound windows initiates a read or write cycle on the PCI bus. The PCI cycle type depends on which translation window the local bus cycle "hits". The read or write decision is based on the internal bus cycle type.

ATU has a window dedicated to the following outbound PCI/PCI-X transaction types:

- Memory reads and Memory writes - Memory Window
- I/O reads and writes - I/O Window

Table 71. Internal Bus-to-PCI Command Translation for Two Memory Windows/DMA Channels

Internal Bus Command	Conventional PCI Command	PCI-X Command
Memory Write	Memory Write	Memory Write
Memory Write Block	Memory Write and Invalidate[a] or Memory Write	Memory Write Block
Memory Read Block	Memory Read Multiple	Memory Read Block
Memory Read DWORD	Memory Read	Memory Read DWORD
Alias to Memory Write Block	Memory Write and Invalidate or Memory Write	Memory Write Block
Alias to Memory Read Block	Memory Read Multiple	Memory Read Block

a. The ATU converts (Alias to) Memory Write Block to Memory Write and Invalidate when the following four conditions are met, otherwise the Memory Write Block is converted to a Memory Write:
   1.) Memory Write and Invalidate transactions are enabled in the "ATU Command Register - ATUCMD".
   2.) The Cache Line size is set to 8 or 16 DWORDS in the "ATU Cacheline Size Register - ATUCLSR".
   3.) Starting address of the Outbound PCI bus request is aligned to the cache line size.
   4.) Byte count intended for the Outbound PCI bus request is a multiple of the cache line size.

Table 72. Internal Bus-to-PCI Command Translation for I/O Window

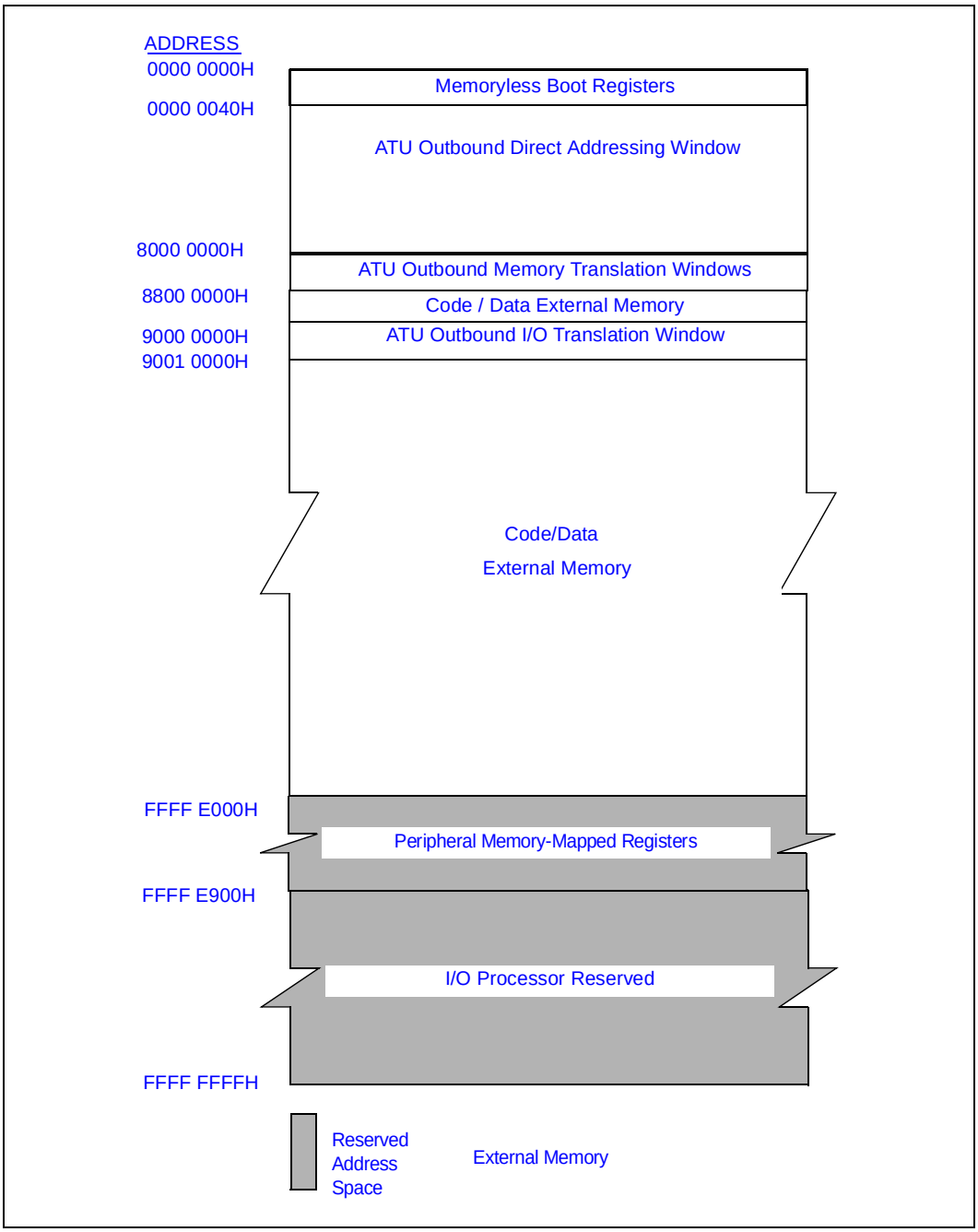
Internal Bus Command[a]	Conventional PCI Command	PCI-X Command
Memory Write	I/O Write	I/O Write
Memory Write Block	I/O Write	I/O Write
Memory Read Block	I/O Read	I/O Read
Memory Read DWORD	I/O Read	I/O Read
Alias to Memory Write Block	I/O Write	I/O Write
Alias to Memory Read Block	I/O Read	I/O Read

a. User should designate memory region containing I/O Window (9000.0000H to 90000.FFFFH) as non-cacheable and non-bufferable from Intel® XScale™ core. This guarantees all load/stores to I/O Window are of DWORD quantities. In event the user inadvertently issues a read to the I/O Window that crosses a DWORD address boundary, the ATU target aborts transaction. All writes are terminated with a Single-Phase-Disconnect and only bytes 3:0 is relevant dependent on the Byte Enables.

The windowing scheme refers to:

- An Internal Bus read cycle that addresses a Memory Window is translated to a Memory Read on the PCI bus
- An Internal Bus write cycle that addresses a Memory Window is translated to a Memory Write on the PCI bus
- An Internal Bus read cycle that addresses the I/O Window is an I/O Read on the PCI bus
- An Internal Bus write cycle that addresses the I/O Window is an I/O Write on the PCI bus

Figure 11. Internal Bus Memory Map



The translation portion of outbound ATU transactions is accomplished with a translate value register in the same manner as inbound translations. Each outbound memory window is associated with two translation registers which provide lower and upper translation addresses (OMWTVR0-1, OUMWTVR-1). When the corresponding OUMWTVRx register is all-zero a SAC transaction is generated on the PCI bus. Otherwise, a DAC is generated on the PCI bus. ATU uses the following registers during outbound address translation:

- Outbound Memory Window Translate Value Register 0 (OMWTVR0)
- Outbound Upper 32-bit Memory Window Translate Value Register 0 (OUMWTVR0)
- Outbound Memory Window Translate Value Register 1 (OMWTVR1)
- Outbound Upper 32-bit Memory Window Translate Value Register 1 (OUMWTVR1)
- Outbound I/O Window Translate Value Register (OIOWTVR)
- Outbound Configuration Cycle Address Register (OCCA)

See Section 3.8 for details on outbound translation register definition and programming constraints.

The translation algorithm used, as stated, is very similar to inbound translation. For memory transactions, the algorithm is:

Equation 3. Outbound Address Translation

```
PCI Address = (Internal_Bus_Address and 03FF.FFFFH) | Window_Translate_Value_Register
```

For memory transactions, the internal bus address is bitwise ANDed with the inverse of 64 Mbytes which clears the upper 6 bits of address. The result is bitwise ORed with the outbound window translate value register to create the lower 32-bits of the PCI address.

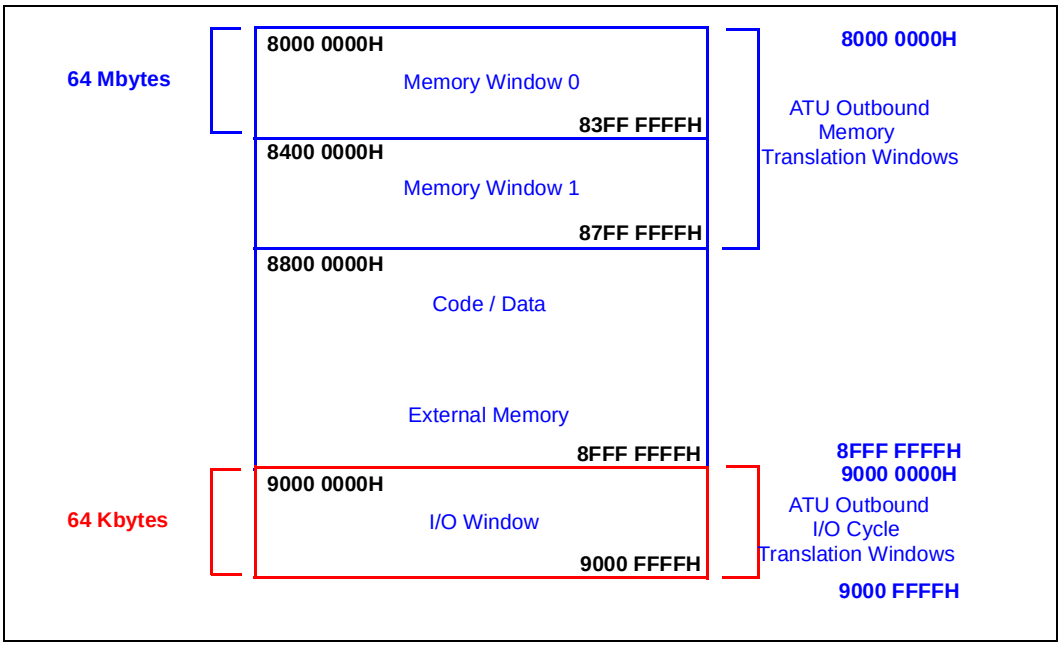
For I/O transactions, the algorithm is:

Equation 4. I/O Transactions

```
PCI Address = (Internal_Bus_Address and 0000.FFFFH) | Window_Translate_Value_Register
```

For I/O transactions, the internal bus address is bitwise ANDed with the inverse of 64 Kbytes which clears the upper 16 bits of address. Address aliasing is prevented by the outbound window translate value registers which only allow values on boundaries equivalent to the window length. PCI I/O addresses are byte addresses and not word addresses. The PCI I/O address two least significant bits are determined by byte enables that the processor issues. For example, when the Intel® XScale™ core performs a 2-byte write and generates byte enables of 0011_2, the ATU sets the two least significant bits of PCI I/O address to 10_2.

Figure 12. Outbound Address Translation Windows

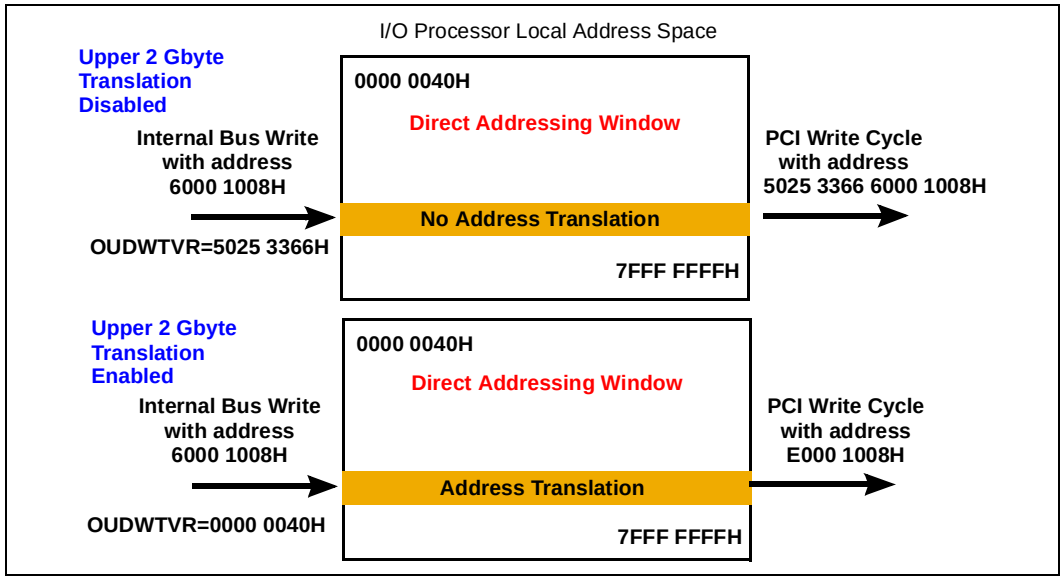


3.2.2.3 Direct Addressing Window - Single Address Cycle (SAC) Transactions

The second method used by outbound cycles from the Intel® XScale™ core to the PCI bus is the direct addressing window. This is a window of addresses in Intel® XScale™ core address space that act in the same manner as the outbound translation windows either without any translation or with the translation of address bit 31 only. This allows the Direct Addressing window to translate to different address ranges on the PCI bus (0000.0040H to 7FFF.FFFFH or 8000.0000H to FFFF.FFFFH). A Intel® XScale™ core read or write to an internal bus address within the direct addressing window initiates a read or write on the PCI bus with the same address (with the possible exception of address bit 31) as used on the internal bus. When the Outbound Upper 32-bit Direct Window Translate Value Register (OUDWTVR) is not all-zero, then a DAC transaction is generated on the PCI bus using the upper-32 bit address provided in OUDWTVR. Otherwise a SAC transaction is generated on the PCI bus. Figure 13 shows two examples of outbound writes that are through the direct addressing window.

Direct Addressing is limited to PCI memory read and write commands only. I/O cycles are not supported with direct addressing.

Figure 13. Direct Addressing Window



The internal bus side of the direct addressing window address range is fixed in the lower 2 Gbytes of the 80331 local address space (except for the first 32 Bytes which are reserved for the Reset and Exception vectors). Internal bus cycles with an address from 0000.0040H to 7FFF.FFFFH are forwarded to the PCI bus, when enabled. The following bits within the ATUCR affect direct addressing operation:

- ATUCR Direct Addressing Enable bit - when set, enables the direct addressing window. When clear, addresses within the direct addressing window are not forwarded to the PCI bus.
- ATUCR Direct Addressing Upper 2G Translation Enable - when set, the ATU forwards internal bus cycles with an address between 0000.0040H and 7FFF.FFFFH to the PCI bus with bit 31 of the address set (8000.0000H - FFFF.FFFFH). When clear, no translation occurs.

3.2.2.4 Outbound DMA Transactions

The ATU provides each DMA channel with its own dedicated decode window that claims all transactions generated by that channel that are destined for the PCI bus.

Depending on the contents of the PCI Upper Address field of a particular DMA descriptor, the ATU may initiate a SAC or a DAC transaction on the PCI bus in response to the DMA channel's request. As the DMA channel transfers data to the ATU, the lower 32-bits of the Internal Bus address are passed to the PCI bus unaltered (see Section 6.3.1, "Chain Descriptors" on page 351 for more details).

3.2.3 Outbound Write Transaction

An outbound write transaction is initiated by the Intel® XScale™ core[1] or by one of the DMAs and is targeted at a PCI target on the PCI bus. The outbound write address and write data are propagated from the 80331 internal bus to a PCI bus through the OWADQ and OWQ, respectively.

The ATUs internal bus target interface claims the write transaction and forwards the write data through to the targeted PCI bus. The data flow for an outbound write transaction on the internal bus is summarized in the following statements:

- For Single Address Cycles (SACs), ATU internal bus target interface latches the address from the internal bus into the OWADQ when that address is inside one of the outbound translate windows (see Section 3.5) and the OWQ is not full.
- For Dual Address Cycles (DACs), ATU internal bus target interface latches address from the internal bus into the OWADQ when the OWQ is not full and OWADQ is not full.
- Once outbound address is latched, internal bus target interface stores write data into the OWQ until the internal bus transaction completes or the reaches a buffer boundary. The initiator of the transaction is disconnected at an ADB when the transaction reaches a buffer boundary.
- When the OWADQ is full, the target interface signals a Retry on the internal bus to the outbound cycle initiator.
- When OWADQ latches the address and corresponding data is latched in a buffer in OWQ, the outbound cycle is enabled for transmission on the PCI Bus and PCI interface requests PCI bus.

The PCI interface is responsible for completing the outbound write transaction with the PCI address translated from the OWADQ and the data in the OWQ. The data flow for an outbound write transaction on the PCI bus is summarized in the following statements:

- ATU PCI interface requests PCI bus when completed internal bus transaction is in OWADQ and data associated with transfer in OWQ. Once bus is granted, PCI master interface writes PCI translated address from OWADQ to the PCI bus and wait for the transaction to be claimed.
- When Master Abort seen during address phase, transaction flushed and OWADQ/OWQ are cleared. Refer Section 3.7.3 for full details on PCI master abort conditions during outbound transactions.
- In the conventional PCI mode, once the PCI write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
  - The PCI target signals a Retry or Disconnect. The ATU PCI master attempts to reacquire the PCI bus to complete the write transaction.
  - The **GNT#** signal is deasserted and the master latency timer has expired. In this case, the master interface attempts to reacquire the PCI bus and complete the write transaction.
  - The PCI target signals a Target-Abort. In this case, the OWQ and OTQ are cleared and the transaction is aborted. The appropriate error bits are set as defined in Section 3.7.4.
  - The transaction terminates normally by transferring all data (full byte count) associated with it. The write address is removed from the OWADQ and the interface returns to idle.

1. For best performance, the user should designate the two Outbound Memory Windows (8000.0000H to 83FF.FFFFH, 8400.0000H to 87FF.FFFFH) as non-cacheable and bufferable from the Intel® XScale™ core. This assignment enables the Intel® XScale™ core to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in Table 78, "ATU Outbound Data Flow Ordering Rules" on page 168. In the event that the user requires strict ordering to be maintained, the user can either change the designation of this region of memory to be non-cacheable and non-bufferable or enforce the requirement in software.

- In the PCI-X mode, once the PCI memory write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
  - The PCI target signals a Retry or Single Data Phase Disconnect. The ATU PCI initiator attempts to reacquire the PCI bus to complete the write transaction.
  - reacquire the PCI bus to complete the write transaction.
  - The **GNT#** signal is deasserted and the master latency timer has expired. In this case, the master interface attempts to reacquire the PCI bus and complete the write transaction.
  - The PCI target signals a Target-Abort. In this case, the OWQ and OWADQ are cleared and the transaction is aborted. The appropriate error bits are set as defined in Section 3.7.4.
  - The transaction terminates normally with Satisfaction of Byte Count. The write address is removed from the OWADQ and the interface returns to idle.
- In the PCI-X mode, once the PCI I/O write transaction is claimed, the PCI interface transfers data from the OWQ to the PCI bus until one of the following is true:
  - The PCI target signals a Retry. The ATU PCI initiator attempts to reacquire the PCI bus to complete the write transaction.
  - The transaction terminates normally with Satisfaction of Byte Count or with Single Data Phase Disconnect. The write address is removed from the OWADQ and the interface returns to idle.
  - The PCI target signals a Target-Abort. In this case, the OWQ and OWADQ are cleared and the transaction is aborted. The appropriate error bits are set as defined in Section 3.7.4.
  - The transaction terminates with Split Response Termination. The write address is removed from the OWADQ and the interface returns to idle only when it receives the corresponding Split Completion Message.

  When a data parity error is encountered (**PERR#** detected), the master interface continues writing data to clear the queue.

  In the conventional PCI mode when the PCI target deasserts TRDY#, no action is taken by the ATU master other than inserting wait states.

3.2.4 Outbound Read Transaction

An outbound read transaction is initiated by the Intel® XScale™ core[1] or one of the DMAs and is targeted at a PCI slave on the PCI bus. The read transaction is propagated through the outbound transaction queue (OTQ) and read data is returned through the outbound read queue (ORQ).

The ATUs internal bus target interface claims the Memory Read Dword and Memory Read Block and Alias to Memory Read Block transaction and forwards the read request through to the PCI bus and returns the read data to the internal bus.

Data flow for an outbound read transaction on the internal bus is summarized as follows:

- For Single Address Cycles (SACs), the ATU internal bus interface latches the internal bus address when the address is inside an outbound address translation window (or direct addressing window, when enabled) and the OTQ is not full. For Dual Address Cycles (DACs), ATU internal bus target interface latches the address from the internal bus into the OTQ irrespective of the address. All read transactions are handled as split transactions. When the OTQ is full (previous outbound transactions in progress), the internal bus interface signals a Retry to the transaction initiator.
- When during the completion cycle on the PCI interface, a master abort is encountered, a flag is set and the ATU notifies the internal bus requester about the aborted transaction by generating a Split Completion Error Message (with message class=1h - PCI bus error and message index=00h - master abort). The OTQ is cleared of the transaction.
- When during the completion cycle on the PCI interface, a target abort is encountered, a flag is set and the ATU notifies the internal bus requester about the aborted transaction by generating a Split Completion Error Message (with message class=1h - PCI bus error and message index=01h - target abort). The OTQ is cleared of the transaction.
- Once the transaction completes on the PCI bus, the ATU generates a split completion transaction to return data to the internal bus requester.
- When operating in the PCI-X mode, ATU may receive a split completion error message when attempting to read data on the PCI bus. In this case, ATU notifies the internal bus requester about the error by forwarding the Split Completion Error Message from the PCI bus back to the Internal Requester. The OTQ is cleared of the transaction.

Data flow for an outbound read transaction on the PCI bus is summarized as follows:

- The ATU PCI interface requests the PCI bus when the head of the OTQ has at least one entry and the ordering rules are satisfied. Once the bus is granted, the PCI interface transfers the PCI translated address from the OTQ to the PCI bus and wait for the transaction to be claimed.
- When no **DEVSEL#** is asserted, a master abort is signaled. This is passed through to the internal bus target interface.
- When a target abort is signaled from the PCI target, the target abort is returned to the internal bus and the PCI interface returns to idle.
- When operating in the PCI-X mode the read transaction may terminate as a split response termination. Then the ATU receives data during the corresponding split completion transaction. When an error occurs, the ATU may receive a split completion error message.

1. For best performance, the user should designate the two Outbound Memory Windows (8000.0000H to 83FF.FFFFH, 8400.0000H to 87FF.FFFFH) as non-cacheable and bufferable from the Intel® XScale™ core. This assignment enables the Intel® XScale™ core to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in Table 78, "ATU Outbound Data Flow Ordering Rules" on page 168. In the event that the user requires strict ordering to be maintained, the user can either change the designation of this region of memory to be non-cacheable and non-bufferable or enforce the requirement in software.

3.2.5 Outbound Configuration Cycle Translation

The outbound ATU provides a port programming model for outbound configuration cycles.

Performing an outbound configuration cycle to the PCI bus involves up to two internal bus cycles:

1. Writing Outbound Configuration Cycle Address Register (OCCAR) with PCI address used during configuration cycle. See the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a for information regarding configuration address cycle formats. This IB bus cycle enables the transaction.
2. Writing or reading the Outbound Configuration Cycle Data Register (OCCDR). A read causes a configuration cycle read to the PCI bus with the address in the outbound configuration cycle address register. Note that the Internal Bus read is executed as a split transaction. Similarly, a write initiates a configuration cycle write to PCI with the write data from the second processor cycle. Configuration cycles are non-burst and restricted to a single DWORD cycle. [1]

Note: Bits 15:11 of the configuration cycle address for Type 0 configuration cycles are defined differently for Conventional versus PCI-X modes. When 80331 software programs the OCCAR to initiate a Type 0 configuration cycle, the OCCAR should always be loaded based on the PCI-X definition for the Type 0 configuration cycle address. When operating in Conventional mode, the 80331 clears bits 15:11 of the OCCAR prior to initiating an outbound Type 0 configuration cycle.

Note: During the attribute phase of a Type 0 configuration transaction, the Secondary Bus Number field (bits 7:0) is set equal to the Requester Bus Number (bits 15:8 of the "PCI-X Status Register - PX_SR" on page 279).

Master aborts during outbound configuration reads result in ATU generating a Split Completion Error Message (class=1h - bridge error and index=00h - master abort on PCI) on internal bus.

Target aborts during outbound configuration reads result in ATU generating a Split Completion Error Message (class=1h - bridge error and index=01h - target abort on PCI) on the internal bus.

Parity errors during outbound configuration reads result in ATU generating a Split Completion Error Message (class=2h - completer error and index=82h - parity error on PCI) on the internal bus.

Parity errors detected by target of an outbound configuration write may result in the ATU receiving either of the two Split Completion Write Data Parity Error Messages (with message class=2h -completer error and message index=01h - split write data parity error or with message class=1h - bridge error and message index=02h - write data parity error) on the PCI bus. When Parity Checking is enabled, the ATU sets error bits in the ATUSR and the PCIXSR. The Intel® XScale™ core is interrupted when the Split Completion Error and/or Master Data Parity interrupt(s) are unmasked.

When the Configuration Cycle Data Register is written, the data is latched and forwarded to the PCI bus with the internal target issuing a single data phase disconnect with data for 32-bits only. This cycle does not receive an **ACK64#** from the ATU and therefore is defined as 32-bit only.

Note that while the programming model uses the register interface for outbound configuration cycles, from a hardware standpoint, the address is entered into the OTQ (reads) or OWADQ (writes), configuration write data goes through the OWQ and configuration read data is returned in the ORQ.

Note: Outbound configuration cycle data registers are not physical registers. They are a 80331 memory mapped addresses used to initiate a transaction with the address in the associated address register.

1. The user should designate the memory region containing the OCCDR as non-cacheable and non-bufferable from the Intel® XScale™ core. This guarantees that all load/stores to the OCCDR is only of DWORD quantities. In event the user inadvertently issues a read to the OCCDR that crosses a DWORD address boundary, the ATU target aborts the transaction. All writes is terminated with a Single-Phase-Disconnect and only bytes 3:0 is relevant.

3.3 Messaging Unit

The Messaging Unit (MU) transfers data between the PCI system and the 80331 and notifies the respective system when new data arrives. The MU is described in Chapter 4, "Messaging Unit".

The PCI window for messaging transactions is always the *first* 4 Kbytes of the inbound translation window defined by the Inbound ATU Base Address Register 0 (IABAR0) and the Inbound ATU Limit Register 0 (IALR0).

All of the Messaging Unit errors are reported in the same manner as ATU errors. Error conditions and status can be found in the ATUSR and the ATUISR, see Section 3.7, "ATU Error Conditions" on page 174.

3.4 Expansion ROM Translation Unit

The inbound ATU supports one address range (defined by a base/limit register pair) used for the Expansion ROM. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details on Expansion ROM format and usage.

During a powerup sequence, initialization code from Expansion ROM is executed once by the host processor to initialize the associated device. The code can be discarded once executed. Expansion ROM registers are described in Section , and Section 3.10.29.

The inbound ATU supports an inbound Expansion ROM window which works like the inbound translation window. A read from the expansion ROM windows is forwarded to the internal bus. The address translation algorithm is the same as the inbound translation; see Section 3.2.1.1, "Inbound Address Translation" on page 139. As a PCI target, the Expansion ROM interface behaves as a standard ATU interface and is capable of returning a 64-bit access by the assertion of **ACK64#** in response to a 64-bit request.

The Expansion ROM unit uses the ATU inbound transaction queue and the inbound read data queue.

When operating in the conventional PCI mode, the address of the inbound delayed read cycle is entered into the ITQ queue and the delayed read completion data is returned in the IRQ. That is, inbound reads to the Expansion ROM window are handled as delayed transactions on the PCI bus.

When operating in the PCI-X mode, the address of the inbound read cycle is entered into the ITQ queue and the read completion data is returned in the IRQ. That is, inbound reads to the Expansion ROM window are handled as split transactions on the PCI bus. The internal bus initiator interface fills the IRQ read queue with the full byte count before generating the split completion transaction on the PCI bus. That is, the ATU generates a Memory Read Block transaction on the internal bus with byte count set to the byte count specified in the PCI read. The PBI (Flash Interface) terminates the read as a split read request and return data in

- either one or more 1024 byte split completion transactions when byte count is greater than or equal to 1024 bytes or
- one split completion with the full byte count when byte count is less than 1024 bytes.

Expansion ROM writes are not supported and result in a Target Abort.

3.5 ATU Queue Architecture

ATU operation and performance depends on the queueing mechanism implemented between the internal bus interface and PCI bus interface. As indicated in Figure 8, the ATU queue architecture consists of separate inbound and outbound queues. The function of each queue is described in the following sections.

3.5.1 Inbound Queues

The inbound data queues of the ATU support transactions initiated on a PCI bus and targeted at either 80331 local memory or a 80331 memory mapped register. Table 73 details the name and sizes of the ATU inbound data queues.

Table 73. Inbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
IWQ	Inbound Write Data Queue	4 KBytes (4*1KB)
IWADQ	Inbound Write Address Queue	4 Transaction Addresses
IRQ	Inbound Read Data Queue	4 KBytes (4*1KB)
IDWQ	Inbound Delayed Write address/data Queue	1 Transaction
ITQ	Inbound Transaction Queue	8 Addresses/Commands

3.5.1.1 Inbound Write Queue Structure

The ATU Inbound Write Queues consist of the inbound write data queue and the inbound write address queue. The inbound write data queue holds the data for memory write transactions moving from a PCI Bus to the internal bus and the address queue holds the corresponding address of the transactions in the data queues. The inbound write queue, IWQ, has a queue depth of 4 KBytes and moves write transactions from the PCI bus to the internal bus. The corresponding address queue, IWADQ, is capable of holding 4 address entries. The queue pair is capable of holding up to 4 memory write (or MWI when operating in the conventional PCI mode) transactions.

The following rules apply to the PCI bus interface and govern the acceptance of data into IWQ and address into the tail of the IWADQ:

- A memory write operation claimed by the target PCI interface on the PCI bus is accepted into the address and data queues when the IWADQ has at least one address entry available.
- When operating in the conventional PCI mode, when the IWQ reaches a full state while filling, a disconnect with data is signaled to the master of the transaction.
- When operating in the PCI-X mode, when the IWQ reaches a full state while filling, a disconnect at next ADB is signaled to the master of that transaction.

Memory write transactions are drained from the head of the queue when the master interface has acquired bus ownership and transaction ordering and priority have been satisfied (see Section 3.5.3). A memory write transaction is considered drained from the queue when the entire amount of data entered on the PCI bus has been accepted by the internal bus target. Error conditions resulting in the cancellation of a write transaction only flush the transaction at the head of the data and address queue. All other transactions within the queues are considered still valid.

3.5.1.2 Inbound Read Queue Structure

The inbound read queues are responsible for retrieving data from local memory and returning it to the PCI bus in response to a read transaction initiated from a PCI master. When operating in the conventional PCI mode, reads are handled as delayed transactions. When operating in the PCI-X mode reads are handled as split transactions.The address of the transactions are held in the ITQ. Up to 8 read requests can be stored in the ITQ. The read data is returned through IRQ.

When operating in the conventional PCI mode, the IRQ holds data from four PCI bus read transactions. The read request cycle on PCI latches the read command and the address into the ITQ when the cycle is first initiated by the PCI master. The ATU internal bus initiator interface takes the translated address and the command and performs a read on the internal bus. Reads can be any of the PCI memory read command types using the ATU inbound translation or an inbound configuration read using the specific configuration cycle translation. The data from the read on the internal bus is stored in the IRQ until the PCI master initiates a read cycle that matches the initial request cycle in both command and address. Any data left in an IRQ after the delivery of a completion cycle on PCI is flushed. This is possible since all internal bus memory is considered prefetchable with no read side effects.

When operating in the PCI-X mode, the IRQ may hold data from up to four PCI bus read transactions. The read request cycle on PCI latches the read command and the address into the ITQ when the cycle is first initiated by the PCI master. The ATU internal bus initiator interface takes the translated address and the command and performs a read on the internal bus. Reads can be any of the PCI memory read command types using the ATU inbound translation or an inbound configuration read using the specific configuration cycle translation. Once read data is available in the IRQ, the ATU generates one or more split completions to return read data to the PCI requester.

When operating in the conventional PCI mode, the exact amount of data (byte count) read by the master state machine on the internal bus interface depends upon the read command used and how much data the Internal Bus target device delivers. Table 74 shows the amount of data attempted to be read for the different memory read commands for the ATU, when operating in the conventional PCI mode.

Internal bus error conditions override all prefetch amounts. i.e. a master-abort and target-abort conditions.

Table 74. Inbound Read Prefetch Data Sizes

PCI Read Command	Prefetch Size (Bytes)
Memory Read	4 to 32
Memory Read Line	4 to 128
Memory Read Multiple	4 to 1024

3.5.1.3 Inbound Delayed Write Queue

The IDWQ is used specifically for inbound configuration write cycles to the ATU. I/O Write transactions are not accepted by the ATU and result in a Master Abort.

The IDWQ contains both the address and data of a configuration write cycle. When operating in the conventional PCI mode, the configuration writes are handled as delayed writes. When operating in the PCI-X mode, the configuration writes are handled as split writes.When the write cycle is initiated on the PCI bus, the address and data are entered into the 8 byte queue, and forwarded to the internal bus. The transaction is forwarded to the internal bus once transaction ordering has been satisfied. The status of the transaction (normal completion) is maintained in the IDWQ for return to the PCI master on the initiating bus. When operating in the PCI-X mode, a write completion message is generated by the ATU to indicate the successful execution of the configuration write transaction.

The IDWQ can only hold 32-bits of data and should never be accessed with **REQ64#** asserted per the *PCI Local Bus Specification*, Revision 2.3 which states that "only memory transactions support 64-bit data transfers". In addition, the cycle should always return only 32-bits of data on the internal bus.

3.5.1.4 Inbound Transaction Queues Command Translation Summary

Table 75. PCI to Internal Bus Command Translation for All Inbound Transactions

PCI Command	Conventional Mode	PCI-X Mode	Internal Bus Command
Memory Write	✔	✔	Memory Write
Memory Write and Invalidate	✔		Memory Write Block
Memory Write Block		✔	Memory Write Block
Alias to Memory Write Block		✔	Memory Write Block
Memory Read	✔		Memory Read Block
Memory Read Line	✔		Memory Read Block
Memory Read Multiple	✔		Memory Read Block
Memory Read Block		✔	Memory Read Block
Alias to Memory Read Block		✔	Memory Read Block
Memory Read DWORD		✔	Memory Read DWORD
Configuration Read	✔	✔	Configuration Read
Configuration Write	✔	✔	Configuration Write
Split Read Completion		✔	Split Read Completion
Split Write Completion		✔	None
Split Completion Message		✔	Split Completion Message
All Other Commands Not Claimed by the ATU	✔	✔	N/A

3.5.2 Outbound Queues

The outbound queues of the ATU are used to hold read and write transactions from the core processor directed at the PCI bus. Each ATU outbound queue structure has a separate read queue, write queue, and address queue. Table 76 contains information about ATU outbound queues.

Table 76. Outbound Queues

Queue Mnemonic	Queue Name	Queue Size (Bytes)
OWQ	Outbound Write Data Queue	4 KBytes (4*1024B)
OWADQ	Outbound Write Address Queue	4 Transaction Addresses
ORQ	Outbound Read Data Queue	2 or 4 KBytes (4* 512B or 4*1024B)[a]
OTQ	Outbound Transaction Queue	8 Addresses/Commands

a. The ORQ can be throttled between 2 Kbytes or 4 Kbytes depending on the setting of the Maximum Memory Read Byte Count (MMRBC) field of the PCI-X Command register (see Section 3.10.60, "PCI-X Command Register - PX_CMD" on page 278). When the MMRBC is set to 512 bytes (default value), the ORQ is only capable of handling 2 Kbytes of SRC data, otherwise, the ORQ handles 4 Kbytes of SRC data.

The outbound queues are capable of holding outbound memory read, memory write, I/O read, and I/O write transactions. The type of transaction used is defined by the internal bus address and the command used on the internal bus. See Section 3.2.2 and Section 3.2.3 for details on outbound address translation.

When an internal bus agent initiates an outbound write transaction, the address is entered into the OWADQ (when not full). The data from the internal bus write is then entered into the OWQ and the transaction is forwarded to the PCI bus. When the write completes (or an error occurs), the address is flushed from the OWADQ. Data is flushed only for the master abort or target abort cases.

For outbound reads, the address is entered into the OTQ (when not full) and a split response termination is signaled to the requester on the internal bus. Read data is fetched and returned to the requester on the internal bus.

3.5.3 Transaction Ordering

Because the ATU can process multiple transactions, they must maintain proper ordering to avoid deadlock conditions and improve throughput. The ATU transaction ordering rules used by the 80331 are listed in Table 77 for the inbound direction and Table 78 on page 168 for the outbound direction. The tables are based on the direction the transaction is moving, i.e. the data for outbound delayed read moves in the same direction as the data for an inbound write or the address/command for an inbound read. When operating in the PCI-X mode, the ATU ignores the Relaxed Ordering Attribute.

Note: Outbound Non-Posted Writes are the result of Internal Bus Memory writes that are claimed by either the I/O translation window or the Outbound Configuration Cycle Data Register - OCCDR. Though these write requests arrive on the PCI bus as non-posted write requests, it is important to note that from the Intel® XScale™ core point of view, these internal bus memory write requests are posted into the Outbound ATU transaction queue. Furthermore, in PCI-X mode non-posted write requests have the potential to be split. Thus, even though a split write completion may be returned to the ATU on the PCI bus for a given outbound non-posted write request, the split write completion is not passed back through to the internal bus. Additionally, strong ordering between outbound memory (posted) write requests and outbound non-posted write requests is **not** maintained as indicated in Table 78 on page 168.

For best performance, the user should designate the two Outbound Memory Windows (8000.0000H to 83FF.FFFFH, 8400.0000H to 87FF.FFFFH) as non-cacheable and bufferable from the Intel® XScale™ core. This assignment enables the Intel® XScale™ core to issue multiple outstanding transactions to the Outbound Memory Windows, thereby, taking full advantage of the ATU outbound queue architecture. However, the user needs to be aware that the Outbound ATU queue architecture does not maintain strict ordering between read and write requests as described in Table 78, "ATU Outbound Data Flow Ordering Rules" on page 168. In the event that the user requires strict ordering to be maintained, the user can either change the designation of this region of memory to be non-cacheable and non-bufferable or enforce the requirement in software.

Table 77. ATU Inbound Data Flow Ordering Rules

Row Pass Column?[a]	ATU Inbound Writes	Inbound Delayed Read Request (PCI mode)	Inbound Split Read Request (PCI-X mode)	Inbound Configu-ration Write Request	Outbound Split Read Completion
ATU Inbound Writes	No	Yes	Yes	Yes	Yes
Inbound Delayed Read Request (PCI mode)	No	No	NA	No	Yes
Inbound Split Read Request (PCI-X mode)	No	NA	No	No	Yes
Inbound Configuration Write Request	No	No	No	NA	Yes
Outbound Split Read Completion	No	Yes	Yes	Yes	Yes

a. Outbound Non-Posted Write Completions are not included in this table since these transactions are never passed back to the Internal Bus Requester (Intel® XScale™ core). The reason is that from the Intel® XScale™ core point of view, these write requests are posted into the Outbound ATU transaction queue.

Table 78. ATU Outbound Data Flow Ordering Rules

Row Pass Column?	Outbound Write		Outbound Read Request	Inbound Read Completion		Inbound Write Completion	
	Posted	Non-Posted		DRC	SRC	DWC[a]	SWC[b]
Outbound Posted Write Request	No	Yes	Yes	Yes	Yes	Yes	Yes
Outbound Non-posted[c] Write Request	No	No	No	Yes	Yes	Yes	Yes
Outbound Read Request	No	No	No	Yes	Yes	Yes	Yes
Inbound Delayed Read Completion (DRC)	No	Yes	Yes	Yes	NA	Yes	NA
Inbound Split Read Completion (SRC)	No	Yes	Yes	NA	Yes	NA	No
Inbound Delayed Write Completion (DWC)	No	Yes	Yes	Yes	NA	NA	NA
Inbound Split Write Completion (SWC)	No	Yes	Yes	NA	Yes	NA	NA

a. Since the Inbound DWR transaction queue is one-deep, the passing rule associated with DWC vs. DWC is moot (i.e., NA).
b. Since the Inbound SWR transaction queue is one-deep, the passing rule associated with SWC vs. SWC is moot (i.e., NA).
c. Outbound Non-posted writes include I/O writes and Configuration writes.

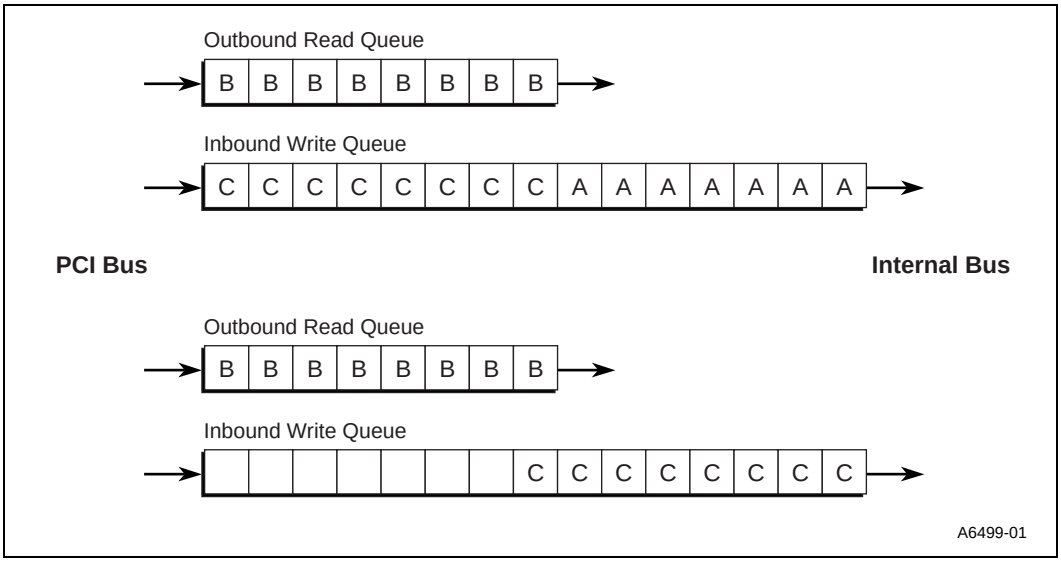
Definitions of the terms used in Table 77 and Table 78 are as follows. PCI terms are noted in parenthesis:

- Inbound Write (PMW) - Data from a write cycle initiated on PCI and targeted at the internal bus. Note that the address is in a separate transaction queue and is not referenced. Inbound writes can also come in through the Messaging Unit which is part of the ATU.
- Inbound Read Request (DRR-PCI mode and SRR-PCI-X mode) - address information from a read transactions retried or split on the PCI bus. is mastered on the internal bus to retrieve data for the Inbound Read Completion.
- Inbound Configuration Write Request - (DWR- PCI mode and SWR - PCI-X mode) - The address and data associated with a configuration write transaction from PCI and targeted at the ATU PCI configuration address space. Once completed on the internal bus, creates an Inbound Configuration Write Completion.
- Outbound Read Completion (SRC) - The data read on PCI in the process of being returned to the internal bus. This data is the completion cycle that results from an Outbound Read Request.
- Outbound Write (PMW) - The address and data from a write initiated on the internal bus and eventually completing on the PCI bus.
- Outbound Read Request (SRR) - The address/command of a split read cycle initiated on the internal bus. The read data is returned in the Outbound Read Completion cycle.
- Inbound Read Completion (DRC-PCI mode and SRC-PCI-X mode) - The data read on the internal bus in the process of being returned to the PCI bus. This data is the completion cycle for an Inbound Read Request.
- Inbound Configuration Write Completion (DWC-PCI mode and SWC-PCI-X mode) - The status of an inbound write configuration cycle traveling from the internal bus back towards the PCI bus.

These transaction ordering rules define the way in which data moves in both directions through the ATU. In Table 77 and Table 78 a **NO** response in a box means that based on ordering rules, the current transaction (the row) can not pass the previous transaction (the column) under any circumstance. A **Yes** response in the box means that the current transaction is *allowed* to pass the previous transaction but is not required to, based on whether a consistent view of data or prevention of deadlocks is needed.

In the case of inbound write operations, multiple transactions may exist within the IWQ and the corresponding IWADQ at any point in time. The ordering of these transactions is based on a time stamp basis. Transactions entering the queue are stamped with a relative time in relation to all other transactions moving in a similar direction.

Example 1. Inbound Queue Completion



In Example 1 on page 169, the inbound write and outbound read queues of the ATU are shown. In this example, transaction A entered the write queue at **Time 0**. Next, the ATU entered read data into the outbound read queue at **Time 1** (Transaction B). Finally, before the previous transactions could be cleared, another inbound write, Transaction C, was entered into the IWQ. The ordering in Table 77 states that nothing can pass an inbound write and therefore Transaction A must complete on the internal bus before Transaction B since an outbound read completion can not pass an inbound write. Also, Transaction A must complete before Transaction C since an inbound write can not pass another inbound write. Once Transaction A completes, Transaction C moves to the head of the IWQ. The two transactions at the head of the queues moving data in an inbound direction are now Transaction C, an inbound write, and Transaction B, an outbound read completion. Ordering states that an inbound write may pass an outbound read completion. This means that the arbitration mechanism now takes over to decide which completes. Note that ordering enforced the completion of Transaction A but arbitration dictated the completion of Transactions B and C.

The first action performed to determine which transaction is allowed to proceed (either inbound or outbound) is to apply the rules of ordering as defined in Table 77 and Table 78. Any box marked **No** must be satisfied first. For example, when an inbound read request is in ITQ and it was latched *after* the data in the IDWQ arrived (this is a configuration write), then ordering states that an Inbound Read Request may not pass an Inbound Configuration Write Request. Therefore, the Inbound Configuration Write Request must be cleared out of IDWQ before the Inbound Read Request is attempted on the internal bus. Once transaction ordering is satisfied, the boxes marked **Yes** are now resolved.

3.5.3.1 Transaction Ordering Summary

Table 79 and Table 80, define transaction ordering in relation to token assignment of the priority mechanism (this is discussed in Section 3.5.3). These tables are read as follows:

1. As the transaction enters the head of the respective queue, the question in column 2 is asked.
2. When all the answers in column 3 for a given transaction type assigns a token to the transaction at the head of the queue, a token is assigned. Otherwise, no token is assigned signifying that transaction ordering must first be satisfied. Any transaction with a token may be initiated on the bus.

Table 79. Inbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Inbound Write in IWQ	Is there an Inbound Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Read Request in ITQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Configuration Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Configuration Write Request in IDWQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Read Completion in ORQ	Is there an Inbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token

Table 80. Outbound Transaction Ordering Summary

Transaction at Head of Queue	Question	Answer	Action
Outbound Write in OWQ	Is there an Outbound Write Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Outbound Read Request in OTQ	Is there an Outbound Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Outbound Read Request with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
Inbound Configuration Write Completion in IRQ	Is there an Outbound Posted Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token
Inbound Read Completion in IRQ	Is there an Outbound Posted Write with an earlier time stamp?	Yes	Do Not Assign Token Allow previous Transaction to Complete
		No	Assign Token
	Is there an Inbound Read Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token
	Is there a Configuration Write Completion with an earlier time stamp?	Yes	Do Not Assign Token and allow previous Transaction to Complete when in PCI-X Mode. Assign Token when in Conventional Mode
		No	Assign Token

3.6 Private Device Control

Private devices are hidden from PCI configuration software but are accessible from the Address Translation Unit. To include private devices on the secondary PCI interface, 80331 bridge unit and ATU must be configured properly prior to system initialization following **P_RST#** deassertion.

To configure the 80331 to control private devices, the PCI-to-PCI Bridge must be configured to:

- Enable private Type 0 commands on the Secondary Interface
- Enable the private memory space on the Secondary Interface

These capabilities are enabled through the reset straps **PRIVDEV** and **PRIVMEM** respectively. The ATU provides status information for these reset straps in the Table 125, "PCI Configuration and Status Register - PCSR" on page 253.

The ATU is internally mapped to PCI Device Number 14 (0EH) with AD30 used as the ATU IDSEL input. This mapping also has the ATU interrupt wired to the **P_INTC#** output pin.

3.6.1 Private Type 0 Commands on the Secondary Interface

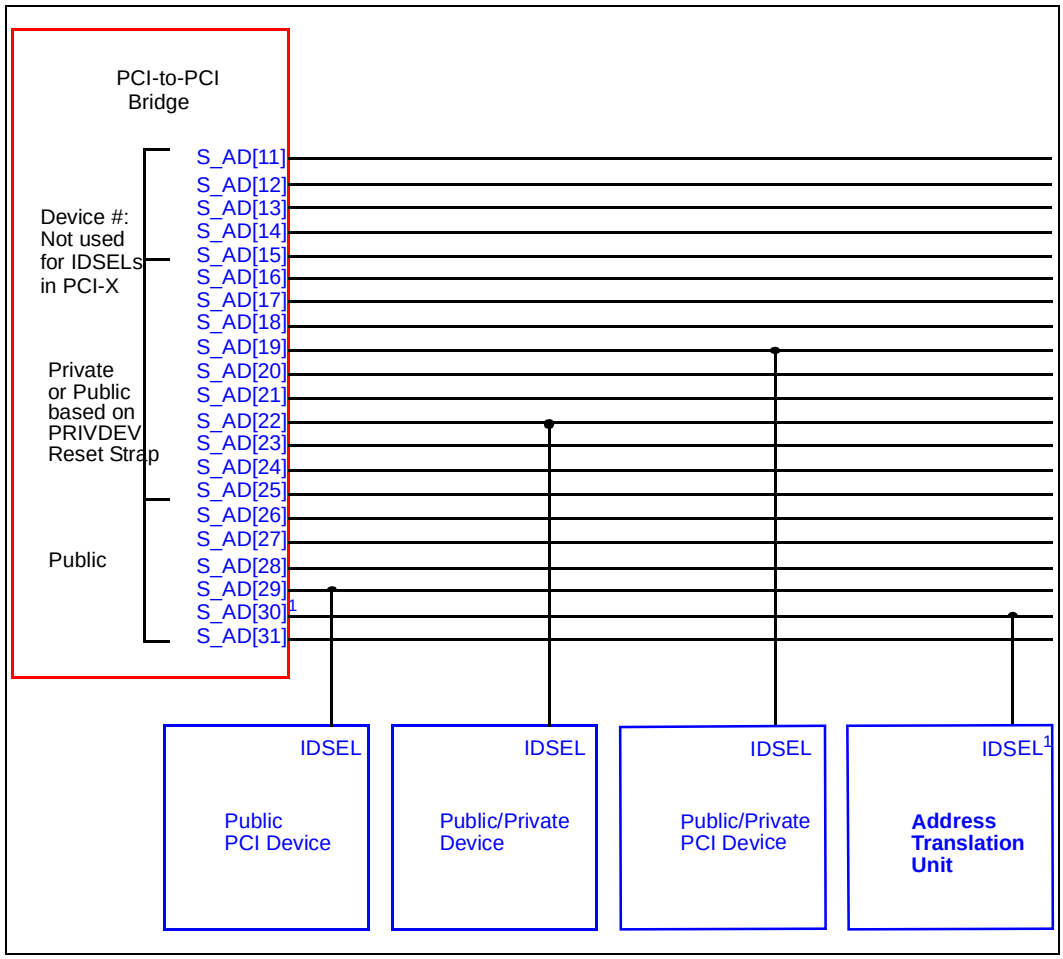
Type 0 configuration reads and write commands can be generated by the Address Translation Unit of the 80331. These Type 0 configuration commands are required to configure private PCI devices on the Secondary bus which are in private PCI address space. These commands are initiated by the Address Translation Unit and not by Type 1 commands on the Primary bus. Any device mapped into this private address space *is not* part of the standard Secondary PCI address space and therefore is not configured by the system host processor. These devices are hidden from PCI configuration software but are accessible from the Address Translation Unit.

In Type 0 commands on the Secondary interface, **S_AD[31:11]** are used to select the **IDSEL** input of the target device. In Type 1 to Type 0 conversions, **P_AD[15:11]** are decoded to assert a unique address line from **S_AD[31:16]** on the Secondary interface.

The bridge can be configured to block 10 address lines during Configuration transactions from the Primary interface. Secondary address bits **S_AD[25:16]** are the address lines that can be masked by the bridge private device control.

Figure 14 shows an example of connecting PCI Address/Data lines to **IDSEL** inputs of PCI devices and private PCI devices.

Figure 14. Private Device Example



1: AD30 is dedicated to the 80331 Address Translation Unit IDSEL input, and must not be used for an external Secondary PCI device.

3.6.2 Private Memory Space

When Private Memory Space Enable in the Bridge SDER (see Section 2.5.2.10, "Secondary Decode Enable Register - SDER" on page 116) is set, the bridge overrides its secondary inverse decode logic and not forward upstream any secondary bus initiated DAC Memory transactions with address in the upper half of 64-bit address space (i.e., AD(63)=1b). This establishes a private memory space for secondary bus usage, independent of bridge downstream forwarding configuration.

3.7 ATU Error Conditions

PCI and internal bus error conditions cause ATU state machines to exit normal operation and return to idle states. In addition, status bits are set to inform error handling code of exact cause of error condition. All of the Messaging Unit errors are reported in the same manner as ATU errors. Error conditions and status can be found in the ATUSR. The basic flow for a PCI error is as follows:

- Set the bit in the ATU Status Register which corresponds to the error condition (master abort, target abort, etc.)
- Set the bit in the ATU Interrupt Status Register which corresponds to the error condition (master abort, target abort, etc.). This function is maskable for all PCI error conditions.
- The setting of the bit in the ATU Interrupt Status Register results in an interrupt being driven to the Intel® XScale™ core.

Error conditions on one side of the ATU are generally propagated to the other side of the ATU and have different effects depending on the error. Error conditions and their effects are described in the following sections.

PCI bus error conditions and the action taken on the bus are defined within the *PCI Local Bus Specification*, Revision 2.3, and the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a. The ATU adheres to the error conditions defined within the PCI specification for both requester and target operation. Error conditions on the internal bus are caused by an ECC error from the Memory Controller (see Section 8.5, "Interrupts/Error Conditions" on page 493 for details on memory controller error conditions) or by incorrect addressing resulting in an internal master abort. All actions on the PCI Bus for error situations are dependent on the error control bits found in the ATU Command Register (see Section 3.10.3, "ATU Command Register - ATUCMD" on page 214) for both Conventional and PCI-X modes. For PCI-X mode, the error response is also dependent on an error control bit in the PCI-X Command Register (see Section 3.10.60, "PCI-X Command Register - PX_CMD" on page 278).

The following sections detail all ATU error conditions on the PCI and 80331 internal bus, action taken on these conditions, and status and control bits associated with error handling.

3.7.1 Address and Attribute Parity Errors on the PCI Interface

The ATUs must detect and report address and attribute (PCI-X mode only) parity errors for transactions on the PCI bus. When an address or attribute parity error occurs on the PCI interface of the ATU, the 80331 performs the following actions based on the constraints specified:

- In Conventional mode, when the Parity Error Response bit in ATUCMD is set, the ATU ignores (Master-Abort) the transaction by not asserting **DEVSEL#**. When clear, the transaction proceeds normally.
- In PCI-X mode, when the Parity Error Response bit in ATUCMD is set, the ATU completes the transaction on the PCI bus as though no error had occurred, but the request or completion is not forwarded to the internal bus. When clear, the transaction proceeds normally.
- Assert **SERR#** when the **SERR#** Enable bit and the Parity Error Response bit in the ATUCMD are set. When the ATU asserts **SERR#,** additional actions is taken:
  - Set the **SERR#** Asserted bit in the ATUSR
  - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
  - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR (see Section 3.10.39, "ATU Configuration Register - ATUCR" on page 252) is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- Set the Detected Address or Attribute Parity Error bit in PCSR (PCI Configuration and Status Register).
- Set the Detected Parity Error bit in the ATUSR. When the ATU sets the Detected Parity Error bit, additional actions is taken:
  - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

Note: The Detected Parity Error bit with its associated interrupt along with the Detected Address or Attribute Parity Error bit provides software with the ability to distinguish between an Address or Attribute Parity error versus a Data Parity Error during a Detected Parity error interrupt.

3.7.2 Data Parity Errors on the PCI Interface

Two kinds of data parity errors can occur on the PCI interface: errors as an initiator and errors as a target.

Errors encountered as an initiator:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completions
- Inbound Configuration Write Completion Messages

As an initiator, the ATU provides an error response for data parity errors on outbound reads, and data parity errors occurring at the target for outbound writes. However, there is no error response for data parity errors on inbound configuration write completion messages and inbound read completions.

Errors encountered as a target:

- Inbound Read Request (Immediate Data Transfer)
- Inbound Write Request
- Outbound Read Completions
- Outbound Split (I/O or Configuration) Write Data Parity Error Messages
- Inbound Configuration Write
- Split Completion Messages

As a target, the ATU provides an error response for data parity errors on inbound writes, outbound read completions, outbound split write data parity error messages, inbound configuration writes, and split completion messages. However, there is no error response for data parity errors on inbound reads.

3.7.2.1 Outbound Read Request Data Parity Errors

3.7.2.1.1 Immediate Data Transfer

As an initiator, the ATU may encounter this error condition in Conventional or PCI-X mode when the target transfers data immediately rather than signalling a Retry[1] (Conventional Delayed Read Request) or a Split Response Termination (PCI-X Split Read Request).

Data parity errors occurring during read operations initiated by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the data phase in which the data parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions is taken:
  - The Master Parity Error bit in the ATUSR is set.
  - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - When the ATU is operating in the PCI-X mode, the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
  - A completion error message (with message class=2h - completer error and message index=82h - PCI bus read parity error) is generated on the internal bus of the 80331.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

1. Retry terminations may also be signaled in PCI-X mode when the target is too busy to handle the current request. However, this is not the same as a Delayed Read Request in Conventional PCI mode since the requester is not required or expected by the target to return with the same read request.

3.7.2.1.2 Split Response Termination

As an initiator, the ATU may encounter this error condition in PCI-X mode when the target signals a Split Response Termination.

Parity errors occurring during Split Response Terminations of Read Requests by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the Split Response Termination in which the parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions is taken:
  - — The Master Parity Error bit in the ATUSR is set.
  - — When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - — When the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - — When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

The Outbound Read Request remains enqueued in the ATU as the completer is initiating completion transactions that are associated with this request.

3.7.2.2 Outbound Write Request Data Parity Errors

3.7.2.2.1 Outbound Writes that are not MSI (Message Signaled Interrupts)

As an initiator, the ATU may encounter this error condition when operating in either the Conventional or PCI-X modes.

Data parity errors occurring during write operations initiated by the ATU may record the assertion of **PERR#** from the target on the PCI Bus. In PCI-X mode, this includes the assertion of **PERR#** from the target on the PCI Bus following the split response termination of a non-posted write request. When an error occurs, the ATUs continues writing data to the target to clear the OWQ of the current outbound write transaction. Specifically, the following actions with the given constraints are taken by the ATU:

- When **PERR#** is sampled active and the Parity Error Response bit in the ATUCMD is set, set the Master Parity Error bit in the ATUSR. When the Parity Error Response bit in the ATUCMD is clear, no action is taken. When the Master Parity Error bit in the ATUSR is set, additional actions is taken:
  - — When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - — When the ATU is operating in the PCI-X mode, the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action

Outbound write parity errors, does not result in a master completion. In addition, when the target terminates the transaction (disconnect), the ATU master must reinitiate the transaction to clear the data from the OWQ.

3.7.2.2.2 MSI Outbound Writes

As an initiator, the ATU may encounter this error condition when operating in either the Conventional or PCI-X modes.

Data parity errors occurring during MSI write operations initiated by the ATU may record the assertion of **PERR#** from the target on the PCI Bus. When an error occurs, the ATU completes the transaction normally. Then, the following actions with the given constraints are taken by the ATU:

- When **PERR#** is sampled active and the Parity Error Response bit in the ATUCMD is set, set the Master Parity Error bit in the ATUSR. When the Parity Error Response bit in the ATUCMD is clear, no action is taken. When the Master Parity Error bit in the ATUSR is set, additional actions is taken:
  - — When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - — When the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.

3.7.2.3 Inbound Read Completions Data Parity Errors

As an initiator, the ATU may encounter this error condition when operating in the PCI-X mode.

When as the completer of a Split Read Request the ATU observes **PERR#** assertion during the split completion transaction**,** the ATU attempts to complete the transaction normally and no further action is taken.

3.7.2.4 Inbound Configuration Write Completion Message Data Parity Errors

As an initiator, the ATU may encounter this error condition when operating in the PCI-X mode.

When as the completer of a Configuration (Split) Write Request the ATU observes **PERR#** assertion during the split completion transaction**,** the ATU attempts to complete the transaction normally and no further action is taken.

3.7.2.5 Inbound Read Request Data Parity Errors

3.7.2.5.1 Immediate Data Transfer

As a target, the ATU may encounter this error when operating in the Conventional PCI or PCI-X modes.

Inbound read data parity errors occur when read data delivered from the IRQ is detected as having bad parity by the initiator of the transaction who is receiving the data. The initiator may optionally report the error to the system by asserting **PERR#**. As a target device in this scenario, no action is required and no error bits are set.

3.7.2.5.2 Split Response Termination

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Inbound read data parity errors occur during the Split Response Termination. The initiator may optionally report the error to the system by asserting **PERR#**. As a target device in this scenario, no action is required and no error bits are set.

3.7.2.6 Inbound Write Request Data Parity Errors

As a target, the ATU may encounter this error when operating in the Conventional or PCI-X modes.

Data parity errors occurring during write operations received by the ATU may assert **PERR#** on the PCI Bus. When an error occurs, the ATU continues accepting data until the initiator of the write transaction completes or a queue fill condition is reached. Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the data phase in which the data parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - — When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

3.7.2.7 Outbound Read Completion Data Parity Errors

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Data parity errors occurring during read completion transactions that are claimed by the ATU are recorded, **PERR#** is asserted (when enabled) and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted two clocks cycles (three clock cycles when operating in the PCI-X mode) following the data phase in which the data parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions is taken:
  - — The Master Parity Error bit in the ATUSR is set.
  - — When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - — When the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
  - — A completion error message (with message class=2h - completer error and message index=82h - PCI bus read parity error) is generated on the internal bus of the 80331.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - — When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

3.7.2.8 Outbound Split Write Data Parity Error Message

The ATU claims a Split Completion Error Message that indicates a data parity error has occurred on one of the ATUs non-posted (I/O or Configuration) write requests (Message Class = 2h, Message Index = 01h -- Split Write Data Parity Error or Message Class = 1h, Message Index = 02h --Write Data Parity Error).

When the ATU receives a Split Completion Error Message indicating a data parity error for an outstanding Configuration or I/O (split) write request, the error is recorded, and **SERR#** is asserted (when enabled). Specifically, the following actions with the given constraints are taken by the ATU:

- When the Parity Error Response bit in the ATUCMD is set, these actions is taken:
  - The Master Parity Error bit in the ATUSR is set.
  - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - When the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- The Received Split Completion Error Message bit in the PCIXSR is set (based on bit 30 of the completer attributes being set). When the ATU sets this bit, additional actions is taken:
  - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- The transaction associated with the Split Completion Error Message is discarded.

3.7.2.9 Inbound Configuration Write Request

As a target, the ATU may encounter this error when operating in the Conventional or PCI-X modes.

3.7.2.9.1 Conventional PCI Mode

To allow for correct data parity calculations for delayed write transactions, the ATU delays the assertion of STOP# (signalling a Retry) until **PAR** is driven by the master. A parity error during a delayed write transaction (inbound configuration write cycle) can occur in any of the following parts of the transactions:

- During the initial Delayed Write Request cycle on the PCI bus when the ATU latches the address/command and data for delayed delivery to the internal configuration register.
- During the Delayed Write Completion cycle on the PCI bus when the ATU delivers the status of the operation back to the original master.

The 80331 ATU PCI interface has the following responses to a delayed write parity error for inbound transactions during Delayed Write Request cycles with the given constraints:

- When the Parity Error Response bit in the ATUCMD is set, the ATU asserts TRDY# (disconnects with data) and two clock cycles later asserts **PERR#** notifying the initiator of the parity error. The delayed write cycle is not enqueued and forwarded to the internal bus.

  When the Parity Error Response bit in the ATUCMD is cleared, the ATU retries the transaction by asserting STOP# and enqueues the Delayed Write Request cycle to be forwarded to the internal bus. **PERR#** is not asserted.

- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:

  — When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

For the original write transaction to be completed, the initiator retries the transaction on the PCI bus and the ATU returns the status from the internal bus, completing the transaction.

For the Delayed Write Completion transaction on the PCI bus where a data parity error occurs and therefore does not agree with the status being returned from the internal bus (i.e. status being returned is normal completion) the ATU performs the following actions with the given constraints:

- When the Parity Error Response Bit is set in the ATUCMD, the ATU asserts TRDY# (disconnects with data) and two clocks later asserts **PERR#**. The Delayed Completion cycle in the IDWQ remains since the data of retried command did not match the data within the queue.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:

  — When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

3.7.2.9.2 PCI-X Mode

Data parity errors occurring during configuration write operations received by the ATU may cause **PERR#** assertion and delivery of a Split Completion Error Message on the PCI Bus. When an error occurs, the ATU accepts the write data and complete with a Split Response Termination. Specifically, the following actions with the given constraints are then taken by the ATU:

- When the Parity Error Response bit in the ATUCMD is set, **PERR#** is asserted three clocks cycles following the Split Response Termination in which the data parity error is detected on the bus. When the ATU asserts **PERR#**, additional actions is taken:
  - A Split Write Data Parity Error message (with message class=2h - completer error and message index=01h - Split Write Data Parity Error) is initiated by the ATU on the PCI bus that addresses the requester of the configuration write.
  - When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.
  - The Split Write Request is not enqueued and forwarded to the internal bus.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.

3.7.2.10 Split Completion Messages

As a target, the ATU may encounter this error when operating in the PCI-X mode.

Data parity errors occurring during Split Completion Messages claimed by the ATU may assert **PERR#** (when enabled) or **SERR# (when** enabled) on the PCI Bus. When an error occurs, the ATU accepts the data and complete normally. Specifically, the following actions with the given constraints are taken by the ATU:

- **PERR#** is asserted three clocks cycles following the data phase in which the data parity error is detected on the bus. This is only done when the Parity Error Response bit in the ATUCMD is set. When the ATU asserts **PERR#**, additional actions is taken:
  - The Master Parity Error bit in the ATUSR is set.
  - When the ATU PCI Master Parity Error Interrupt Mask Bit in the ATUIMR is clear, set the PCI Master Parity Error bit in the ATUISR. When set, no action.
  - When the **SERR#** Enable bit in the ATUCMD is set, and the Data Parity Error Recover Enable bit in the PCIXCMD register is clear, assert **SERR#**; otherwise no action is taken. When the ATU asserts **SERR#,** additional actions is taken:

    Set the **SERR#** Asserted bit in the ATUSR.

    When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.

    When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- When the SCE bit (Split Completion Error -- bit 30 of the Completer Attributes) is set during the Attribute phase, the Received Split Completion Error Message bit in the PCIXSR is set. When the ATU sets this bit, additional actions is taken:
  - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- The Detected Parity Error bit in the ATUSR is set. When the ATU sets this bit, additional actions is taken:
  - When the ATU Detected Parity Error Interrupt Mask bit in the ATUIMR is clear, set the Detected Parity Error bit in the ATUISR. When set, no action.
- The transaction associated with the Split Completion Message is discarded.
- When the discarded transaction was a read, a completion error message (with message class=2h - completer error and message index=82h - PCI bus read parity error) is generated on the internal bus of the 80331.

3.7.3 Master Aborts on the PCI Interface

As an initiator on the PCI bus, the ATU can encounter master abort conditions during:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completion
- Inbound Configuration Write Completion Message

As a target, the ATU PCI interface is capable of signaling a master abort case during:

- Address Parity Error (Conventional Mode)
- Inbound Read Request (PCI-X Mode)

3.7.3.1 Master Aborts for Outbound Read or Write Request

This error may be encountered in both the Conventional and the PCI-X modes. For an Outbound transaction, there are two ways in which a Master-Abort may be signaled to the ATU:

1. In the Conventional or PCI-X modes, a master abort is signaled when the target of the transaction does not assert **DEVSEL#** within 5 clocks (7 clocks when operating in the PCI-X mode) of the assertion of **FRAME#**.
2. In PCI-X mode, ATU may enqueue a Split request (Read or Write) on target-side interface of a PCI-to-PCI Bridge. When PCI-to-PCI Bridge detects a Master Abort on requester-side interface for that Split Request, master abort is signaled to ATU through a Master-Abort Split Completion Error Message (class=1h - bridge error and index=00h - Master Abort). The following actions with given constraints are performed by ATU when a master abort is detected by the PCI initiator interface or the PCI target interface receives a Master-Abort Split Completion error message:

- Set the Master Abort bit (bit 13) in the ATUSR.
- When the ATU PCI Master Abort Interrupt Mask bit in the ATUIMR is clear, set the PCI Master Abort bit in the ATUISR. When set, no action.
- When an outbound write or inbound completion, flush data and address.
- When the transaction is an MSI outbound write and the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**, otherwise no action. When the ATU asserts **SERR#,** additional actions is taken:
  - Set the **SERR#** Asserted bit in the ATUSR
  - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
  - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action
- When operating in PCI-X mode and Master-Abort is signaled via a Split Completion Error Message, the Received Split Completion Error Message bit in PCIXSR is set. When ATU sets this bit, additional actions is taken:
  - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- For an Outbound Read request, generate a split completion error message (class=1h - 80331 Outbound Request error and index=00h - master abort) on the internal bus.
- Flush the address from the OTQ.

3.7.3.2 Inbound Read Completion or Inbound Configuration Write Completion Message

The ATU encounters this error only in the PCI-X mode.

A master abort is signaled when the target of the transaction does not assert **DEVSEL#** within 7 clocks of the assertion of **FRAME#**.

When the ATU is signaled a Master-Abort while initiating either a Split Read Completion Transaction or a Split Write Completion Message, the ATU discards the Split Completion and take no further action.

3.7.3.3 Master-Aborts Signaled by the ATU as a Target

3.7.3.3.1 Address Parity Errors

The ATU can only signal this error during an Address Parity Error in the Conventional mode.

Please see Section 3.7.1, "Address and Attribute Parity Errors on the PCI Interface" on page 175 for details on the ATU response to an Address Parity Error in the Conventional mode.

3.7.3.3.2 Internal Bus Master-Abort

The ATU can only signal this error during an internal bus master abort in the PCI-X mode.

Please see Section 3.7.7.1, "Master Abort on the Internal Bus" on page 193 for details on the ATU response to an Internal Bus Master Abort in the PCI-X mode.

3.7.4 Target Aborts on the PCI Interface

As an initiator on the PCI bus, the ATU can encounter target abort conditions during:

- Outbound Read Request
- Outbound Write Request
- Inbound Read Completion
- Inbound Configuration Write Completion Message

As a target, the ATU PCI interface is capable of signaling a target abort case during:

- Inbound Read Request (PCI-X and Conventional Modes)
- Inbound Write Request to EROM memory space (PCI-X and Conventional Modes)

3.7.4.1 Target Aborts for Outbound Read Request or Outbound Write Request

This error can be encountered by the ATU in both the Conventional and PCI-X modes. For an Outbound transaction, there are two ways in which a Target-Abort may be signaled to the ATU:

1. In the Conventional or PCI-X modes, a target abort is signaled when the target of the transaction simultaneously deasserts **DEVSEL#**, deasserts TRDY#**,** and asserts STOP#.
2. In PCI-X mode, ATU may enqueue a Split request (Read or Write) on target-side interface of a PCI-to-PCI Bridge. When PCI-to-PCI Bridge detects a Target Abort on requester-side interface for that Split Request, target abort is signaled to ATU through a Target-Abort Split Completion Error Message (class=1h - bridge error and index=01h - Target Abort).The following actions with the given constraints are performed by the ATU when a target abort is detected by the PCI initiator interface or the PCI target interface receives a Target-Abort Split Completion error message:

- Set the Target Abort (master) bit (bit 12) in the ATUSR.
- When the ATU PCI Target Abort (master) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (master) bit in the ATUISR. When set, no action.
- When the transaction is an MSI outbound write and the **SERR#** Enable bit in the ATUCMD is set, assert **SERR#**; otherwise, no action is taken. When the ATU asserts **SERR#,** additional actions is taken:
  - Set the **SERR#** Asserted bit in the ATUSR.
  - When the ATU **SERR#** Asserted Interrupt Mask Bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
  - When the ATU **SERR#** Detected Interrupt Enable Bit in the ATUCR is set, set the **SERR#** Detected bit in the ATUISR. When clear, no action.
- When operating in the PCI-X mode and the Target-Abort is signaled via a Split Completion Error Message, the Received Split Completion Error Message bit in the PCIXSR is set. When the ATU sets this bit, additional actions is taken:
  - When the ATU Received Split Completion Error Message Interrupt Mask bit in the ATUIMR is clear, set the Received Split Completion Error Message bit in the ATUISR. When set, no action.
- For an Outbound Read request, generate a split completion error message (message class=1h - 80331 Outbound Request error and message index=01h - target abort) on the internal bus.
- Flush the address from the OTQ.

3.7.4.2 Inbound Read Completion or Inbound Configuration Write Completion Message

The ATU encounters this error only in the PCI-X mode.

A target abort is signaled when the target of the transaction simultaneously deasserts **DEVSEL#**, deasserts TRDY#**,** and asserts STOP#.

When the ATU is signaled a Target-Abort while initiating either a Split Read Completion Transaction or a Split Write Completion Message, the ATU discards the Split Completion and take no further action

3.7.4.3 Target-Aborts Signaled by the ATU as a Target

3.7.4.3.1 Internal Bus Master Abort

A target abort can be signaled by the ATU during an inbound read request where the internal bus cycle resulted in a master abort on the Internal Bus.

Please see Section 3.7.7.1, "Master Abort on the Internal Bus" on page 193 for details on the ATU response to an Internal Bus Master Abort.

3.7.4.3.2 Internal Bus Target Abort

A target abort can be signaled by the ATU during an inbound read request where the internal bus cycle resulted in a Target Abort from the memory controller due to a non-recoverable multi-bit ECC error.

Please see Section 3.7.7.2, "Target Abort on the Internal Bus" on page 195 for details on the ATU response to an Internal Bus Target Abort.

3.7.4.3.3 Inbound EROM Memory Write

Since the EROM memory window is defined to be read-only by the *PCI Local Bus Specification*, Revision 2.3, the ATU target-aborts when an inbound write transaction is claimed by the EROM memory window.

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI target interface during an inbound EROM Memory write transaction:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
  - — When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.

3.7.5 Corrupted or Unexpected Split Completions

Warning: When any of the errors discussed in this section actually occur, a catastrophic system failure is likely to result from which the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a provides no recovery mechanism. In these cases, the ATU may be communicating with a non-compliant target device or the system may not be configured properly.

3.7.5.1 Completer Address

The ATU only asserts **DEVSEL#** for split completion transactions where the Sequence ID (Requester ID and Tag) matches that of a currently outstanding split request in the OTQ.

Conversely, the ATU does not assert **DEVSEL#** for any split completion transaction where either the Requester ID does not match that of the ATU or the Tag does not match that of any currently outstanding split request. No further action is taken.

When the Sequence ID of a split completion transaction matches that of an outstanding request, but the Lower Address field is not valid, the ATU accepts the split completion transaction in its' entirety according to the invalid Lower Address field and set the Unexpected Split Completion bit in the PCIXSR. No further action is taken.

3.7.5.2 Completer Attributes

When the Sequence ID of a split completion transaction matches that of an outstanding request, but the Byte Count is not valid, the ATU accepts the split completion transaction in its' entirety according to the invalid byte count field and set the Unexpected Split Completion bit in the PCIXSR. In this case, the ATU discards all the data. No further action is taken.

3.7.6 SERR# Assertion and Detection

The ATU is capable of reporting error conditions through the use of the **SERR#** output.

The following conditions may result in the assertion of **SERR#** by the ATU:

- An address parity error (or an attribute parity error when operating in the PCI-X mode) is detected by the ATU PCI interface (see Section 3.7.1, "Address and Attribute Parity Errors on the PCI Interface" on page 175 for details).
- A Master Data Parity Error is recorded in the ATUSR while operating in the PCI-X mode (see Section 3.7.2, "Data Parity Errors on the PCI Interface" on page 176 for details).
- An outbound MSI write transaction is either signaled a Master-Abort or a Target-Abort by the target.
- An inbound write transaction is master aborted on the internal bus (see Section 3.7.7.1, "Master Abort on the Internal Bus" on page 193 for details).
- The **SERR#** Manual Assertion bit in the ATUCR has been set and the **SERR#** Enable bit is set in the ATUCMD.

Note that the **SERR#** manual assertion bits must be cleared manually before they can be set again resulting in **SERR#** asserted. Refer to Section 3.10.39, "ATU Configuration Register - ATUCR" on page 252 for details.

The following actions with the given constraints are performed by the ATU when **SERR#** is asserted by the PCI interface:

- Set the **SERR#** Asserted bit in the ATUSR.
- When the ATU **SERR#** Asserted Interrupt Mask bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action.
- When **SERR#** is asserted and the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action.

The following actions with the given constraints are performed by the ATU when **SERR#** is detected by the PCI interface:

- When **SERR#** is detected and the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action.

Note: Whenever the ATU asserts **SERR#**, both the asserted and detected status bits may be set in the corresponding ISR. To mask an interrupt to the core when the ATU asserts **SERR#**, the **SERR#** asserted mask bit must be set and the **SERR#** detected interrupt enable bit must be clear.

3.7.7 Internal Bus Error Conditions

The 80331 internal bus uses a protocol similar to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a. As such, master abort and target abort conditions are valid error states on the bus. The error handling protocol for internal bus conditions is similar to the PCI bus error protocol. An internal bus error results in a bit being set in the Interrupt Status Registers at which time an interrupt is driven to the Intel® XScale™ core. Unlike PCI errors, internal bus error conditions are not maskable.

The following sections detail internal bus error conditions for the ATU.

3.7.7.1 Master Abort on the Internal Bus

A master abort on the internal bus is seen by the ATU when the inbound translated address presented on the internal bus is not claimed by the assertion of **I_DEVSEL#**.

3.7.7.1.1 Inbound Write Request

The following action with the given constraints are performed by the ATU when a master abort is detected by the internal master interface during an inbound write request transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR.
- When the Inbound Error **SERR#** Enable bit is set in the ATUIMR and the **SERR#** Enable bit is set in the ATUCMD, assert **SERR#** on the PCI interface. When both bits are not set, take no action. When **SERR#** is asserted, additional actions are taken:
  - Set the **SERR#** Asserted bit in the ATUSR
  - When the ATU **SERR#** Asserted Interrupt Mask bit in the ATUIMR is clear, set the **SERR#** Asserted bit in the ATUISR. When set, no action
  - When the ATU **SERR#** Detected interrupt enable is set in the ATUCR, set the **SERR#** Detected bit in the ATUISR. When clear, no action
- Flush the transaction that was master aborted from the IWQ.

The Internal Bus Master Abort bit is non-maskable and always results in an interrupt being driven to the core processor.

3.7.7.1.2 Inbound Read Request

When operating in the Conventional mode, the following actions with the given constraints are performed by the ATU when a master abort is detected by the internal initiator interface during an inbound read transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR
- Return a target abort condition to the initiating master during the delayed completion cycle on the PCI bus. No data is ever read from the internal bus and returned to the PCI bus.

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI interface during an inbound delayed read completion cycle:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
- When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.
- Flush the transaction that was master aborted from the ITQ after the target abort is delivered on the PCI interface.

When operating in the PCI-X mode, the following actions with the given constraints are performed by the ATU when a master abort is detected by the internal master interface during an inbound split read transaction:

- Set the Internal Bus Master Abort bit (bit 7) in the ATUISR.
- Generate a Split Completion Error Message (with message class=2h - completer error and message index=80h - 80331 internal bus master abort) on the PCI bus.
- When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.
- Flush the transaction that was master aborted.

Note: This split completion error message includes a device specific message index. The error handler would need to have knowledge of the device specific error messages of the 80331 in order to fully diagnose the problem.

The Internal Bus Master Abort bit is non-maskable and always results in an interrupt being driven to the core processor.

3.7.7.2 Target Abort on the Internal Bus

Target Aborts can be seen by the internal bus requester interface during inbound read operations to the memory controller. During inbound read operations, the memory controller is capable of signalling a target abort when a multi-bit, unrecoverable ECC error is encountered. This can occur during any read operation.

Note target aborts are signalled on a Qword basis. When either Dword of a Qword target aborts, both is considered to have target aborted.

The Memory Controller is responsible for creating an interrupt to the Intel® XScale™ core for any multi-bit ECC errors.

3.7.7.2.1 Conventional Mode

When operating in the Conventional PCI mode, when the data word which was target aborted on the internal bus is actually requested and delivered on the PCI Bus, and the ATU ECC Target Abort Enable bit is set in the ATUIMR, a target abort is returned to the PCI initiator on that data word. When the ATU ECC Target Abort Enable bit is clear in the ATUIMR, a disconnect with data is returned to the PCI initiator during the data word that was target aborted on the internal bus. In both cases, the IRQ is flushed after the completion cycle is performed on the PCI bus

The following additional actions with the given constraints are performed by the ATU when a target abort is signaled by the PCI target interface during an inbound read transaction:

- Set the Target Abort (target) bit (bit 11) in the ATUSR.
- When the ATU PCI Target Abort (target) Interrupt Mask bit in the ATUIMR is clear, set the PCI Target Abort (target) bit in the ATUISR. When set, no action.

3.7.7.2.2 PCI-X Mode

When operating in the PCI-X mode, a Target-Abort of an inbound read transaction (split read request) on the Internal Bus results in the following actions.

- The ATU initiates a Split Completion Error Message (with message class=2h - completer error and message index=81h - 80331 internal bus target abort) on the PCI bus.
- When the Initiated Split Completion Error Message Interrupt Mask in the ATUIMR is clear, set the Initiated Split Completion Error Message bit in the ATUISR. When set, no action.

Note: This split completion error message includes a device specific message index. The error handler would need to have knowledge of the device specific error messages of the 80331 in order to fully diagnose the problem.

3.7.8 ATU Error Summary

Table 81 summarizes the ATU error reporting for PCI bus errors and Table 82 summarizes the ATU error reporting for internal bus errors. The tables assume that all error reporting is enabled through the appropriate command registers (unless otherwise noted). The ATU Status Register records PCI bus errors. Note that the **SERR#** Asserted bit in the Status Register is set only when the **SERR#** Enable bit in the Command Register is set. The ATU Interrupt Status Registers record Intel® XScale™ core interrupt status information.

Table 81. ATU Error Reporting Summary - PCI Interface (Sheet 1 of 3)

Error Condition[a] (Bus Mode[b])	Bits Set in ATU Status Register (ATUSR[c]) or PCI-X Status Register (PCIXSR[d])	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Address or Attribute Parity Error (Both)	In Conventional Mode, ignore (Master-Abort) the transaction, and then signal **SERR#**. In PCI-X Mode, claim the transaction and complete as though no error had occurred; and, signal **SERR#** upon parity error detection.		
(Both)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(Both)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(Both)	Detected Address or Attribute Parity Error - bit 18 of the PCI Configuration and Status Register	N/A	N/A
(Both)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Outbound Read Request Parity Error (Both)	Signal **PERR#** and **SERR#** (PCI-X Mode Only).		
(Both)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(Both)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Outbound Write Request Parity Error (Both)	Signal **SERR#** (only for PCI-X or MSI Writes).		
(Both)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X or MSI)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(PCI-X or MSI)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
Inbound Read Completions Parity Error (PCI-X)	None.		
Inbound Configuration Write Completion Message Parity Error (PCI-X)	None.		
Inbound Read Request Parity Error (Both)	None.		

Table 81. ATU Error Reporting Summary - PCI Interface (Sheet 2 of 3)

Error Condition[a] (Bus Mode[b])	Bits Set in ATU Status Register (ATUSR[c]) or PCI-X Status Register (PCIXSR[d])	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Inbound Write Request Parity Error (Both)	Signal **PERR#**.		
(Both)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Outbound Split Write Data Parity Error Message (PCI-X)	Signal **SERR#**.		
(PCI-X)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(PCI-X)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Inbound Configuration Write Request Parity Error (Both)	Signal **PERR#**. Initiate a Split Write Data Parity Error Message addressed to the Requester (PCI-X Mode Only).		
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10
(Both)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Split Completion Message Parity Error (PCI-X)	Signal **PERR#** and **SERR#**.		
(PCI-X)	Master Parity Error - bit 8	Master Parity Error - bit 0	ATUIMR bit 2
(PCI-X)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(PCI-X)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(PCI-X and SCE[e])	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
(PCI-X)	Detected Parity Error - bit 15	Detected Parity Error - bit 9	ATUIMR bit 7
Outbound Read Request Master-Abort (Both)	None.		
(Both)	Master Abort - bit 13	PCI Master Abort - bit 3	ATUIMR bit 5
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Outbound Write Request Master-Abort (Both)	None.		
(Both)	Master Abort - bit 13	PCI Master Abort - bit 3	ATUIMR bit 5
(MSI)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(MSI)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9

Table 81. ATU Error Reporting Summary - PCI Interface (Sheet 3 of 3)

Error Condition[a] (Bus Mode[b])	Bits Set in ATU Status Register (ATUSR[c]) or PCI-X Status Register (PCIXSR[d])	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Inbound Read Completions Master-Abort (PCI-X)	None.		
Inbound Configuration Write Completion Message Master-Abort (PCI-X)	None.		
Outbound Read Request Target-Abort (Both)	None.		
(Both)	Target Abort (master) - bit 12	PCI Target Abort (master) - bit 2	ATUIMR bit 4
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Outbound Write Request Target-Abort (Both)	None.		
(Both)	Target Abort (master) - bit 12	PCI Target Abort (master) - bit 2	ATUIMR bit 4
(MSI)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(MSI)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
(PCI-X and SCE)	Received Split Completion Error Message - bit 29	Received Split Completion Error Message - bit 12	ATUIMR bit 9
Inbound EROM Write Request Target-Abort (Both)	Signal Target-Abort.		
(Both)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
Unexpected Split Completion (PCI-X)	In the PCI-X mode, the transaction completes normally according to the invalid lower address field or invalid byte count.		
(PCI-X)	Unexpected Split Completion - bit 19	N/A	N/A

a. All parity errors refer to data parity errors except where otherwise noted.
b. Codes for bus mode in which this error response applies: PCI-X means PCI-X Mode Only, Conventional means Conventional PCI Mode Only, and Both means that the error response applies both in the Conventional and PCI-X mode of operation. MSI stands for Message-Signaled Interrupts and refers to an Outbound Write transaction that is actually an MSI write transaction.
c. Table assumes that Parity Error Response - bit 6 of the ATUCMD register is set.
d. Table assumes that Data Parity Recovery Enable - bit 0 of the PCIXCMD is clear.
e. When the SCE bit (bit 30 of the Completer Attributes) and the SCM bit (bit 29 of the Completer Attributes) are set during the Attribute phase of a Split Completion Transaction, the transaction is a Split Completion Message that is an Error Message. In this case, the Received Split Completion Error Message - bit 29 of the PCIXSR is set.

Table 82. ATU Error Reporting Summary - Internal Bus Interface

Error Condition[a] (Bus Mode[b])	Bits Set in ATU Status Register (ATUSR[c])	Bits Set in ATU Interrupt Status Register (ATUISR)	Interrupt Mask Bit in ATUIMR or ATUCR
	PCI Bus Error Response (i.e., signal Target-Abort, signal Master-Abort etc.)		
Inbound Write Request Master-Abort (Both)	Assert **SERR#**.		
(Both)	N/A	Internal Bus Master Abort - bit 7	N/A
(Both)	**SERR#** Asserted - bit 14	**SERR#** Asserted - bit 10	ATUIMR bit 6
(Both)	N/A	**SERR#** Detected - bit 4	ATUCR bit 9
Inbound Read Request Master-Abort (Both)	In the Conventional Mode signal Target-Abort. In the PCI-X Mode send a device specific Split Completion Error Message to the Requester.		
(Both)	N/A	Internal Bus Master Abort - bit 7	N/A
(Conventional)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10
Inbound Read Request Target-Abort (Both)	In the Conventional Mode signal Target-Abort. In the PCI-X Mode send a device specific Split Completion Error Message to the Requester.		
(Conventional)	Target Abort (target) - bit 11	PCI Target Abort (target) - bit 1	ATUIMR bit 3
(PCI-X)	N/A	Initiated Split Completion Error Message - bit 13	ATUIMR bit 10

a. There are no Inbound Write Request Target-Abort Error Conditions.
b. Codes for bus mode in which this error response applies: PCI-X means PCI-X Mode Only, Conventional means Conventional PCI Mode Only, and Both means that the error response applies both in the Conventional and PCI-X mode of operation. MSI stands for Message-Signaled Interrupts and refers to an Outbound Write transaction that is actually an MSI write transaction.
c. Table assumes that the ATU Inbound **SERR#** Enable bit (bit 1 of the ATUIMR), the ATU ECC Target Abort Enable (bit 0 of the ATUIMR), and the **SERR#** Enable bit (bit 8 of the ATUCMD) are set.

3.8 Message-Signaled Interrupts

The Messaging Unit is responsible for the generation of all of the Outbound Interrupts from the 80331. These interrupts can be delivered to the Host Processor via the **P_INTC#** output pin or the Message Signaled Interrupt (MSI) mechanism.

When a host processor enables Message-Signaled Interrupts (MSI) on the 80331, an outbound interrupt is signaled to the host via a PCI write instead of the assertion of the **P_INTC#** output pin.

In support of MSI, the 80331 implements the MSI capability structure. The capability structure includes the Section 4.9.20, "MSI Capability Identifier Register - MSI_CAPID" on page 323, the Section 4.9.21, "MSI Next Item Pointer Register - MSI_NXTP" on page 324, the Section 4.9.23, "MSI Message Address Register - MSI_MAR" on page 326, the Section 4.9.24, "MSI Message Upper Address Register - MSI_MUAR" on page 327 and the Section 4.9.25, "MSI Message Data Register- MSI_MDR" on page 328.

The Message Unit generates MSIs by writing to the MSI port via the internal bus. The ATU generates a write transaction whenever the Message Unit writes to the MSI port, using the address specified in the Section 4.9.23, "MSI Message Address Register - MSI_MAR" on page 326, the Section 4.9.24, "MSI Message Upper Address Register - MSI_MUAR" on page 327 and the Section 4.9.25, "MSI Message Data Register- MSI_MDR" on page 328.

3.9 Vital Product Data

Vital Product Data (VPD) provides detailed information to the system regarding the hardware, software and microcode elements of a device. This information may include Part Number, Serial Number or other detailed information. This information resides on a non-volatile storage device (i.e., Flash Memory) attached to the 80331. In addition VPD also provides a mechanism for storing information such as performance or failure data on the device being monitored.

Support of VPD involves the implementation of the VPD Extended Capabilities List Item in the Primary ATU. The VPD Extended capabilities header consists of five registers, the "VPD Capability Identifier Register - VPD_CAPID" on page 267, the "VPD Next Item Pointer Register - VPD_NXTP" on page 268, the "VPD Address Register - VPD_AR" on page 269, and the "VPD Data Register - VPD_DR" on page 270.

Scheduled by Intel® XScale™ core interrupts, the 80331 may be used to retrieve or store VPD information through the VPD extended capabilities list item.

Please consult Appendix I of the *PCI Local Bus Specification*, Revision 2.3 for the definitions of compliant VPD format.

3.9.1 Configuring Vital Product Data Operation

By default, the 80331 VPD functionality is not configured for operation. Specifically, the VPD Extended Capabilities List Item is not discovered during a PCI bus scan and the ATUs VPD interrupt status bit in the "ATU Interrupt Status Register - ATUISR" on page 257 is masked by the "ATU Interrupt Mask Register - ATUIMR" on page 259. The following steps should be followed to properly configure the Intel® 80331 I/O processor support for VPD:

1. The 80331 must be strapped to Retry Type 0 Configuration cycles following the deassertion of **P_RST#**. Enabling this configuration cycle retry mechanism guarantees that the Intel® XScale™ core can make the VPD Extended Capabilities List Item visible before the system configures the 80331. The configuration retry mechanism is controlled through bit 2 of the Table 125, "PCI Configuration and Status Register - PCSR" on page 253.
2. When the configuration retry mechanism is strapped enabled as described in step 1, typically, the 80331 would also be strapped such that the Intel® XScale™ core would immediately boot following the deassertion of **P_RST#** (bit 1 of the PCSR), though this is not required.
3. The Intel® XScale™ core writes E8H to the "PCI-X Next Item Pointer Register - PX_NXTP" on page 277. This links the PCI-X Capabilities List Item to the VPD Capabilities List Item.
4. The Intel® XScale™ core clears bit 12 of the ATUIMR to enable the ATUs VPD interrupt status bit.

3.9.2 Accessing Vital Product Data

The VPD Capabilities List Item provides three fields which the system uses to access the Vital Product Data:

VPD Address — DWORD Aligned Byte address of the VPD to be accessed which is represented by VPDAR[14:0]. Note that this means that the maximum size of the VPD is 128 Kbytes. The user may pick any 128 Kbyte block of memory in the storage component for the VPD.

Flag — The flag register is used to indicate when the transfer between the VPD Data Register and the storage component is completed. The flag is in VPDAR[15] which means that the Flag is written at the same time that VPD address is written.

VPD Data — Four bytes of VPD Data can be read or written through this field which is represented by VPDDR[31:0]. The least significant byte of this register represents the byte at the VPD Address (VPDAR[14:0]). Four bytes are always transferred between this register and the VPD storage component.

3.9.2.1 Reading Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 80331 reads Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag cleared.
2. An interrupt to the Intel® XScale™ core is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be set.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being set, the results of the original read operation are unpredictable.

3. Using the VPD Address, the Intel® XScale™ core retrieves the Vital Product Data from the VPD storage component (i.e., Flash Memory).
4. The Intel® XScale™ core then writes this data to VPD Data Register (VPDDR).
5. The Intel® XScale™ core clears the VPD interrupt status bit in the ATUISR.
6. The Intel® XScale™ core then sets the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been set, the host processor then reads the retrieved VPD from the VPDDR.

3.9.2.2 Writing Vital Product Data

Using the fields defined in the VPD Capabilities List Item, the 80331 writes Vital Product Data using the following sequence of events:

1. Host processor executes a configuration write of the VPD data to be written to the VPDDR.
2. Host processor executes a configuration write of the VPD address to the VPDAR with the Flag set.
3. An interrupt to the Intel® XScale™ core is triggered and bit 17 of the ATUISR is set. Meanwhile, the host processor polls the VPDAR register waiting for the Flag to be cleared.

Warning: When any configuration writes to either the VPDAR or the VPDDR occur prior to the Flag being cleared, the results of the original write operation are unpredictable.

4. Using the VPD Address, the Intel® XScale™ core writes the Vital Product Data from the VPDDR to the VPD storage component (i.e., Flash Memory).
5. The Intel® XScale™ core clears the VPD interrupt status bit in the ATUISR.
6. The Intel® XScale™ core then clears the Flag in the VPDAR register.
7. When the host processor detects that the Flag has been cleared, the host processor has been informed that the VPD write operation is complete.

3.10 Register Definitions

Every PCI device implements its own separate configuration address space and configuration registers. The *PCI Local Bus Specification*, Revision 2.3 requires that configuration space be 256 bytes, and the first 64 bytes must adhere to a predefined header format.

Figure 15 defines the header format. Table 84 shows the PCI configuration registers, listed by internal bus address offset. Table 84 shows the entire ATU configuration space (including header and extended registers) and the corresponding section that describes each register. Note that all configuration read and write transactions is accepted on the internal bus as 32-bit transactions. Refer to Chapter 17, "Peripheral Registers".

Figure 15. ATU Interface Configuration Header Format

ATU Device ID		Vendor ID		00H
Status		Command		04H
ATU Class Code			Revision ID	08H
ATUBISTR	Header Type	Latency Timer	Cacheline Size	0CH
Inbound ATU Base Address 0				10H
Inbound ATU Upper Base Address 0				14H
Inbound ATU Base Address 1				18H
Inbound ATU Upper Base Address 1				1CH
Inbound ATU Base Address 2				20H
Inbound ATU Upper Base Address 2				24H
Reserved				28H
ATU Subsystem ID		ATU Subsystem Vendor ID		2CH
Expansion ROM Base Address				30H
Reserved			Capabilities Pointer	34H
Reserved				38H
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	3CH

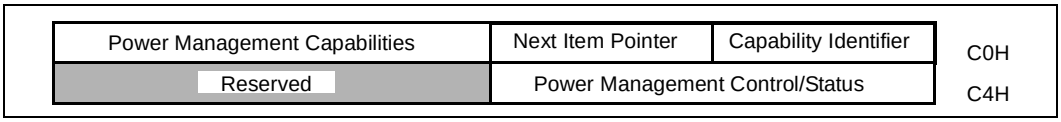
The ATU is programmed via a Type 0 configuration command on the PCI interface. See Section 3.2.1.4, "Inbound Configuration Cycle Translation" on page 147. ATU configuration space is function number zero of the 80331 single-function PCI device.

Beyond the required 64 byte header format, ATU configuration space implements extended register space in support of the units functionality. Refer to the *PCI Local Bus Specification*, Revision 2.3 for details on accessing and programming configuration register space.

The ATU unit includes two 8 byte and one 16 byte extended capability configuration spaces beginning at configuration offset C0H, D0H and E0H. The extended configuration spaces can be accessed by a device on the PCI interface through a mechanism defined in the *PCI Local Bus Specification*, Revision 2.3.

In the ATU Status Register (Section 3.10.4) the appropriate bit is set indicating that the Extended Capability Configuration space is supported. When this bit is read, the device can then read the Capabilities Pointer register (Section 3.10.20) to determine the configuration offset of the Extended Capabilities Configuration Header. The format of these headers are depicted in Figure 16, Figure 17, Figure 18and Figure 19.

Figure 16. ATU Interface Extended Configuration Header Format (Power Management)

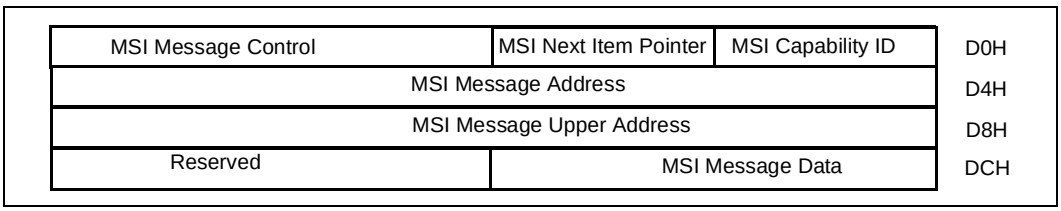


The first byte at the Extended Configuration Offset C0H is the ATU Capability Identifier Register (Section). This identifies this Extended Configuration Header space as the type defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 3.10.54) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to D0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

To enable the *PCI Bus Power Management Interface Specification*, Revision 1.1 compliance support, the Power State Transition interrupt mask in bit 8 of the ATUIMR needs to be cleared. It is the configuration software's responsibility to properly enable and initialize the ATUs Power Management Interface before the Configuration Cycle Retry Bit in the Table 125, "PCI Configuration and Status Register - PCSR" on page 253 is cleared in order for the ATU to be *Advanced Configuration and Power Interface Specification*, Revision 2.0 compliant.

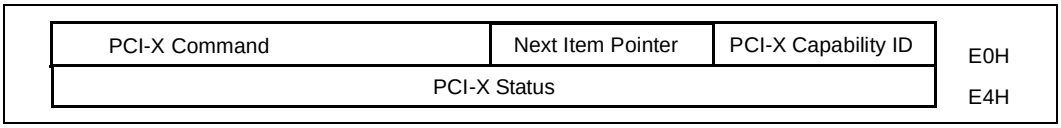
Figure 17. ATU Interface Extended Configuration Header Format (MSI Capability)



The first byte at the Extended Configuration Offset D0H is the "MSI Capability Identifier Register - MSI_CAPID" on page 323. This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte "MSI Next Item Pointer Register - MSI_NXTP" on page 324, which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to E0H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

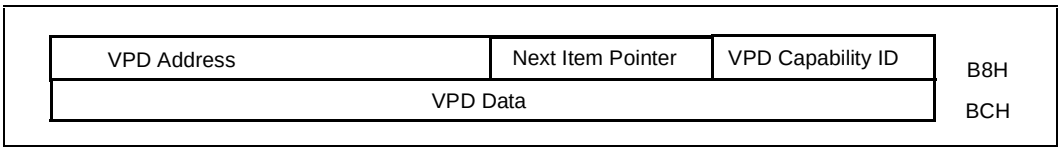
Figure 18. ATU Interface Extended Configuration Header Format (PCI-X Capability)



The first byte at the Extended Configuration Offset E0H is the PCI-X Capability Identifier Register (Section 3.10.58). This identifies this Extended Configuration Header space as the type defined by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 3.10.59) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H by default to indicate there are no additional Extended Capabilities Headers in the ATUs configuration space. Software can set this pointer to B8H indicating that there is an additional Extended Capabilities Headers supported in the ATUs configuration space.

Figure 19. ATU Interface Extended Configuration Header Format (VPD Capability)



The first byte at the Extended Configuration Offset B8H is the VPD Capability Identifier Register (Section 3.10.49). This identifies this Extended Configuration Header space as the type defined by the *PCI Local Bus Specification*, Revision 2.3.

Following the Capability Identifier Register is the single byte Next Item Pointer Register (Section 3.10.50) which indicates the configuration offset of an additional Extended Capabilities Header, when supported. In the ATU, the Next Item Pointer Register is set to 00H indicating that there are no additional Extended Capabilities Headers supported in the ATUs configuration space.

The following sections describe the ATU and Expansion ROM configuration registers. Configuration space consists of 8, 16, 24, and 32-bit registers arranged in a predefined format. Each register is described in functionality, access type (read/write, read/clear, read only) and reset default condition.

See Section 1.4, "Terminology and Conventions" on page 41 for a description of *reserved*, *read only*, and *read/clear*. All registers adhere to the definitions found in the *PCI Local Bus Specification*, Revision 2.3 unless otherwise noted.

The PCI register number for each register is given in Table 84. As stated, a Type 0 configuration command on the bus with an active **IDSEL** or a memory-mapped internal bus access is required to read or write these registers.

Note: Each configuration register's access type is individually defined for PCI configuration accesses. Some PCI read-only configuration registers have read/write capability from the 80331 core CPU. See also Chapter 17, "Peripheral Registers".

Table 83. Address Translation Unit Registers (Sheet 1 of 3)

Register Name	Bits	PCI Configuration Cycle Register Number	Internal Bus Address
ATU Vendor ID Register - ATUVID	16	0	FFFF.E100H
ATU Device ID Register - ATUDID	16	0	FFFF.E102H
ATU Command Register - ATUCMD	16	1	FFFF.E104H
ATU Status Register - ATUSR	16	1	FFFF.E106H
ATU Revision ID Register - ATURID	8	2	FFFF.E108H
ATU Class Code Register - ATUCCR	24	2	FFFF.E109H
ATU Cacheline Size Register - ATUCLSR	8	3	FFFF.E10CH
ATU Latency Timer Register - ATULT	8	3	FFFF.E10DH
ATU Header Type Register - ATUHTR	8	3	FFFF.E10EH
ATU BIST Register - ATUBISTR	8	3	FFFF.E10FH
Inbound ATU Base Address Register 0 - IABAR0	32	4	FFFF.E110H
Inbound ATU Upper Base Address Register 0 - IAUBAR0	32	5	FFFF.E114H
Inbound ATU Base Address Register 1 - IABAR1	32	6	FFFF.E118H
Inbound ATU Upper Base Address Register 1 - IAUBAR1	32	7	FFFF.E11CH
Inbound ATU Base Address Register 2 - IABAR2	32	8	FFFF.E120H
Inbound ATU Upper Base Address Register 2 - IAUBAR2	32	9	FFFF.E124H
Reserved	32	10	FFFF.E128H
ATU Subsystem Vendor ID Register - ASVIR	16	11	FFFF.E12CH
ATU Subsystem ID Register - ASIR	16	11	FFFF.E12EH
Expansion ROM Base Address Register -ERBAR	32	12	FFFF.E130H
ATU Capabilities Pointer Register - ATU_Cap_Ptr	8	13	FFFF.E134H
Reserved	8	13	FFFF.E135H
Reserved	16	13	FFFF.E136H
Reserved	32	14	FFFF.E138H
ATU Interrupt Line Register - ATUILR	8	15	FFFF.E13CH
ATU Interrupt Pin Register - ATUIPR	8	15	FFFF.E13DH
ATU Minimum Grant Register - ATUMGNT	8	15	FFFF.E13EH
ATU Maximum Latency Register - ATUMLAT	8	15	FFFF.E13FH
Inbound ATU Limit Register 0 - IALR0	32	16	FFFF.E140H
Inbound ATU Translate Value Register 0 - IATVR0	32	17	FFFF.E144H
Expansion ROM Limit Register - ERLR	32	18	FFFF.E148H
Expansion ROM Translate Value Register - ERTVR	32	19	FFFF.E14CH
Inbound ATU Limit Register 1 - IALR1	32	20	FFFF.E150H
Inbound ATU Limit Register 2 - IALR2	32	21	FFFF.E154H
Inbound ATU Translate Value Register 2 - IATVR2	32	22	FFFF.E158H
Outbound I/O Window Translate Value Register - OIOWTVR	32	23	FFFF.E15CH

Table 83. Address Translation Unit Registers (Sheet 2 of 3)

Register Name	Bits	PCI Configuration Cycle Register Number	Internal Bus Address
Outbound Memory Window Translate Value Register 0- OMWTVR0	32	24	FFFF.E160H
Outbound Upper 32-bit Memory Window Translate Value Register 0- OUMWTVR0	32	25	FFFF.E164H
Outbound Memory Window Translate Value Register 1- OMWTVR1	32	26	FFFF.E168H
Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1	32	27	FFFF.E16CH
Reserved	32	28	FFFF.E170H
Reserved	32	29	FFFF.E174H
Outbound Upper 32-bit Direct Window Translate Value Register - OUDWTVR	32	30	FFFF.E178H
Reserved	32	31	FFFF.E17CH
ATU Configuration Register - ATUCR	32	32	FFFF.E180H
PCI Configuration and Status Register - PCSR	32	33	FFFF.E184H
ATU Interrupt Status Register - ATUISR	32	34	FFFF.E188H
ATU Interrupt Mask Register - ATUIMR	32	35	FFFF.E18CH
Inbound ATU Base Address Register 3 - IABAR3	32	36	FFFF.E190H
Inbound ATU Upper Base Address Register 3 - IAUBAR3	32	37	FFFF.E194H
Inbound ATU Limit Register 3 - IALR3	32	38	FFFF.E198H
Inbound ATU Translate Value Register 3 - IATVR3	32	39	FFFF.E19CH
Reserved	32	40	FFFF.E1A0H
Outbound Configuration Cycle Address Register - OCCAR	32	41	FFFF.E1A4H
Reserved	32	42	FFFF.E1A8H
Outbound Configuration Cycle Data Register - OCCDR	32	Not Available in PCI Configuration Space	FFFF.E1ACH
Reserved	32	44	FFFF.E1B0H
Reserved	32	45	FFFF.E1B4H
VPD Capability Identifier Register - VPD_CAPID	8	46	FFFF.E1B8H
VPD Next Item Pointer Register - VPD_NXTP	8	46	FFFF.E1B9H
VPD Address Register - VPD_AR	16	47	FFFF.E1BAH
VPD Data Register - VPD_DR	32	47	FFFF.E1BCH
Power Management Capability Identifier Register - PM_CAPID	8	48	FFFF.E1C0H
Power Management Next Item Pointer Register - PM_NXTP	8	48	FFFF.E1C1H
Power Management Capabilities Register - PM_CAP	16	48	FFFF.E1C2H
Power Management Control/Status Register - PM_CSR	16	49	FFFF.E1C4H
Reserved	16	49	FFFF.E1C6H
Reserved	32	50	FFFF.E1C8H
Reserved	32	51	FFFF.E1CCH

Table 83. Address Translation Unit Registers (Sheet 3 of 3)

Register Name	Bits	PCI Configu-ration Cycle Register Number	Internal Bus Address
MSI Capability Identifier Register - MSI_CAPID	8	52	FFFF.E1D0H
MSI Next Item Pointer Register - MSI_NXTP	8	52	FFFF.E1D1H
MSI Message Control Register - MSI_MCR	16	52	FFFF.E1D2H
MSI Message Address Register - MSI_MAR	32	53	FFFF.E1D4H
MSI Message Upper Address Register - MSI_MUAR	32	54	FFFF.E1D8H
MSI Message Data Register- MSI_MDR	16	55	FFFF.E1DCH
Reserved	16	55	FFFF.E1DEH
PCI-X_Capability Identifier Register - PX_CAPID	8	56	FFFF.E1E0H
PCI-X Next Item Pointer Register - PX_NXTP	8	56	FFFF.E1E1H
PCI-X Command Register - PX_CMD	16	56	FFFF.E1E2H
PCI-X Status Register - PX_SR	32	57	FFFF.E1E4H
"PCI Interrupt Routing Select Register - PIRSR" on page 717	32	59	FFFF.E1ECH
Secondary PCIDrive Strength Control Register - SPDSCR	32	n/a	FFFF.F5C0H
Primary PCIDrive Strength Control Register - PPDSCR	32	n/a	FFFF.F5C8H

Table 84. ATU PCI Configuration Register Space (Sheet 1 of 2)

Address Offset	ATU PCI Configuration Register Section, Name, Page
00H	Section 3.10.1, "ATU Vendor ID Register - ATUVID" on page 212
02H	Section 3.10.2, "ATU Device ID Register - ATUDID" on page 213
04H	Section 3.10.3, "ATU Command Register - ATUCMD" on page 214
06H	Section 3.10.4, "ATU Status Register - ATUSR" on page 215
08H	Section 3.10.5, "ATU Revision ID Register - ATURID" on page 217
09H	Section 3.10.6, "ATU Class Code Register - ATUCCR" on page 218
0CH	Section 3.10.7, "ATU Cacheline Size Register - ATUCLSR" on page 219
0DH	Section 3.10.8, "ATU Latency Timer Register - ATULT" on page 220
0EH	Section 3.10.9, "ATU Header Type Register - ATUHTR" on page 221
0FH	Section 3.10.10, "ATU BIST Register - ATUBISTR" on page 222
10H	Section 3.10.11, "Inbound ATU Base Address Register 0 - IABAR0" on page 223
14H	Section 3.10.12, "Inbound ATU Upper Base Address Register 0 - IAUBAR0" on page 224
18H	Section 3.10.13, "Inbound ATU Base Address Register 1 - IABAR1" on page 225
1CH	Section 3.10.14, "Inbound ATU Upper Base Address Register 1 - IAUBAR1" on page 226
20H	Section 3.10.15, "Inbound ATU Base Address Register 2 - IABAR2" on page 227
24H	Section 3.10.16, "Inbound ATU Upper Base Address Register 2 - IAUBAR2" on page 228
2CH	Section 3.10.17, "ATU Subsystem Vendor ID Register - ASVIR" on page 229
2EH	Section 3.10.18, "ATU Subsystem ID Register - ASIR" on page 230
30H	Section 3.10.19, "Expansion ROM Base Address Register - ERBAR" on page 231
34H	Section 3.10.20, "ATU Capabilities Pointer Register - ATU_Cap_Ptr" on page 232
3CH	Section 3.10.22, "ATU Interrupt Line Register - ATUILR" on page 235
3DH	Section 3.10.23, "ATU Interrupt Pin Register - ATUIPR" on page 236
3EH	Section 3.10.24, "ATU Minimum Grant Register - ATUMGNT" on page 237
3FH	Section 3.10.25, "ATU Maximum Latency Register - ATUMLAT" on page 238
40H	Section 3.10.26, "Inbound ATU Limit Register 0 - IALR0" on page 239
44H	Section 3.10.27, "Inbound ATU Translate Value Register 0 - IATVR0" on page 240
48H	Section 3.10.28, "Expansion ROM Limit Register - ERLR" on page 241
4CH	Section 3.10.29, "Expansion ROM Translate Value Register - ERTVR" on page 242
50H	Section 3.10.30, "Inbound ATU Limit Register 1 - IALR1" on page 243
54H	Section 3.10.31, "Inbound ATU Limit Register 2 - IALR2" on page 244
58H	Section 3.10.32, "Inbound ATU Translate Value Register 2 - IATVR2" on page 245
5CH	Section 3.10.33, "Outbound I/O Window Translate Value Register - OIOWTVR" on page 246
60H	Section 3.10.34, "Outbound Memory Window Translate Value Register 0 - OMWTVR0" on page 247
64H	Section 3.10.35, "Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0" on page 248
68H	Section 3.10.36, "Outbound Memory Window Translate Value Register 1 - OMWTVR1" on page 249
6CH	Section 3.10.37, "Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1" on page 250
80H	Section 3.10.39, "ATU Configuration Register - ATUCR" on page 252
84H	Section 3.10.40, "PCI Configuration and Status Register - PCSR" on page 253
88H	Section 3.10.41, "ATU Interrupt Status Register - ATUISR" on page 257
8CH	Section 3.10.42, "ATU Interrupt Mask Register - ATUIMR" on page 259
90H	Section 3.10.43, "Inbound ATU Base Address Register 3 - IABAR3" on page 261

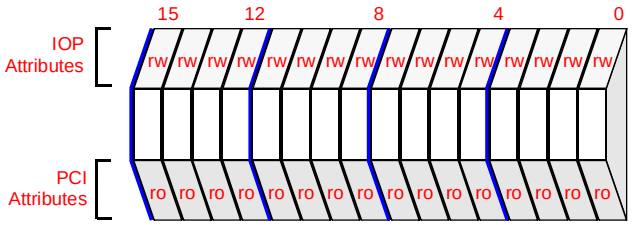
Table 84. ATU PCI Configuration Register Space (Sheet 2 of 2)

Address Offset	ATU PCI Configuration Register Section, Name, Page
94H	Section 3.10.44, "Inbound ATU Upper Base Address Register 3 - IAUBAR3" on page 262
98H	Section 3.10.45, "Inbound ATU Limit Register 3 - IALR3" on page 263
9CH	Section 3.10.46, "Inbound ATU Translate Value Register 3 - IATVR3" on page 264
A4H	Section 3.10.47, "Outbound Configuration Cycle Address Register - OCCAR" on page 265
ACH	Section 3.10.48, "Outbound Configuration Cycle Data Register - OCCDR" on page 266
B8H	Section 3.10.49, "VPD Capability Identifier Register - VPD_CAPID" on page 267
B9H	Section 3.10.50, "VPD Next Item Pointer Register - VPD_NXTP" on page 268
BAH	Section 3.10.51, "VPD Address Register - VPD_AR" on page 269
BCH	Section 3.10.52, "VPD Data Register - VPD_DR" on page 270
C0H	Section 3.10.53, "Power Management Capability Identifier Register - PM_CAPID" on page 271
C1H	Section 3.10.54, "Power Management Next Item Pointer Register - PM_NXTP" on page 272
C2H	Section 3.10.55, "Power Management Capabilities Register - PM_CAP" on page 273
C4H	Section 3.10.56, "Power Management Control/Status Register - PM_CSR" on page 274
D0H	Section 4.9.20, "MSI Capability Identifier Register - MSI_CAPID" on page 323
D1H	Section 4.9.21, "MSI Next Item Pointer Register - MSI_NXTP" on page 324
D2H	Section 4.9.22, "MSI Message Control Register - MSI_MCR" on page 325
D4H	Section 4.9.23, "MSI Message Address Register - MSI_MAR" on page 326
D8H	Section 4.9.24, "MSI Message Upper Address Register - MSI_MUAR" on page 327
DCH	Section 4.9.25, "MSI Message Data Register- MSI_MDR" on page 328
E0H	Section 3.10.58, "PCI-X Capability Identifier Register - PX_CAPID" on page 276
E1H	Section 3.10.59, "PCI-X Next Item Pointer Register - PX_NXTP" on page 277
E2H	Section 3.10.60, "PCI-X Command Register - PX_CMD" on page 278
E4H	Section 3.10.61, "PCI-X Status Register - PX_SR" on page 279
ECH	Section 15.7.17, "PCI Interrupt Routing Select Register - PIRSR" on page 717
n/a	Section 3.10.63, "Secondary PCIDrive Strength Control Register - SPDSCR" on page 282
n/a	Section 3.10.64, "Primary PCIDrive Strength Control Register - PPDSCR" on page 283

3.10.1 ATU Vendor ID Register - ATUVID

ATU Vendor ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 85. ATU Vendor ID Register - ATUVID



Internal Bus Address
FFFF.E100H

PCI Configuration Address Offset
00H - 01H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

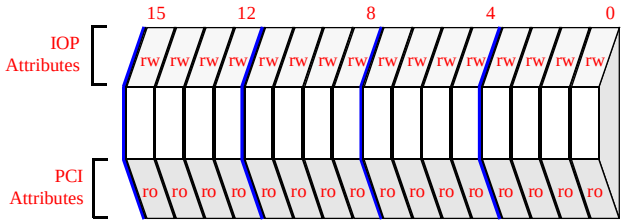
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:00	8086H	ATU Vendor ID - This is a 16-bit value assigned to Intel. This register, combined with the DID, uniquely identify the PCI device. Access type is Read/Write to allow the 80331 to configure the register as a different vendor ID to simulate the interface of a standard mechanism currently used by existing application software.

3.10.2 ATU Device ID Register - ATUDID

ATU Device ID Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3.

Table 86. ATU Device ID Register - ATUDID



Internal Bus Address
FFFF.E102H

PCI Configuration Address Offset
02H - 03H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

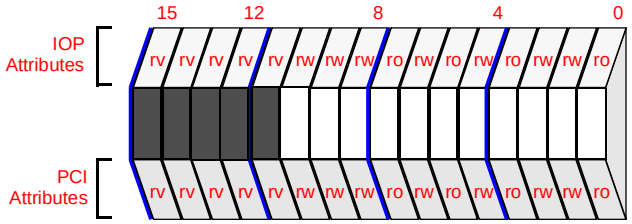
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:00	0336H	ATU Device ID - This is a 16-bit value assigned to the ATU. This ID, combined with the VID, uniquely identify any PCI device.

3.10.3 ATU Command Register - ATUCMD

ATU Command Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3 and in most cases, affect the behavior of the PCI ATU and devices on the PCI bus.

Table 87. ATU Command Register - ATUCMD



Internal Bus Address
FFFF.E104H

PCI Configuration Address Offset
04H - 05H

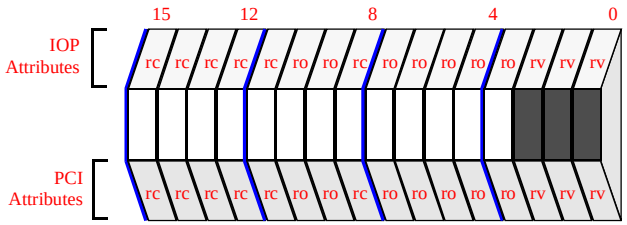
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:11	000000_2	Reserved
10	0	Interrupt Disable - This bit disables 80331 from asserting the ATU interrupt signal. 0 = enables the assertion of interrupt signal. 1 = disables the assertion of its interrupt signal. Refer to Section 3.10.23, "ATU Interrupt Pin Register - ATUIPR" on page 236 for details on the ATU interrupt signal.
09	0_2	Fast Back to Back Enable - When cleared, the ATU interface is not allowed to generate fast back-to-back cycles on its bus. Ignored when operating in the PCI-X mode.
08	0_2	**SERR#** Enable - When cleared, the ATU interface is not allowed to assert **SERR#** on the PCI interface.
07	1_2	Address/Data Stepping Control - Address stepping is implemented for configuration transactions. The ATU inserts 2 clock cycles of address stepping for Conventional Mode and 4 clock cycles of address stepping for PCI-X mode.
06	0_2	Parity Error Response - When set, the ATU takes normal action when a parity error is detected. When cleared, parity checking is disabled.
05	0_2	VGA Palette Snoop Enable - The ATU interface does not support I/O writes and therefore, does not perform VGA palette snooping.
04	0_2	Memory Write and Invalidate Enable - When set, ATU may generate MWI commands. When clear, ATU use Memory Write commands instead of MWI. Ignored when operating in the PCI-X mode.
03	0_2	Special Cycle Enable - The ATU interface does not respond to special cycle commands in any way. Not implemented and a reserved bit field.
02	0_2	Bus Master Enable - The ATU interface can act as a master on the PCI bus. When cleared, disables the device from generating PCI accesses. When set, allows the device to behave as a PCI bus master. When operating in the PCI-X mode, ATU initiates a split completion transaction regardless of the state of this bit.
01	0_2	Memory Enable - Controls the ATU interface's response to PCI memory addresses. When cleared, the ATU interface does not respond to any memory access on the PCI bus.
00	0_2	I/O Space Enable - Controls the ATU interface response to I/O transactions. Not implemented and a reserved bit field.

3.10.4 ATU Status Register - ATUSR

The ATU Status Register bits adhere to the *PCI Local Bus Specification*, Revision 2.3 definitions. The *read/clear* bits can only be set by internal hardware and cleared by either a reset condition or by writing a 1_2 to the register.

Table 88. ATU Status Register - ATUSR (Sheet 1 of 2)



Internal Bus Address
FFFF.E106H

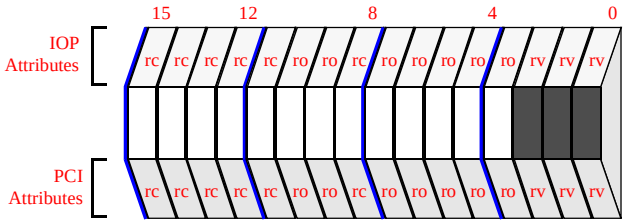
PCI Configuration Address Offset
06H - 07H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0_2	Detected Parity Error - set when a parity error is detected in data received by the ATU on the PCI bus even when the ATUCMD register's Parity Error Response bit is cleared. Set under the following conditions: • Write Data Parity Error when the ATU is a target (inbound write). • Read Data Parity Error when the ATU is a requester (outbound read). • Any Address or Attribute (PCI-X Only) Parity Error on the Bus (including one generated by the ATU).
14	0_2	**SERR#** Asserted - set when **SERR#** is asserted on the PCI bus by the ATU.
13	0_2	Master Abort - set when a transaction initiated by the ATU PCI master interface, ends in a Master-Abort or when the ATU receives a Master Abort Split Completion Error Message in PCI-X mode.
12	0_2	Target Abort (master) - set when a transaction initiated by the ATU PCI master interface, ends in a target abort or when the ATU receives a Target Abort Split Completion Error Message in PCI-X mode.
11	0_2	Target Abort (target) - set when the ATU interface, acting as a target, terminates the transaction on the PCI bus with a target abort.
10:09	01_2	DEVSEL# Timing - These bits are read-only and define the slowest DEVSEL# timing for a target device in Conventional PCI Mode regardless of the operating mode (except configuration accesses). 00_2 = Fast 01_2 = Medium 10_2 = Slow 11_2 = Reserved The ATU interface uses Medium timing.
08	0_2	Master Parity Error - The ATU interface sets this bit under the following conditions: • The ATU asserted **PERR#** itself or the ATU observed **PERR#** asserted. • And the ATU acted as the requester for the operation in which the error occurred. • And the ATUCMD register's Parity Error Response bit is set • Or (PCI-X Mode Only) the ATU received a Write Data Parity Error Message • And the ATUCMD register's Parity Error Response bit is set
07	1_2 (Conventional mode) 0_2 (PCI-X mode)	Fast Back-to-Back - The ATU/Messaging Unit interface is capable of accepting fast back-to-back transactions in Conventional PCI mode when the transactions are not to the same target. Since fast back-to-back transactions do not exist in PCI-X mode, this bit is forced to 0 in the PCI-X mode.
06	0_2	UDF Supported - User Definable Features are not supported
05	1_2	66 MHz. Capable - 66 MHz operation is supported.
04	1_2	**Capabilities** - When set, this function implements extended capabilities.

Table 88. ATU Status Register - ATUSR (Sheet 2 of 2)



Internal Bus Address
FFFF.E106H

PCI Configuration Address Offset
06H - 07H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

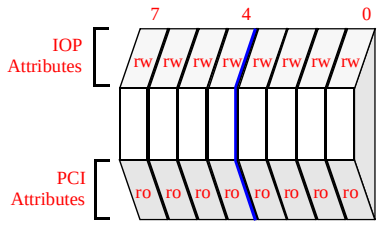
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
03	0	Interrupt Status - reflects the state of the ATU interrupt when the Interrupt Disable bit in the command register is a 0. 0 = ATU interrupt signal deasserted. 1 = ATU interrupt signal asserted. **NOTE:** Setting the Interrupt Disable bit to a 1 has no effect on the state of this bit. Refer to Section 3.10.23, "ATU Interrupt Pin Register - ATUIPR" on page 236 for details on the ATU interrupt signal.
02:00	00000_2	Reserved.

3.10.5 ATU Revision ID Register - ATURID

Revision ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 89. ATU Revision ID Register - ATURID



Internal Bus Address
FFFF.E108H

PCI Configuration Address Offset
08H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

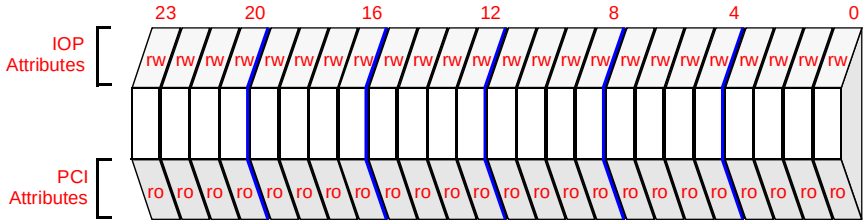
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	00H	ATU Revision - identifies the 80331 revision number.

3.10.6 ATU Class Code Register - ATUCCR

Class Code Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. Auto configuration software reads this register to determine the PCI device function.

Table 90. ATU Class Code Register - ATUCCR



Internal Bus Address
FFFF.E109H

PCI Configuration Address Offset
09H - 0BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

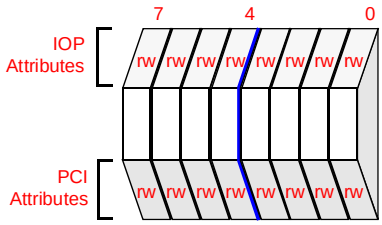
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
23:16	05H	Base Class - Memory Controller
15:08	80H	Sub Class - Other Memory Controller
07:00	00H	Programming Interface - None defined

3.10.7 ATU Cacheline Size Register - ATUCLSR

Cacheline Size Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register is programmed with the system cacheline size in DWORDs (32-bit words). Cacheline Size is restricted to either 0, 8 or 16 DWORDs; the ATU interprets any other value as "0".

Table 91. ATU Cacheline Size Register - ATUCLSR



Internal Bus Address: FFFF.E10CH

PCI Configuration Address Offset: 0CH

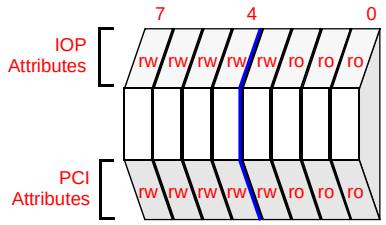
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	00H	ATU Cacheline Size - specifies the system cacheline size in DWORDs. Cacheline size is restricted to either 0, 8 or 16 DWORDs.

3.10.8 ATU Latency Timer Register - ATULT

ATU Latency Timer Register bit definitions apply to the PCI interface.

Table 92. ATU Latency Timer Register - ATULT



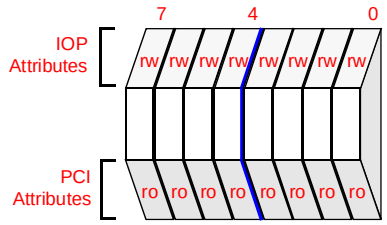
Internal Bus Address	PCI Configuration Address Offset	Attribute Legend:	
FFFF.E10DH	0DH	RV = Reserved	RW = Read/Write
		PR = Preserved	RC = Read Clear
		RS = Read/Set	RO = Read Only
			NA = Not Accessible

Bit	Default	Description
07:03	00000_2 (for Conventional mode) 01000_2 (for PCI-X mode)	Programmable Latency Timer - This field varies the latency timer for the interface from 0 to 248 clocks. The default value is 0 clocks for Conventional PCI mode, and 64 clocks for PCI-X mode.
02:00	000_2	Latency Timer Granularity - These Bits are read only giving a programmable granularity of 8 clocks for the latency timer.

3.10.9 ATU Header Type Register - ATUHTR

Header Type Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register indicates the layout of ATU configuration space bytes 10H to 3FH. The MSB indicates whether or not the device is multi-function.

Table 93. ATU Header Type Register - ATUHTR



Internal Bus Address
FFFF.E10EH

PCI Configuration Address Offset
0EH

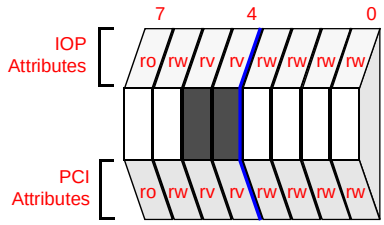
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07	0_2	Single Function/Multi-Function Device - Identifies the 80331 as a single-function PCI device.
06:00	000000_2	PCI Header Type - This bit field indicates the type of PCI header implemented. The ATU interface header conforms to *PCI Local Bus Specification*, Revision 2.3.

3.10.10 ATU BIST Register - ATUBISTR

The ATU BIST Register controls the functions the Intel® XScale™ core performs when BIST is initiated. This register is the interface between the host processor requesting BIST functions and the 80331 replying with the results from the software implementation of the BIST functionality.

Table 94. ATU BIST Register - ATUBISTR



Internal Bus Address
FFFF.E10FH

PCI Configuration Address Offset
0FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07	0_2	BIST Capable - This bit value is always equal to the ATUCR ATU BIST Interrupt Enable bit. See Section 3.10.39, "ATU Configuration Register - ATUCR" on page 252.
06	0_2	Start BIST - When the ATUCR BIST Interrupt Enable bit is set: Setting this bit generates an interrupt to the Intel® XScale™ core to perform a software BIST function. The Intel® XScale™ core clears this bit when the BIST software has completed with the BIST results found in ATUBISTR register bits [3:0]. When the ATUCR BIST Interrupt Enable bit is clear: Setting this bit does not generate an interrupt to the Intel® XScale™ core and no BIST functions is performed. The Intel® XScale™ core does not clear this bit.
05:04	00_2	Reserved
03:00	0000_2	BIST Completion Code - when the ATUCR BIST Interrupt Enable bit is set and the ATUBISTR Start BIST bit is set (bit 6): The Intel® XScale™ core places the results of the software BIST in these bits. A nonzero value indicates a device-specific error.

3.10.11 Inbound ATU Base Address Register 0 - IABAR0

The Inbound ATU Base Address Register 0 (IABAR0) together with the Inbound ATU Upper Base Address Register 0 (IAUBAR0) defines the block of memory addresses where the inbound translation window 0 begins. The inbound ATU decodes and forwards the bus request to the 80331 internal bus with a translated address to map into 80331 local memory. The IABAR0 and IAUBAR0 define the base address and describes the required memory block size; see Section 3.10.21, "Determining Block Sizes for Base Address Registers" on page 233. Bits 31 through 12 of the IABAR0 is either read/write bits or read only with a value of 0 depending on the value located within the IALR0. This configuration allows the IABAR0 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The first 4 Kbytes of memory defined by the IABAR0, IAUBAR0 and the IALR0 is reserved for the Messaging Unit.

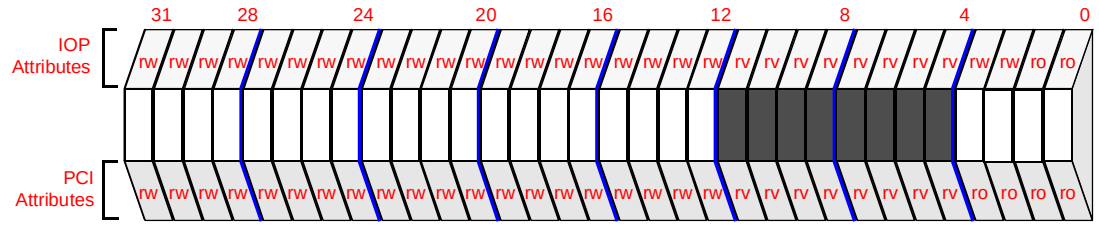
The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Warning: When IALR0 is cleared prior to host configuration, the user should also clear the Prefetchable Indicator and the Type Indicator. Assuming IALR0 is not cleared:

a. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is cleared prior to host configuration, the user should also set the Type Indicator for 32 bit addressability.

b. For compliance to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability. This is the default for IABAR0.

Table 95. Inbound ATU Base Address Register 0 - IABAR0



Internal Bus Address
FFFF.E110H

PCI Configuration Address Offset
10H - 13H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Translation Base Address 0 - These bits define the actual location the translation function is to respond to when addressed from the PCI bus.
11:04	00H	Reserved.
03	1_2	Prefetchable Indicator - When set, defines the memory space as prefetchable.
02:01	10_2	Type Indicator - Defines the width of the addressability for this memory window: 00 - Memory Window is locatable anywhere in 32 bit address space 10 - Memory Window is locatable anywhere in 64 bit address space
00	0_2	Memory Space Indicator - This bit field describes memory or I/O space base address. The ATU does not occupy I/O space, thus this bit must be zero.

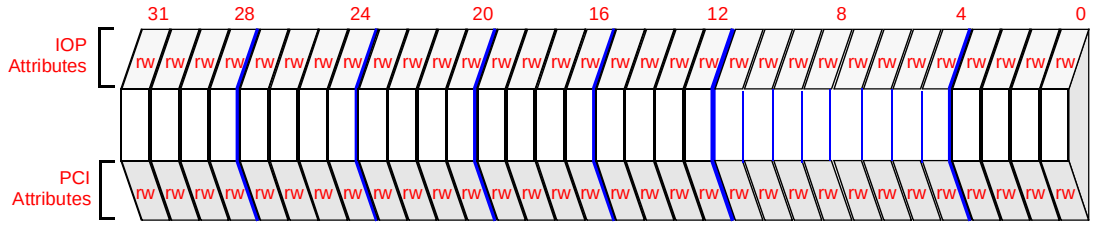
3.10.12 Inbound ATU Upper Base Address Register 0 - IAUBAR0

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR0 is set to indicate 32 bit addressability, the IAUBAR0 register attributes are read-only.

Table 96. Inbound ATU Upper Base Address Register 0 - IAUBAR0



Internal Bus Address
FFFF.E114H

PCI Configuration Address Offset
14H - 17H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:0	00000H	Translation Upper Base Address 0 - Together with the Translation Base Address 0 these bits define the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes.

3.10.13 Inbound ATU Base Address Register 1 - IABAR1

The Inbound ATU Base Address Register (IABAR1) together with the Inbound ATU Upper Base Address Register 1 (IAUBAR1) defines the block of memory addresses where the inbound translation window 1 begins. This window is used merely to allocate memory on the PCI bus and, the ATU does not process any PCI bus transactions to this memory range.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

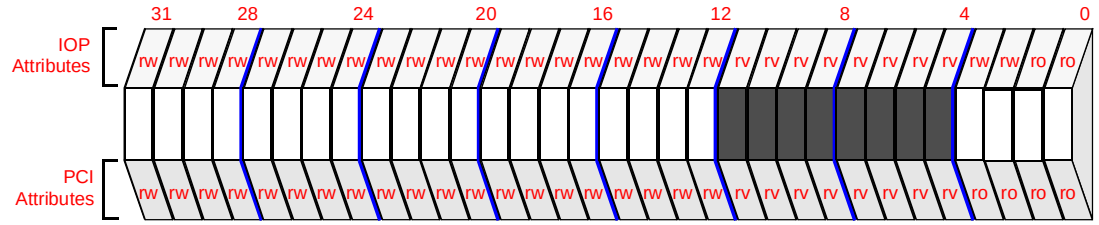
When enabled, the ATU interrupts the Intel® XScale™ core when the IABAR1 register is written from the PCI bus. Please see Section 3.10.41, "ATU Interrupt Status Register - ATUISR" on page 257 for more details.

Warning: When a non-zero value is not written to IALR1 prior to host configuration, the user should not set either the Prefetchable Indicator or the Type Indicator for 64 bit addressability. This is the default for IABAR1. Assuming a non-zero value is written to IALR1, the user may set the Prefetchable Indicator or the Type Indicator:

c. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is not set prior to host configuration, the user should also leave the Type Indicator set for 32 bit addressability. This is the default for IABAR1.

d. For compliance to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability.

Table 97. Inbound ATU Base Address Register 1 - IABAR1



Internal Bus Address
FFFF.E118H

PCI Configuration Address Offset
18H - 1BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Translation Base Address 1 - These bits define the actual location of window 1 on the PCI bus.
11:04	00H	Reserved.
03	0_2	Prefetchable Indicator - When set, defines the memory space as prefetchable.
02:01	00_2	Type Indicator - Defines the width of the addressability for this memory window: 00 - Memory Window is locatable anywhere in 32 bit address space 10 - Memory Window is locatable anywhere in 64 bit address space
00	0_2	Memory Space Indicator - This bit field describes memory or I/O space base address. The ATU does not occupy I/O space, thus this bit must be zero.

3.10.14 Inbound ATU Upper Base Address Register 1 - IAUBAR1

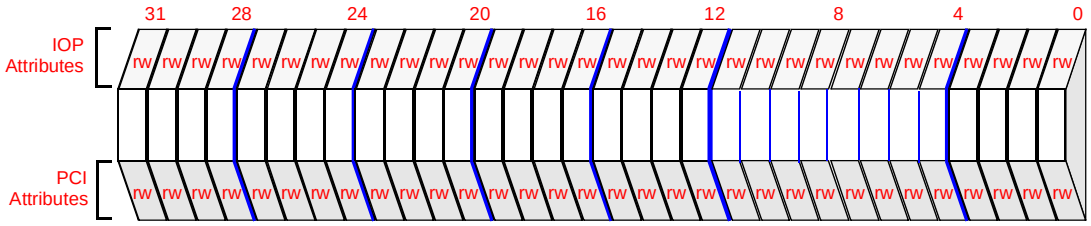
This register contains the upper base address when locating this window for PCI addresses beyond 4 GBytes. Together with the IABAR1 this register defines the actual location for this memory window for addresses > 4GBytes (for DACs). This window is used merely to allocate memory on the PCI bus and, the ATU does not process any PCI bus transactions to this memory range.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

When enabled, the ATU interrupts the Intel® XScale™ core when the IAUBAR1 register is written from the PCI bus. Please see Section 3.10.41, "ATU Interrupt Status Register - ATUISR" on page 257 for more details.

Note: When the Type indicator of IABAR1 is set to indicate 32 bit addressability, the IAUBAR1 register attributes are read-only. This is the default for IABAR1.

Table 98. Inbound ATU Upper Base Address Register 1 - IAUBAR1



Internal Bus Address
FFFF.E11CH

PCI Configuration Address Offset
1CH - 1FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:0	00000H	Translation Upper Base Address 1 - Together with the Translation Base Address 1 these bits define the actual location for this memory window on the PCI bus for addresses > 4GBytes.

3.10.15 Inbound ATU Base Address Register 2 - IABAR2

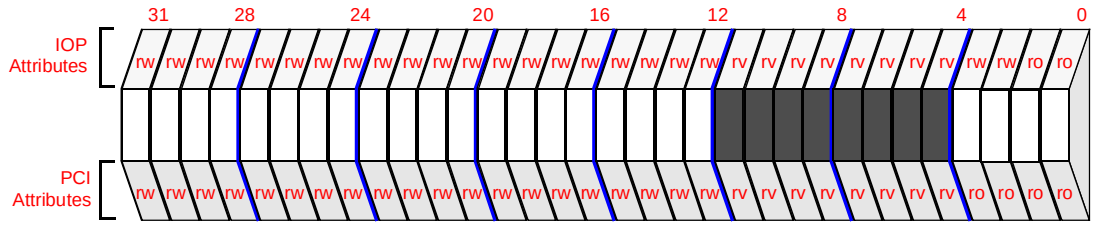
The Inbound ATU Base Address Register 2 (IABAR2) together with the Inbound ATU Upper Base Address Register 2 (IAUBAR2) defines the block of memory addresses where the inbound translation window 2 begins. The inbound ATU decodes and forwards the bus request to the 80331 internal bus with a translated address to map into 80331 local memory. The IABAR2 and IAUBAR2 define the base address and describes the required memory block size; see Section 3.10.21, "Determining Block Sizes for Base Address Registers" on page 233. Bits 31 through 12 of the IABAR2 is either read/write bits or read only with a value of 0 depending on the value located within the IALR2. This configuration allows the IABAR2 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Warning: When a non-zero value is not written to IALR2 prior to host configuration, the user should not set either the Prefetchable Indicator or the Type Indicator for 64 bit addressability. This is the default for IABAR2. Assuming a non-zero value is written to IALR2, the user may set the Prefetchable Indicator or the Type Indicator:

1. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is not set prior to host configuration, the user should also leave the Type Indicator set for 32 bit addressability. This is the default for IABAR2.
2. For compliance to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, when the Prefetchable Indicator is set prior to host configuration, the user should also set the Type Indicator for 64 bit addressability.

Table 99. Inbound ATU Base Address Register 2 - IABAR2



Internal Bus Address
FFFF.E120H

PCI Configuration Address Offset
20H - 23H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Translation Base Address 2 - These bits define the actual location the translation function is to respond to when addressed from the PCI bus.
11:04	00H	Reserved.
03	0_2	Prefetchable Indicator - When set, defines the memory space as prefetchable.
02:01	00_2	Type Indicator - Defines the width of the addressability for this memory window: 00 - Memory Window is locatable anywhere in 32 bit address space 10 - Memory Window is locatable anywhere in 64 bit address space
00	0_2	Memory Space Indicator - This bit field describes memory or I/O space base address. The ATU does not occupy I/O space, thus this bit must be zero.

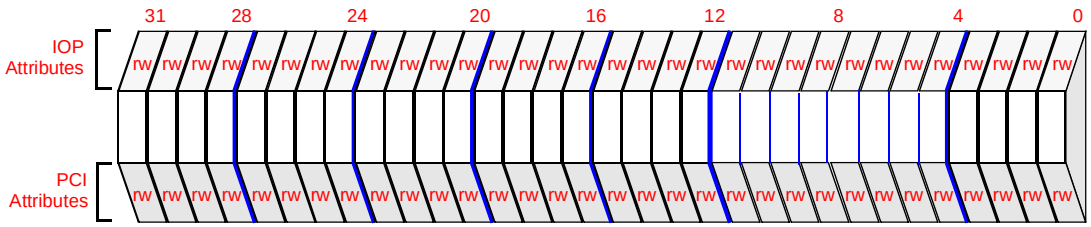
3.10.16 Inbound ATU Upper Base Address Register 2 - IAUBAR2

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR2 is set to indicate 32 bit addressability, the IAUBAR2 register attributes are read-only. This is the default for IABAR2.

Table 100. Inbound ATU Upper Base Address Register 2 - IAUBAR2



Internal Bus Address
FFFF.E124H

PCI Configuration Address Offset
24H - 27H

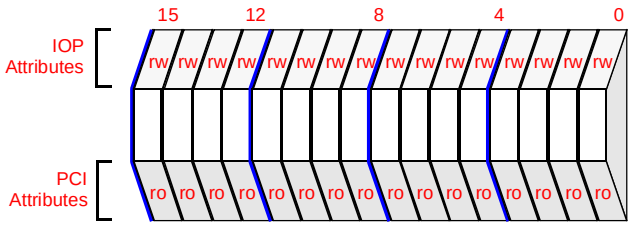
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:0	00000H	Translation Upper Base Address 2 - Together with the Translation Base Address 2 these bits define the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes.

3.10.17 ATU Subsystem Vendor ID Register - ASVIR

ATU Subsystem Vendor ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 101. ATU Subsystem Vendor ID Register - ASVIR



Internal Bus Address
FFFF.E12CH

PCI Configuration Address Offset
2CH - 2DH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

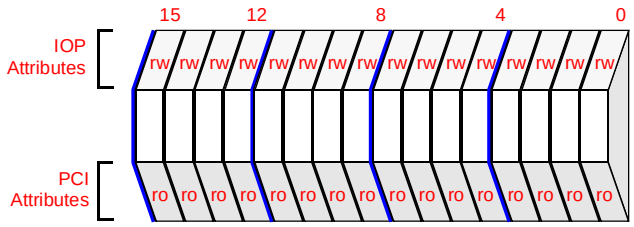
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:0	0000H	Subsystem Vendor ID - This register uniquely identifies the add-in board or subsystem vendor.

3.10.18 ATU Subsystem ID Register - ASIR

ATU Subsystem ID Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3.

Table 102. ATU Subsystem ID Register - ASIR



Internal Bus Address
FFFF.E12EH

PCI Configuration Address Offset
2EH - 2FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

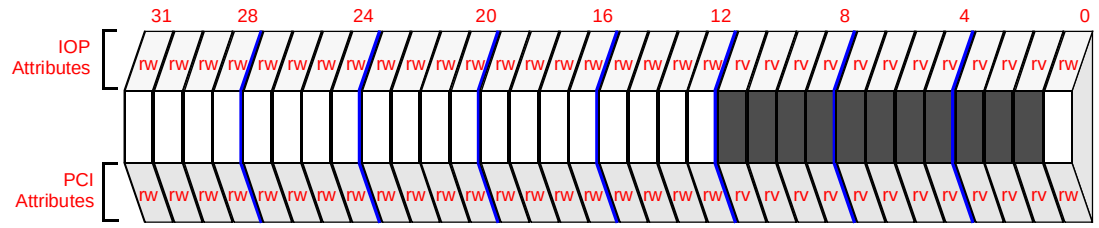
Bit	Default	Description
15:0	0000H	Subsystem ID - uniquely identifies the add-in board or subsystem.

3.10.19 Expansion ROM Base Address Register - ERBAR

The Expansion ROM Base Address Register defines the block of memory addresses used for containing the Expansion ROM. It permits the inclusion of multiple code images, allowing the device to be initialized. The code image supplied consists of either executable code or an interpreted code. Each code image must start on a 512 byte boundary and each must contain the PCI Expansion ROM header. Image placement in ROM space depends on the length of code images which precede it within ROM. ERBAR defines the base address and describes the required memory block size; see Section 3.10.21. Expansion ROM address space (limit size) can be a maximum of 16 MBytes. Bits 31 through 12 of the ERBAR is either read/write bits or read only with a value of 0 depending on the value located within the ERLR. This configuration allows the ERBAR to be programmed per *PCI Local Bus Specification*, Revision 2.3.

The Expansion ROM Base Address Register's programmed value must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming Expansion ROM base address registers.

Table 103. Expansion ROM Base Address Register -ERBAR



Internal Bus Address
FFFF.E130H

PCI Configuration Address Offset
30H - 33H

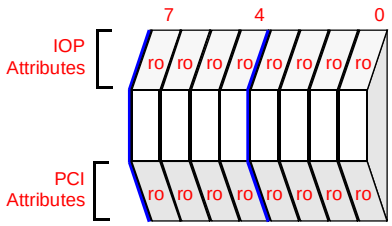
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Expansion ROM Base Address - These bits define the actual location where the Expansion ROM address window resides when addressed from the PCI bus on any 4 Kbyte boundary.
11:01	000H	Reserved
00	0_2	Address Decode Enable - This bit field shows the ROM address decoder is enabled or disabled. When cleared, indicates the address decoder is disabled.

3.10.20 ATU Capabilities Pointer Register - ATU_Cap_Ptr

The Capabilities Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register provides an offset in this function's PCI Configuration Space for the location of the first item in the first Capability list. In the case of the 80331, this is the PCI Bus Power Management extended capability as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 104. ATU Capabilities Pointer Register - ATU_Cap_Ptr



Internal Bus Address
FFFF.E134H

PCI Configuration Address Offset
34H

Attribute Legend:
RW = Read/Write
RV = Reserved
RC = Read Clear
PR = Preserved
RO = Read Only
RS = Read/Set
NA = Not Accessible

36 Bit	Default	Description
07:00	C0H	Capability List Pointer - This provides an offset in this function's configuration space that points to the 80331 PCI Bus Power Management extended capability.

3.10.21 Determining Block Sizes for Base Address Registers

The required address size and type can be determined by writing ones to a base address register and reading from the registers. By scanning the returned value from the least-significant bit of the base address registers upwards, the programmer can determine the required address space size. The binary-weighted value of the first non-zero bit found indicates the required amount of space. Table 105 describes the relationship between the values read back and the byte sizes the base address register requires.

Table 105. Memory Block Size Read Response

Response After Writing all 1s to the Base Address Register	Size (in Bytes)	Response After Writing all 1s to the Base Address Register	Size (in Bytes)
FFFFFFF0H	16	FFF00000H	1 M
FFFFFFE0H	32	FFE00000H	2 M
FFFFFFC0H	64	FFC00000H	4 M
FFFFFF80H	128	FF800000H	8 M
FFFFFF00H	256	FF000000H	16 M
FFFFFE00H	512	FE000000H	32 M
FFFFFC00H	1K	FC000000H	64 M
FFFFF800H	2K	F8000000H	128 M
FFFFF000H	4K	F0000000H	256 M
FFFFE000H	8K	E0000000H	512 M
FFFFC000H	16K	C0000000H	1 G
FFFF8000H	32K	80000000H	2 G
FFFF0000H	64K	00000000H	Register not imple-mented, no address space required.
FFFE0000H	128K		
FFFC0000H	256K		
FFF80000H	512K		

As an example, assume that FFFF.FFFFH is written to the ATU Inbound Base Address Register 0 (IABAR0) and the value read back is FFF0.0008H. Bit zero is a zero, so the device requires memory address space. Bit three is one, so the memory does supports prefetching. Scanning upwards starting at bit four, bit twenty is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicated that the device requires 1 Mbyte of memory space.

The ATU Base Address Registers and the Expansion ROM Base Address Register use their associated limit registers to enable which bits within the base address register are read/write and which bits are read only (0). This allows the programming of these registers in a manner similar to other PCI devices even though the limit is variable.

Table 106. ATU Base Registers and Associated Limit Registers

Base Address Register	Limit Register	Description
Inbound ATU Base Address Register 0	Inbound ATU Limit Register 0	Defines the inbound translation window 0 from the PCI bus.
Inbound ATU Upper Base Address Register 0	N/A	Together with ATU Base Address Register 0 defines the inbound translation window 0 from the PCI bus for DACs.
Inbound ATU Base Address Register 1 [a]	Inbound ATU Limit Register 1	Defines inbound window 1 from the PCI bus.
Inbound ATU Upper Base Address Register 1	N/A	Together with ATU Base Address Register 1 defines inbound window 1 from the PCI bus for DACs.
Inbound ATU Base Address Register 2	Inbound ATU Limit Register 2	Defines the inbound translation window 2 from the PCI bus.
Inbound ATU Upper Base Address Register 2	N/A	Together with ATU Base Address Register 2 defines the inbound translation window 2 from the PCI bus for DACs.
Inbound ATU Base Address Register 3	Inbound ATU Limit Register 3	Defines the inbound translation window 3 from the PCI bus.
Inbound ATU Upper Base Address Register 3	N/A	Together with ATU Base Address Register 3 defines the inbound translation window 3 from the PCI bus for DACs. **NOTE:** This is a private BAR that resides outside of the standard PCI configuration header space (offsets 00H-3FH).
Expansion ROM Base Address Register	Expansion ROM Limit Register	Defines the window of addresses used by a bus master for reading from an Expansion ROM.

a. ATU Inbound Window 1 is **not** a translate window. The ATU does not claim any PCI accesses that fall within this range. This window is used to allocate host memory for use by Private Devices. When enabled, the ATU interrupts the Intel® XScale™ core when either the IABAR1 register or the IAUBAR1 register is written from the PCI bus. Please see Section 3.10.41, "ATU Interrupt Status Register - ATUISR" on page 257 for more details.

3.10.22 ATU Interrupt Line Register - ATUILR

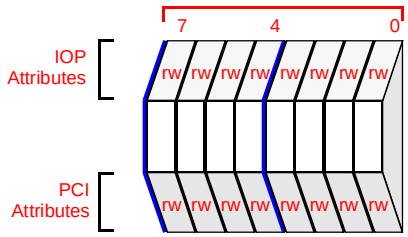
ATU Interrupt Line Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the system interrupt controller's interrupt request lines which connect to the device's PCI interrupt request lines (as specified in the interrupt pin register).

In a PC environment, for example, the register values and corresponding connections are:

- 0 (00H) through 15 (0FH) correspond to IRQ0 through IRQ15
- 16 (10H) through 254 (FEH) are reserved
- 255 (FFH) indicates “unknown” or “no connection”

The operating system or device driver can examine each device’s interrupt pin and interrupt line register to determine which system interrupt request line the device uses to issue requests for service.

Table 107. ATU Interrupt Line Register - ATUILR



Internal Bus Address
FFFF.E13CH

PCI Configuration Address Offset
3CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

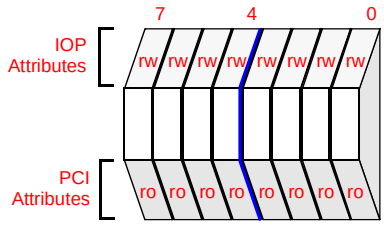
Bit	Default	Description
07:00	FFH	Interrupt Assigned - system-assigned value identifies which system interrupt controller’s interrupt request line connects to the device's PCI interrupt request lines (as specified in the interrupt pin register). A value of FFH signifies “no connection” or “unknown”.

3.10.23 ATU Interrupt Pin Register - ATUIPR

ATU Interrupt Pin Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register identifies the interrupt pin the ATU and Messaging Unit interface uses. The 80331 is, a PCI single-function device and, as such, generates only one interrupt output. The interrupt output is for the Messaging Unit on **INTA#**.

Because the ATU is integrated within 80331 and is attached to the Secondary PCI bus, its **INTA#** interrupt is wired internally to **P_INTC#**. This interrupt wiring coincides with the ATU device number 0xE based on S_AD30 used as the IDSEL input to the ATU device.

Table 108. ATU Interrupt Pin Register - ATUIPR



Internal Bus Address
FFFF.E13DH

PCI Configuration Address Offset
3DH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

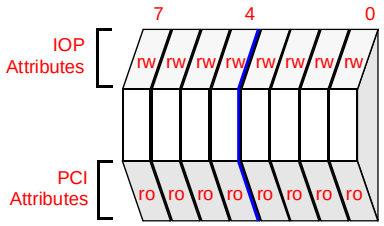
Bit	Default	Description
07:00	01H	Interrupt Used - A value of 01H signifies that the ATU interface unit uses **INTA#** as the interrupt pin.

3.10.24 ATU Minimum Grant Register - ATUMGNT

ATU Minimum Grant Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register specifies the burst period the device requires in increments of 8 PCI clocks.

This register and the ATU Maximum Latency register are information-only registers which the configuration uses to determine how often a bus master typically requires access to the PCI bus and the duration of a typical transfer when it does acquire the bus. This information is useful in determining the values to be programmed into the bus master latency timers and in programming the algorithm to be used by the PCI bus arbiter.

Table 109. ATU Minimum Grant Register - ATUMGNT



Internal Bus Address
FFFF.E13EH

PCI Configuration Address Offset
3EH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

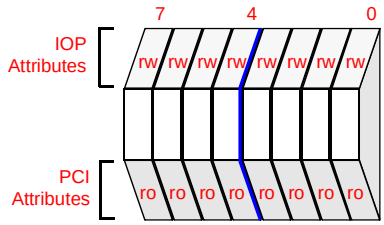
Bit	Default	Description
07:00	80H	This register specifies how long a burst period the device needs in increments of 8 PCI clocks.

3.10.25 ATU Maximum Latency Register - ATUMLAT

ATU Maximum Latency Register bit definitions adhere to *PCI Local Bus Specification*, Revision 2.3. This register specifies how often the device needs to access the PCI bus in increments of 8 PCI clocks.

This register and the Minimum Grant Register are information-only registers which the configuration uses to determine how often a bus master typically requires access to the PCI bus and the duration of a typical transfer when it does acquire the bus. This information is useful in determining the values to be programmed into the bus master latency timers and in programming the algorithm to be used by the PCI bus arbiter.

Table 110. ATU Maximum Latency Register - ATUMLAT



Internal Bus Address
FFFF.E13FH

PCI Configuration Address Offset
3FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	00H	Specifies frequency (how often) the device needs to access the PCI bus in increments of 8 PCI clocks. A zero value indicates the device has no stringent requirement.

3.10.26 Inbound ATU Limit Register 0 - IALR0

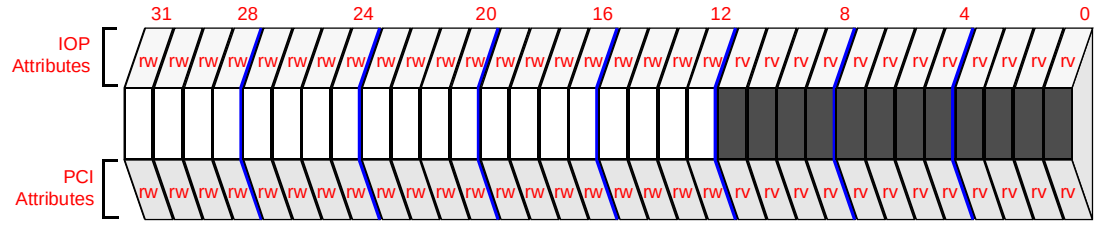
Inbound address translation for memory window 0 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 80331 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The inbound translation base address for inbound window 0 is specified in Section 3.10.11. When determining block size requirements — as described in Section 3.10.21 — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in Section 3.2.1.1.

The 80331 translate value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 80331 translate value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR0 have a direct effect on the IABAR0 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR0 makes the corresponding bit within the IABAR0 a read only bit which always returns 0. A value of 1 in a bit within the IALR0 makes the corresponding bit within the IABAR0 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR0, all writes to the IABAR0 has no effect since a value of all zeros within the IALR0 makes the IABAR0 a read only register.

Table 111. Inbound ATU Limit Register 0 - IALR0



Internal Bus Address
FFFF.E140H

PCI Configuration Address Offset
40H - 43H

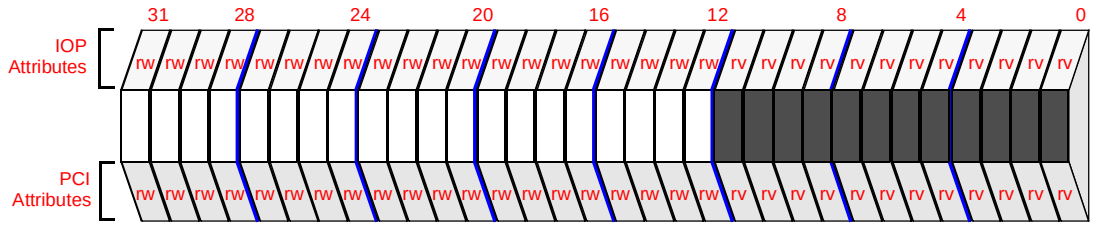
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	FF000H	Inbound Translation Limit 0 - This readback value determines the memory block size required for inbound memory window 0 of the address translation unit. This defaults to an inbound window of 16MB.
11:00	000H	Reserved

3.10.27 Inbound ATU Translate Value Register 0 - IATVR0

The Inbound ATU Translate Value Register 0 (IATVR0) contains the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 112. Inbound ATU Translate Value Register 0 - IATVR0



Internal Bus Address
FFFF.E144H

PCI Configuration Address Offset
44H - 47H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

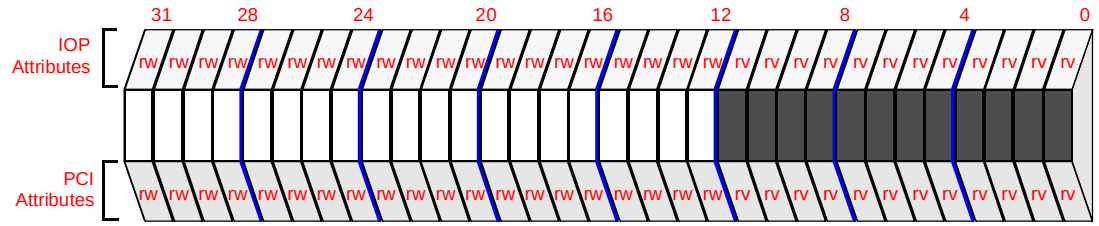
Bit	Default	Description
31:12	FF000H	Inbound ATU Translation Value 0 - This value is used to convert the PCI address to internal bus addresses. This value must be 64-bit aligned on the internal bus. The default address allows the ATU to access the internal 80331 memory-mapped registers.
11:00	000H	Reserved

3.10.28 Expansion ROM Limit Register - ERLR

The Expansion ROM Limit Register (ERLR) defines the block size of addresses the ATU defines as Expansion ROM address space. The block size is programmed by writing a value into the ERLR.

Bits 31 to 12 within the ERLR have a direct effect on the ERBAR register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the ERLR makes the corresponding bit within the ERBAR a read only bit which always returns 0. A value of 1 in a bit within the ERLR makes the corresponding bit within the ERBAR read/write from PCI.

Table 113. Expansion ROM Limit Register - ERLR



Internal Bus Address
FFFF.E148H

PCI Configuration Address Offset
48H - 4BH

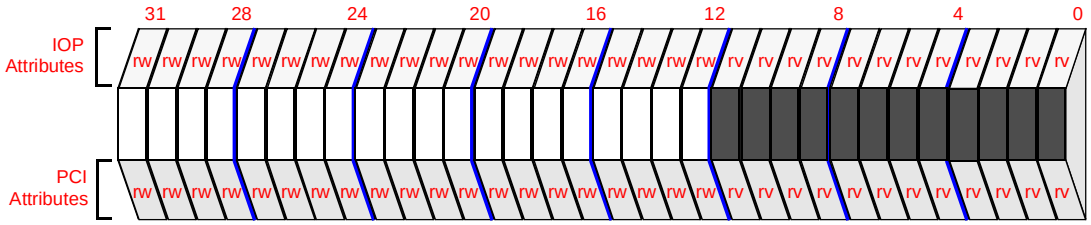
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	000000H	Expansion ROM Limit - Block size of memory required for the Expansion ROM translation unit. Default value is 0, which indicates no Expansion ROM address space and all bits within the ERBAR are read only with a value of 0.
11:00	000H	Reserved.

3.10.29 Expansion ROM Translate Value Register - ERTVR

The Expansion ROM Translate Value Register contains the 80331 internal bus address which the ATU converts the PCI bus access. This address is driven on the internal bus as a result of the Expansion ROM address translation.

Table 114. Expansion ROM Translate Value Register - ERTVR



Internal Bus Address
FFFF.E14CH

PCI Configuration Address Offset
4CH - 4FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Expansion ROM Translation Value - Used to convert PCI addresses to 80331 internal bus addresses for Expansion ROM accesses. The Expansion ROM address translation value must be word aligned on the internal bus.
11:00	000H	Reserved

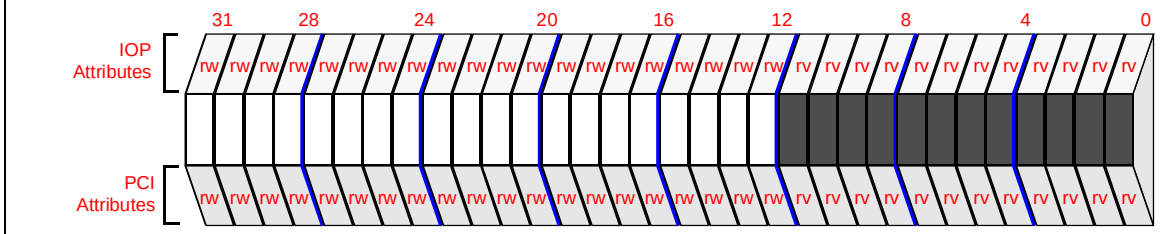
3.10.30 Inbound ATU Limit Register 1 - IALR1

Bits 31 to 12 within the IALR1 have a direct effect on the IABAR1 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR1 makes the corresponding bit within the IABAR1 a read only bit which always returns 0. A value of 1 in a bit within the IALR1 makes the corresponding bit within the IABAR1 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR1, all writes to the IABAR1 has no effect since a value of all zeros within the IALR1 makes the IABAR1 a read only register.

The inbound memory window 1 is used merely to allocate memory on the PCI bus. The ATU does not process any PCI bus transactions to this memory range.

Warning: The ATU does not claim any PCI accesses that fall within the range defined by IABAR1, IAUBAR1, and IALR1.

Table 115. Inbound ATU Limit Register 1 - IALR1



Internal Bus Address
FFFF.E150H

PCI Configuration Address Offset
50H - 53H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Inbound Translation Limit 1 - This readback value determines the memory block size required for the ATUs memory window 1.
11:00	000H	Reserved

3.10.31 Inbound ATU Limit Register 2 - IALR2

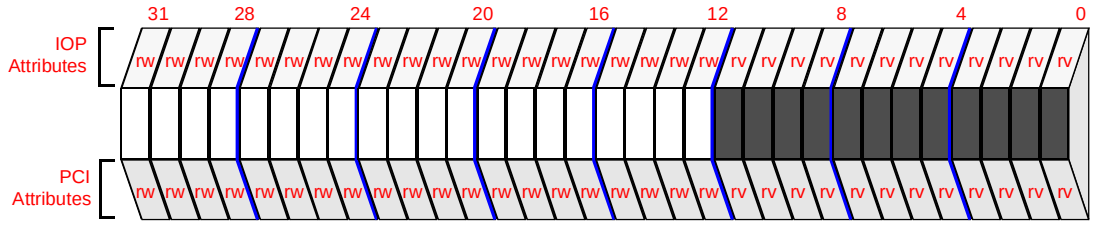
Inbound address translation for memory window 2 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 80331 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The inbound translation base address for inbound window 2 is specified in Section 3.10.15. When determining block size requirements — as described in Section 3.10.21 — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in Section 3.2.1.1.

The 80331 translate value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 80331 translate value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR2 have a direct effect on the IABAR2 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR2 makes the corresponding bit within the IABAR2 a read only bit which always returns 0. A value of 1 in a bit within the IALR2 makes the corresponding bit within the IABAR2 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR2, all writes to the IABAR2 has no effect since a value of all zeros within the IALR2 makes the IABAR2 a read only register.

Table 116. Inbound ATU Limit Register 2 - IALR2



Internal Bus Address
FFFF.E154H

PCI Configuration Address Offset
54H - 57H

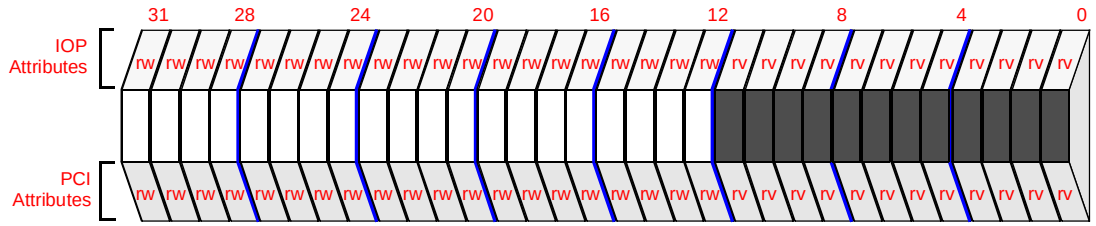
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Inbound Translation Limit 2 - This readback value determines the memory block size required for the ATUs memory window 2.
11:00	000H	Reserved

3.10.32 Inbound ATU Translate Value Register 2 - IATVR2

The Inbound ATU Translate Value Register 2 (IATVR2) contains the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 117. Inbound ATU Translate Value Register 2 - IATVR2



Internal Bus Address
FFFF.E158H

PCI Configuration Address Offset
58H - 5BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

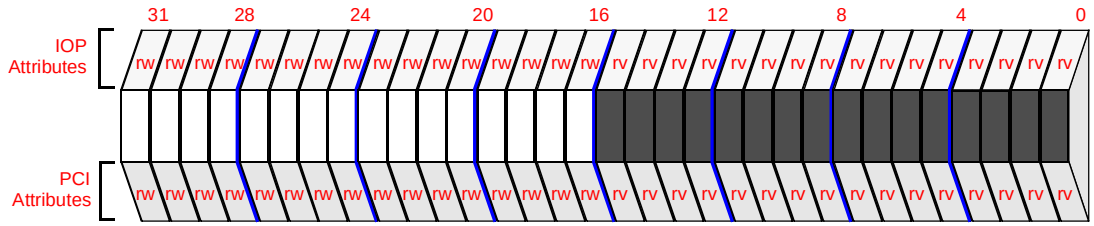
Bit	Default	Description
31:12	00000H	Inbound ATU Translation Value 2 - This value is used to convert the PCI address to internal bus addresses. This value must be 64-bit aligned on the internal bus. The default address allows the ATU to access the internal 80331 memory-mapped registers.
11:00	000H	Reserved

3.10.33 Outbound I/O Window Translate Value Register - OIOWTVR

The Outbound I/O Window Translate Value Register (OIOWTVR) contains the PCI I/O address used to convert the internal bus access to a PCI address. This address is driven on the PCI bus as a result of the outbound ATU address translation. See Section 3.2.2.1, "Outbound Address Translation - Single Address Cycle (SAC) Internal Bus Transactions" on page 149 for details on outbound address translation.

The I/O window is from 80331 internal bus address 9000 000H to 9000 FFFFH with the fixed length of 64 Kbytes.

Table 118. Outbound I/O Window Translate Value Register - OIOWTVR



Internal Bus Address
FFFF.E15CH

PCI Configuration Address Offset
5CH - 5FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

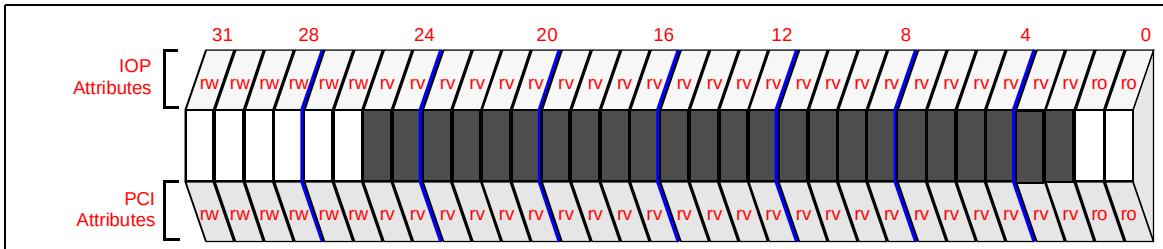
Bit	Default	Description
31:16	0000H	Outbound I/O Window Translate Value - Used to convert internal bus addresses to PCI addresses.
15:00	0000H	Reserved

3.10.34 Outbound Memory Window Translate Value Register 0 - OMWTVR0

The Outbound Memory Window Translate Value Register 0 (OMWTVR0) contains the PCI address used to convert 80331 internal bus addresses for outbound transactions. This address is driven on the PCI bus as a result of the outbound ATU address translation. See Section 3.2.2.1, "Outbound Address Translation - Single Address Cycle (SAC) Internal Bus Transactions" on page 149 for details on outbound address translation.

The memory window is from internal bus address 8000 000H to 83FF FFFFH with the fixed length of 64 Mbytes.

Table 119. Outbound Memory Window Translate Value Register 0- OMWTVR0



Internal Bus Address
FFFF.E160H

PCI Configuration Address Offset
60H - 63H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

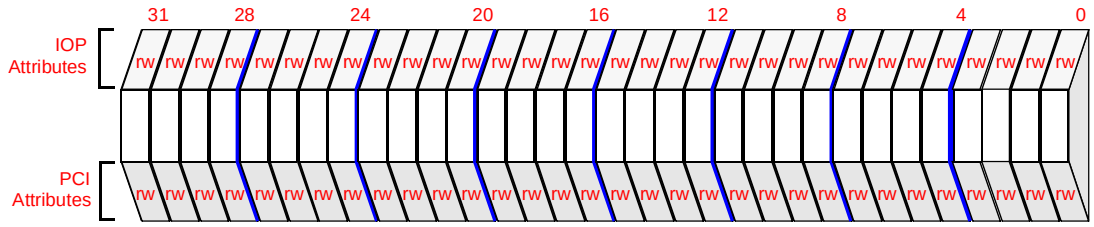
Bit	Default	Description
31:26	00H	Outbound MW Translate Value - Used to convert 80331 internal bus addresses to PCI addresses.
25:02	00 0000H	Reserved
01:00	00_2	Burst Order - This bit field shows the address sequence during a memory burst. Only linear incrementing mode is supported.

3.10.35 Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0

The Outbound Upper 32-bit Memory Window Translate Value Register 0 (OUMWTVR0) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

The memory window is from internal bus address 8000 000H to 83FF FFFFH with the fixed length of 64 Mbytes.

Table 120. Outbound Upper 32-bit Memory Window Translate Value Register 0- OUMWTVR0



Internal Bus Address
FFFF.E164H

PCI Configuration Address Offset
64H - 67H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

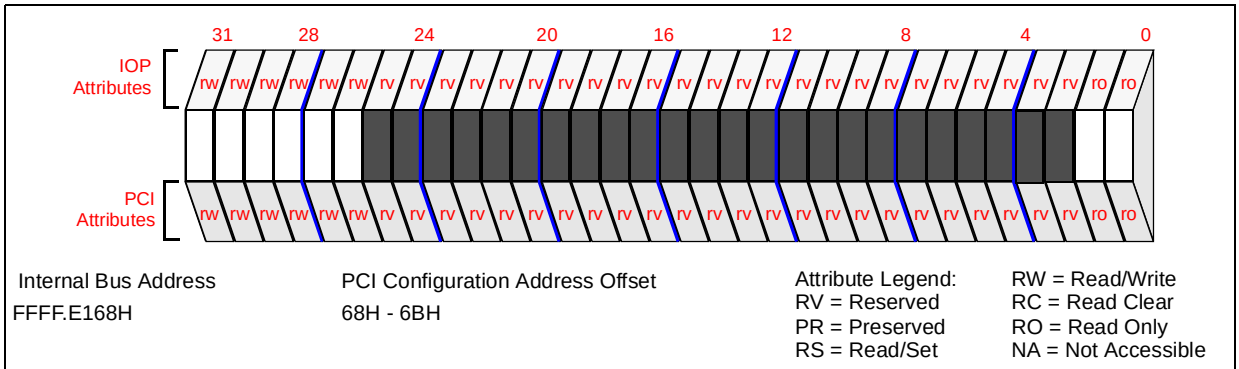
Bit	Default	Description
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).

3.10.36 Outbound Memory Window Translate Value Register 1 - OMWTVR1

The Outbound Memory Window Translate Value Register 1 (OMWTVR1) contains the PCI address used to convert 80331 internal bus addresses for outbound transactions. This address is driven on the PCI bus as a result of the outbound ATU address translation. See Section 3.2.2.1, "Outbound Address Translation - Single Address Cycle (SAC) Internal Bus Transactions" on page 149 for details on outbound address translation.

The memory window is from internal bus address 8400 000H to 87FF FFFFH with the fixed length of 64 Mbytes.

Table 121. Outbound Memory Window Translate Value Register 1- OMWTVR1



Internal Bus Address
FFFF.E168H

PCI Configuration Address Offset
68H - 6BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

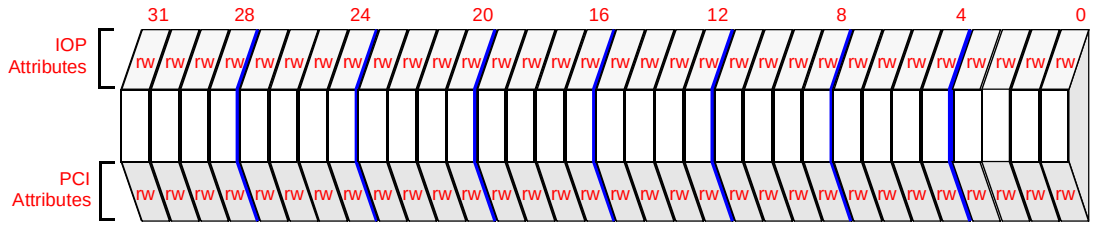
Bit	Default	Description
31:26	00H	Outbound MW Translate Value - Used to convert 80331 internal bus addresses to PCI addresses.
25:02	00 0000H	Reserved
01:00	00_2	Burst Order - This bit field shows the address sequence during a memory burst. Only linear incrementing mode is supported.

3.10.37 Outbound Upper 32-bit Memory Window Translate Value Register 1 - OUMWTVR1

The Outbound Upper 32-bit Memory Window Translate Value Register 1 (OUMWTVR1) defines the upper 32-bits of address used during a dual address cycle. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

The memory window is from internal bus address 8400 000H to 87FF FFFFH with the fixed length of 64 Mbytes.

Table 122. Outbound Upper 32-bit Memory Window Translate Value Register 1- OUMWTVR1



Internal Bus Address
FFFF.E16CH

PCI Configuration Address Offset
6CH - 6FH

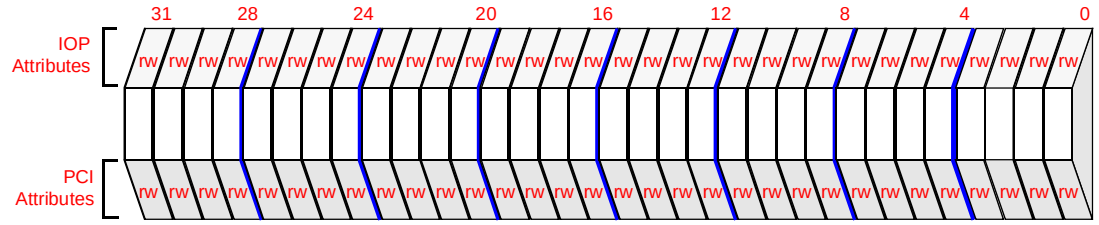
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).

3.10.38 Outbound Upper 32-bit Direct Window Translate Value Register - OUDWTVR

The Outbound Upper 32-bit Direct Window Translate Value Register (OUDWTVR) defines the upper 32-bits of address used during a dual address cycle for the transactions via Direct Addressing Window. This enables the outbound ATU to directly address anywhere within the 64-bit host address space. When this register is all-zero, then a SAC is generated on the PCI bus.

Table 123. Outbound Upper 32-bit Direct Window Translate Value Register - OUDWTVR



Internal Bus Address
FFFF.E178H

PCI Configuration Address Offset
78H - 7BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

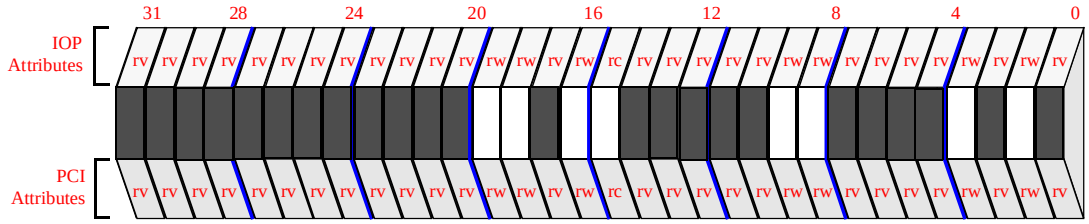
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	These bits define the upper 32-bits of address driven during the dual address cycle (DAC).

3.10.39 ATU Configuration Register - ATUCR

The ATU Configuration Register controls the outbound address translation for address translation unit. It also contains bits for Conventional PCI Delayed Read Command (DRC) aliasing, discard timer status, **SERR#** manual assertion, **SERR#** detection interrupt masking, and ATU BIST interrupt enabling.

Table 124. ATU Configuration Register - ATUCR



Internal Bus Address
FFFF.E180H

PCI Configuration Address Offset
80H - 83H

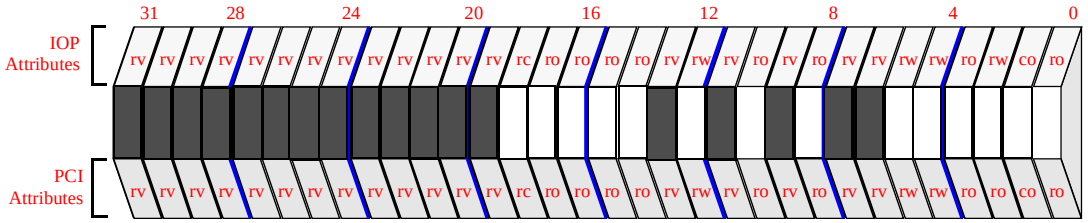
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	00H	Reserved
19	0_2	ATU DRC Alias - when set, the ATU does not distinguish read commands when attempting to match a current PCI read transaction with read data enqueued within the DRC buffer. When clear, a current read transaction must have the exact same read command as the DRR for the ATU to deliver DRC data. Not applicable in the PCI-X mode.
18	0_2	Direct Addressing Upper 2Gbytes Translation Enable - When set, with Direct Addressing enabled (bit 7 of the ATUCR set), the ATU forwards internal bus cycles with an address between 0000.0040H and 7FFF.FFFFH to the PCI bus with bit 31 of the address set (8000.0000H - FFFF.FFFFH). When clear, no translation occurs.
17	0_2	Reserved
16	0_2	**SERR#** Manual Assertion - when set, the ATU asserts **SERR#** for one clock on the PCI interface. Until cleared, **SERR#** may not be manually asserted again. Once cleared, operation proceeds as specified.
15	0_2	ATU Discard Timer Status - when set, one of the 4 discard timers within the ATU has expired and discarded the delayed completion transaction within the queue. When clear, no timer has expired.
14:10	00000_2	Reserved
09	0_2	**SERR#** Detected Interrupt Enable - When set, the Intel® XScale™ core is signalled an **HPI#** interrupt when the ATU detects that **SERR#** was asserted. When clear, the Intel® XScale™ core is not interrupted when **SERR#** is detected.
08	0_2	Direct Addressing Enable - Setting this bit enables direct outbound addressing through the ATU. Internal bus cycles with an address between 0000.0040H and 7FFF.FFFFH automatically forwards to the PCI bus with or without translation of address bit 31 based on the setting of bit 18 of the ATUCR.
07:04	0000_2	Reserved
03	0_2	ATU BIST Interrupt Enable - When set, enables an interrupt to the Intel® XScale™ core when the start BIST bit is set in the ATUBISTR register. This bit is also reflected as the BIST Capable bit 7 in the ATUBISTR register.
02	0_2	Reserved
01	0_2	Outbound ATU Enable - When set, enables the outbound address translation unit. When cleared, disables the outbound ATU.
00	0_2	Reserved

3.10.40 PCI Configuration and Status Register - PCSR

The PCI Configuration and Status Register has additional bits for controlling and monitoring various features of the PCI bus interface.

Table 125. PCI Configuration and Status Register - PCSR (Sheet 1 of 4)



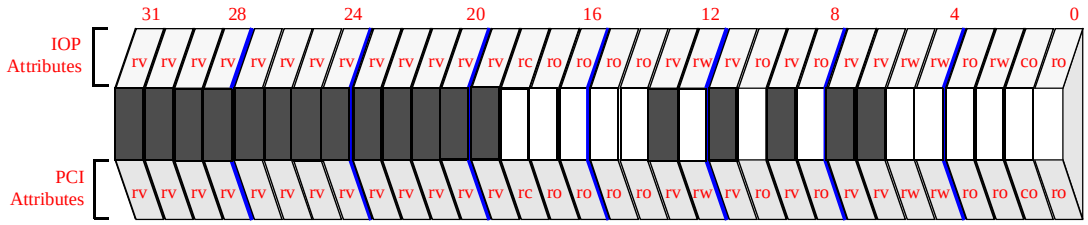
Internal Bus Address
FFFF.E184H

PCI Configuration Address Offset
84H - 87H

Attribute Legend:
RV = Reserved
CO = Clear Only
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:19	0000H	Reserved
18	0_2	Detected Address or Attribute Parity Error - set when a parity error is detected during either the address or attribute phase of a transaction on the PCI bus even when the ATUCMD register Parity Error Response bit is cleared. Set under the following conditions: • Any Address or Attribute (PCI-X Only) Parity Error on the Bus (including one generated by the ATU).
17:16	Varies with external state of DEVSEL#, STOP#, and TRDY#, during **P_RST#**	PCI-X capability - These two bits define the mode of the PCI bus (conventional or PCI-X) as well as the operating frequency in the case of PCI-X mode. 00 - Conventional PCI mode 01 - PCI-X 66 10 - PCI-X 100 11 - PCI-X 133 As defined by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, the operating mode is determined by an initialization pattern on the PCI bus during **P_RST#** assertion: DEVSEL# STOP# TRDY# **Mode** Deasserted Deasserted Deasserted Conventional Deasserted Deasserted Asserted PCI-X 66 Deasserted Asserted Deasserted PCI-X 100 Deasserted Asserted Asserted PCI-X 133 All other patterns are reserved.
15	0_2	Outbound Transaction Queue Busy: 0 = Outbound Transaction Queue Empty 1 = Outbound Transaction Queue Busy
14	0_2	Inbound Transaction Queue Busy: 0 = Inbound Transaction Queue Empty 1 = Inbound Transaction Queue Busy
13	0_2	Reserved.
12	0_2	Discard Timer Value - This bit controls the time-out value for the four discard timers attached to the queues holding read data. A value of 0 indicates the time-out value is 2^{15} clocks. A value of 1 indicates the time-out value is 2^{10} clocks.
11	0_2	Reserved.

Table 125. PCI Configuration and Status Register - PCSR (Sheet 2 of 4)



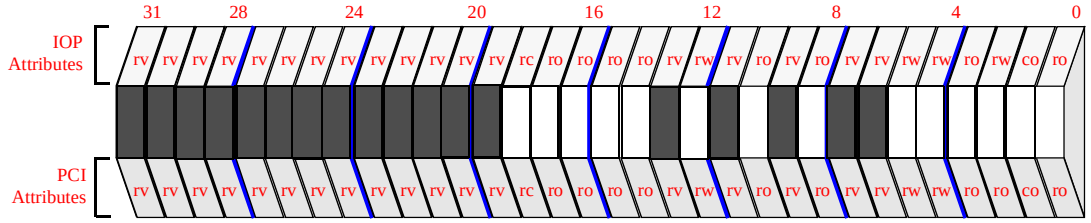
Internal Bus Address
FFFF.E184H

PCI Configuration Address Offset
84H - 87H

Attribute Legend:
RV = Reserved
CO = Clear Only
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
10	Varies with external state of **M66EN** during **P_RST#**	Bus Operating at 66 MHz - When set, the interface has been initialized to function at 66 MHz in Conventional PCI mode by the assertion of **M66EN** during bus initialization. When clear, the interface has been initialized as a 33 MHz bus. **NOTE:** When PCSR bits 17:16 are not equal to zero, then this bit is meaningless since the 80331 is operating in PCI-X mode.
09	0_2	Reserved
08	Varies with external state of **REQ64#** during **P_RST#**	PCI Bus 64-Bit Capable - When clear, the PCI bus interface has been configured as 64-bit capable by the assertion of **REQ64#** on the rising edge of **P_RST#**. When set, the PCI interface is configured as 32-bit only.
07:06	00_2	Reserved.

Table 125. PCI Configuration and Status Register - PCSR (Sheet 3 of 4)



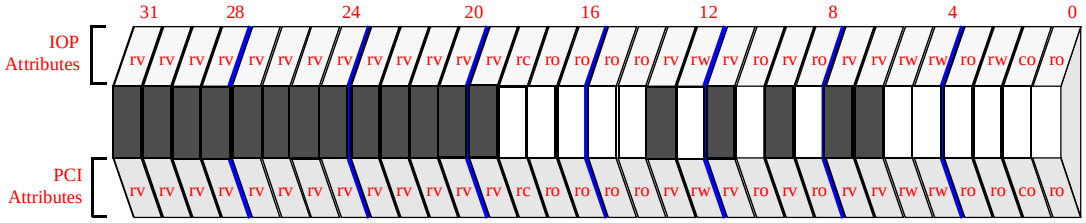
Internal Bus Address
FFFF.E184H

PCI Configuration Address Offset
84H - 87H

Attribute Legend:
RV = Reserved
CO = Clear Only
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
05	0_2	Reset Internal Bus - This bit controls the reset of the Intel® XScale™ core and all units on the internal bus. In addition to the internal bus initialization, this bit triggers the assertion of the **M_RST#** pin for initialization of registered DIMMs. When set: When operating in the conventional PCI mode: • All current PCI transactions being mastered by the ATU completes, and the ATU master interfaces proceeds to an idle state. No additional transactions is mastered by these units until the internal bus reset is complete. • All current transactions being slaved by the ATU on either the PCI bus or the internal bus completes, and the ATU target interfaces proceeds to an idle state. All future slave transactions master aborts, with the exception of the completion cycle for the transaction that set the Reset Internal Bus bit in the PCSR. • When the value of the Core Processor Reset bit in the PCSR (upon **P_RST#** assertion) is set, the Intel® XScale™ core is held in reset when the internal bus reset is complete. • The ATU ignores configuration cycles, and they appears as master aborts for: 32 Internal Bus clocks. • The 80331 hardware clears this bit after the reset operation completes. When operating in the PCI-X mode: The ATU hardware responds the same as in Conventional PCI-X mode. However, this may create a problem in PCI-X mode for split requests in that there may still be an outstanding split completion that the ATU is either waiting to receive (Outbound Request) or initiate (Inbound Read Request). For a cleaner internal bus reset, host software can take the following steps prior to asserting Reset Internal bus: 1. Clear the Bus Master (bit 2 of the ATUCMD) and the Memory Enable (bit 1 of the ATUCMD) bits in the ATUCMD. This ensures that no new transactions, either outbound or inbound are enqueued. 2. Wait for both the Outbound (bit 15 of the PCSR) and Inbound Read (bit 14 of the PCSR) Transaction queue busy bits to be clear. 3. Set the Reset Internal Bus bit As a result, the ATU hardware resets the internal bus using the same logic as in conventional mode, however the user is now assured that the ATU no longer has any pending inbound or outbound split completion transactions. **NOTE:** Since the Reset Internal Bus bit is set using an inbound configuration cycle, the user is guaranteed that any prior configuration cycles have properly completed since there is only a one deep transaction queue for configuration transaction requests. The ATU sends the appropriate Split Write Completion Message to the Requester prior to the onset of Internal Bus Reset.

Table 125. PCI Configuration and Status Register - PCSR (Sheet 4 of 4)



Internal Bus Address
FFFF.E184H

PCI Configuration Address Offset
84H - 87H

Attribute Legend:
RV = Reserved
CO = Clear Only
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
04	0_2	Bus Master Indicator Enable: Provides software control for the Bus Master Indicator signal **P_BMI** used for external RAIDIOS logic control of private devices. Only valid when operating with the bridge and central resource/arbiter disabled (**BRG_EN** =low, **ARB_EN** = low).
03	Varies with external state of **PRIVDEV** during **P_RST#**	Private Device Enable - This bit indicates the state of the reset strap which enables the private device control mechanism within the PCI-to-PCI Bridge SISR configuration register. 0 = Private Device control Disabled - SISR register bits default to zero 1 = Private Device control Enabled - SISR register bits default to one
02	Varies with external state of **RETRY** during **P_RST#**	Configuration Cycle Retry - When this bit is set, the PCI interface of the 80331 responds to all configuration cycles with a Retry condition. When clear, the 80331 responds to the appropriate configuration cycles. The default condition for this bit is based on the external state of the **RETRY** pin at the rising edge of **P_RST#**. When the external state of the pin is high, the bit is set. When the external state of the pin is low, the bit is cleared.
01	Varies with external state of **CORE_RST# during P_RST#**	Core Processor Reset - This bit is set to its default value by the hardware when either **P_RST#** is asserted or the Reset Internal Bus bit in PCSR is set. When this bit is set, the Intel® XScale™ core is being held in reset. Software cannot set this bit. Software is required to clear this bit to deassert Intel® XScale™ core reset. The default condition for this bit is based on the external state of the **CORE_RST#** pin at the rising edge of **P_RST#**. When the external state of the pin is low, the bit is set. When the external state of the pin is high, the bit is clear.
00	Varies with external state of **PRIVMEM** during **P_RST#**	Private Memory Enable - This bit indicates the state of the reset strap which enables the private device control mechanism within the PCI-to-PCI Bridge SDER configuration register. 0 = Private Memory control Disabled - SDER register bit 2 default to zero 1 = Private Memory control Enabled - SDER register bits 2 default to one

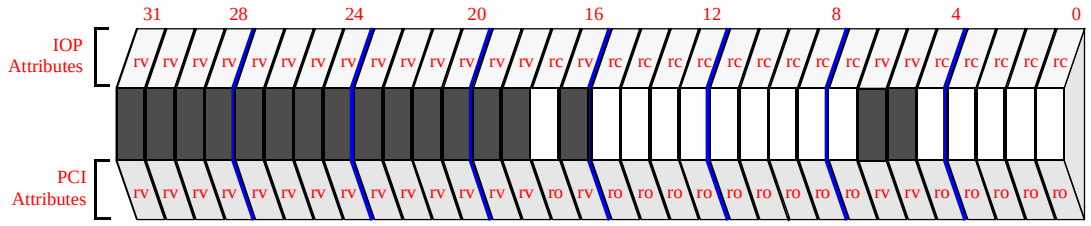
3.10.41 ATU Interrupt Status Register - ATUISR

The ATU Interrupt Status Register is used to notify the core processor of the source of an ATU interrupt. In addition, this register is written to clear the source of the interrupt to the interrupt unit of the 80331. All bits in this register are Read/Clear.

Bits 4:0 are a direct reflection of bits 14:11 and bit 8 (respectively) of the ATU Status Register (these bits are set at the same time by hardware but need to be cleared independently). Bit 7 is set by an error associated with the internal bus of the 80331. Bit 8 is for software BIST. The conditions that result in an ATU interrupt are cleared by writing a 1 to the appropriate bits in this register.

Note: Bits 4:0, and bits 15 and 13:7 can result in an interrupt being driven to the Intel® XScale™ core.

Table 126. ATU Interrupt Status Register - ATUISR (Sheet 1 of 2)



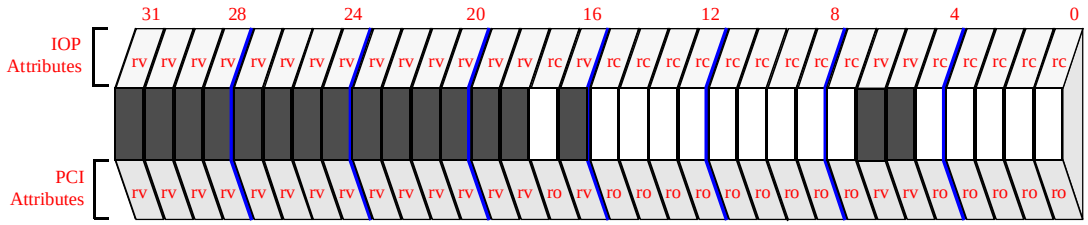
Internal Bus Address
FFFF.E188H

PCI Configuration Address Offset
88H - 8BH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:18	0000H	Reserved
17	0_2	VPD Address Register Updated - This bit is set when a PCI bus configuration write occurs to the VPDAR register. Configuration register writes to the VPDAR does NOT result in bit 15 also being set. When set, this bit results in the assertion of the ATU Configure Register Write Interrupt.
16	0_2	Reserved
15	0_2	ATU Configuration Write - This bit is set when a PCI bus configuration write occurs to any ATU register. When set, this bit results in the assertion of the ATU Configure Register Write Interrupt.
14	0_2	ATU Inbound Memory Window 1 Base Updated - This bit is set when a PCI bus configuration write occurs to either the IABAR1 register or the IAUBAR1 register. Configuration register writes to these registers deos NOT result in bit 15 also being set. When set, this bit results in the assertion of the ATU Configure Register Write Interrupt.
13	0_2	Initiated Split Completion Error Message - This bit is set when the device initiates a Split Completion Message on the PCI Bus with the Split Completion Error attribute bit set.
12	0_2	Received Split Completion Error Message - This bit is set when the device receives a Split Completion Message from the PCI Bus with the Split Completion Error attribute bit set.
11	0_2	Power State Transition - When the Power State Field of the ATU Power Management Control/Status Register is written to transition the ATU function Power State from D0 to D3, D0 to D1, or D3 to D0 and the ATU Power State Transition Interrupt mask bit is cleared, this bit is set.
10	0_2	**P_SERR#** Asserted - set when **P_SERR#** is asserted on the PCI bus by the ATU.

Table 126. ATU Interrupt Status Register - ATUISR (Sheet 2 of 2)



Internal Bus Address
FFFF.E188H

PCI Configuration Address Offset
88H - 8BH

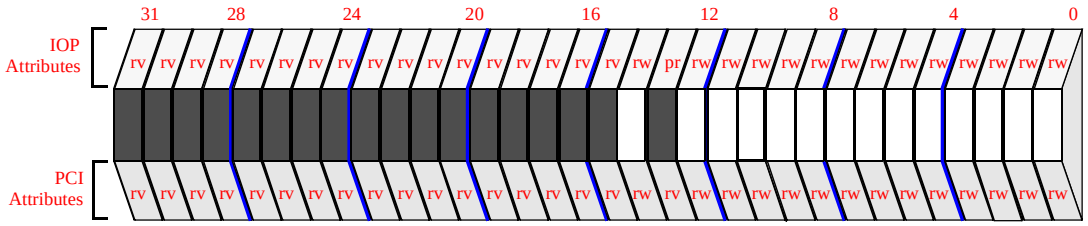
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
09	0_2	Detected Parity Error - set when a parity error is detected on the PCI bus even when the ATUCMD register's Parity Error Response bit is cleared. Set under the following conditions: • Write Data Parity Error when the ATU is a target (inbound write). • Read Data Parity Error when the ATU is an initiator (outbound read). • Any Address or Attribute (PCI-X Only) Parity Error on the Bus.
08	0_2	ATU BIST Interrupt - When set, generates the ATU BIST Start Interrupt and indicates the host processor has set the Start BIST bit (ATUBISTR register bit 6), when the ATU BIST interrupt is enabled (ATUCR register bit 3). The Intel® XScale™ core can initiate the software BIST and store the result in ATUBISTR register bits 3:0. Configuration register writes to the ATUBISTR does NOT result in bit 15 also being set or the assertion of the ATU Configure Register Write Interrupt.
07	0_2	Internal Bus Master Abort - set when a transaction initiated by the ATU internal bus initiator interface ends in a Master-abort.
06:05	00_2	Reserved.
04	0_2	**P_SERR#** Detected - set when **P_SERR#** is detected on the PCI bus by the ATU.
03	0_2	PCI Master Abort - set when a transaction initiated by the ATU PCI initiator interface ends in a Master-abort.
02	0_2	PCI Target Abort (master) - set when a transaction initiated by the ATU PCI master interface ends in a Target-abort.
01	0_2	PCI Target Abort (target) - set when the ATU interface, acting as a target, terminates the transaction on the PCI bus with a target abort.
00	0_2	PCI Master Parity Error - Master Parity Error - The ATU interface sets this bit under the following conditions: • The ATU asserted **PERR#** itself or the ATU observed **PERR#** asserted. • And the ATU acted as the requester for the operation in which the error occurred. • And the ATUCMD register's Parity Error Response bit is set • Or (PCI-X Mode Only) the ATU received a Write Data Parity Error Message • And the ATUCMD register's Parity Error Response bit is set

3.10.42 ATU Interrupt Mask Register - ATUIMR

The ATU Interrupt Mask Register contains the control bit to enable and disable interrupts generated by the ATU.

Table 127. ATU Interrupt Mask Register - ATUIMR (Sheet 1 of 2)



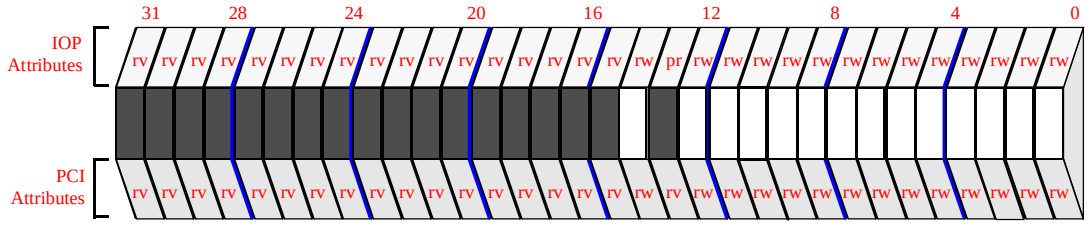
Internal Bus Address
FFFF.E18CH

PCI Configuration Address Offset
8CH - 8FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:15	0 0000H	Reserved
14	0_2	VPD Address Register Updated Mask - Controls the setting of bit 17 of the ATUISR and generation of the ATU Configuration Register Write interrupt when a PCI bus write occurs to the VPDAR register. 0 = Not Masked 1 = Masked
13	0_2	Reserved
12	0_2	Configuration Register Write Mask - Controls the setting of bit 15 of the ATUISR and generation of the ATU Configuration Register Write interrupt when a PCI bus write occurs to any ATU configuration register except those covered by mask bit 11 and bit 14 of this register, and ATU BIST enable bit 3 of the ATUCR. 0 = Not Masked 1 = Masked
11	1_2	ATU Inbound Memory Window 1 Base Updated Mask - Controls the setting of bit 14 of the ATUISR and generation of the ATU Configuration Register Write interrupt when a PCI bus write occurs to either the IABAR1 register or the IAUBAR1 register. 0 = Not Masked 1 = Masked
10	0_2	Initiated Split Completion Error Message Interrupt Mask - Controls the setting of bit 13 of the ATUISR and generation of the ATU Error interrupt when the ATU initiates a Split Completion Error Message. 0 = Not Masked 1 = Masked
09	0_2	Received Split Completion Error Message Interrupt Mask- Controls the setting of bit 12 of the ATUISR and generation of the ATU Error interrupt when a Split Completion Error Message results in bit 29 of the PCIXSR being set. 0 = Not Masked 1 = Masked
08	1_2	Power State Transition Interrupt Mask - Controls the setting of bit 12 of the ATUISR and generation of the ATU Error interrupt when ATU Power Management Control/Status Register is written to transition the ATU Function Power State from D0 to D3, D0 to D1, D1 to D3 or D3 to D0. 0 = Not Masked 1 = Masked
07	0_2	ATU Detected Parity Error Interrupt Mask - Controls the setting of bit 9 of the ATUISR and generation of the ATU Error interrupt when a parity error detected on the PCI bus that sets bit 15 of the ATUSR. 0 = Not Masked 1 = Masked

Table 127. ATU Interrupt Mask Register - ATUIMR (Sheet 2 of 2)



Internal Bus Address
FFFF.E18CH

PCI Configuration Address Offset
8CH - 8FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
06	0_2	ATU **SERR#** Asserted Interrupt Mask - Controls the setting of bit 10 of the ATUISR and generation of the ATU Error interrupt when **SERR#** is asserted on the PCI interface resulting in bit 14 of the ATUSR being set. 0 = Not Masked 1 = Masked **NOTE:** This bit is specific to the ATU asserting **SERR#** and not detecting **SERR#** from another master.
05	0_2	ATU PCI Master Abort Interrupt Mask - Controls the setting of bit 3 of the ATUISR and generation of the ATU Error interrupt when a master abort error resulting in bit 13 of the ATUSR being set. 0 = Not Masked 1 = Masked
04	0_2	ATU PCI Target Abort (Master) Interrupt Mask- Controls the setting of bit 12 of the ATUISR and ATU Error generation of the interrupt when a target abort error resulting in bit 12 of the ATUSR being set 0 = Not Masked 1 = Masked
03	0_2	ATU PCI Target Abort (Target) Interrupt Mask- Controls the setting of bit 1 of the ATUISR and generation of the ATU Error interrupt when a target abort error resulting in bit 11 of the ATUSR being set. 0 = Not Masked 1 = Masked
02	0_2	ATU PCI Master Parity Error Interrupt Mask - Controls the setting of bit 0 of the ATUISR and generation of the ATU Error interrupt when a parity error resulting in bit 8 of the ATUSR being set. 0 = Not Masked 1 = Masked
01	0_2	ATU Inbound Error **SERR#** Enable - Controls when the ATU asserts (when enabled through the ATUCMD) **SERR#** on the PCI interface in response to a master abort on the internal bus during an inbound write transaction. 0 = **SERR#** Not Asserted due to error 1 = **SERR#** Asserted due to error
00	0_2	ATU ECC Target Abort Enable - Controls the ATU response on the PCI interface to a target abort (ECC error) from the memory controller on the internal bus. In conventional mode, this action only occurs during an inbound read transaction where the data phase that was target aborted on the internal bus is actually requested from the inbound read queue. 0 = Disconnect with data (the data being up to 64 bits of 1's) 1 = Target Abort **NOTE:** In PCI-X Mode, The ATU initiates a Split Completion Error Message (with message class=2h - completer error and message index=81h - 80331 internal bus target abort) on the PCI bus, independent of the setting of this bit.

3.10.43 Inbound ATU Base Address Register 3 - IABAR3

The Inbound ATU Base Address Register 3 (IABAR3) together with the Inbound ATU Upper Base Address Register 3 (IAUBAR3) defines the block of memory addresses where the inbound translation window 3 begins. The inbound ATU decodes and forwards the bus request to the 80331 internal bus with a translated address to map into 80331 local memory. The IABAR3 and IAUBAR3 define the base address and describes the required memory block size; see Section 3.10.21, "Determining Block Sizes for Base Address Registers" on page 233. Bits 31 through 12 of the IABAR3 is either read/write bits or read only with a value of 0 depending on the value located within the IALR3. This configuration allows the IABAR3 to be programmed per *PCI Local Bus Specification*, Revision 2.3.

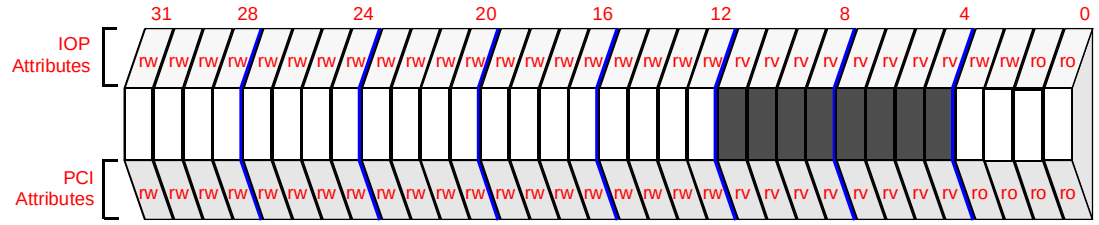
The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: Since IABAR3 does not appear in the standard PCI configuration header space (offsets 00H - 3CH), IABAR3 is not configured by the host during normal system initialization.

Warning: When a non-zero value is not written to IALR3, the user should not set either the Prefetchable Indicator or the Type Indicator for 64 bit addressability. This is the default for IABAR3. Assuming a non-zero value is written to IALR3, the user may set the Prefetchable Indicator or the Type Indicator:

1. Since non prefetchable memory windows can never be placed above the 4 Gbyte address boundary, when the Prefetchable Indicator is not set, the user should also leave the Type Indicator set for 32 bit addressability. This is the default for IABAR3.
2. For compliance to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a, when the Prefetchable Indicator is set, the user should also set the Type Indicator for 64 bit addressability.

Table 128. Inbound ATU Base Address Register 3 - IABAR3



Internal Bus Address
FFFF.E190H

PCI Configuration Address Offset
90H - 93H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Translation Base Address 3 - These bits define the actual location the translation function is to respond to when addressed from the PCI bus.
11:04	00H	Reserved.
03	0_2	Prefetchable Indicator - When set, defines the memory space as prefetchable.
02:01	00_2	Type Indicator - Defines the width of the addressability for this memory window: 00 - Memory Window is locatable anywhere in 32 bit address space 10 - Memory Window is locatable anywhere in 64 bit address space
00	0_2	Memory Space Indicator - This bit field describes memory or I/O space base address. The ATU does not occupy I/O space, thus this bit must be zero.

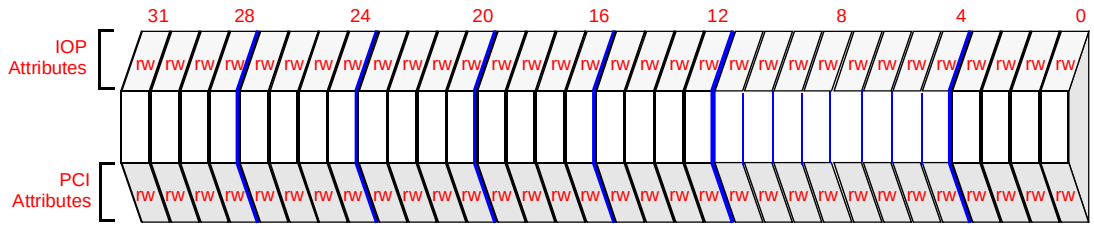
3.10.44 Inbound ATU Upper Base Address Register 3 - IAUBAR3

This register contains the upper base address when decoding PCI addresses beyond 4 GBytes. Together with the Translation Base Address this register defines the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes (for DACs).

The programmed value within the base address register must comply with the PCI programming requirements for address alignment. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Note: When the Type indicator of IABAR3 is set to indicate 32 bit addressability, the IAUBAR3 register attributes are read-only. This is the default for IABAR3.

Table 129. Inbound ATU Upper Base Address Register 3 - IAUBAR3



Internal Bus Address
FFFF.E194H

PCI Configuration Address Offset
94H - 97H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:0	00000H	Translation Upper Base Address 3 - Together with the Translation Base Address 3 these bits define the actual location the translation function is to respond to when addressed from the PCI bus for addresses > 4GBytes.

3.10.45 Inbound ATU Limit Register 3 - IALR3

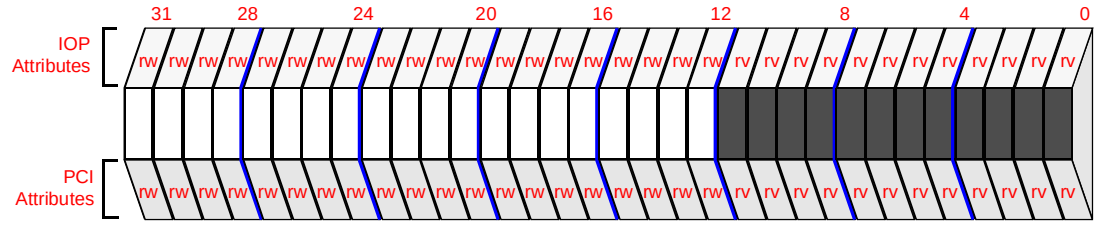
Inbound address translation for memory window 3 occurs for data transfers occurring from the PCI bus (originated from the PCI bus) to the 80331 internal bus. The address translation block converts PCI addresses to internal bus addresses.

The inbound translation base address for inbound window 3 is specified in Section 3.10.15. When determining block size requirements — as described in Section 3.10.21 — the translation limit register provides the block size requirements for the base address register. The remaining registers used for performing address translation are discussed in Section 3.2.1.1.

The 80331 translate value register's programmed value must be naturally aligned with the base address register's programmed value. The limit register is used as a mask; thus, the lower address bits programmed into the 80331 translate value register are invalid. Refer to the *PCI Local Bus Specification*, Revision 2.3 for additional information on programming base address registers.

Bits 31 to 12 within the IALR3 have a direct effect on the IABAR3 register, bits 31 to 12, with a one to one correspondence. A value of 0 in a bit within the IALR3 makes the corresponding bit within the IABAR3 a read only bit which always returns 0. A value of 1 in a bit within the IALR3 makes the corresponding bit within the IABAR3 read/write from PCI. Note that a consequence of this programming scheme is that unless a valid value exists within the IALR3, all writes to the IABAR3 has no effect since a value of all zeros within the IALR3 makes the IABAR3 a read only register.

Table 130. Inbound ATU Limit Register 3 - IALR3



Internal Bus Address
FFFF.E198H

PCI Configuration Address Offset
98H - 9BH

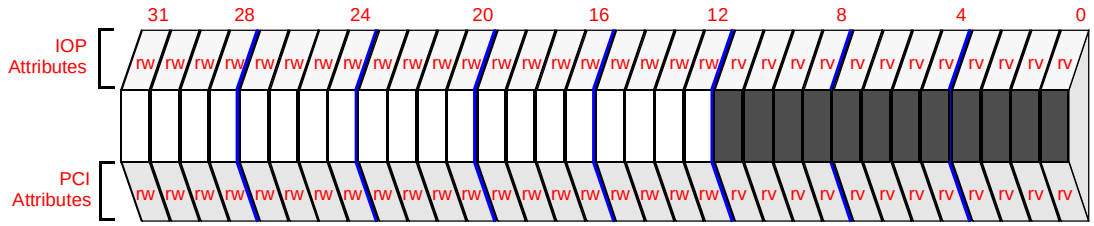
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	Inbound Translation Limit 3 - This readback value determines the memory block size required for the ATUs memory window 3.
11:00	000H	Reserved

3.10.46 Inbound ATU Translate Value Register 3 - IATVR3

The Inbound ATU Translate Value Register 3 (IATVR3) contains the internal bus address used to convert PCI bus addresses. The converted address is driven on the internal bus as a result of the inbound ATU address translation.

Table 131. Inbound ATU Translate Value Register 3 - IATVR3



Internal Bus Address
FFFF.E19CH

PCI Configuration Address Offset
9CH-9FH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

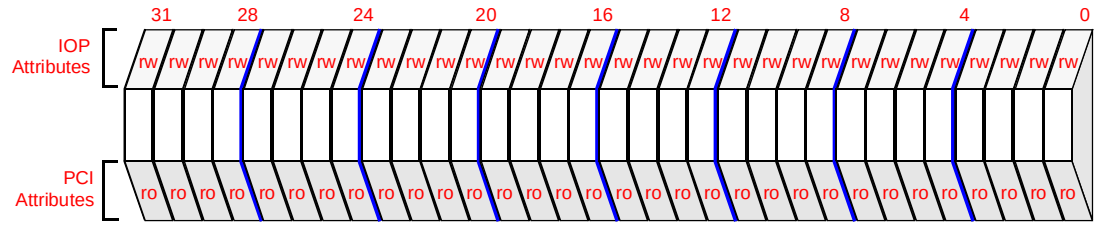
Bit	Default	Description
31:12	00000H	Inbound ATU Translation Value 3 - This value is used to convert the PCI address to internal bus addresses. This value must be 64-bit aligned on the internal bus. The default address allows the ATU to access the internal 80331 memory-mapped registers.
11:00	000H	Reserved

3.10.47 Outbound Configuration Cycle Address Register - OCCAR

The Outbound Configuration Cycle Address Register is used to hold the 32-bit PCI configuration cycle address. The Intel® XScale™ core writes the PCI configuration cycles address which then enables the outbound configuration read or write. The Intel® XScale™ core then performs a read or write to the Outbound Configuration Cycle Data Register to initiate the configuration cycle on the PCI bus.

Note: Bits 15:11 of the configuration cycle address for Type 0 configuration cycles are defined differently for Conventional versus PCI-X modes. When 80331 software programs the OCCAR to initiate a Type 0 configuration cycle, the OCCAR should always be loaded based on the PCI-X definition for the Type 0 configuration cycle address. When operating in Conventional mode, the 80331 clears bits 15:11 of the OCCAR prior to initiating an outbound Type 0 configuration cycle. See the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a for details on the two formats.

Table 132. Outbound Configuration Cycle Address Register - OCCAR



Internal Bus Address
FFFF.E1A4H

PCI Configuration Address Offset
A4H - A7H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

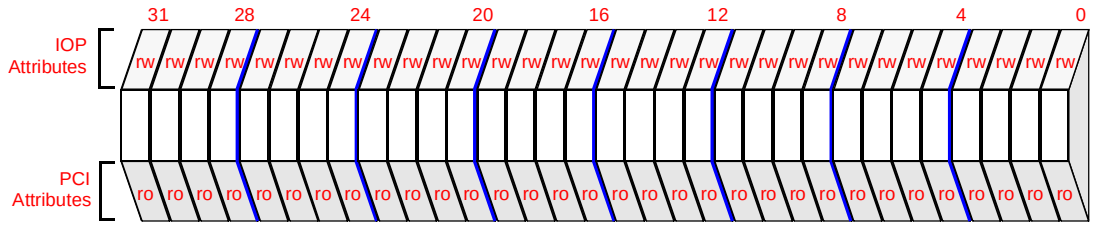
Bit	Default	Description
31:00	0000 0000H	Configuration Cycle Address - These bits define the 32-bit PCI address used during an outbound configuration read or write cycle.

3.10.48 Outbound Configuration Cycle Data Register - OCCDR

The Outbound Configuration Cycle Data Register is used to initiate a configuration read or write on the PCI bus. The register is logical rather than physical meaning that it is an address not a register. The Intel® XScale™ core reads or writes the data registers memory-mapped address to initiate the configuration cycle on the PCI bus with the address found in the OCCAR. For a configuration write, the data is latched from the internal bus and forwarded directly to the OWQ. For a read, the data is returned directly from the ORQ to the Intel® XScale™ core and is never actually entered into the data register (which does not physically exist).

The OCCDR is only visible from 80331 internal bus address space and appears as a reserved value within the ATU configuration space.

Table 133. Outbound Configuration Cycle Data Register - OCCDR



Internal Bus Address
FFFF.E1ACH

PCI Configuration Address Offset
ACH - AFH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

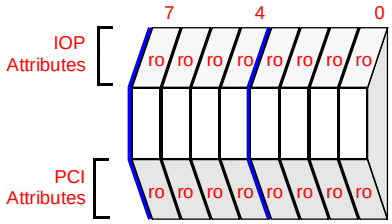
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	Configuration Cycle Data - These bits define the data used during an outbound configuration read or write cycle.

3.10.49 VPD Capability Identifier Register - VPD_CAPID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 80331, this is the VPD extended capability with an ID of 03H as defined by the *PCI Local Bus Specification*, Revision 2.3.

Table 134. VPD Capability Identifier Register - VPD_CAPID



Internal Bus Address
FFFF.E1B8H

PCI Configuration Offset
B8H

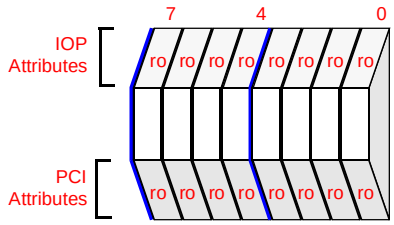
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	03H	Cap_Id - This field with its' 03H value identifies this item in the linked list of Extended Capability Headers as being the VPD capability registers.

3.10.50 VPD Next Item Pointer Register - VPD_NXTP

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 80331, this the final capability list, and hence, this register is set to 00H.

Table 135. VPD Next Item Pointer Register - VPD_NXTP



Internal Bus Address
FFFF.E1B9H

PCI Configuration Offset
B9H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

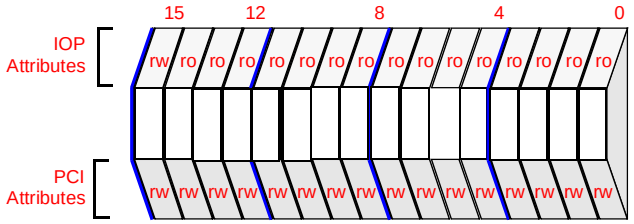
Bit	Default	Description
07:00	00H	Next_ Item_ Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list. Since the VPD capabilities are the last in the linked list of extended capabilities in the 80331, the register is set to 00H.

3.10.51 VPD Address Register - VPD_AR

The VPD Address register (VPDAR) contains the DWORD-aligned byte address of the VPD to be accessed. The register is read/write and the initial value at power-up is indeterminate.

A PCI Configuration Write to the VPDAR interrupts the Intel® XScale™ core. Software can use the Flag setting to determine whether the configuration write was intended to initiate a read or write of the VPD through the VPD Data Register.

Table 136. VPD Address Register - VPD_AR



Internal Bus Address
FFFF.E1BAH

PCI Configuration Offset
BA- BBH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

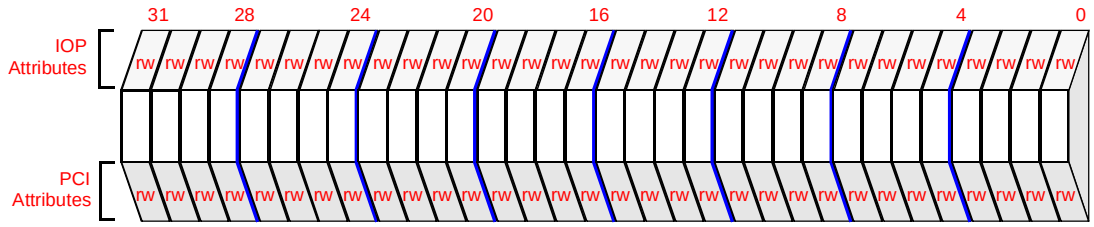
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0_2	Flag - A flag is used to indicate when a transfer of data between the VPD Data Register and the storage component has completed. Please see Section 3.9, "Vital Product Data" on page 201 for more details on how the 80331 handles the data transfer.
14:0	0000H	VPD Address - This register is written to set the DWORD-aligned byte address used to read or write Vital Product Data from the VPD storage component.

3.10.52 VPD Data Register - VPD_DR

This register is used to transfer data between the 80331 and the VPD storage component.

Table 137. VPD Data Register - VPD_DR



Internal Bus Address
FFFF.E1BCH

PCI Configuration Offset
BCH-BFH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

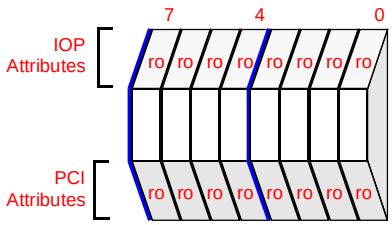
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000H	VPD Data - Four bytes are always read or written through this register to/from the VPD storage component.

3.10.53 Power Management Capability Identifier Register - PM_CAPID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 80331, this is the PCI Bus Power Management extended capability with an ID of 01H as defined by the *PCI Bus Power Management Interface Specification*, Revision 1.1.

Table 138. Power Management Capability Identifier Register - PM_CAPID



Internal Bus Address
FFFF.E1C0H

PCI Configuration Offset
C0H

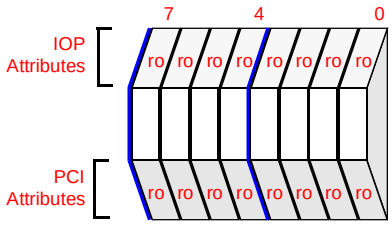
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	01H	Cap_Id - This field with its' 01H value identifies this item in the linked list of Extended Capability Headers as being the PCI Power Management Registers.

3.10.54 Power Management Next Item Pointer Register - PM_NXTP

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list. For the 80331, the next capability (MSI capability list) is located at off-set D0H.

Table 139. Power Management Next Item Pointer Register - PM_NXTP



Internal Bus Address
FFFF.E1C1H

PCI Configuration Offset
C1H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

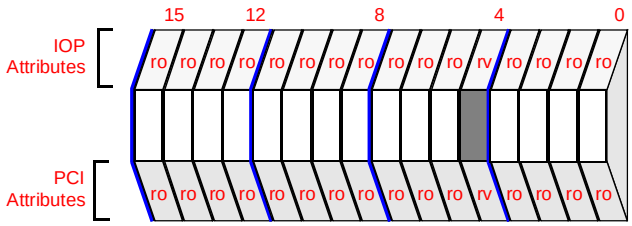
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	D0H	Next_ Item_ Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list which in the 80331 is the MSI extended capabilities header.

3.10.55 Power Management Capabilities Register - PM_CAP

Power Management Capabilities bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This register is a 16-bit read-only register which provides information on the capabilities of the ATU function related to power management.

Table 140. Power Management Capabilities Register - PM_CAP



Internal Bus Address
FFFF.E1C2H

PCI Configuration Offset
C2- C3H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

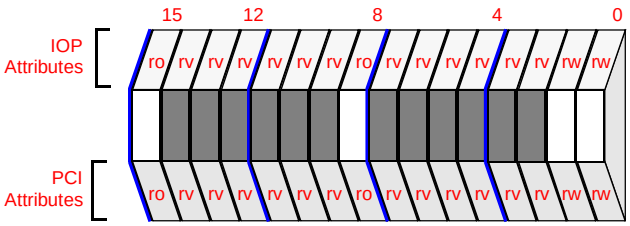
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:11	00000_2	PME_Support - This function is not capable of asserting the **PME#** signal in any state, since **PME#** is not supported by the 80331.
10	0_2	D2_Support - This bit is set to 0_2 indicating that the 80331 does not support the D2 Power Management State
9	1_2	D1_Support - This bit is set to 1_2 indicating that the 80331 supports the D1 Power Management State
8:6	000_2	Aux_Current - This field is set to 000_2 indicating that the 80331 has no current requirements for the 3.3Vaux signal as defined in the *PCI Bus Power Management Interface Specification*, Revision 1.1
5	0_2	DSI - This field is set to 0_2 meaning that this function requires a device specific initialization sequence following the transition to the D0 uninitialized state.
4	0_2	Reserved.
3	0_2	PME Clock - Since the 80331 does not support **PME#** signal generation this bit is cleared to 0_2.
2:0	010_2	Version - Setting these bits to 010_2 means that this function complies with *PCI Bus Power Management Interface Specification*, Revision 1.1

3.10.56 Power Management Control/Status Register - PM_CSR

Power Management Control/Status bits adhere to the definitions in the *PCI Bus Power Management Interface Specification*, Revision 1.1. This 16-bit register is the control and status interface for the power management extended capability.

Table 141. Power Management Control/Status Register - PM_CSR



Internal Bus Address
FFFF.E1C4H

PCI Configuration Offset
C4- C5H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15	0_2	PME_Status - This function is not capable of asserting the PME# signal in any state, since **PME##** is not supported by the 80331.
14:9	00H	Reserved
8	0_2	PME_En - This bit is hardwired to read-only 0_2 since this function does not support **PME#** generation from any power state.
7:2	000000_2	Reserved
1:0	00_2	Power State - This 2-bit field is used both to determine the current power state of a function and to set the function into a new power state. The definition of the values is: 00_2 - D0 01_2 - D1 10_2 - D2 (Unsupported) 11_2 - $D3_{hot}$ The 80331 supports only the D0 and $D3_{hot}$ states.

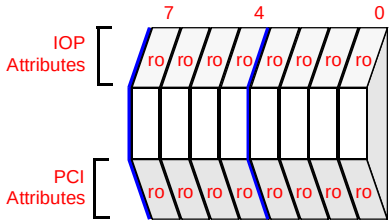
3.10.57 MSI Capability Registers

The MSI capability registers are defined in the Messaging Unit. See "MSI Capability Identifier Register - MSI_CAPID" on page 323, "MSI Next Item Pointer Register - MSI_NXTP" on page 324, "MSI Message Control Register - MSI_MCR" on page 325, "MSI Message Address Register - MSI_MAR" on page 326, "MSI Message Upper Address Register - MSI_MUAR" on page 327 and "MSI Message Data Register- MSI_MDR" on page 328.

3.10.58 PCI-X Capability Identifier Register - PX_CAPID

The Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. In the case of the 80331, this is the PCI-X extended capability with an ID of 07H as defined by the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a.

Table 142. PCI-X_Capability Identifier Register - PX_CAPID



Internal Bus Address
FFFF.E1E0H

PCI Configuration Offset
E0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	07H	Cap_Id - This field with its' 07H value identifies this item in the linked list of Extended Capability Headers as being the PCI-X capability registers.

3.10.59 PCI-X Next Item Pointer Register - PX_NXTP

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.3. This register describes the location of the next item in the function's capability list.

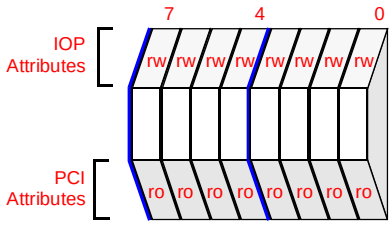
By default, the PCI-X capability is the last capabilities list for the 80331, thus this register defaults to 00H.

However, this register may be written to B8H prior to host configuration to include the VPD capability located at off-set B8H.

Warning: Writing this register to any value other than 00H (default) or B8H is not supported and may produce unpredictable system behavior.

In order to guarantee that this register is written prior to host configuration, the 80331 must be initialized at **P_RST#** assertion to Retry Type 0 configuration cycles (bit 2 of PCSR). Typically, the Intel® XScale™ core would be enabled to boot immediately following **P_RST#** assertion in this case (bit 1 of PCSR), as well. Please see Table 125, "PCI Configuration and Status Register - PCSR" on page 253 for more details on the 80331 initialization modes.

Table 143. PCI-X Next Item Pointer Register - PX_NXTP



Internal Bus Address
FFFF.E1E1H

PCI Configuration Offset
E1H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

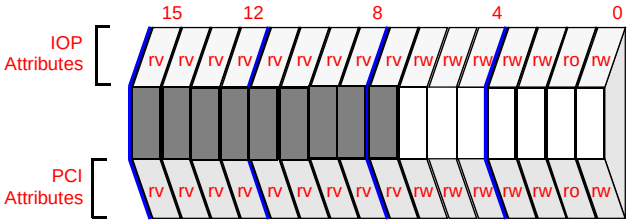
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	00H	Next_ Item_ Pointer - This field provides an offset into the function's configuration space pointing to the next item in the function's capability list. Since the PCI-X capabilities are the last in the linked list of extended capabilities in the 80331, the register is set to 00H. However, this field may be written prior to host configuration with B8H to extend the list to include the VPD extended capabilities header.

3.10.60 PCI-X Command Register - PX_CMD

This register controls various modes and features of ATU and Message Unit when operating in the PCI-X mode.

Table 144. PCI-X Command Register - PX_CMD



Internal Bus Address
FFFF.E1E2H

PCI Configuration Offset
E2- E3H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

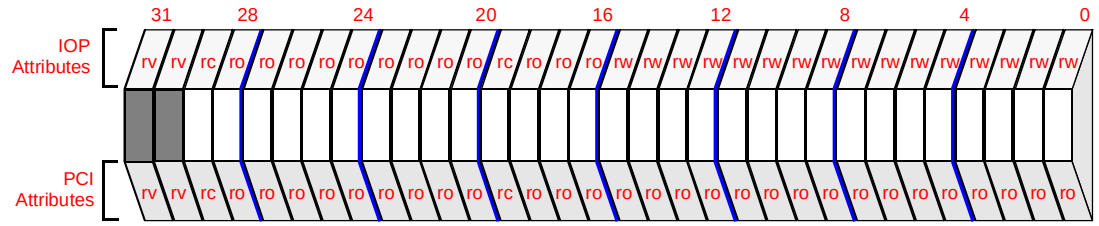
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:7	000000000_2	Reserved.
6:4	011_2	Maximum Outstanding Split Transactions - This register sets the maximum number of Split Transactions the device is permitted to have outstanding at one time. **Register Maximum Outstanding** 0 1 1 2 2 3 3 4 4 8 5 12 6 16 7 32
3:2	00_2	Maximum Memory Read Byte Count - This register sets the maximum byte count the device uses when initiating a Sequence with one of the burst memory read commands. **Register Maximum Byte Count** 0 512 1 1024 2 2048 3 4096
1	0_2	Enable Relaxed Ordering - The 80331 **does not** set the relaxed ordering bit in the Requester Attributes of Transactions.
0	0_2	Data Parity Error Recovery Enable - The device driver sets this bit to enable the device to attempt to recover from data parity errors. When this bit is 0 and the device is in PCI-X mode, the device asserts **SERR#** (when enabled) whenever the Master Data Parity Error bit (Status register, bit 8) is set.

3.10.61 PCI-X Status Register - PX_SR

This register identifies the capabilities and current operating mode of ATU, DMAs and Message Unit when operating in the PCI-X mode.

Table 145. PCI-X Status Register - PX_SR (Sheet 1 of 2)



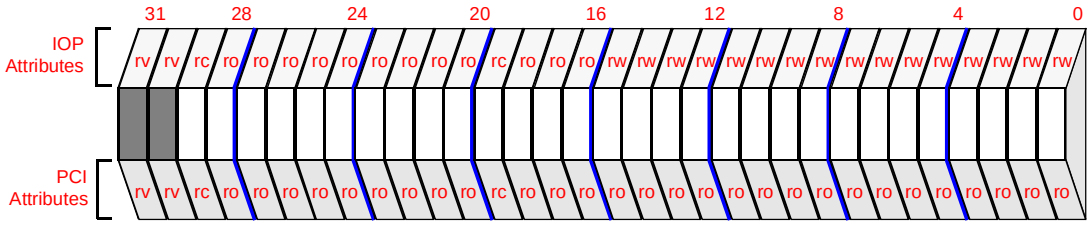
Internal Bus Address
FFFF.E1E4H

PCI Configuration Offset
E4H-E7H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:30	00_2	Reserved
29	0_2	Received Split Completion Error Message - This bit is set when the device receives a Split Completion Message with the Split Completion Error attribute bit set. Once set, this bit remains set until software writes a 1 to this location. 0 = no Split Completion error message received. 1 = a Split Completion error message has been received.
28:26	001_2	Designed Maximum Cumulative Read Size (DMCRS) - The value of this register depends on the setting of the Maximum Memory Read Byte Count field of the PCIXCMD register: **DMCRS** **Max ADQs** ***Maximum Memory Read Byte Count Register Setting*** 1 16 512 (Default) 2 32 1024 2 32 2048 2 32 4096
25:23	011_2	Designed Maximum Outstanding Split Transactions - The 80331 can have up to four outstanding split transactions.
22:21	01_2	Designed Maximum Memory Read Byte Count - The 80331 can generate memory reads with byte counts up to 1024 bytes.
20	1_2	80331 is a complex device.
19	0_2	Unexpected Split Completion - This bit is set when an unexpected Split Completion with this device's Requester ID is received. Once set, this bit remains set until software writes a 1 to this location. 0 = no unexpected Split Completion has been received. 1 = an unexpected Split Completion has been received.
18	0_2	Split Completion Discarded - This bit is set when the device discards a Split Completion because the requester would not accept it. See Section 5.4.4 of the *PCI-X Addendum to the PCI Local Bus Specification,* Revision 1.0a for details. Once set, this bit remains set until software writes a 1 to this location. 0 = no Split Completion has been discarded. 1 = a Split Completion has been discarded. **NOTE:** The 80331 does not set this bit since there is no Inbound address responding to Inbound Read Requests with Split Responses (Memory or Register) that has "read side effects."

Table 145. PCI-X Status Register - PX_SR (Sheet 2 of 2)



Internal Bus Address
FFFF.E1E4H

PCI Configuration Offset
E4H-E7H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
17	1_2	80331 is a 133 MHz capable device.
16	$1_{2\ or}$ **P_32BITPCI#**	80331 with bridge enabled (BRG_EN = 1) implements the ATU with a 64-bit interface on the secondary PCI bus, therefore this bit is always set. 80331 with no bridge and central resource disabled (BRG_EN = 0, ARB_EN = 0), use this bit to identify the add-in card to the system as 64-bit or 32-bit wide via a user-configurable strap (**P_32BITPCI#**). This strap, by default, identifies the add in card based on 80331 with bridge disabled as 64-bit unless the user attaches the appropriate pull-down resistor to the strap. 0 = The bus is 32 bits wide. 1 = The bus is 64 bits wide.
15:8	FFH	Bus Number - This register is read for diagnostic purposes only. It indicates the number of the bus segment for the device containing this function. The function uses this number as part of its Requester ID and Completer ID. For all devices other than the source bridge, each time the function is addressed by a Configuration Write transaction, the function must update this register with the contents of AD[7::0] of the attribute phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction when all of the following are true: 1. The transaction uses a Configuration Write command. 2. IDSEL is asserted during the address phase. 3. AD[1::0] are 00b (Type 0 configuration transaction). 4. AD[10::08] of the configuration address contain the appropriate function number.
7:3	1FH	Device Number - This register is read for diagnostic purposes only. It indicates the number of the device containing this function, i.e., the number in the Device Number field (AD[15::11]) of the address of a Type 0 configuration transaction that is assigned to the device containing this function by the connection of the system hardware. The system must assign a device number other than 00h (00h is reserved for the source bridge). The function uses this number as part of its Requester ID and Completer ID. Each time the function is addressed by a Configuration Write transaction, the device must update this register with the contents of AD[15::11] of the address phase of the Configuration Write, regardless of which register in the function is addressed by the transaction. The function is addressed by a Configuration Write transaction when all of the following are true: 1. The transaction uses a Configuration Write command. 2. IDSEL is asserted during the address phase. 3. AD[1::0] are 00b (Type 0 configuration transaction). 4. AD[10::08] of the configuration address contain the appropriate function number.
2:0	000_2	Function Number - This register is read for diagnostic purposes only. It indicates the number of this function; i.e., the number in the Function Number field (AD[10::08]) of the address of a Type 0 configuration transaction to which this function responds. The function uses this number as part of its Requester ID and Completer ID.

3.10.62 PCI Interrupt Routing Select Register - PIRSR

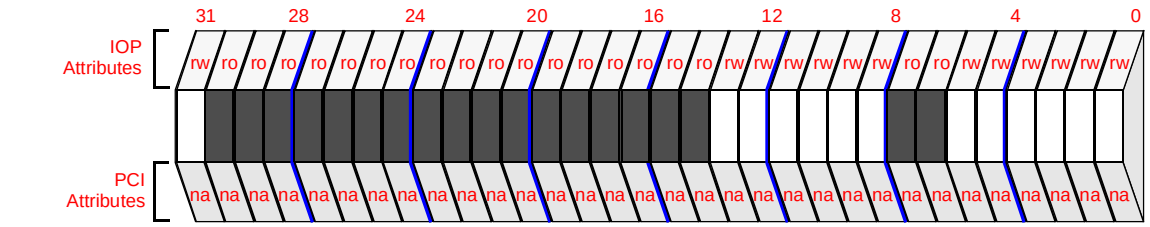
The PCI Interrupt Routing Select Register (PIRSR) determines the routing of the external input pins. This register is addressed in the ATU register space, but is defined in Section 15.7.17, "PCI Interrupt Routing Select Register - PIRSR" on page 717. Refer to that page for functional details and programming information.

3.10.63 Secondary PCIDrive Strength Control Register - SPDSCR

When necessary, the Secondary drive strength control register is used to manually control the slew rate and drive strength of the PCI bus interface.

Note: By default, the user is **not** required to program this register.

Table 146. Secondary PCIDrive Strength Control Register - SPDSCR



Intel® XScale™ Core Local Bus Address
FFFF F5C0H

PCI Configuration Offset
Not Accessible

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

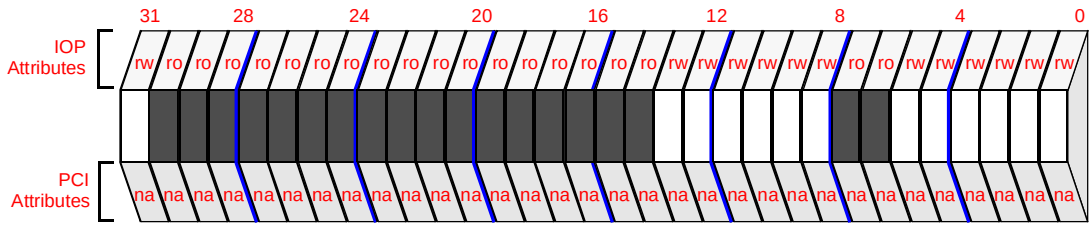
Bit	Default	Description
31	0	**Overdrive Enable**: When set, enables the **Secondary** PCIBus interface drive strengths to be programmed with the values in this register.
30:14	0 0000H	**Reserved**
13:8	11 1111	**Pull-up Drive Strength**: When bit 31 is set, this field programs the strength of the n-drivers for the **Secondary** PCIBus interface.
7:6	00	**Reserved**
5:0	11 1111	**Pull-Down Drive Strength**: When bit 31 is set, this field programs the strength of the n-drivers for the **Secondary** PCIBus interface.

3.10.64 Primary PCIDrive Strength Control Register - PPDSCR

When necessary, the Primary drive strength control register is used to manually control the slew rate and drive strength of the PCI bus interface.

Note: By default, the user is **not** required to program this register.

Table 147. Primary PCIDrive Strength Control Register - PPDSCR



Intel® XScale™ Core Local Bus Address
FFFF F5C8H

PCI Configuration Offset
Not Accessible

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0	**Overdrive Enable**: When set, enables the **Primary PCI**Bus interface drive strengths to be programmed with the values in this register.
30:14	000 0000H	**Reserved**
13:8	11 1111	**Pull-up Drive Strength**: When bit 31 is set, this field programs the strength of the n-drivers for the **Primary PCI**Bus interface.
7:6	00	**Reserved**
5:0	11 1111	**Pull-Down Drive Strength**: When bit 31 is set, this field programs the strength of the n-drivers for the **Primary PCI**Bus interface.

This Page Intentionally Left Blank

Messaging Unit 4

This chapter describes the Messaging Unit (MU) of the Intel® 80331 I/O processor (80331). The MU is closely related to the Address Translation Unit (ATU) described in Chapter 3, "Address Translation Unit".

4.1 Overview

The Messaging Unit (MU) provides a mechanism for data to be transferred between the PCI bus and the Intel® XScale™ core and notifying the respective system of the arrival of new data through an interrupt. The MU can be used to send and receive messages.

The MU has four distinct messaging mechanisms. Each allows a host processor or external PCI agent and the 80331 to communicate through message passing and interrupt generation. The four mechanisms are:

- **Message Registers** — allow the 80331 and external PCI agents to communicate by passing messages in one of four 32-bit Message Registers. In this context, a message is any 32-bit data value. Message registers combine aspects of mailbox registers and doorbell registers. Writes to the message registers may optionally cause interrupts.
- **Doorbell Registers** — allow the 80331 to assert the PCI interrupt signal and allow external PCI agents to generate an interrupt to the Intel® XScale™ core.
- **Circular Queues** — support a message passing scheme that uses four circular queues.
- **Index Registers** — support a message passing scheme that uses a portion of the 80331 local memory to implement a large set of message registers.

Each of the above are available to the system designer at the same time. No special mode selection is needed.

4.2 Theory of Operation

The MU has four independent messaging mechanisms.

There are four Message Registers that are similar to a combination of mailbox and doorbell registers. Each holds a 32-bit value and generates an interrupt when written.

The two Doorbell Registers support software interrupts. When a bit is set in a Doorbell Register, an interrupt is generated.

The Circular Queues support a message passing scheme that uses 4 circular queues. The 4 circular queues are implemented in 80331 local memory. Two queues are used for inbound messages and two are used for outbound messages. Interrupts may be generated when the queue is written.

The Index Registers use a portion of the 80331 local memory to implement a large set of message registers. When one of the Index Registers is written, an interrupt is generated and the address of the register written is captured.

Interrupt status for all interrupts is recorded in the Inbound Interrupt Status Register and the Outbound Interrupt Status Register. Each interrupt generated by the Messaging Unit can be masked.

Multi-DWORD PCI burst accesses are not supported by the Messaging Unit, with the exception of Multi-DWORD reads to the index registers. In Conventional mode, the MU terminates Multi-DWORD PCI transactions (other than index register reads) with a disconnect at the next Qword boundary, with the exception of queue ports. In PCI-X mode, the MU terminates a Multi-DWORD PCI read transaction with a Split Response and the data is returned through split completion transaction(s); however, when the burst request crosses into or through the range of offsets 40h to 4Ch (e.g., this includes the queue ports) the transaction is signaled target-abort immediately on the PCI bus. In PCI-X mode, Multi-DWORD PCI writes is signaled a Single-Data-Phase Disconnect which means that no data beyond the first Qword (Dword when the MU does not assert **P_ACK64#**) is written.

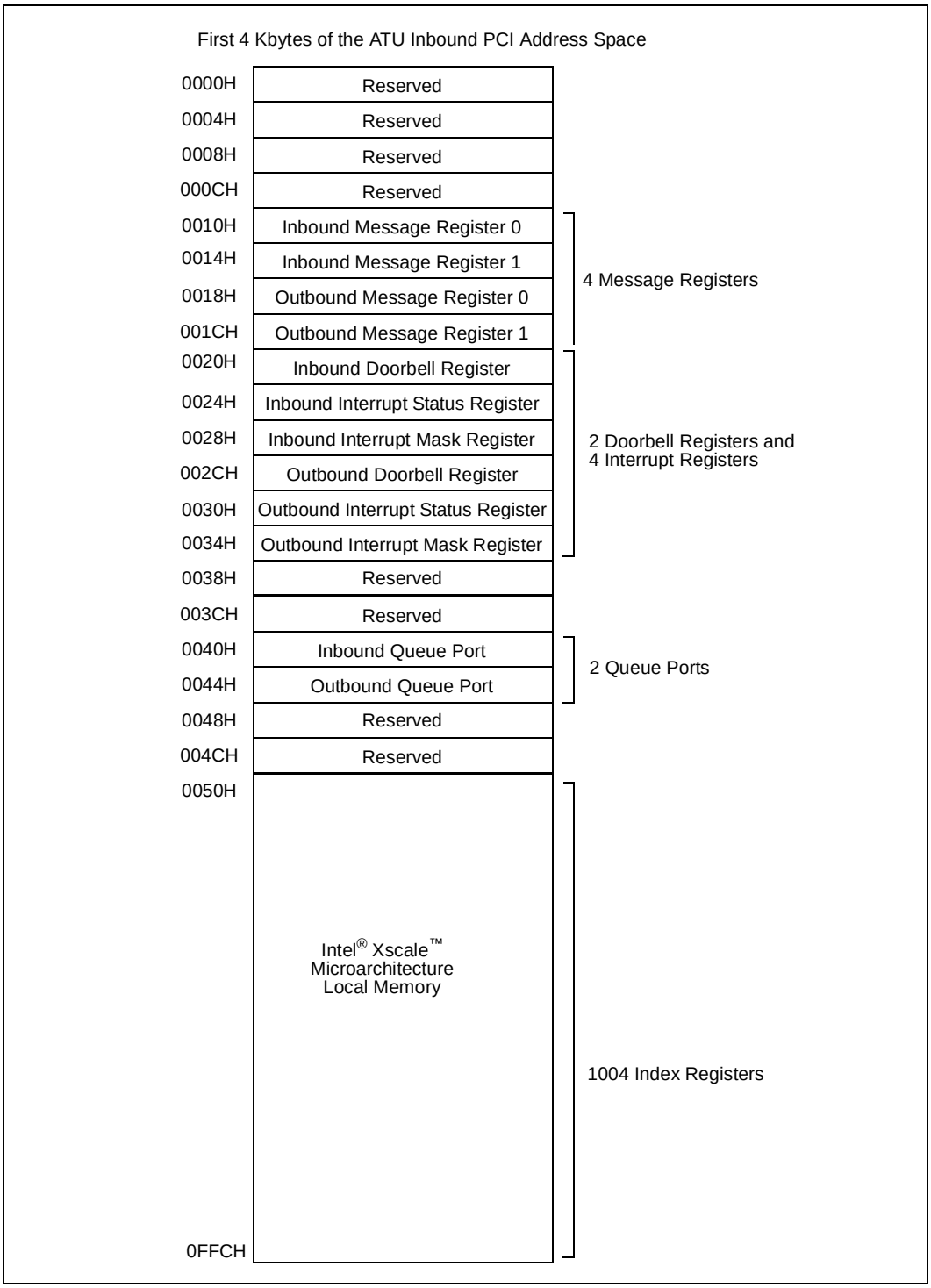
All registers needed to configure and control the Messaging Unit are memory-mapped registers.

The MU uses the first 4 Kbytes of the inbound translation window in the Address Translation Unit (ATU). This PCI address window is used for PCI transactions that access the 80331 local memory. The PCI address of the inbound translation window is contained in the Inbound ATU Base Address Register. See Chapter 3, "Address Translation Unit" for more details on inbound ATU addressing and the ATU.

From the PCI perspective, the Messaging Unit is part of the Address Translation Unit. The Messaging Unit uses the PCI configuration registers of the ATU for control and status information. The Messaging Unit must observe all PCI control bits in the ATU Command Register and ATU Configuration Register. The Messaging Unit reports all PCI errors in the ATU Status Register.

Parts of the Messaging Unit can be accessed as a 64-bit PCI device. The register interface, message registers, doorbell registers, and index registers returns a **P_ACK64#** in response to a **P_REQ64#** on the PCI interface. Up to 1 Qword of data can be read or written per transaction (except Index Register reads, see Section 4.6, "Index Registers" on page 300). The Inbound and Outbound Queue Ports are always 32-bit addresses and the MU does not assert **P_ACK64#** to offsets 40H and 44H.

Figure 20. PCI Memory Map



Note: See Table 154, "Message Unit Register".

Figure 21. Internal Bus Memory Map

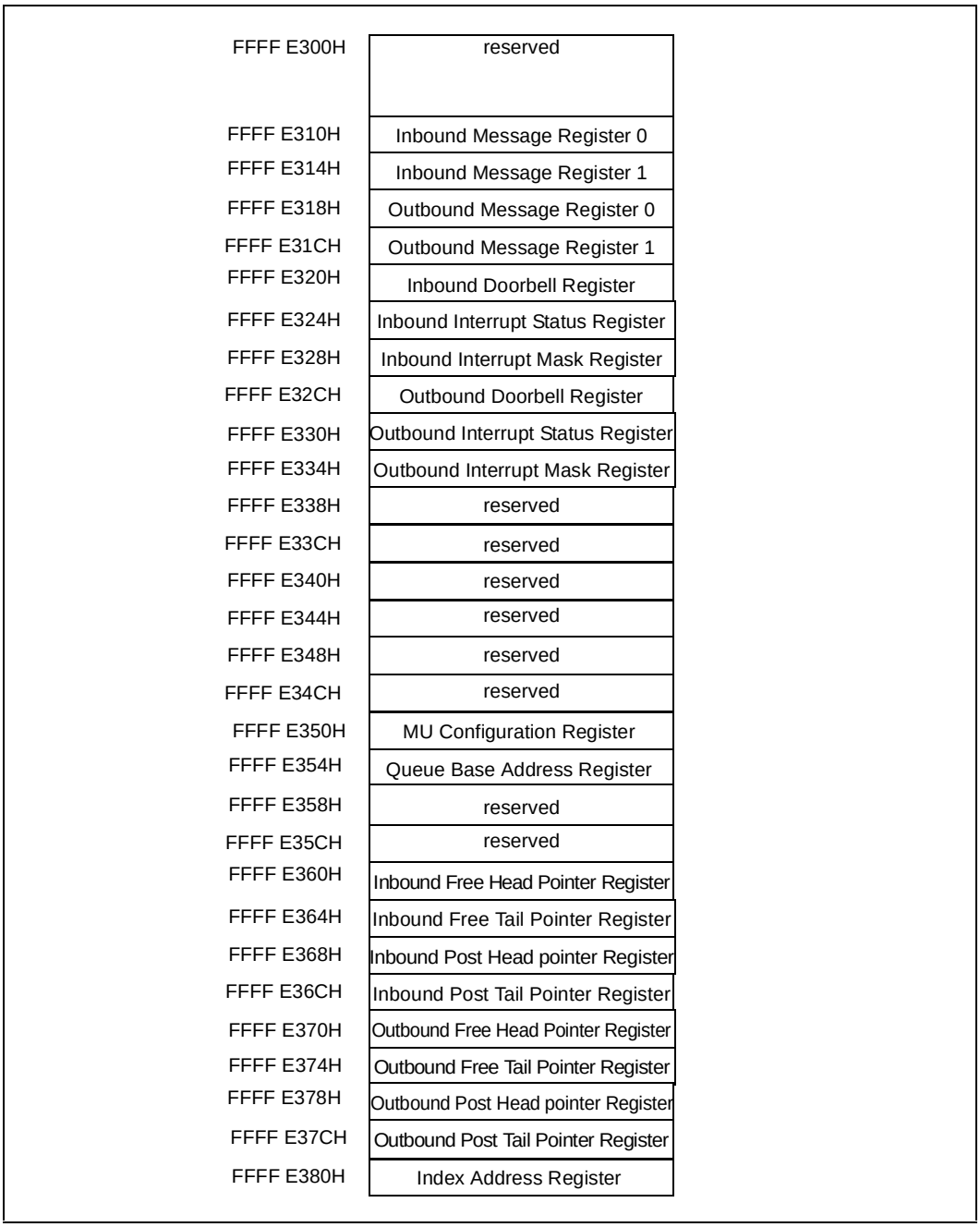


Table 148 provides a summary of the four messaging mechanisms used in the Messaging Unit.

Table 148. MU Summary

Mechanism	Quantity	Assert PCI Interrupt Signals	Generate I/O Processor Interrupt
Message Registers	2 Inbound 2 Outbound	Optional	Optional
Doorbell Registers	1 Inbound 1 Outbound	Optional	Optional
Circular Queues	4 Circular Queues	Under certain conditions	Under certain conditions
Index Registers	1004 32-bit Memory Locations	No	Optional

4.2.1 Transaction Ordering

From a PCI standpoint, the Messaging Unit is a piece of the ATU and therefore must maintain ordering requirements against ATU transactions. Transaction ordering is achieved for the Index Registers, the Doorbell Register, and the Message Registers since these transactions are routed through the standard set of ATU read/write queues.

The Circular Queues (Inbound/Outbound Queue Port) are separate queue structures and therefore require ordering. The Inbound Post Queue (contains PCI writes) must be ordered against the inbound write queue of the ATU to allow the data that is represented by the Inbound Post interrupt to be written to local memory before the interrupt is delivered. See Table 149 for a summary of Messaging Unit transaction ordering.

Table 149. Circular Queue Ordering Requirements

Messaging Unit Feature		Transaction Ordering Mechanism
Message Registers		Through ATU Queues
Doorbell Registers		Through ATU Queues
Index Registers		Through ATU Queues
Circular Queues	Inbound Post	Ordered Against ATU Inbound Write Queue (PMW Can Not Pass Another PMW)
	Inbound Free	No Specific Hardware Ordering
	Outbound Post	No Specific Hardware Ordering
	Outbound Free	No Specific Hardware Ordering

4.3 Message Registers

Messages can be sent and received by the 80331 through the use of the Message Registers. When written, the message registers may cause an interrupt to be generated to either the Intel® XScale™ core or the host processor. Inbound messages are sent by the host processor and received by the 80331. Outbound messages are sent by the 80331 and received by the host processor.

The interrupt status for outbound messages is recorded in the Outbound Interrupt Status Register. Interrupt status for inbound messages is recorded in the Inbound Interrupt Status Register.

4.3.1 Outbound Messages

When an outbound message register is written by the Intel® XScale™ core, an interrupt may be generated on the **P_INTC#** interrupt pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) interrupt pin or a message signaled interrupt is generated when MSI is enabled.

The PCI interrupt is recorded in the Outbound Interrupt Status Register. The interrupt causes the Outbound Message Interrupt bit to be set in the Outbound Interrupt Status Register. This is a Read/Clear bit that is set by the MU hardware and cleared by software.

The interrupt is cleared when an external PCI agent writes a value of 1 to the Outbound Message Interrupt bit in the Outbound Interrupt Status Register to clear the bit.

The interrupt may be masked by the mask bits in the Outbound Interrupt Mask Register.

4.3.2 Inbound Messages

When an inbound message register is written by an external PCI agent, an interrupt may be generated to the Intel® XScale™ core. The interrupt may be masked by the mask bits in the Inbound Interrupt Mask Register.

The Intel® XScale™ core interrupt is recorded in the Inbound Interrupt Status Register. The interrupt causes the Inbound Message Interrupt bit to be set in the Inbound Interrupt Status Register. This is a Read/Clear bit that is set by the MU hardware and cleared by software.

The interrupt is cleared when the Intel® XScale™ core writes a value of 1 to the Inbound Message Interrupt bit in the Inbound Interrupt Status Register.

4.4 Doorbell Registers

There are two Doorbell Registers: the Inbound Doorbell Register and the Outbound Doorbell Register. The Inbound Doorbell Register allows external PCI agents to generate interrupts to the Intel® XScale™ core. The Outbound Doorbell Register allows the Intel® XScale™ core to generate a PCI interrupt. Both Doorbell Registers may generate interrupts whenever a bit in the register is set.

4.4.1 Outbound Doorbells

When the Outbound Doorbell Register is written by the Intel® XScale™ core, an interrupt may be generated on the **P_INTA#** interrupt pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) or a message signaled interrupt is generated when MSI is enabled. An interrupt is generated when any of the bits in the doorbell register is written to a value of 1. Writing a value of 0 to any bit does not change the value of that bit and does not cause an interrupt to be generated. Once a bit is set in the Outbound Doorbell Register, it cannot be cleared by the Intel® XScale™ core.

The interrupt is recorded in the Outbound Interrupt Status Register.

The interrupt may be masked by the mask bits in the Outbound Interrupt Mask Register. When the mask bit is set for a particular bit, no interrupt is generated for that bit. The Outbound Interrupt Mask Register affects only the generation of the interrupt and not the values written to the Outbound Doorbell Register.

The interrupt is cleared when an external PCI agent writes a value of 1 to the bits in the Outbound Doorbell Register that are set. Writing a value of 0 to any bit does not change the value of that bit and does not clear the interrupt.

In summary, the Intel® XScale™ core generates an interrupt and external PCI agents clear the interrupt by setting bits in the Outbound Doorbell Register.

4.4.2 Inbound Doorbells

When the Inbound Doorbell Register is written by an external PCI agent, an interrupt may be generated to the Intel® XScale™ core. An interrupt is generated when any of the bits in the doorbell register is written to a value of 1. Writing a value of 0 to any bit does not change the value of that bit and does not cause an interrupt to be generated. Once a bit is set in the Inbound Doorbell Register, it cannot be cleared by any external PCI agent. The interrupt is recorded in the Inbound Interrupt Status Register.

The interrupt may be masked by the Inbound Doorbell Interrupt mask bit in the Inbound Interrupt Mask Register. When the mask bit is set for a particular bit, no interrupt is generated for that bit. The Inbound Interrupt Mask Register affects only the generation of the normal messaging unit interrupt and not the values written to the Inbound Doorbell Register. One bit in the Inbound Doorbell Register is reserved for an Error Doorbell interrupt.

The interrupt is cleared when the Intel® XScale™ core writes a value of 1 to the bits in the Inbound Doorbell Register that are set. Writing a value of 0 to any bit does not change the value of that bit and does not clear the interrupt.

4.5 Circular Queues

The MU implements four circular queues. There are 2 inbound queues and 2 outbound queues. In this case, inbound and outbound refer to the direction of the flow of posted messages.

Inbound messages are either:

- *posted* messages by other processors for the Intel® XScale™ core to process or
- *free* (or empty) messages that can be reused by other processors.

Outbound messages are either:

- *posted* messages by the Intel® XScale™ core for other processors to process or
- *free* (or empty) messages that can be reused by the Intel® XScale™ core.

Therefore, free inbound messages flow away from the and free outbound messages flow toward the 80331.

The four Circular Queues are used to pass messages in the following manner. The two inbound queues are used to handle inbound messages and the two outbound queues are used to handle outbound messages. One of the inbound queues is designated the Free queue and it contains inbound free messages. The other inbound queue is designated the Post queue and it contains inbound posted messages. Similarly, one of the outbound queues is designated the Free queue and the other outbound queue is designated the Post queue. Table 150 contains a summary of the queues.

Table 150. Circular Queue Summary

Queue Name	Purpose	Action on PCI Interface
Inbound Post Queue	Queue for inbound messages from other processors waiting to be processed by the 80331	Written
Inbound Free Queue	Queue for empty inbound messages from the 80331 available for use by other processors	Read
Outbound Post Queue	Queue for outbound messages from the 80331 that are being posted to the other processors	Read
Outbound Free Queue	Queue for empty outbound messages from other processors available for use by the 80331	Written

The two outbound queues allow the Intel® XScale™ core to post outbound messages in one queue and to receive free messages returning from the host processor. The Intel® XScale™ core posts outbound messages, the host processor receives the posted message and when it is finished with the message, places it back on the outbound free queue for reuse by the Intel® XScale™ core.

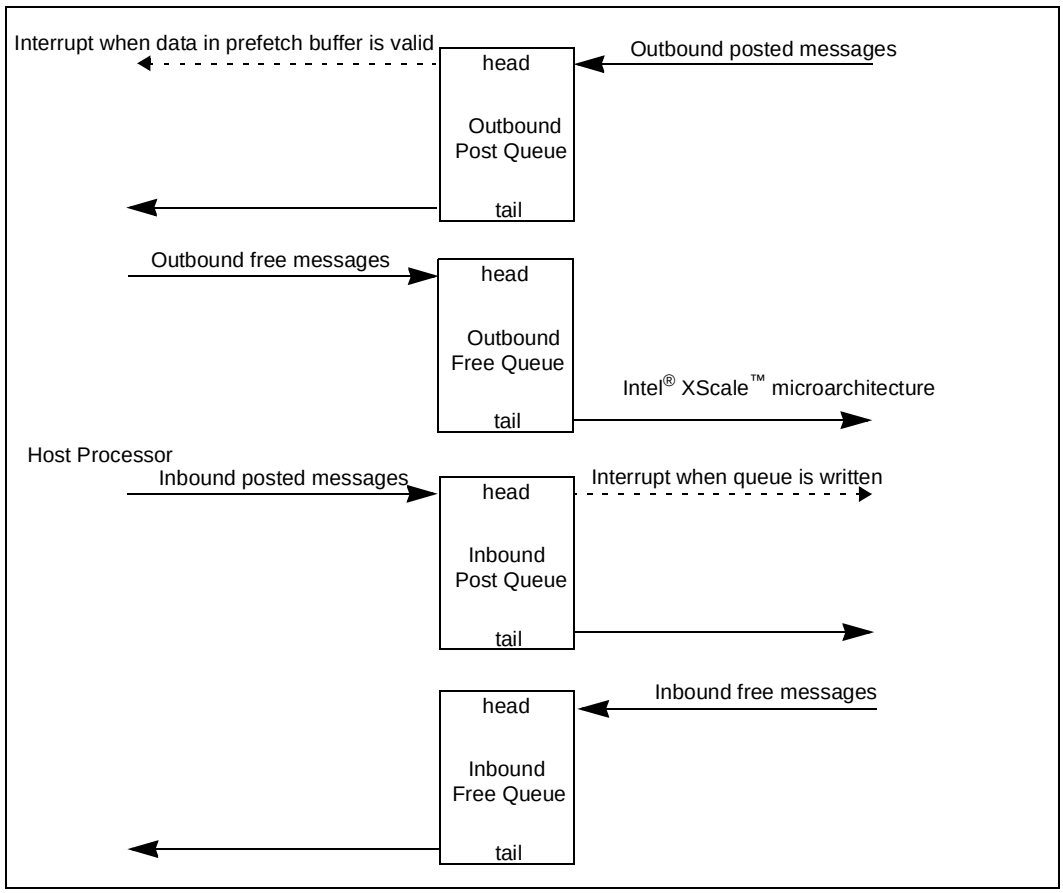
The two inbound queues allow the host processor to post inbound messages for the 80331 in one queue and to receive free messages returning from the 80331. The host processor posts inbound messages, the Intel® XScale™ core receives the posted message and when it is finished with the message, places it back on the inbound free queue for reuse by the host processor.

Figure 22 provides an overview of the Circular Queue operation.

The circular queues are accessed by external PCI agents through two port locations in the PCI address space: Inbound Queue Port and Outbound Queue Port. The Inbound Queue Port is used by external PCI agents to read the Inbound Free Queue and write the Inbound Post Queue. The

Outbound Queue Port is used by external PCI agents to read the Outbound Post Queue and write the Outbound Free Queue. Note that a PCI transaction to the inbound or outbound queue ports with null byte enables (**P_C/BE[3:0]#** = 1111_2) does not cause the MU hardware to increment the queue pointers. This is treated as when the PCI transaction did not occur. The Inbound and Outbound Queue Ports never respond with **P_ACK64#** on the PCI interface.

Figure 22. Overview of Circular Queue Operation



The data storage for the circular queues must be provided by the 80331 local memory. The base address of the circular queues is contained in the Queue Base Address Register (Section 4.9.10, "Queue Base Address Register - QBAR" on page 313). Each entry in the queue is a 32-bit data value. Each read from or write to the queue may access only one queue entry. Multi-DWORD accesses to the circular queues are not allowed. Sub-DWORD accesses are promoted to DWORD accesses.

Each circular queue has a head pointer and a tail pointer. The pointers are offsets from the Queue Base Address. Writes to a queue occur at the head of the queue and reads occur from the tail. The head and tail pointers are incremented by either the Intel® XScale™ core or the Messaging Unit hardware. Which unit maintains the pointer is determined by the writer of the queue. More details about the pointers are given in the queue descriptions below. The pointers are incremented after the queue access. Both pointers wrap around to the first address of the circular queue when they reach the circular queue size.

The Messaging Unit generates an interrupt to the Intel® XScale™ core or generate a PCI interrupt under certain conditions. In general, when a Post queue is written, an interrupt is generated to notify the receiver that a message was posted.

The size of each circular queue can range from 4K entries (16 Kbytes) to 64K entries (256 Kbytes). All four queues must be the same size and may be contiguous. Therefore, the total amount of local memory needed by the circular queues ranges from 64 Kbytes to 1 Mbytes. The Queue size is determined by the Queue Size field in the MU Configuration Register.

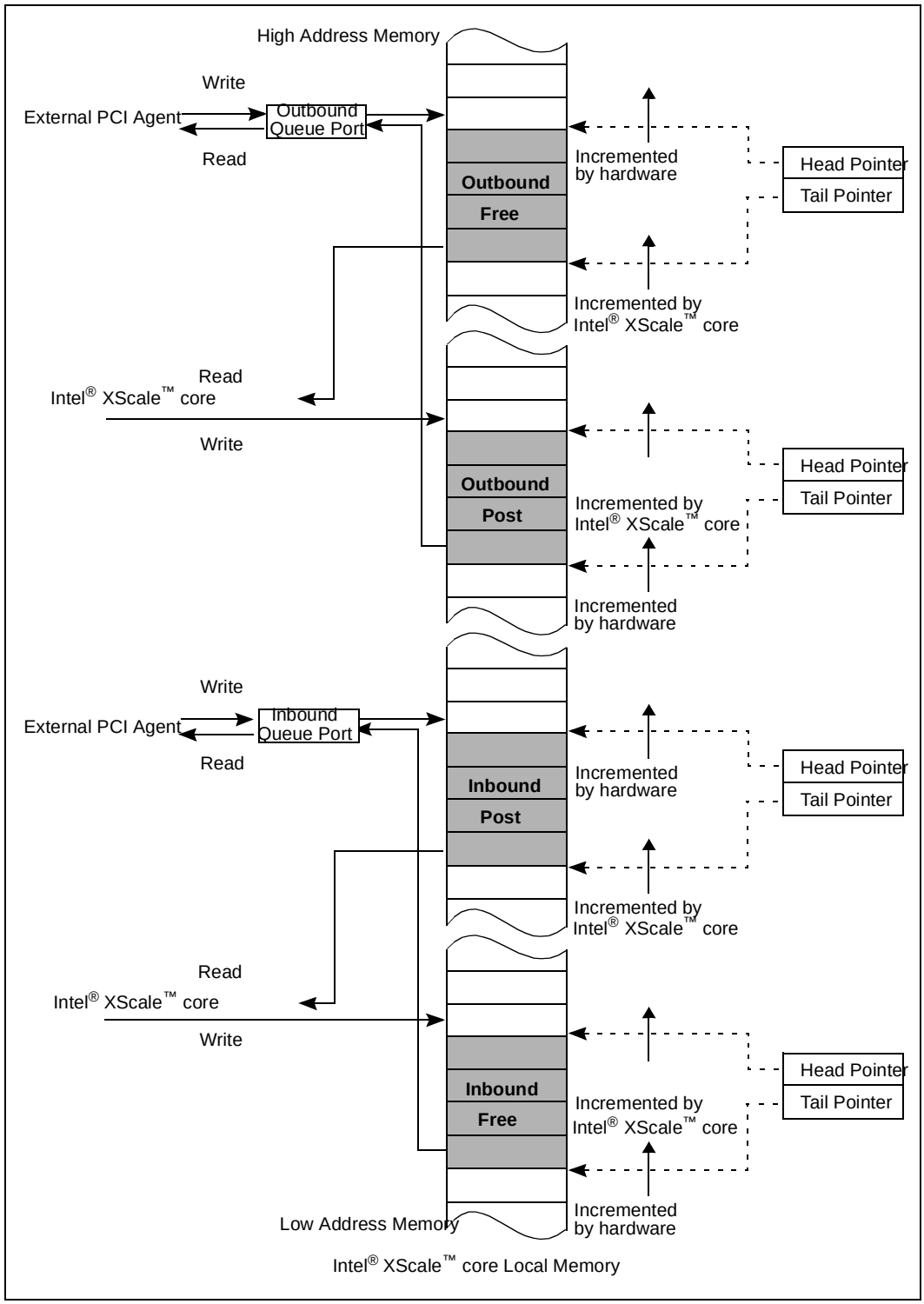
There is one base address for all four queues. It is stored in the Queue Base Address Register (QBAR). The starting addresses of each queue is based on the Queue Base Address and the Queue Size field. Table 151 shows an example of how the circular queues should be set up based on the *Intelligent I/O (I_2O) Architecture Specification*. Other ordering of the circular queues is possible.

Table 151. Queue Starting Addresses

Queue	Starting Address
Inbound Free Queue	QBAR
Inbound Post Queue	QBAR + Queue Size
Outbound Post Queue	QBAR + 2 * Queue Size
Outbound Free Queue	QBAR + 3 * Queue Size

Figure 23 provides a more detailed diagram of the usage of the Circular Queues.

Figure 23. Circular Queue Operation



4.5.1 Inbound Free Queue

The Inbound Free Queue holds free inbound messages placed there by the Intel® XScale™ core for other processors to use. This queue is read from the queue tail by external PCI agents. It is written to the queue head by the Intel® XScale™ core. The tail pointer is maintained by the MU hardware. The head pointer is maintained by the Intel® XScale™ core.

For a PCI read transaction that accesses the Inbound Queue Port, the MU attempts to read the data at the local memory address in the Inbound Free Tail Pointer. When the queue is not empty (head and tail pointers are not equal) or full (head and tail pointers are equal but the head pointer was last written by software), the data is returned. When the queue is empty (head and tail pointers are equal and the head pointer was last updated by hardware), the value of -1 (FFFF.FFFFH) is returned. When the queue was not empty and the MU succeeded in returning the data at the tail, the MU hardware must increment the value in the Inbound Free Tail Pointer Register.

To reduce latency for the PCI read access, the MU implements a prefetch mechanism to anticipate accesses to the Inbound Free Queue. The MU hardware prefetches the data at the tail of the Inbound Free Queue and load it into an internal prefetch register. When the PCI read access occurs, the data is read directly from the prefetch register.

The prefetch mechanism loads a value of -1 (FFFF.FFFFH) into the prefetch register when the head and tail pointers are equal and the queue is empty. In order to update the prefetch register when messages are added to the queue and it becomes non-empty, the prefetch mechanism automatically starts a prefetch when the prefetch register contains FFFF.FFFFH and the Inbound Free Head Pointer Register is written. The Intel® XScale™ core needs to update the Inbound Free Head Pointer Register when it adds messages to the queue.

A prefetch must appear atomic from the perspective of the external PCI agent. When a prefetch is started, any PCI transaction that attempts to access the Inbound Free Queue is signalled a Retry until the prefetch is completed.

The Intel® XScale™ core may place messages in the Inbound Free Queue by writing the data to the local memory location pointed to by the Inbound Free Head Pointer Register. The processor must then increment the Inbound Free Head Pointer Register.

4.5.2 Inbound Post Queue

The Inbound Post Queue holds posted messages placed there by other processors for the Intel® XScale™ core to process. This queue is read from the queue tail by the Intel® XScale™ core. It is written to the queue head by external PCI agents. The tail pointer is maintained by the Intel® XScale™ core. The head pointer is maintained by the MU hardware.

For a PCI write transaction that accesses the Inbound Queue Port, the MU writes the data to the local memory location address in the Inbound Post Head Pointer Register.

When the data written to the Inbound Queue Port is written to local memory, the MU hardware increments the Inbound Post Head Pointer Register.

An Intel® XScale™ core interrupt may be generated when the Inbound Post Queue is written. The Inbound Post Queue Interrupt bit in the Inbound Interrupt Status Register indicates the interrupt status. The interrupt is cleared when the Inbound Post Queue Interrupt bit is cleared. The interrupt can be masked by the Inbound Interrupt Mask Register. Software must be aware of the state of the Inbound Post Queue Interrupt Mask bit to guarantee that the full condition is recognized by the core processor. In addition, to guarantee that the queue does not get overwritten, software must process messages from the tail of the queue before incrementing the tail pointer and clearing this interrupt. Once cleared, an interrupt is NOT generated when the head and tail pointers remain unequal (i.e. queue status is Not Empty). Only a new message posting the in the inbound queue generates a new interrupt. Therefore, when software leaves any unprocessed messages in the post queue when the interrupt is cleared, software must retain the information that the Inbound Post queue status.

From the time that the PCI write transaction is received until the data is written in local memory and the Inbound Post Head Pointer Register is incremented, any PCI transaction that attempts to access the Inbound Post Queue Port is signalled a Retry.

The Intel® XScale™ core may read messages from the Inbound Post Queue by reading the data from the local memory location pointed to by the Inbound Post Tail Pointer Register. The Intel® XScale™ core must then increment the Inbound Post Tail Pointer Register. When the Inbound Post Queue is full (head and tail pointers are equal and the head pointer was last updated by hardware), the hardware retries any PCI writes until a slot in the queue becomes available. A slot in the post queue becomes available by the Intel® XScale™ core incrementing the tail pointer.

4.5.3 Outbound Post Queue

The Outbound Post Queue holds outbound posted messages placed there by the Intel® XScale™ core for other processors to process. This queue is read from the queue tail by external PCI agents. It is written to the queue head by the Intel® XScale™ core. The tail pointer is maintained by the MU hardware. The head pointer is maintained by the Intel® XScale™ core.

For a PCI read transaction that accesses the Outbound Queue Port, the MU attempts to read the data at the local memory address in the Outbound Post Tail Pointer Register. When the queue is not empty (head and tail pointers are not equal) or full (head and tail pointers are equal but the head pointer was last written by software), the data is returned. When the queue is empty (head and tail pointers are equal and the head pointer was last updated by hardware), the value of -1 (FFFF.FFFFH) is returned. When the queue was not empty and the MU succeeded in returning the data at the tail, the MU hardware must increment the value in the Outbound Post Tail Pointer Register.

To reduce latency for the PCI read access, the MU implements a prefetch mechanism to anticipate accesses to the Outbound Post Queue. The MU hardware prefetches the data at the tail of the Outbound Post Queue and load it into an internal prefetch register. When the PCI read access occurs, the data is read directly from the prefetch register.

The prefetch mechanism loads a value of -1 (FFFF.FFFFH) into the prefetch register when the head and tail pointers are equal and the queue is empty. In order to update the prefetch register when messages are added to the queue and it becomes non-empty, the prefetch mechanism automatically starts a prefetch when the prefetch register contains FFFF.FFFFH and the Outbound Post Head Pointer Register is written. The Intel® XScale™ core needs to update the Outbound Post Head Pointer Register when it adds messages to the queue.

A prefetch must appear atomic from the perspective of the external PCI agent. When a prefetch is started, any PCI transaction that attempts to access the Outbound Post Queue is signalled a Retry until the prefetch is completed.

A PCI interrupt may be generated when data in the prefetch buffer is valid. When the prefetch queue is clear, no interrupt is generated. The Outbound Post Queue Interrupt bit in the Outbound Interrupt Status Register shall indicate the status of the prefetch buffer data and therefore the interrupt status. The interrupt is cleared when any prefetched data has been read from the Outbound Queue Port. The interrupt can be masked by the Outbound Interrupt Mask Register.

The Intel® XScale™ core may place messages in the Outbound Post Queue by writing the data to the local memory address in the Outbound Post Head Pointer Register. The processor must then increment the Outbound Post Head Pointer Register.

4.5.4 Outbound Free Queue

The Outbound Free Queue holds free messages placed there by other processors for the Intel® XScale™ core to use. This queue is read from the queue tail by the Intel® XScale™ core. It is written to the queue head by external PCI agents. The tail pointer is maintained by the Intel® XScale™ core. The head pointer is maintained by the MU hardware.

For a PCI write transaction that accesses the Outbound Queue Port, the MU writes the data to the local memory address in the Outbound Free Head Pointer Register. When the data written to the Outbound Queue Port is written to local memory, the MU hardware increments the Outbound Free Head Pointer Register.

When the head pointer and the tail pointer become equal and the queue is full, the MU may signal an interrupt to the Intel® XScale™ core to register the queue full condition. This interrupt is recorded in the Inbound Interrupt Status Register. The interrupt is cleared when the Outbound Free Queue Full Interrupt bit is cleared and not by writing to the head or tail pointers. The interrupt can be masked by the Inbound Interrupt Mask Register. Software must be aware of the state of the Outbound Free Queue Interrupt Mask bit to guarantee that the full condition is recognized by the core processor.

From the time that a PCI write transaction is received until the data is written in local memory and the Outbound Free Head Pointer Register is incremented, any PCI transaction that attempts to access the Outbound Free Queue Port is signalled a retry.

The Intel® XScale™ core may read messages from the Outbound Free Queue by reading the data from the local memory address in the Outbound Free Tail Pointer Register. The processor must then increment the Outbound Free Tail Pointer Register. When the Outbound Free Queue is full, the hardware must retry any PCI writes until a slot in the queue becomes available.

Table 152. Circular Queue Summary

Queue Name	PCI Port	Generate PCI Interrupt	Generate Intel® Xscale™ Core Interrupt	Head Pointer maintained by	Tail Pointer maintained by
Inbound Post Queue	Inbound Queue Port	No	Yes, when queue is written	MU hardware	Intel® XScale™ microarchitecture
Inbound Free Queue		No	No	Intel® XScale™ microarchitecture	MU hardware
Outbound Post Queue	Outbound Queue Port	Yes, when prefetch buffer data is valid	No	Intel® XScale™ microarchitecture	MU hardware
Outbound Free Queue		No	Yes, when queue is full	MU hardware	Intel® XScale™ microarchitecture

Table 153. Circular Queue Status Summary

Queue Name	Queue Status	Head & Tail Pointer	Last Pointer Update
Inbound Post Queue & Outbound Free Queue	Empty	Equal	Tail pointer last updated by software
	Not Empty	Not Equal	N/A
	Full	Equal	Head pointer last updated by hardware
Inbound Free Queue & Outbound Post Queue	Empty	Equal	Head pointer last updated by hardware
	Not Empty	Not Equal	
	Full	Equal	Tail pointer last updated by software

4.6 Index Registers

The Index Registers are a set of 1004 registers that when written by an external PCI agent can generate an interrupt to the Intel® XScale™ core. These registers are for inbound messages only. The interrupt is recorded in the Inbound Interrupt Status Register.

The storage for the Index Registers is allocated from the 80331 local memory. PCI write accesses to the Index Registers write the data to local memory. PCI read accesses to the Index Registers read the data from local memory. The local memory used for the Index Registers ranges from Inbound ATU Translate Value Register + 050H to Inbound ATU Translate Value Register + FFFH. Chapter 3, "Address Translation Unit" describes how PCI addresses are translated to local memory addresses.

The address of the first write access is stored in the Index Address Register. This register is written during the earliest write access and provides a means to determine which Index Register was written. Once updated by the MU, the Index Address Register is not updated until the Index Register Interrupt bit in the Inbound Interrupt Status Register is cleared. When the interrupt is cleared, the Index Address Register is re-enabled and stores the address of the next Index Register write access.

Writes by the Intel® XScale™ core to the local memory used by the Index Registers does not cause an interrupt and does not update the Index Address Register.

The index registers can be accessed with Multi-DWORD reads and single QWORD aligned writes.

4.7 Messaging Unit Error Conditions

The Messaging Unit, like the ATU, encounters error conditions on the PCI interface as well as the internal bus interface. As a PCI target, all PCI errors (parity and aborts) are captured and recorded in the ATU Status Register and can be masked using the ATU mechanisms. Refer to Chapter 3, "Address Translation Unit" for further details.

4.8 Message-Signaled Interrupts

The Messaging Unit is responsible for the generation of all of the Outbound Interrupts from the 80331. These interrupts can be delivered to the Host Processor via the **P_INTC#** output pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) or the Message Signaled Interrupt (MSI) mechanism.

When a host processor enables Message-Signaled Interrupts (MSI) on the 80331, an outbound interrupt is signaled to the host via a PCI write instead of the assertion of the **P_INTC#** output pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low).

The *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a states that "PCI-X devices that generate interrupts are required to support message-signaled interrupts, as defined by the *PCI Local Bus Specification*, Revision 2.2 and must support a 64-bit message address." "Devices that require interrupts in systems that do not support message-signaled interrupts, must implement interrupt pins." Thus, the 80331 needs to implement both wired and message-signaled interrupt delivery mechanisms.

In support of MSI, the 80331 implements the MSI capability structure. The capability structure includes the "MSI Message Control Register - MSI_MCR" on page 325, the "MSI Message Address Register - MSI_MAR" on page 326, the "MSI Message Upper Address Register - MSI_MUAR" on page 327 and the"MSI Message Data Register- MSI_MDR" on page 328.

During system initialization, the configuration software for an MSI system reads the Message Control Register to determine that the 80331 supports a 64-bit Message Address, and that it is capable of generating two unique interrupt messages.

After gathering this data from all of the MSI capable devices in the system, the configuration software decides where to initialize the Message Address and how many unique messages each MSI capable device is allowed. Then, software writes the Message Address Registers (and the Message Upper Address Registers when Message Address is above the 4G address boundary[1]), and the Message Data Register. This system specified data is used to route the interrupt request message to the appropriate entry in a host processor Local APIC table.

Configuration of MSI completes with a write to the Message Control Register which includes an update to the Multiple Message Enable field and the MSI enable bit of each device. This informs the device how many unique messages (Local APIC table entries) have been allocated for exclusive use by that device and enable that device for MSI. Device hardware is required to handle allocation of fewer unique interrupt messages than requested by the Multiple Message Capable field.

80331 is able to handle generating only one message, even though the device is capable of generating two unique messages. When two unique messages are enabled, one message is reserved for Outbound Post Queue Interrupt and the other message represents all of Outbound Doorbell and Outbound Message Interrupts. When only one message is enabled, all interrupts are represented by a single message. Interrupt handler software needs to read 80331 Outbound Interrupt Status Register to determine the interrupt cause when more than one source is represented by a single message.

To signal an Outbound Interrupt with MSI enabled, the 80331 creates an outbound write transaction using the Message Address and the Message Data. When two unique messages are enabled, the lowest order bit of the Message Data is modified by hardware so that the host processor can distinguish between them.

Note: When host software enables MSI, a Messaging Unit Interrupt does not result in the assertion of the **P_INTC#** output pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low). However, all the **S_INT[D:A]#** pins is functional for steering of interrupts from other PCI devices that may not be MSI capable.

1. When host software writes the Message Upper address register to a non-zero value, device hardware uses a write transaction with a Dual Address Cycle (DAC) to present the full 64-bit address to the bus.

4.8.1 Level-Triggered Versus Edge-Triggered Interrupts

When MSI is disabled, the **P_INTC#** output pin (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) remains asserted and pended to the host when **any** of the MU interrupt sources requires service (Outbound Post Queue, Outbound Doorbell and Outbound Message). Since the PCI pin signaled interrupt is **level-triggered**, the interrupt service routine does not drop out of the service routine until the interrupt signal is deasserted. This guarantees that an interrupt is not missed.

MSI interrupts are inherently edge-triggered, in that an interrupt is only pended to the host as a write event when any of the MU interrupt sources requires service. The MU generates a MSI message when a interrupt status bit in the OISR is set. Additional MSI messages is generated by the MU when other bits are set in the OISR, that are previously clear.

Note: The MU does NOT generate a MSI message when one pending interrupt is cleared in the OISR and other pending interrupt status bits remain set. Interrupt service routines must service and clear all pending interrupts from the OISR before exiting to insure no interrupts are missed.

4.9 Register Definitions

The following registers are located in the PCI address space and in the Peripheral Memory-Mapped Register (PMMR) address space. They are accessible through PCI bus transactions and through Intel® XScale™ core internal bus accesses. In the PCI address space, they are mapped into the first 80 bytes of the inbound address window of the ATU.

- Inbound Message 0 Register
- Inbound Message 1 Register
- Outbound Message 0 Register
- Outbound Message 1 Register
- Inbound Doorbell Register
- Inbound Interrupt Status Register
- Inbound Interrupt Mask Register
- Outbound Doorbell Register
- Outbound Interrupt Status Register
- Outbound Interrupt Mask Register

The following registers are located in the Peripheral Memory-Mapped Register (PMMR) address space as described in Chapter 17, "Peripheral Registers".

- MU Configuration Register
- Queue Base Address Register
- Inbound Free Head Pointer Register
- Inbound Free Tail Pointer Register
- Inbound Post Head Pointer Register
- Inbound Post Tail Pointer Register
- Outbound Free Head Pointer Register
- Outbound Free Tail Pointer Register
- Outbound Post Head Pointer Register
- Outbound Post Tail Pointer Register
- Index Address Register

Reading or writing a register that is reserved is undefined.

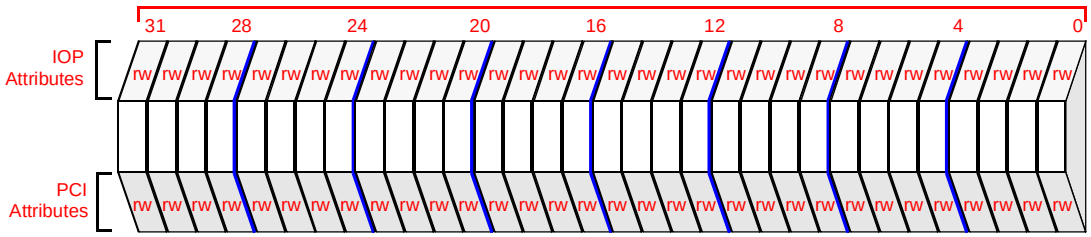
Table 154. Message Unit Register

Internal Bus Address	Section, Register Name - Acronym (Page)
FFFF.E310H	Section 4.9.1, "Inbound Message Register - IMRx" on page 304
FFFF.E314H	Section 4.9.1, "Inbound Message Register - IMRx" on page 304
FFFF.E318H	Section 4.9.2, "Outbound Message Register - OMRx" on page 305
FFFF.E31CH	Section 4.9.2, "Outbound Message Register - OMRx" on page 305
FFFF.E320H	Section 4.9.3, "Inbound Doorbell Register - IDR" on page 306
FFFF.E324H	Section 4.9.4, "Inbound Interrupt Status Register - IISR" on page 307
FFFF.E328H	Section 4.9.5, "Inbound Interrupt Mask Register - IIMR" on page 308
FFFF.E32CH	Section 4.9.6, "Outbound Doorbell Register - ODR" on page 309
FFFF.E330H	Section 4.9.7, "Outbound Interrupt Status Register - OISR" on page 310
FFFF.E334H	Section 4.9.8, "Outbound Interrupt Mask Register - OIMR" on page 311
FFFF.E350H	Section 4.9.9, "MU Configuration Register - MUCR" on page 312
FFFF.E354H	Section 4.9.10, "Queue Base Address Register - QBAR" on page 313
FFFF.E360H	Section 4.9.11, "Inbound Free Head Pointer Register - IFHPR" on page 314
FFFF.E364H	Section 4.9.12, "Inbound Free Tail Pointer Register - IFTPR" on page 315
FFFF.E368H	Section 4.9.13, "Inbound Post Head Pointer Register - IPHPR" on page 316
FFFF.E36CH	Section 4.9.14, "Inbound Post Tail Pointer Register - IPTPR" on page 317
FFFF.E370H	Section 4.9.15, "Outbound Free Head Pointer Register - OFHPR" on page 318
FFFF.E374H	Section 4.9.16, "Outbound Free Tail Pointer Register - OFTPR" on page 319
FFFF.E378H	Section 4.9.17, "Outbound Post Head Pointer Register - OPHPR" on page 320
FFFF.E37CH	Section 4.9.18, "Outbound Post Tail Pointer Register - OPTPR" on page 321
FFFF.E380H	Section 4.9.19, "Index Address Register - IAR" on page 322

4.9.1 Inbound Message Register - IMRx

There are two Inbound Message Registers: IMR0 and IMR1. When the IMR register is written, an interrupt to the Intel® XScale™ core may be generated. The interrupt is recorded in the Inbound Interrupt Status Register and may be masked by the Inbound Message Interrupt Mask bit in the Inbound Interrupt Mask Register.

Table 155. Inbound Message Register - IMRx



	internal bus address
IMR0	FFFF.E310H
IMR1	FFFF.E314H

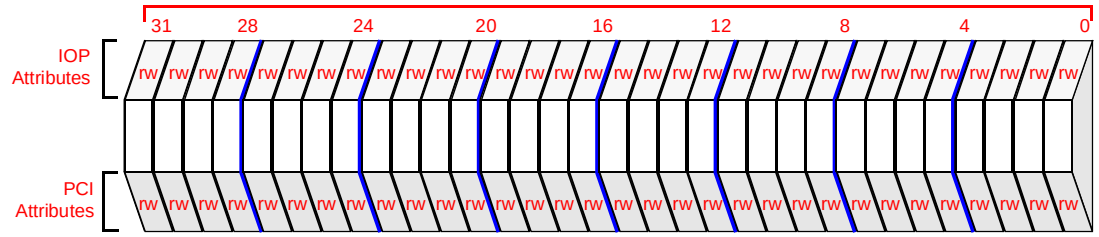
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	Inbound Message - This is a 32-bit message written by an external PCI agent. When written, an interrupt to the Intel® XScale™ core may be generated.

4.9.2 Outbound Message Register - OMRx

There are two Outbound Message Registers: OMR0 and OMR1. When the OMR register is written, a PCI interrupt may be generated. The interrupt is recorded in the Outbound Interrupt Status Register and may be masked by the Outbound Message Interrupt Mask bit in the Outbound Interrupt Mask Register.

Table 156. Outbound Message Register - OMRx



	internal bus address
OMR0	FFFF.E318H
OMR1	FFFF.E31CH

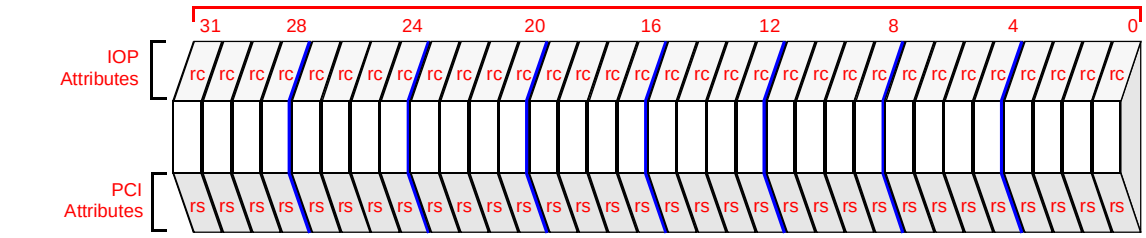
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	Outbound Message - This is 32-bit message written by the Intel® XScale™ core. When written, an interrupt may be generated on the PCI Interrupt pin determined by the ATU Interrupt Pin Register.

4.9.3 Inbound Doorbell Register - IDR

The Inbound Doorbell Register (IDR) is used to generate interrupts to the Intel® XScale™ core. Bit 31 is reserved for generating an Error Doorbell interrupt. When bit 31 is set, an Error interrupt may be generated to the Intel® XScale™ core. All other bits, when set, cause the Normal Messaging Unit interrupt line of the Intel® XScale™ core to be asserted, when the interrupt is not masked by the Inbound Doorbell Interrupt Mask bit in the Inbound Interrupt Mask Register. The bits in the IDR register can only be set by an external PCI agent and can only be cleared by the Intel® XScale™ core.

Table 157. Inbound Doorbell Register - IDR



IDR — internal bus address: FFFF.E320H

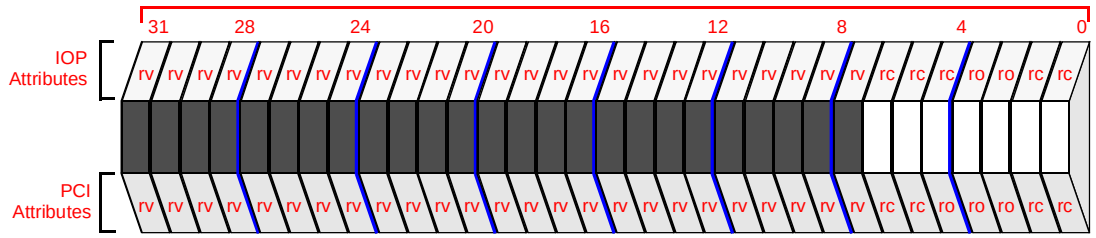
Attribute Legend:
RW = Read/Write
RV = Reserved
RC = Read Clear
PR = Preserved
RO = Read Only
RS = Read/Set
NA = Not Accessible

Bit	Default	Description
31	0_2	Error Interrupt - Generate an Error Interrupt to the Intel® XScale™ core.
30:00	00000000H	Normal Interrupt - When any bit is set, generate a Normal interrupt to the Intel® XScale™ core. When all bits are clear, do not generate a Normal Interrupt.

4.9.4 Inbound Interrupt Status Register - IISR

The Inbound Interrupt Status Register (IISR) contains hardware interrupt status. It records the status of Intel® XScale™ core interrupts generated by the Message Registers, Doorbell Registers, and the Circular Queues. All interrupts are routed to the Normal Messaging Unit interrupt input of the Intel® XScale™ core, except for the Error Doorbell Interrupt and the Outbound Free Queue Full interrupt; these two are routed to the Messaging Unit Error interrupt input. The generation of interrupts recorded in the Inbound Interrupt Status Register may be masked by setting the corresponding bit in the Inbound Interrupt Mask Register. Some of the bits in this register are Read Only. For those bits, the interrupt must be cleared through another register.

Table 158. Inbound Interrupt Status Register - IISR



IISR — internal bus address: FFFF.E324H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

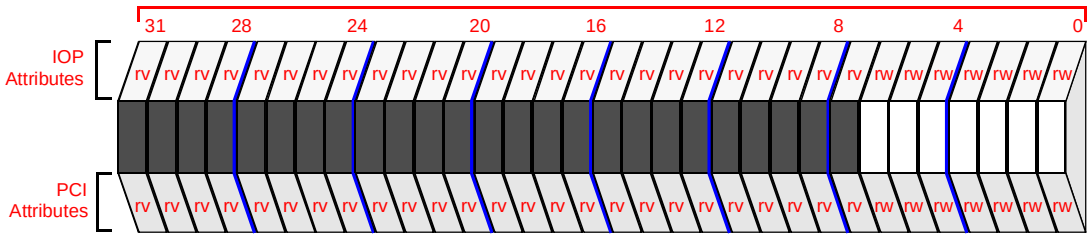
Bit	Default	Description
31:07	0000000H 0_2	Reserved
06	0_2	Index Register Interrupt - This bit is set by the MU hardware when an Index Register has been written after a PCI transaction.
05	0_2	Outbound Free Queue Full Interrupt - This bit is set when the Outbound Free Head Pointer becomes equal to the Tail Pointer and the queue is full. An Error interrupt is generated for this condition.
04	0_2	Inbound Post Queue Interrupt - This bit is set by the MU hardware when the Inbound Post Queue has been written. Once cleared, an interrupt does NOT be generated when the head and tail pointers remain unequal (i.e. queue status is Not Empty). Therefore, when software leaves any unprocessed messages in the post queue when the interrupt is cleared, software must retain the information that the Inbound Post queue status is not empty. **NOTE:** This interrupt is provided with dedicated support in the 80331 Interrupt Controller.
03	0_2	Error Doorbell Interrupt - This bit is set when the Error Interrupt of the Inbound Doorbell Register is set. To clear this bit (and the interrupt), the Error Interrupt bit of the Inbound Doorbell Register must be clear.
02	0_2	Inbound Doorbell Interrupt - This bit is set when at least one Normal Interrupt bit in the Inbound Doorbell Register is set. To clear this bit (and the interrupt), the Normal Interrupt bits in the Inbound Doorbell Register must all be clear.
01	0_2	Inbound Message 1 Interrupt - This bit is set by the MU hardware when the Inbound Message 1 Register has been written.
00	0_2	Inbound Message 0 Interrupt - This bit is set by the MU hardware when the Inbound Message 0 Register has been written.

4.9.5 Inbound Interrupt Mask Register - IIMR

The Inbound Interrupt Mask Register (IIMR) provides the ability to mask Intel® XScale™ core interrupts generated by the Messaging Unit. Each bit in the Mask register corresponds to an interrupt bit in the Inbound Interrupt Status Register.

Setting or clearing bits in this register does not affect the Inbound Interrupt Status Register. They only affect the generation of the Intel® XScale™ core interrupt.

Table 159. Inbound Interrupt Mask Register - IIMR



IIMR — internal bus address: FFFF.E328H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

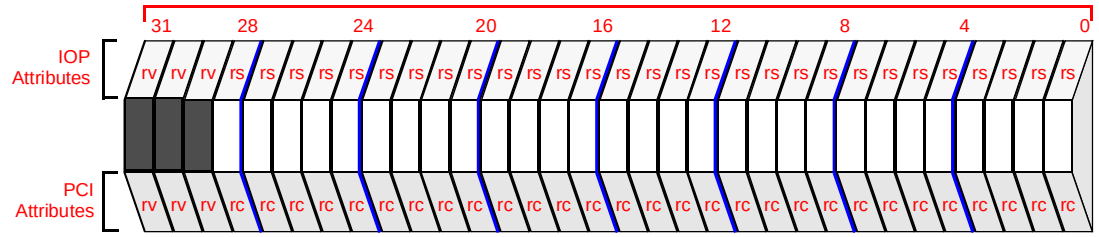
Bit	Default	Description
31:07	000000H 0_2	Reserved
06	0_2	Index Register Interrupt Mask - When set, this bit masks the interrupt generated by the MU hardware when an Index Register has been written after a PCI transaction.
05	0_2	Outbound Free Queue Full Interrupt Mask - When set, this bit masks the Error interrupt generated when the Outbound Free Head Pointer becomes equal to the Tail Pointer and the queue is full.
04	0_2	Inbound Post Queue Interrupt Mask - When set, this bit masks the interrupt generated by the MU hardware when the Inbound Post Queue has been written.
03	0_2	Error Doorbell Interrupt Mask - When set, this bit masks the Error Interrupt when the Error Interrupt bit of the Inbound Doorbell Register is set.
02	0_2	Inbound Doorbell Interrupt Mask - When set, this bit masks the interrupt generated when at least one Normal Interrupt bit in the Inbound Doorbell Register is set.
01	0_2	Inbound Message 1 Interrupt Mask - When set, this bit masks the Inbound Message 1 Interrupt generated by a write to the Inbound Message 1 Register.
00	0_2	Inbound Message 0 Interrupt Mask - When set, this bit masks the Inbound Message 0 Interrupt generated by a write to the Inbound Message 0 Register.

4.9.6 Outbound Doorbell Register - ODR

The Outbound Doorbell Register (ODR) allows software interrupt generation. It allows the Intel® XScale™ core to generate PCI interrupts to the host processor by writing to this register. The generation of PCI interrupts through the Outbound Doorbell Register may be masked by setting the Outbound Doorbell Interrupt Mask bit in the Outbound Interrupt Mask Register.

The Software Interrupt bits in this register can only be set by the Intel® XScale™ core and can only be cleared by an external PCI agent.

Table 160. Outbound Doorbell Register - ODR



internal bus address

ODR FFFF.E32CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

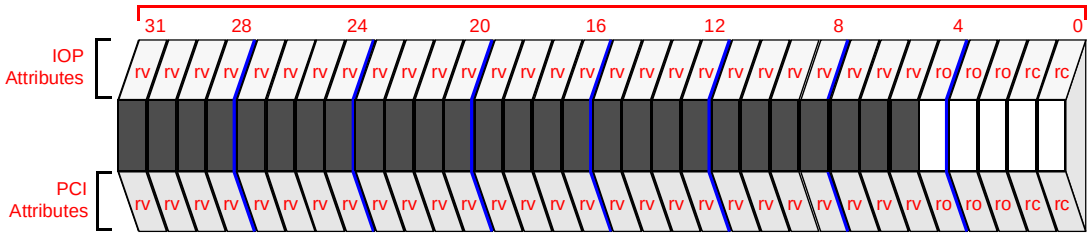
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	Reserved
30	0_2	Reserved.
29	0_2	Reserved
28	0000 0000H	PCI Interrupt - When set, this bit causes the **P_INTC#** interrupt output (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) signal to be asserted or a Message-signaled Interrupt is generated (when enabled). When this bit is cleared, the **P_INTC#** interrupt output (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) signal is deasserted.
27:00	000 0000H	Software Interrupts - When any bit is set the **P_INTC#** interrupt output (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) signal is asserted or a Message-signaled Interrupt is generated (when enabled). When all bits are cleared, the **P_INTC#** interrupt output (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) signal is deasserted.

4.9.7 Outbound Interrupt Status Register - OISR

The Outbound Interrupt Status Register (OISR) contains hardware interrupt status. It records the status of PCI interrupts generated by the Message Registers, Doorbell Registers, and the Circular Queues. The generation of PCI interrupts recorded in the Outbound Interrupt Status Register may be masked by setting the corresponding bit in the Outbound Interrupt Mask Register. Some of the bits in this register are Read Only. For those bits, the interrupt must be cleared through another register.

Table 161. Outbound Interrupt Status Register - OISR



internal bus address

OISR FFFF.E330H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

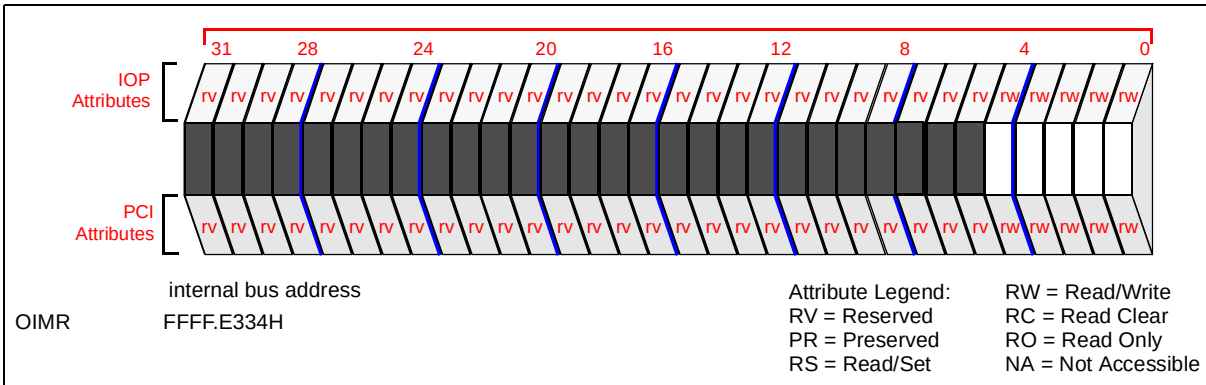
Bit	Default	Description
31:05	000000H 000_2	Reserved
04	0_2	PCI Interrupt - This bit is set when the PCI Interrupt bit (bit 28) is set in the Outbound Doorbell Register. To clear this bit (and the interrupt), the PCI Interrupt bit must be cleared.
03	0_2	Outbound Post Queue Interrupt - This bit is set when data in the prefetch buffer is valid. This bit is cleared when any prefetch data has been read from the Outbound Queue Port.
02	0_2	Outbound Doorbell Interrupt - This bit is set when at least one Software Interrupt bit in the Outbound Doorbell Register is set. To clear this bit (and the interrupt), the Software Interrupt bits in the Outbound Doorbell Register must all be clear.
01	0_2	Outbound Message 1 Interrupt - This bit is set by the MU when the Outbound Message 1 Register is written. Clearing this bit clears the interrupt.
00	0_2	Outbound Message 0 Interrupt - This bit is set by the MU when the Outbound Message 0 Register is written. Clearing this bit clears the interrupt.

4.9.8 Outbound Interrupt Mask Register - OIMR

The Outbound Interrupt Mask Register (OIMR) provides the ability to mask outbound PCI interrupts generated by the Messaging Unit. Each bit in the mask register corresponds to a hardware interrupt bit in the Outbound Interrupt Status Register. When the bit is set, the PCI interrupt is not generated. When the bit is clear, the interrupt is allowed to be generated.

Setting or clearing bits in this register does not affect the Outbound Interrupt Status Register. They only affect the generation of the PCI interrupt.

Table 162. Outbound Interrupt Mask Register - OIMR

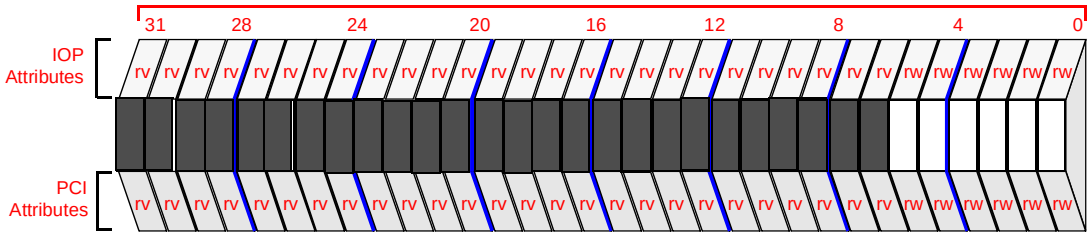


Bit	Default	Description
31:05	000000H	Reserved
04	0_2	PCI Interrupt Mask - When set, this bit masks the interrupt generation when the PCI Interrupt bit (bit 28) in the in the Outbound Doorbell Register is set.
03	0_2	Outbound Post Queue Interrupt Mask - When set, this bit masks the interrupt generated when data in the prefetch buffer is valid.
02	0_2	Outbound Doorbell Interrupt Mask - When set, this bit masks the interrupt generated by the Outbound Doorbell Register.
01	0_2	Outbound Message 1 Interrupt Mask - When set, this bit masks the Outbound Message 1 Interrupt generated by a write to the Outbound Message 1 Register.
00	0_2	Outbound Message 0 Interrupt Mask- When set, this bit masks the Outbound Message 0 Interrupt generated by a write to the Outbound Message 0 Register.

4.9.9 MU Configuration Register - MUCR

The MU Configuration Register (MUCR) contains the Circular Queue Enable bit and the size of one Circular Queue. The Circular Queue Enable bit enables or disables the Circular Queues. The Circular Queues are disabled at reset to allow the software to initialize the head and tail pointer registers before any PCI accesses to the Queue Ports. Each Circular Queue may range from 4 K entries (16 Kbytes) to 64 K entries (256 Kbytes) and there are four Circular Queues.

Table 163. MU Configuration Register - MUCR



internal bus address

MUCR FFFF.E350H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:06	000000H 00_2	Reserved
05:01	00001_2	Circular Queue Size - This field determines the size of each Circular Queue. All four queues are the same size. • 00001_2 - 4K Entries (16 Kbytes) • 00010_2 - 8K Entries (32 Kbytes) • 00100_2 - 16K Entries (64 Kbytes) • 01000_2 - 32K Entries (128 Kbytes) • 10000_2 - 64K Entries (256 Kbytes)
00	0_2	Circular Queue Enable - This bit enables or disables the Circular Queues. When clear the Circular Queues are disabled, however the MU accepts PCI accesses to the Circular Queue Ports but ignores the data for Writes and return FFFF.FFFFH for Reads. Interrupts are not generated to the core when disabled. When set, the Circular Queues are fully enabled.

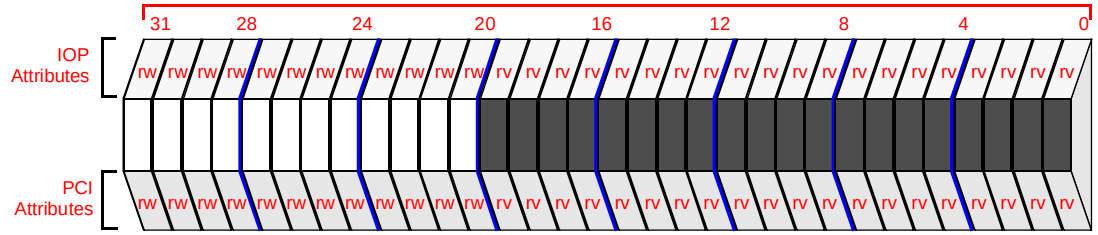
4.9.10 Queue Base Address Register - QBAR

The Queue Base Address Register (QBAR) contains the local memory address of the Circular Queues. The base address is required to be located on a 1 Mbyte address boundary.

All Circular Queue head and tail pointers are based on the QBAR. When the head and tail pointer registers are read, the Queue Base Address is returned in the upper 12 bits. Writing to the upper 12 bits of the head and tail pointer registers does not affect the Queue Base Address or Queue Base Address Register.

Warning: The QBAR must designate a range allocated to the 80331 DDR SDRAM interface (Chapter 8, "Memory Controller").

Table 164. Queue Base Address Register - QBAR



internal bus address

QBAR FFFF.E354H

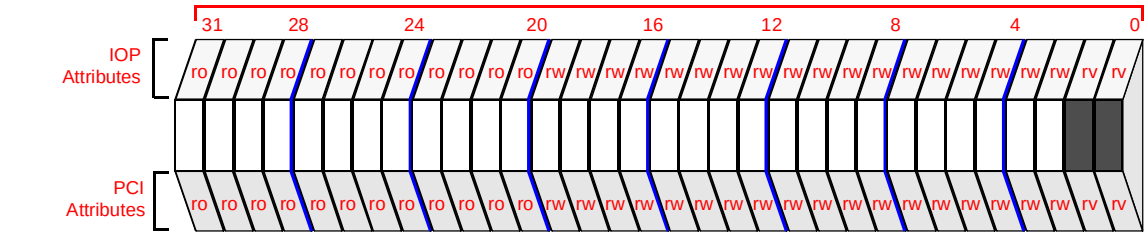
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:00	00000H	Reserved

4.9.11 Inbound Free Head Pointer Register - IFHPR

The Inbound Free Head Pointer Register (IFHPR) contains the local memory offset from the Queue Base Address of the head pointer for the Inbound Free Queue. The Head Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored. This register is maintained by software.

Table 165. Inbound Free Head Pointer Register - IFHPR



internal bus address
IFHPR FFFF.E360H

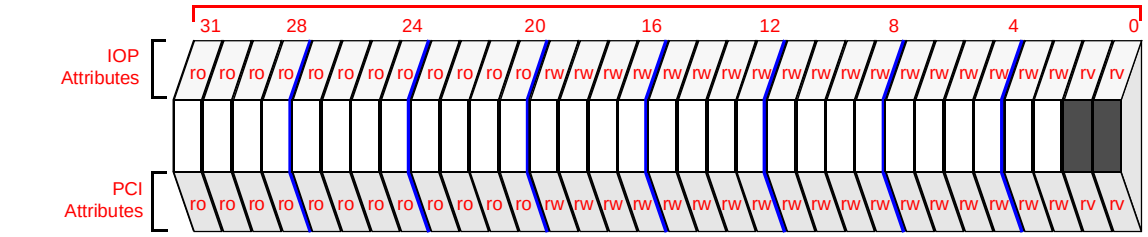
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Inbound Free Head Pointer - Local memory offset of the head pointer for the Inbound Free Queue.
01:00	00_2	Reserved

4.9.12 Inbound Free Tail Pointer Register - IFTPR

The Inbound Free Tail Pointer Register (IFTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Inbound Free Queue. The Tail Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 166. Inbound Free Tail Pointer Register - IFTPR



internal bus address

IFTPR FFFF.E364H

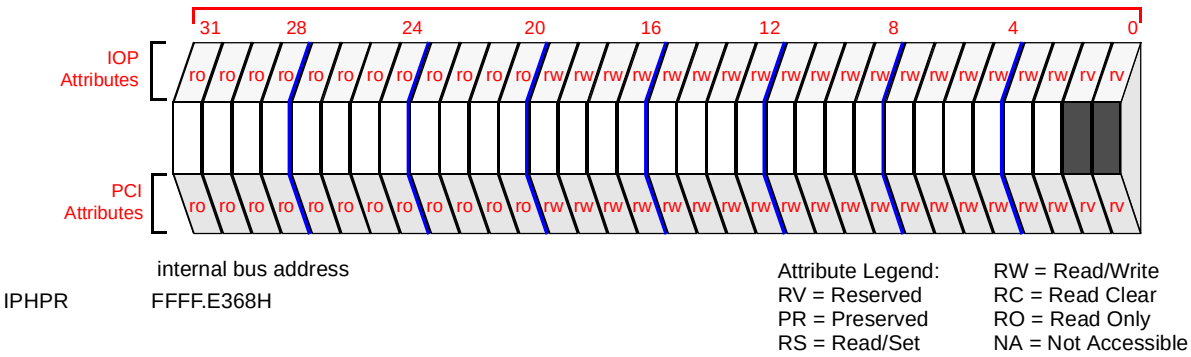
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Inbound Free Tail Pointer - Local memory offset of the tail pointer for the Inbound Free Queue.
01:00	00_2	Reserved

4.9.13 Inbound Post Head Pointer Register - IPHPR

The Inbound Post Head Pointer Register (IPHPR) contains the local memory offset from the Queue Base Address of the head pointer for the Inbound Post Queue. The Head Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 167. Inbound Post Head Pointer Register - IPHPR

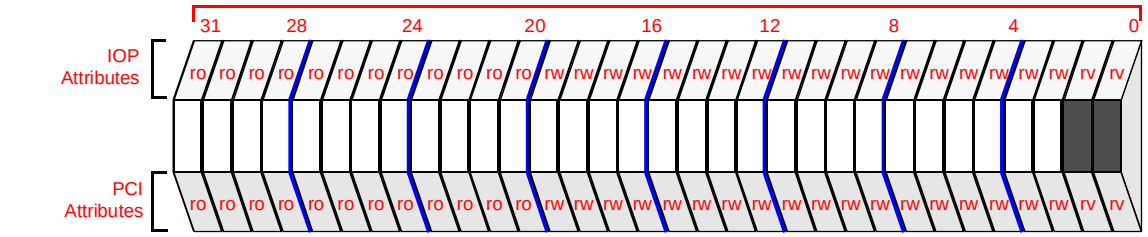


Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Inbound Post Head Pointer - Local memory offset of the head pointer for the Inbound Post Queue.
01:00	00_2	Reserved

4.9.14 Inbound Post Tail Pointer Register - IPTPR

The Inbound Post Tail Pointer Register (IPTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Inbound Post Queue. The Tail Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 168. Inbound Post Tail Pointer Register - IPTPR



internal bus address

IPTPR FFFF.E36CH

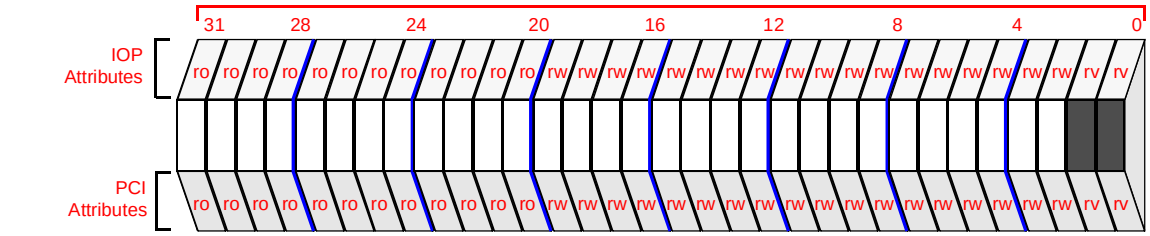
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Inbound Post Tail Pointer - Local memory offset of the tail pointer for the Inbound Post Queue.
01:00	00_2	Reserved

4.9.15 Outbound Free Head Pointer Register - OFHPR

The Outbound Free Head Pointer Register (OFHPR) contains the local memory offset from the Queue Base Address of the head pointer for the Outbound Free Queue. The Head Pointer must be aligned on a DWORD address boundary. This register is maintained by software. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 169. Outbound Free Head Pointer Register - OFHPR



internal bus address
OFHPR FFFF.E370H

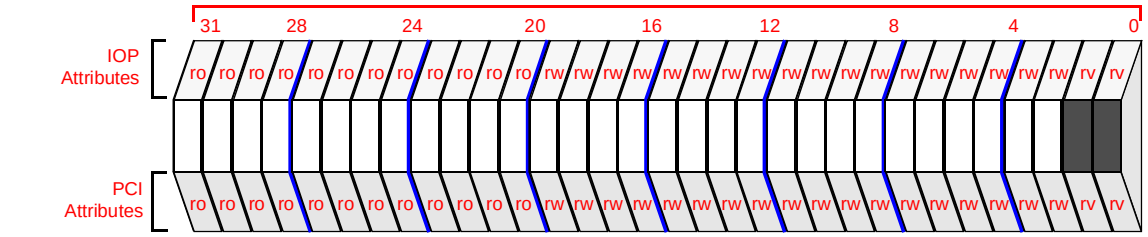
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Outbound Free Head Pointer - Local memory offset of the head pointer for the Outbound Free Queue.
01:00	00_2	Reserved

4.9.16 Outbound Free Tail Pointer Register - OFTPR

The Outbound Free Tail Pointer Register (OFTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Outbound Free Queue. The Tail Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 170. Outbound Free Tail Pointer Register - OFTPR



internal bus address

OFTPR FFFF.E374H

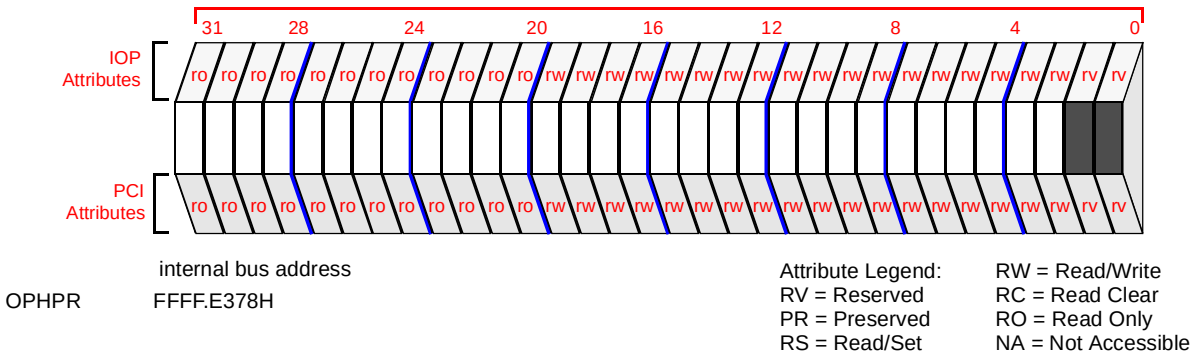
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Outbound Free Tail Pointer - Local memory offset of the tail pointer for the Outbound Free Queue.
01:00	00_2	Reserved

4.9.17 Outbound Post Head Pointer Register - OPHPR

The Outbound Post Head Pointer Register (OPHPR) contains the local memory offset from the Queue Base Address of the head pointer for the Outbound Post Queue. The Head Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 171. Outbound Post Head Pointer Register - OPHPR

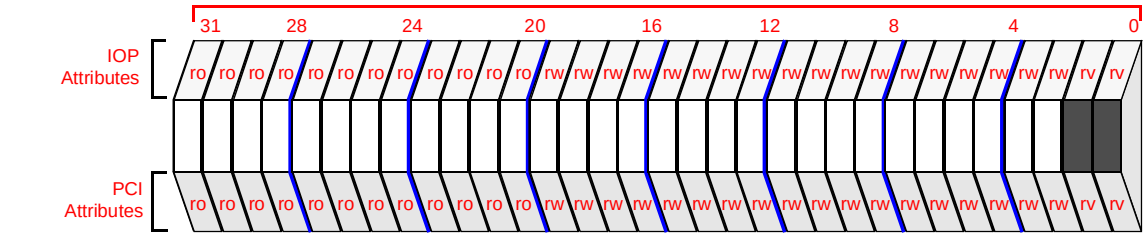


Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Outbound Post Head Pointer - Local memory offset of the head pointer for the Outbound Post Queue.
01:00	00_2	Reserved

4.9.18 Outbound Post Tail Pointer Register - OPTPR

The Outbound Post Tail Pointer Register (OPTPR) contains the local memory offset from the Queue Base Address of the tail pointer for the Outbound Post Queue. The Tail Pointer must be aligned on a DWORD address boundary. When read, the Queue Base Address is provided in the upper 12 bits of the register. Writes to the upper 12 bits of the register are ignored.

Table 172. Outbound Post Tail Pointer Register - OPTPR



internal bus address

OPTPR FFFF.E37CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

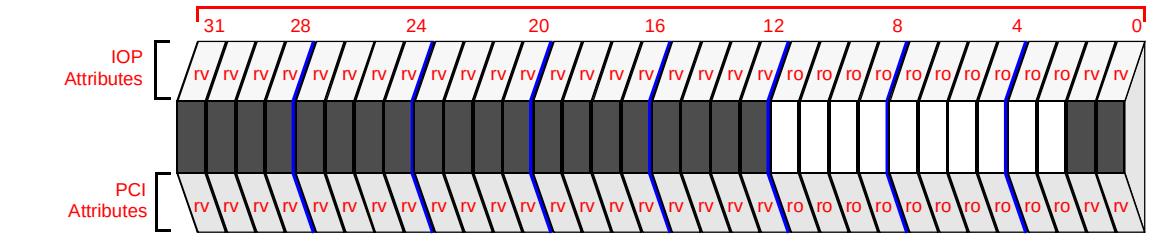
Bit	Default	Description
31:20	000H	Queue Base Address - Local memory address of the circular queues.
19:02	0000H 00_2	Outbound Post Tail Pointer - Local memory offset of the tail pointer for the Outbound Post Queue.
01:00	00_2	Reserved

4.9.19 Index Address Register - IAR

The Index Address Register (IAR) contains the offset of the least recently accessed Index Register. It is written by the MU when the Index Registers are written by a PCI agent. The register is not updated until the Index Interrupt bit in the Inbound Interrupt Status Register is cleared.

The local memory address of the Index Register least recently accessed is computed by adding the Index Address Register to the Inbound ATU Translate Value Register.

Table 173. Index Address Register - IAR



internal bus address

IAR FFFF.E380H

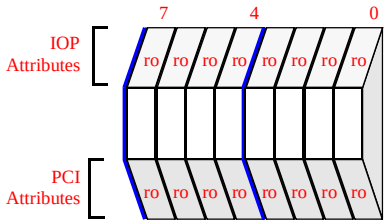
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	000000H	Reserved
11:02	00H 00_2	Index Address - is the local memory offset of the Index Register written (050H to FFCH)
01:00	00_2	Reserved

4.9.20 MSI Capability Identifier Register - MSI_CAPID

The MSI Capability Identifier Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.2. This register in the PCI Extended Capability header identifies the type of Extended Capability contained in that header. The value of 05H in this field identifies the function as message signaled interrupt capable.

Table 174. MSI Capability Identifier Register - MSI_CAPID



PCI Configuration Offset
D0H

Intel® Xscale™ Core Local Bus Address
FFFF E1D0H

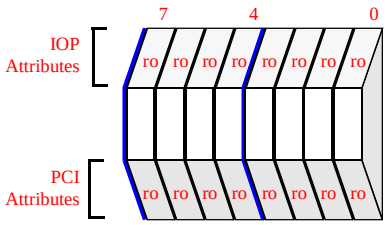
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	05H	MSI_Cap_Id - This field with its 05H value identifies this item in the linked list of Extended Capability Headers as being the message signaled interrupt capability item.

4.9.21 MSI Next Item Pointer Register - MSI_NXTP

The Next Item Pointer Register bits adhere to the definitions in the *PCI Local Bus Specification*, Revision 2.2. This register describes the location of the next item in the function capability list. For the 80331 that is the PCI-X capability header at offset E0H.

Table 175. MSI Next Item Pointer Register - MSI_NXTP



PCI Configuration Offset
D1H

Intel® Xscale™ Core Local Bus Address
FFFF E1D1H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

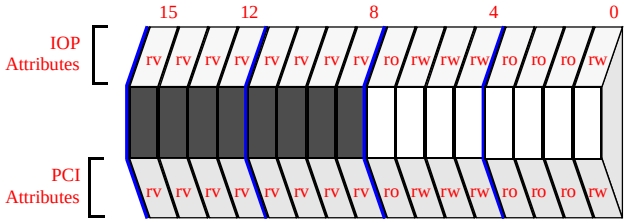
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
07:00	E0H	MSI_Next_ Pointer - This field provides an offset into the function configuration space pointing to the next item in the function capability list which is the PCI-X extended capability header residing at E0H.

4.9.22 MSI Message Control Register - MSI_MCR

The Message Control Register provides system software control over MSI. After reset, MSI is disabled. System software is permitted to modify the Message Control register's read/write bits and fields while a device driver is not permitted to modify them.

Table 176. MSI Message Control Register - MSI_MCR



PCI Configuration Offset
D2H - D3H

Intel® Xscale™ Core Local Bus Address
FFFF E1D2H

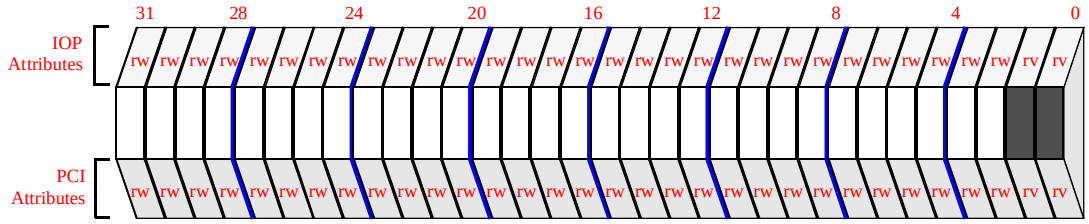
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:8	00H	Reserved
7	1_2	64-bit Address Support - This field is set to 1_2 indicating that the 80331 is capable of generating a 64-bit message address.
6:4	000_2	Multiple Message Enable - System software writes to this field to indicate the number of messages allocated to the 80331. While, the 80331 requests two messages, it is possible that system software only allocates one message. The device hardware is designed to handle both cases.
3:1	001_2	Multiple Message Capable - This field is set to 001_2 indicating that the 80331 can issue up to two unique interrupt messages.
0	0_2	MSI Enable - Setting this bit enables the 80331 MSI functionality and disables the use of the **P_INTC#** interrupt output (**P_INTA#** with **BRG_EN** and **ARB_EN** straps low) for 80331 interrupts.

4.9.23 MSI Message Address Register - MSI_MAR

The Message address register specifies the DWORD aligned address for the MSI memory write transaction. The value is set by system software during initialization.

Table 177. MSI Message Address Register - MSI_MAR



PCI Configuration Offset
D4H- D7H

Intel® Xscale™ Core Local Bus Address
FFFF E1D4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

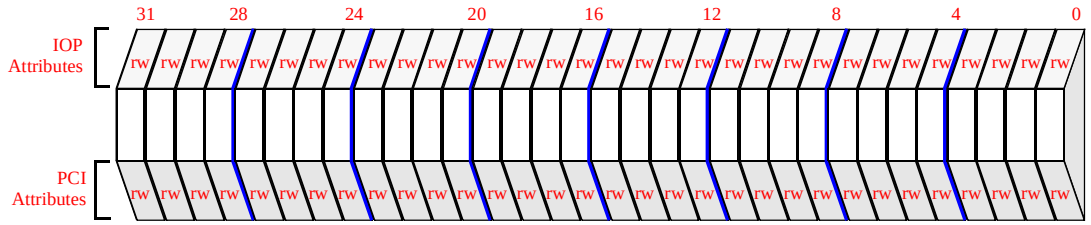
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:2	00000000H	Message Address - DWORD aligned Message Address. This value is set by system software.
1:0	00_2	Reserved.

4.9.24 MSI Message Upper Address Register - MSI_MUAR

The Message Upper Address register is set during system initialization when system software wishes to place the MSI address location above the 4G address boundary. When this register is set to a non-zero value, the 80331 generates a dual address cycle for the MSI write command and uses the contents of this register as the upper 32-bits of that address.

Table 178. MSI Message Upper Address Register - MSI_MUAR



PCI Configuration Offset
D8H - DBH

Intel® Xscale™ Core Local Bus Address
FFFF E1D8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

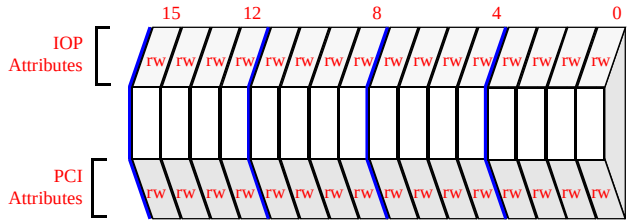
Bit	Default	Description
31:0	00000000H	Message Upper Address - Upper 32-bits of Message Address. Used for a Dual address cycle when programmed to a non-zero value. This value is set by system software.

4.9.25 MSI Message Data Register- MSI_MDR

The value in the Message Data Register contains the data used during an MSI write transaction. When two unique messages are enabled, one message is reserved for the Outbound Post Queue Interrupt and the other message represents all of the Outbound Doorbell and Outbound Message Interrupts. When only one message is enabled, all of these interrupts is represented by a single message. Interrupt handler software needs to read the 80331 Outbound Interrupt Status Register to determine the cause of the interrupt when more than one source is represented by a single message.

During an MSI write data phase, the value in the Message Data Register is driven on to **AD[15:0]** while **AD[31:16]** is driven to zero. **C/BE[3:0]#** are asserted during the data phase of the memory write transaction.

Table 179. MSI Message Data Register - MSI_MDR



PCI Configuration Offset
DCH - DDH

Intel® Xscale™ Core Local Bus Address
FFFF E1DCH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:00	0000H	Message Data - System software specifies a 16-bit value to be transferred during the data phase of an MSI write transaction. During initialization, system software can allocate two messages to the 80331 by writing "001" to the Multiple Message Enable field of the Message Control Register. In this case, the hardware shall modify bit 0 of the Message Data value before generating the MSI write transaction. The PCI Outbound interrupts is distributed across the two messages by the setting and clearing of bit 0 of the Message Data Register's contents before the data is used for the MSI write: 0 = Outbound Post Queue Interrupt 1 = Outbound Doorbell and Outbound Message Interrupts When software leaves the Multiple Message Enable field of the Message Control Register at its' default value of "000", the 80331 has only been allocated one message. Consequently, the value in the MSI data register is transmitted unmodified during an MSI write cycle.

4.10 Power/Default Status

The software is responsible for initializing the Circular Queue Size in the MU Configuration Register and all head and tail pointer registers before setting the Circular Queue Enable bit.

This Page Intentionally Left Blank

Intel® XScale™ Core Bus Interface Unit 5

This chapter describes the Intel® XScale™ core (ARM* architecture compliant) Bus Interface Unit (BIU) for the Intel® 80331 I/O processor (80331).

5.1 Overview

The BIU provides the interfaces between the Intel® XScale™ core and the Internal Bus and Memory Controller outbound ports. The BIU decodes transactions from the core and sends them to the appropriate outbound port. Transactions which address the DDR SDRAM are sent to the MCU port, and all other transactions are sent to the 80331 64-bit internal bus.

The high performance Internal Bus is used in the 80331 to interconnect major peripheral blocks as shown in Figure 24. A detailed block diagram of the BIU and Memory Controller Unit (MCU) is shown in Figure 24.

The BIU acts as a Master for all the read and write requests issued by the Intel® XScale™ core targeting the Internal Bus, and acts as a Slave for split completions of BIU split requests. The BIU has software accessible state information in the form of registers which reside in Peripheral Memory Mapped Register space.

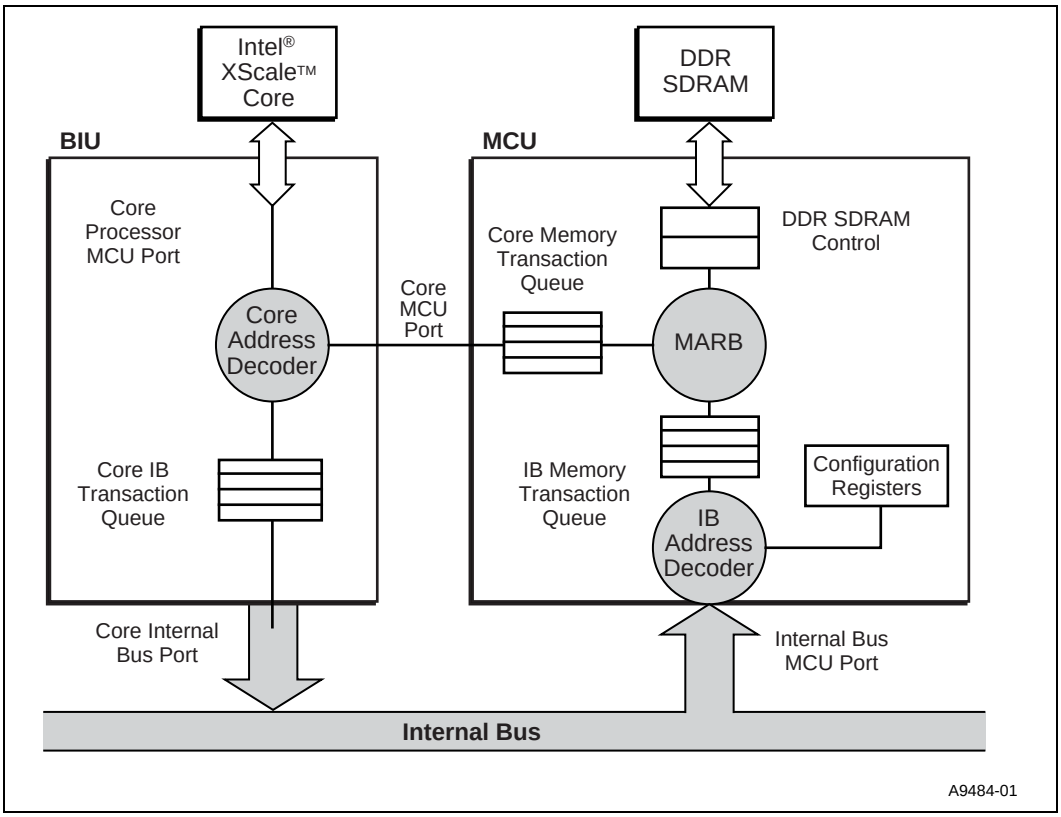
5.2 Theory of Operation

The BIU accepts Intel® XScale™ core bus accesses. It decodes core transactions and forwards them to the memory controller or internal bus outbound ports accordingly. The BIU translates the Intel® XScale™ core bus transactions to the protocol for the internal bus port, and the format for the Core Memory Transaction Queue of the MCU for the Core MCU port.

5.2.1 Functional Blocks

The memory controller and Intel® XScale™ core processor bus interface unit (BIU) logically comprises the blocks illustrated in Figure 24. The memory controller is a multiported unit, supporting an inbound path for both the BIU and Internal Bus (IB) to the DDR SDRAM.

Figure 24. Intel® 80331 I/O Processor BIU and MCU Architecture



5.2.1.1 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 any transactions received from the BIU via the Core MCU Port are decoded to determine when they address the DDR SDRAM Memory Space. When a transaction addresses the DDR SDRAM Memory Space, it is sent to the Core MCU port of the MCU which is a direct connection to the Core Memory Transaction Queue. When the transaction does not address that space, it is sent to the Core Internal Bus Transaction Queue to be issued on the Internal Bus. Core processor transactions addressing the MCU MMR Space, are directed to the Internal Bus, and subsequently claimed by the Internal Bus Port interface of the MCU.

5.2.1.2 Transaction Queues

The BIU has one transaction queue for transactions directed to the Internal Bus. The Core IB Transaction Queue (CIBTQ) has 8 read entries, 4 write entries, and 4 return data entries.The CIBTQ holds 4 outstanding read transactions up to 32-Bytes each, and 4 posted write transactions up to 16-Bytes each. The BIU acts as a Master for all the read and write requests issued by the Intel® XScale™ core targeting the Internal Bus, and acts as a Slave for split completions of BIU split requests. The transaction queue for MCU directed transactions exists in the MCU, and is defined in Section 8.3.1.3, "Memory Transaction Queues" on page 442. The BIU has software accessible state information in the form of memory mapped registers. All BIU registers reside in the Peripheral Memory Mapped Register space.

The Core Internal Bus port propagates read accesses to the Internal Bus and returns the read data to the Intel® XScale™ core. It completes write accesses for the Intel® XScale™ core. The BIU follows the ordering rules below for Internal Bus targeted transactions:

— Writes can pass Read requests
— Read requests does not pass read requests
— Writes does not pass Writes

The BIU may address any target on the Internal Bus. The BIU divides the core processor clock domain from the Internal Bus clock domain. The BIU has several address/data buffers:

- 4-Deep Write Transaction Queue
- 8-Deep Read Transaction Queue
- Write Data Buffer
- Read Data Buffer

In the Read transaction queue, each entry can be associated with an instruction or data cache line fill, a non-cacheable DWORD read. In the Write transaction queue, each entry can be associated with a write varying from 1 to 16 bytes of data.

The Write Data Buffer associated with each transaction queue entry can temporarily store write accesses that are destined for the Internal Bus. The BIU is responsible for forwarding all write accesses to the Internal Bus and ensuring their completion.

The Read Data Buffer associated with each transaction queue entry can temporarily store return data from the Internal Bus. The BIU is responsible for returning all read accesses to the Intel® XScale™ core.

A given transaction queue entry remains busy until the last byte of write data is transmitted or the last byte of read data is received across the Internal Bus.

5.3 Addressing

The BIU byte enables need to be adjusted depending on whether the 32-bit address resides in the even (A2='0') or odd (A2='1') DWORD.

See Table 180 for the Byte Enable encodings.

The BIU only reads and writes data on the Internal Bus as indicated by the Byte address generated by the Intel® XScale™ core. The BIU assumes that all accesses are to non-prefetchable memory. It does not promote Intel® XScale™ microarchitecture Byte or Intel® XScale™ microarchitecture Short accesses to DWORD accesses.

See Table 180 for the Byte Enable encodings.

Table 180. Contiguous Byte Enable Encodings[a]

Access Type[b]	DWORD Address[c]	Intel® XScale™ Core Address	Internal Bus							
		A[2:0]	BE7	BE6	BE5	BE4	BE3	BE2	BE1	BE0
Word	Even	0	1	1	1	1	0	0	0	0
	Odd	4	0	0	0	0	1	1	1	1
Short	Even	0	1	1	1	1	1	1	0	0
		2	1	1	1	1	0	0	1	1
	Odd	4	1	1	0	0	1	1	1	1
		6	0	0	1	1	1	1	1	1
Byte	Even	0	1	1	1	1	1	1	1	0
		1	1	1	1	1	1	1	0	1
		2	1	1	1	1	1	0	1	1
		3	1	1	1	1	0	1	1	1
	Odd	4	1	1	1	0	1	1	1	1
		5	1	1	0	1	1	1	1	1
		6	1	0	1	1	1	1	1	1
		7	0	1	1	1	1	1	1	1

a. Table assumes that write coalescing is turned off. When write coalescing is enabled, many other non-contiguous byte enable patterns are possible including all disabled.
b. For the purposes of this column, definitions are: Word equals 32-bits, Short = 16-bits, Byte = 8-bits.
c. For the purposes of this column, Even and Odd is based on Address bit A2.

5.3.1 Bus Width

The BIU supports a bus width of 32-bit for the core interface, 64-bits for the Internal Bus data bus and a width of 128 bits for the Core MCU port to the Core Memory Transaction Queue in the MCU.

5.4 Internal Bus Commands

For data read accesses, the BIU uses the Memory Read Block command for Intel® XScale™ microarchitecture cacheline (32 bytes) fills, and Memory Read DWORD command for read accesses include Byte, Short, and Word accesses (1-, 2-, or 4-byte). The BIU supports Critical Word First (CWF) address ordering for Intel® XScale™ microarchitecture cacheline (32 bytes) fills. For Intel® XScale™ core transactions targeting the PCI bus through the Address Translation Unit, refer to Chapter 3, "Address Translation Unit" for command translation.

For all write accesses including cache line flushes, the BIU uses the Memory Write command.

Note: The Intel® XScale™ core promotes all Intel® XScale™ microarchitecture Byte and Short read accesses to Intel® XScale™ microarchitecture Word read accesses when the data cache is enabled.

Table 181 contains a summary of the Internal Bus commands used by the BIU.

Table 181. Internal Bus Command Summary

Internal Bus Command	Intel® XScale™ Microarchitecture Word Size	Usage
Memory Read DWORD	Intel® XScale™ microarchitecture Byte, Short, or Word	Read 1, 2, 4 bytes
Memory Read Block	Intel® XScale™ microarchitecture core cacheline fills	Read 32 bytes (CWF)
Split Completion (Target Only)	All	Read data may be returned to the BIU using a Split completion
Memory Write	All	For all writes

5.5 Features

The BIU supports the following additional features.

5.5.1 Multi-Transaction Timer

The BIU has an associated Multi-Transaction Timer (MTT1) in the Internal Bus Arbiter. When programmed properly, the MTT allows for a guaranteed quantum of time for the BIU. See Chapter 12, "Intel® 80331 I/O Processor Arbitration Unit" for more details.

5.5.2 ATU Accesses

All read accesses from the BIU to the Address Translation Units (ATU) are processed as Split Transactions.

All writes from the BIU to the Address Translation Unit (ATU) are posted.

Instruction fetches from the BIU to the ATU are supported.

Atomic accesses from the BIU to the ATU are not supported.

5.6 Interrupts and Error Conditions

The BIU records error conditions that result from accesses initiated by the BIU on behalf of the Intel® XScale™ core on the Internal Bus or MCU port. The errors are recorded in the BIU Status Register (BIUSR).

5.6.1 Internal Bus Errors

There are two ways that the BIU can receive an error from transactions issued on the Internal Bus:

- IB Master-Abort: No target on receives a Target-Abort when accessing the Configuration Outbound Data Register of the ATU when the ATU configuration space is incorrectly configured as cacheable. In this case the BIU would issue a burst read request (cacheline fill), and the ATU responds with a Target-Abort.
- PCI Master-Abort: The ATU, as a PCI master on behalf of the BIU, receives a Master-Abort on the PCI bus and returns the Master-Abort to the BIU with a Split Completion Error Message (Message Class = 1H, Message Index = 00H).
- PCI Target-Abort: The ATU, as a PCI master on behalf of the BIU, receives a Target-Abort on the PCI bus and returns the Target-Abort to the BIU via a Split Completion Error Message (Message Class = 1H, Message Index = 01H).

There are also two ways that the BIU can detect an error from transactions issued on the Internal Bus:

- Split completion Target Abort: On a read completion, the remaining byte count is out of range from the original request.
- Split completion Target Abort: The lower 7-bits of the address from the original request are corrupted on the first Split Completion Transaction.

5.6.1.1 Internal Bus Master-Abort

When an Internal Bus access initiated by the BIU receives a Master-Abort, the BIU records the Master-Abort condition in the BIU Status Register (BIUSR), the address in BIU Error Address Register (BEAR) and signals a Data Abort to the Intel® XScale™ core.

When a Master Abort is received on the Internal Bus, the BIU signals an Imprecise Data Abort to the Intel® XScale™ core. This results in a Data Access Memory Abort exception by the Intel® XScale™ core; for more details on exceptions, please refer to Section 15.3, "The Intel® XScale™ Core Exceptions Architecture" on page 677.

5.6.1.2 Internal Bus Target-Abort

When an Internal Bus access initiated by the BIU receives a Target-Abort, the BIU records the Target-Abort condition in the BIUSR, the address in BEAR and signals a Data Abort to the Intel® XScale™ core.

5.6.1.3 PCI Master-Abort

The PCI Master Abort is recorded in the ATUISR. The ATU generates an interrupt to the Intel® XScale™ core.

For PCI read accesses, the ATU returns a Split Completion Error Message to the BIU. This causes the BIU to signal an imprecise data abort to the Intel® XScale™ core.

The software is able to diagnose that the error occurred via an outbound BIU transaction by examining the BIU Error Address Register (BEAR) and the BIU Control Register.

Because writes are posted, the ATU is responsible for signalling an interrupt to the Intel® XScale™ core when a PCI Master-Abort is received during a write access to the ATU.

5.6.1.4 PCI Target-Abort

The PCI Target Abort is recorded in the ATUISR. The ATU generates an interrupt to the Intel® XScale™ core. When the ATU detects a Target-Abort on the PCI bus for a read access, the ATU returns a Split Completion Error Message to the BIU indicating the Target-Abort.

For PCI read accesses, the ATU returns a Split Completion Error Message to the BIU. This causes the BIU to signal an imprecise data abort to the Intel® XScale™ core.

The software is able to diagnose that the error occurred via an outbound BIU transaction by examining the BIU Error Address Register (BEAR) and the BIU Control Register.

Because writes are posted, the ATU is responsible for signalling an interrupt to the Intel® XScale™ core when a PCI Target-Abort is received during a write access to the ATU.

5.6.1.5 Split Transaction Errors

As a requester on the internal bus, the BIU may detect the following error conditions due to a corrupted split completion:

- On a read completion, the remaining byte count is out of range from the original request.
- The lower 7-bits of the address from the original request are corrupted on the first Split Completion Transaction.

Upon detecting either of these errors, the BIU signals an Imprecise Data Abort to the Intel® XScale™ core and records the error in the BIUSR.

The BIU never acts as a Completer on internal bus for split transactions, and therefore no errors as a completer are possible.

5.6.2 Core MCU Errors

There are two ways that the BIU can receive an error from transactions issued to the MCU port:

- DDR SDRAM Multi-bit ECC Error
- Invalid Address Region Error

5.6.2.1 Multi-bit ECC Error

When a multi-bit ECC error is detected by the MCU, the error is also reported by the BIU. A multi-bit ECC error reported by the BIU can only occur on a read transaction by the Intel® XScale™ core.

For write accesses, the core processor transaction is posted to the MCU. Therefore to the BIU, the transaction appears as when it has completed before the transaction completes to the DDR SDRAM. In this case, the MCU is the only unit that notifies the core processor of any error condition.

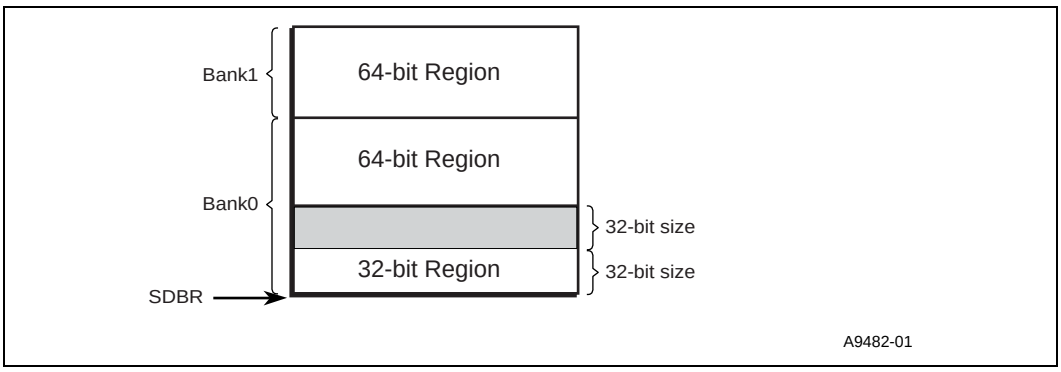
For read accesses, the BIU signals an Imprecise Data Abort to the Intel® XScale™ core. The MCU also records the error in the ELOGx registers and generates an interrupt to the Intel® XScale™ core.

5.6.2.2 Invalid Address Region Error

When a 32-bit region is defined with a 64-bit wide memory, an invalid address region exists. The region is illustrated in Figure 25 and is equal size to the 32-bit region as defined by Section 8.7.7, "SDRAM 32-bit Region Size Register - S32SR" on page 504, and at a contiguous address to the 32-bit region.

Intel® XScale™ core transactions which address the invalid address region are sent to the MCU via the Core MCU port. The MCU detects and reports this error in the Section 8.7.12, "Memory Controller Interrupt Status Register - MCISR" on page 509.

Figure 25. DDR SDRAM 64-bit Memory Address Map



5.7 Reset and Initialization Conditions

5.7.1 Reset

The BIU is reset:

- when **P_RST#** is asserted.
- when the Intel® XScale™ core is reset.

When reset, all Core IB transaction buffers are marked as invalid, and BIU registers return to the reset values.

5.7.2 Initialization

For proper operation of 80331, the configuration of the MCU must be complete prior to enabling the MCU port of the BIU. Once the MCU is fully configured (specifically the SDCR[1:0], SDIR, SDBR, SBR[1:0], and S32SR), the BIU control register can be programmed to enable the MCU port of the BIU.

5.8 Register Definitions

There are two Registers for the BIU. These registers are mapped into Intel® XScale™ core memory space.

Table 182 lists the PMMR registers for the BIU.

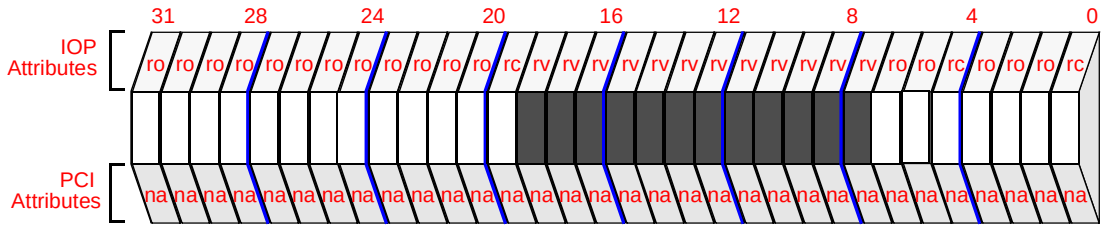
Table 182. Bus Interface Unit Register Tables

Section, Register Name - Acronym (Page)
Section 5.8.1, "BIU Status Register - BIUSR" on page 342
Section 5.8.2, "BIU Error Address Register - BEAR" on page 344
Section 5.8.3, "BIU Control Register - BIUCR" on page 345

5.8.1 BIU Status Register - BIUSR

The BIU Status Register (BIUSR) allows software to determine the cause of any BIU error. When an error is logged and this register is updated, an imprecise data abort is signaled to the Intel® XScale™ core.

Table 183. BIU Status Register - BIUSR (Sheet 1 of 2)

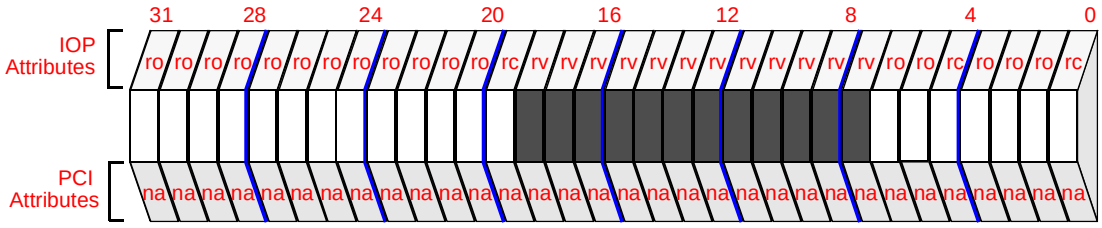


Intel® XScale™ Core Local Bus Address
FFFF E600H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0H	Message Class -- Identifies the Error Class for a Split Completion Error Message: Split Completion Messages are in one of the following classes. All other values are reserved. 0H Write Completion (Not Supported by the BIU) 1H Bridge Error 2H Completer Error All other values undefined
27:20	00H	Message Index -- Identifies the Specific Error with in an Error Class for a Split Completion Message: All values not explicitly identified here are reserved. For Class 1H: 00H Master Abort 01H Target Abort For Class 2H: 00H Byte Count Out of Range 8XH Device Specific Error
19	0_2	Corrupted Split Completion Detected - Indicates when set that a Split Completion received by the BIU was detected as corrupt.
18:07	000H	Reserved
06	0_2	Core Memory Read Transaction Pending -- Indicates when set that at least one pending core read transaction is pending in the MCU Core Memory Transaction Queue. Can be monitored along with use of the "BIU Control Register - BIUCR" on page 345 to force this queue to drain.
05	0_2	Core Memory Write Transaction Pending -- Indicates when set that at least one pending core write transaction is pending in the MCU Core Memory Transaction Queue. Can be monitored along with use of the "BIU Control Register - BIUCR" on page 345 to force this queue to drain.
04	0_2	Error Overflow -- Indicates that more than one error has occurred and has not been serviced by an interrupt routine. 0 = No Errors have occurred that aren't logged. 1 = Errors have occurred beyond the one logged.

Table 183. BIU Status Register - BIUSR (Sheet 2 of 2)



Intel® XScale™ Core Local Bus Address
FFFF E600H

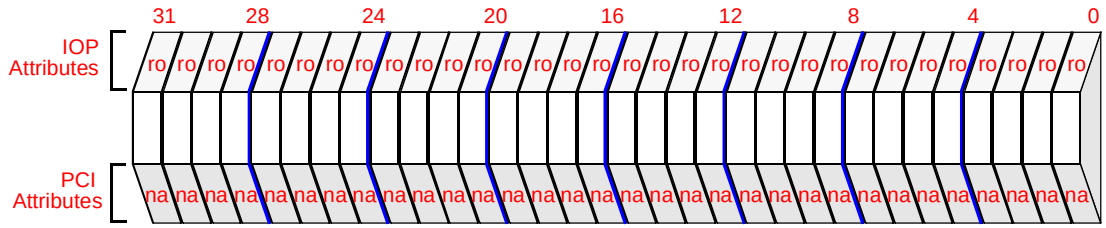
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
03:02	00_2	Error Type 00_2 - Target Abort 01_2 - Master Abort 10_2 - Split Completion Error Message 11_2 - Reserved
01	0_2	Direction -- The direction the data was traveling in the Internal Bus Transaction that resulted in the error condition: 0 = Read Error 1 = Write Error
00	0_2	Error Valid -- Indicates that the values of bits 31:20 (when applicable), 03:01 and the value in the BEAR (BIU Error Address Register) register represent a valid error that has been logged 0 = Valid Error has not been logged 1 = Valid Error has been logged

5.8.2 BIU Error Address Register - BEAR

This register is responsible for logging the addresses where the errors were detected on the Internal Bus. One error can be detected and logged. For error details, see Section 5.6, "Interrupts and Error Conditions" on page 337).

Table 184. BIU Error Address Register - BEAR



Intel® XScale™ Core Local Bus Address
FFFF E604H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

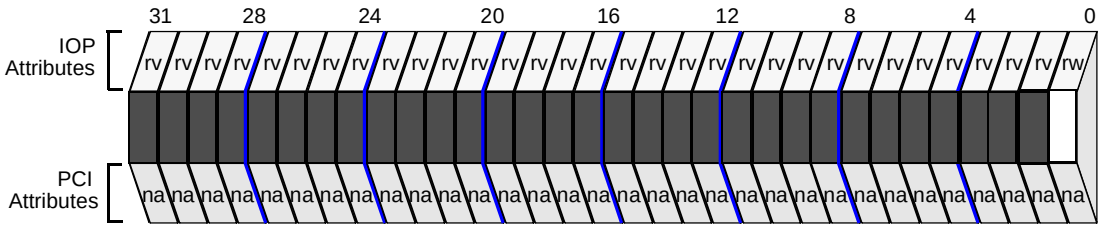
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0	Error Address: Stores the address that resulted in a Internal Bus Error.

5.8.3 BIU Control Register - BIUCR

The BIU Control Register (BIUCR) provides enable control of the Core MCU port interface of the BIU. Monitoring the Read and Write transactions queue depth status bits of the "BIU Status Register - BIUSR" on page 342 to determine when the pending Intel® XScale™ core processor transactions have completed and drained through the Core Memory Transaction Queue of the Memory Controller Unit.

Table 185. BIU Control Register - BIUCR



Intel® XScale™ Core Local Bus Address
FFFF E608H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:01	000 0000H	Reserved
00	0_2	MCU Port Enable: Controls the operation of the Core MCU port of the BIU. 0 = MCU port is disabled and forces all Intel® XScale™ core memory transactions to be issued out the Core IB port. 1 = MCU port is enabled, and Intel® XScale™ core memory transactions are decoded in the BIU to be issued to the IB port or MCU port. This bit must only be set after the MCU is programmed.

This Page Intentionally Left Blank

DMA Controller Unit 6

This chapter describes the integrated Direct Memory Access (DMA) Controller Unit. DMA Controller operation modes, setup, external interface, and implementation are detailed in this chapter.

6.1 Overview

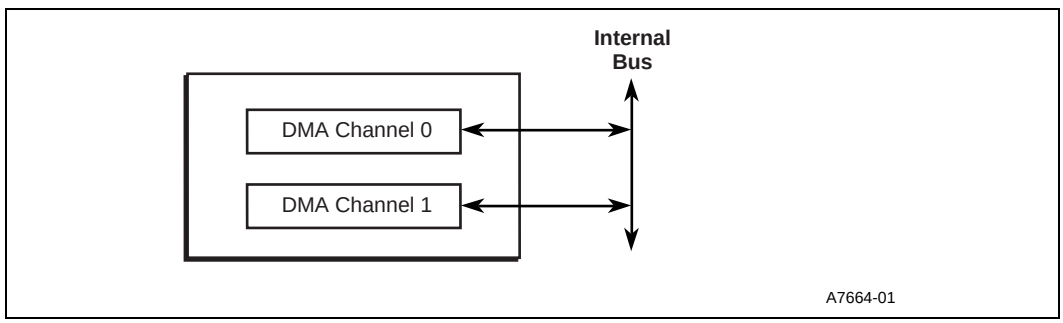
The DMA Controller provides low-latency, high-throughput data transfer capability. The DMA Controller optimizes block transfers of data between the Secondary PCI bus and the local processor memory. The DMA is an initiator on the Internal bus with burst capabilities providing a maximum throughput of 1064 Mbytes/sec. when the Secondary PCI bus is operating in 64-bit/133 MHz PCI-X mode. When addressing host memory space, the PCI-to-PCI Bridge forwards the transaction to the Primary PCI bus . In addition, the DMA provides a hardware assist to the iSCSI* application by calculating CRC-32C on the data during the block transfer.

The DMA Controller hardware is responsible for executing data transfers and for providing the programming interface. The DMA Controller features:

- Two Independent Channels
- Three 1-KByte queues in both Ch-0 and Ch-1
- Utilization of the Intel® 80331 I/O processor (80331) Memory Controller Interface
- 2^{32} addressing range on the Internal Bus interface
- 2^{64} addressing range on the PCI interface by using PCI Dual Address Cycle (DAC)
- Hardware support for unaligned data transfers for both the PCI bus and the Internal Bus
- Up to 1064 Mbytes/sec burst support for the PCI bus in the PCI-X mode
- Direct addressing to and from the PCI bus
- Memory to Memory DMA transfer mode
- Calculation of CRC-32C on the transferred data stream
- Fully programmable directly from the internal bus
- Support for automatic data chaining for gathering and scattering of data blocks
- 64-bit/266 MHz 80331 internal bus interface.

Figure 26 shows the connections of the DMA channels to the Internal Bus.

Figure 26. DMA Controller

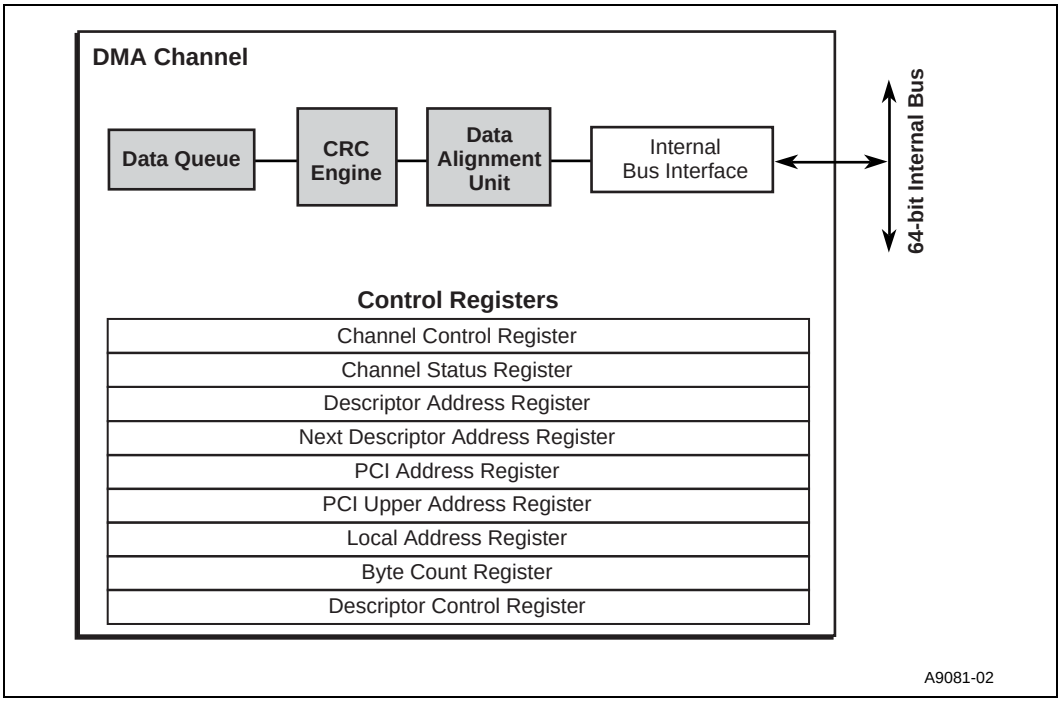


6.2 Theory of Operation

The DMA Controller provides two channels of high throughput PCI-to-Memory or Memory-to-Memory transfers. Channels 0 and 1 transfer blocks of data between the PCI bus and I/O processor local memory.

The two DMA channels operate identically. Each channel has an internal bus interface. Figure 27 shows the block diagram for one channel of the DMA Controller.

Figure 27. DMA Channel Block Diagram



Each DMA channel uses direct addressing for both the PCI bus and the internal bus. It supports data transfers to and from the full 64-bit address range of the PCI bus. This includes 64-bit addressing using the PCI DAC command. The channel provides a special register which contains the upper 32 address bits for the 64-bit address. The DMA channels do not support data transfers that cross a 32-bit address (4 GByte) boundary.

When Memory-to-Memory transfer mode is enabled, the DMA can be programmed to transfer data across the entire 4 Gbyte memory space on the Internal Bus. This includes but is not limited to transferring data to and from the 80331 Memory Controller (MCU), and from the Peripheral Bus Interface (PBI) to the MCU.

Both the PCI interface and the internal bus interface support large burst lengths up to 4 KBytes.

The channel programming interface is accessible from the internal bus through a memory-mapped register interface. Each channel is programmed independently and has its own set of registers. A normal DMA transfer is configured by writing the source address, destination address, number of bytes to transfer, and various control information into a chain descriptor in 80331 local memory. Chain descriptors are described in detail in Section 6.3.

Each channel supports chaining. Chain descriptors that describe one DMA transfer each can be linked together in 80331 local memory to form a linked list. Each chain descriptor contains all the necessary information for transferring a block of data in addition to a pointer to the next chain descriptor. The end of the chain is indicated when the pointer is zero.

Each channel contains a hardware data alignment unit. This unit enables data transfers from or to unaligned addresses in either the PCI address space or the I/O processor local address space. All combinations of unaligned data are supported with the data alignment unit.

Each channel includes a CRC Engine to calculate the CRC-32C algorithm on the data stream being transferred. The CRC engine also initiates the read of a CRC seed and the subsequent write back of the 32-bit result to a CRC Address specified in the descriptor. In addition, a Transfer Complete status bit is updated to the descriptor in memory.

6.3 DMA Transfer

A DMA transfer is a block move of data from one memory address space to another. DMA transfers are configured and initiated through a set of memory-mapped registers and one or more chain descriptors located in local memory. A DMA transfer is defined by the source address, destination address, number of bytes to transfer, and control values. These values are loaded into the chain descriptor before a DMA transfer begins. On the 80331 internal bus, the DMA controller attempts all transactions as 64-bit transfers.

Table 186. DMA Registers

Register	Abbreviation	Description
Channel Control Register x	CCRx	Channel Control Word
Channel Status Register x	CSRx	Channel Status Word
Descriptor Address Register x	DARx	Address of Current Chain Descriptor
Next Descriptor Address Register x	NDARx	Address of Next Chain Descriptor
PCI Address Register x	PADRx	Lower 32-bit PCI Address of Source/Destination
PCI Upper Address Register x	PUADRx	Upper 32-bit PCI Address of Source/Destination
Local Address Register x	LADRx	Local Address of Source/Destination
Byte Count Register x	BCRx	Number of Bytes to transfer
Descriptor Control Register x	DCRx	Chain Descriptor Control Word

6.3.1 Chain Descriptors

All DMA transfers are controlled by chain descriptors located in local memory. A chain descriptor contains necessary information to complete one data transfer. A single DMA transfer has only one chain descriptor in memory. Chain descriptors can be linked together to form more complex DMA operations. To perform a DMA transfer, one or more chain descriptors must first be written to local memory. Every descriptor requires seven contiguous words in local memory and is required to be aligned on an 8-word boundary. When CRC is not enabled, this word is ignored. It is still read, but does not have any effect. Each word in the chain descriptor is analogous to control register values. Bit definitions for words in the chain descriptor are the same as for the DMA control registers.

Warning: Chain descriptors can only be located in DDR SDRAM memory in order for the DMA to function properly. Location of the chain descriptors anywhere else (e.g., either on the Peripheral Bus or on PCI) is not supported and the results would be unpredictable.

Figure 28 shows the format of an individual chain descriptor.

- First word is 80331 memory address of the next chain descriptor. A value of "0" specifies the end of chain. This value is loaded into the Next Descriptor Address Register. Because chain descriptors are aligned on an 8-word boundary, the channel may ignore bits 04:00 of this address.
- Second word is lower 32-bit PCI source/destination address. This address is generated on the PCI bus. This value is loaded into the PCI Address Register.
- Third word is upper 32-bit PCI source/destination address, when needed. When non-zero, this address is used during Dual Address Cycles for driving 64-bit PCI addresses. When zero, Single Address Cycles are used for driving 32-bit PCI addresses. This value is loaded into the PCI Upper Address Register.
- Fourth word is internal bus source/destination address. This address is driven on the internal bus. This value is loaded into the Intel® XScale™ (ARM* architecture compliant) Local Address Register.
- Fifth word is Byte Count value. This value determines the number of bytes to transfer. This value is loaded into the Byte Count Register.
- Sixth word is Descriptor Control word. This word configures the DMA channel for one DMA transfer. It contains the PCI transaction type, which determines the direction of the data transfer. This value is loaded into the Descriptor Control Register.
- Seventh word is CRC Address word. This address is used to load a 32-bit value to seed the CRC calculation (when enabled). Also, this address is used as the destination for the write back of the CRC result.

There are no data alignment requirements for either the PCI address or the internal bus address.

Refer to Section 6.14 for additional descriptions about the DMA Controller registers

Figure 28. DMA Chain Descriptor

Normal Chain Descriptor in Memory Description

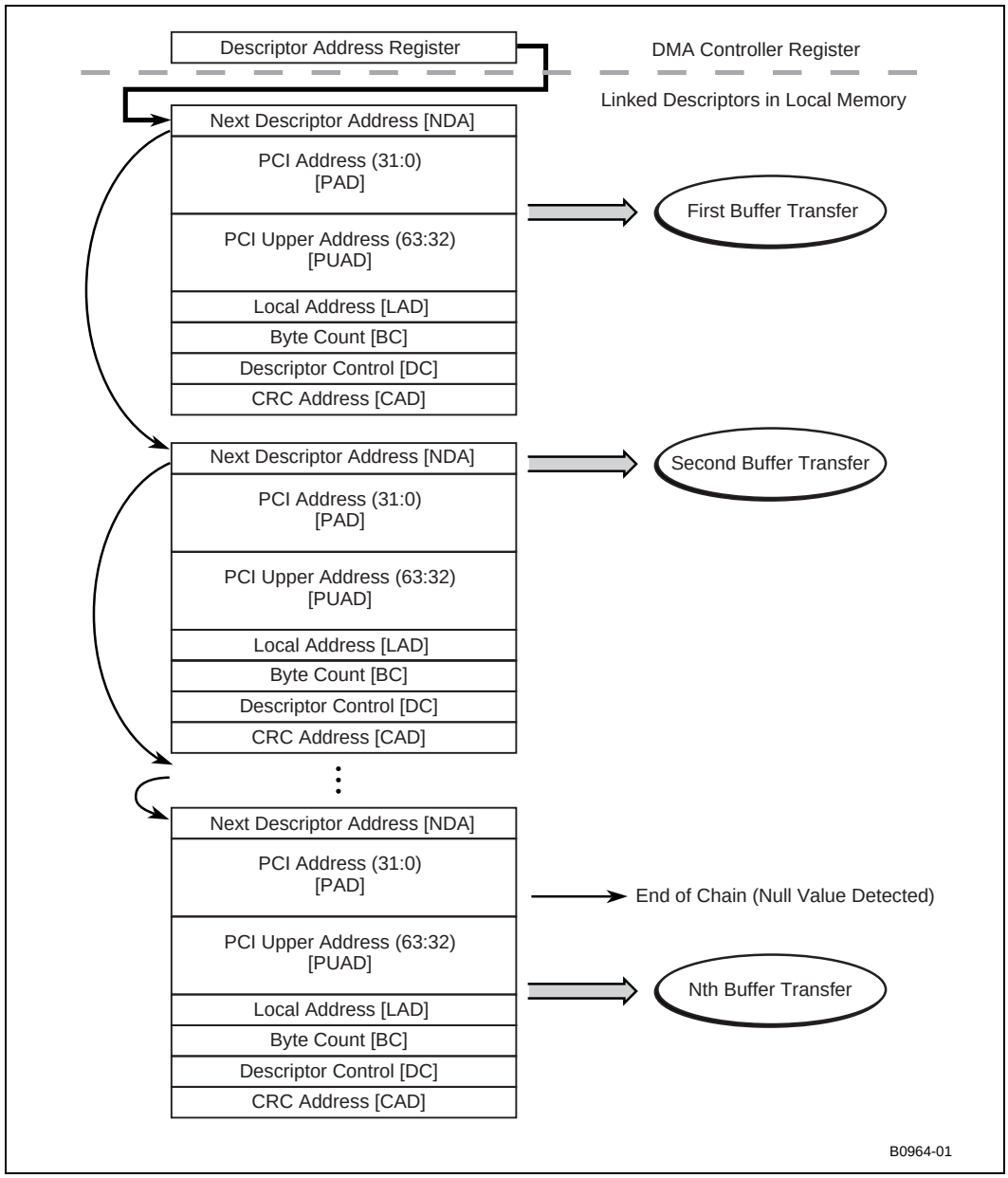
Next Descriptor Address [NDA]	Address of Next Chain Descriptor
PCI Address (31:0) [PAD]	Lower 32-bit PCI Source/Destination Address
PCI Upper Address (63:32) [PUAD]	Upper 32-bit PCI Source/Destination Address
Local Address [LAD]	Local Source/Destination Address
Byte Count [BC]	Number of Bytes to Transfer
Descriptor Control [DC]	Descriptor Control
CRC Address [CAD]	CRC Address

A9082-01

6.3.2 Chaining DMA Descriptors

A series of chain descriptors can be built in local memory to transfer data between the PCI bus and the internal bus. For example, the application can build multiple chain descriptors to transfer many blocks of data which have different source addresses within the local memory. When multiple chain descriptors are built in 80331 local memory, the application can link each of these chain descriptors using the Next Descriptor Address in the chain descriptor. This address logically links chain descriptors together.This allows the application to build a list of DMA transfers which may not require the Intel® XScale™ core until all DMA transfers are complete. Figure 29 shows a list of DMA transfers built in local memory and how they are linked.

Figure 29. DMA Chaining Operation



6.3.3 Initiating DMA Transfers

A DMA transfer is started by building one or more chain descriptors in 80331 local memory. Each chain descriptor takes the form shown in Figure 28. The chain descriptors are required to be aligned on an 8-word boundary in the 80331 local memory.

The following describes the steps for initiating a new DMA transfer:

1. In order to begin the processing of a new chain of DMA transfers, the channel must be inactive. This can be checked by software by reading the *Channel Active* bit in the Channel Status Register (CSR). When this bit is clear, the channel is inactive. When this bit is set, the channel is currently active with a DMA transfer.
2. The CSR must be cleared of all error conditions.
3. The software writes the address of the first chain descriptor to the Next Descriptor Address Register.
4. The software sets the *Channel Enable* bit in the Channel Control Register (CCR). Since this is the start of a new DMA transfer and not the resumption of a previous transfer, the *Chain Resume* bit in the CCR should be clear.
5. The channel starts the DMA transfer by reading the chain descriptor at the address contained in the Next Descriptor Address Register. The channel loads the chain descriptor values into the channel control registers and begins data transfer. The Descriptor Address Register now contains the address of the chain descriptor just read and the Next Descriptor Address Register now contains the Next Descriptor Address from the chain descriptor just read.

The last descriptor in the DMA chain list has zero in the next descriptor address field specifying the last chain descriptor. The NULL value notifies the DMA channel not to read additional chain descriptors from memory.

Once a DMA transfer is active, it may be temporarily suspended by clearing the *Channel Enable* bit in the CCR. Note that this does not abort the DMA transfer. The channel resumes the DMA transfer when the *Channel Enable* bit is set.

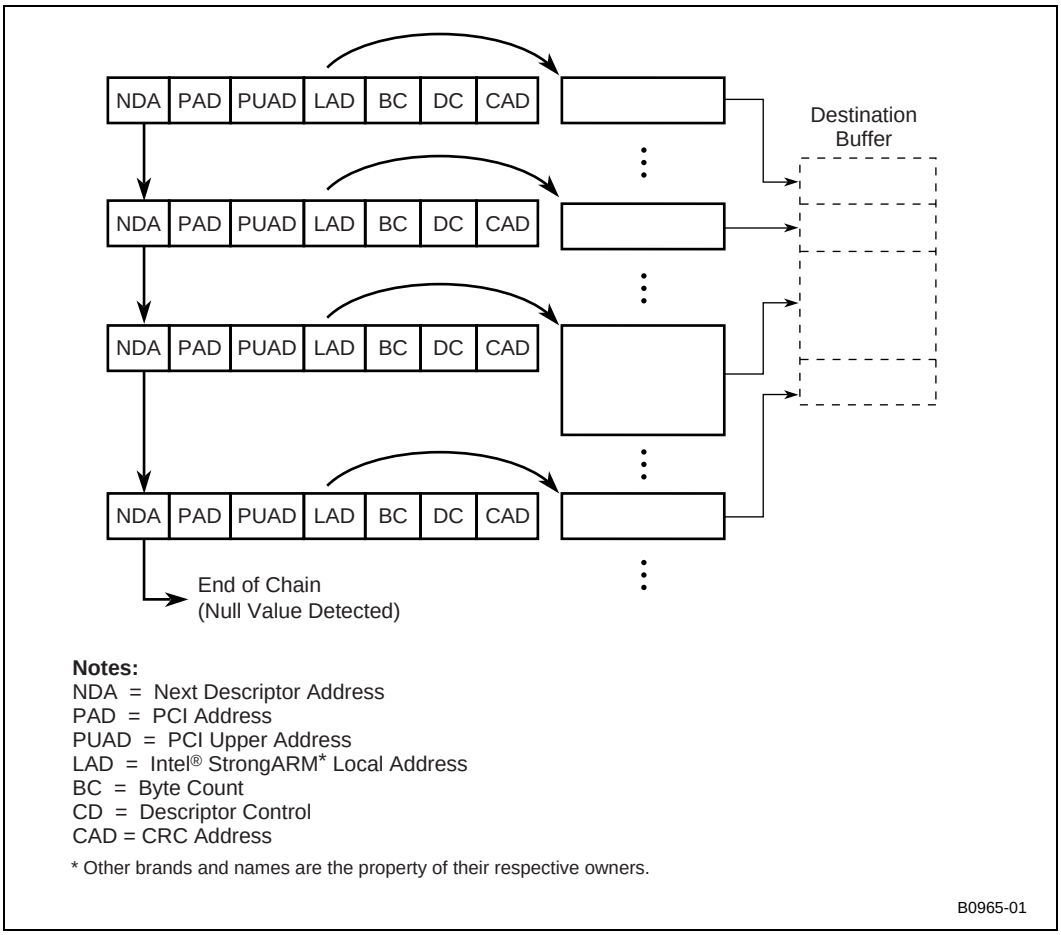
When descriptors are read from local memory, bus latency and memory speed affect chaining latency. Chaining latency is defined as the time required for the channel to access the next chain descriptor plus the time required to set up for the next DMA transfer.

See Section 6.9 for a state diagram of the channel programming model.

6.3.4 Scatter Gather DMA Transfers

The DMA Controller can be used to perform typical scatter gather data transfers. This consists of programming the chain descriptors to gather the data which may be located in non-contiguous blocks of memory. The chain descriptor specifies the destination location such that once all data has been transferred, the data is contiguous in memory. Figure 30 shows how the destination pointers can gather data.

Figure 30. Example of Gather Chaining



6.3.5 Synchronizing a Program to Chained Transfers

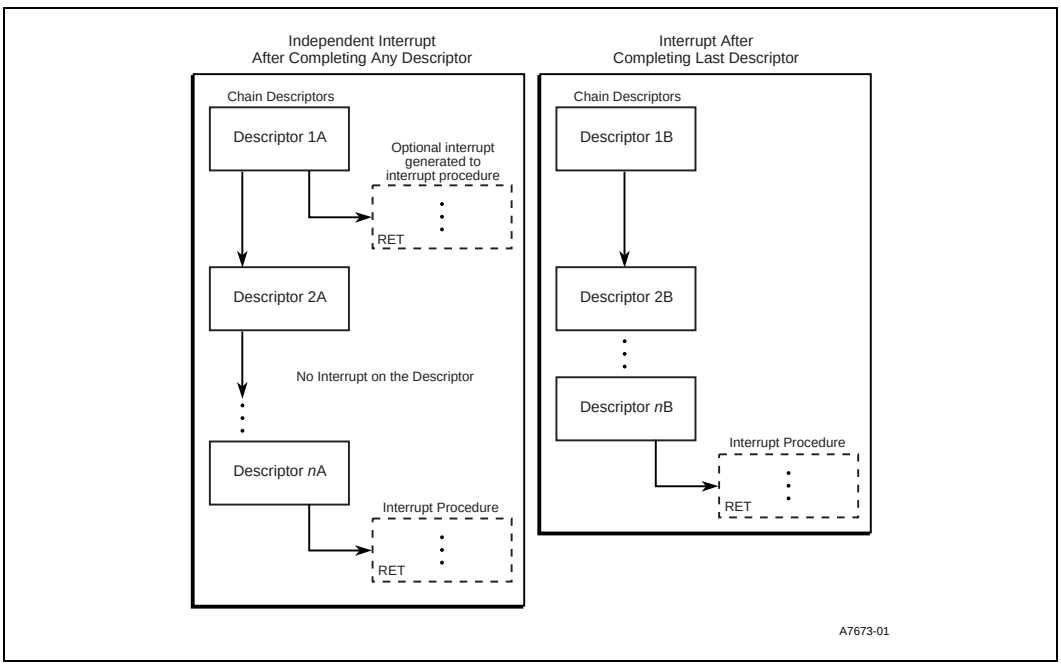
Chained DMA transfers can be synchronized to a program executing on the Intel® XScale™ Core through the use of processor interrupts. The channel generates an interrupt to the Intel® XScale™ Core under certain conditions. They are:

1. [Interrupt and Continue] The channel completes the data transfer for a chain descriptor and the Next Descriptor Address Register is non-zero. When the *Interrupt Enable* bit within the Descriptor Control Register is set, an interrupt is generated to the Intel® XScale™ Core. This interrupt is for synchronization purposes only. The channel sets the *End Of Transfer Interrupt* flag in the Channel Status Register. Since it is not the last chain descriptor in the list, the DMA channel starts to process the next chain descriptor without requiring any processor interaction.
2. [End of Chain] The DMA channel completes the data transfer for a DMA chain descriptor and the Next Descriptor Address Register is zero, and the chain resume bit is clear, specifying the end of the chain. When the *Interrupt Enable* bit within the Descriptor Control Register is set, an interrupt is generated to the Intel® XScale™ core. The channel sets the *End Of Chain Interrupt* flag in the Channel Status Register.
3. [Error] An error condition occurs (refer to Section 6.12 for DMA error conditions) during a DMA transfer. The DMA channel halts operation on the current chain descriptor and not proceed to the next chain descriptor.

Each chain descriptor can independently set the *Interrupt Enable* bit in the Descriptor Control word. This bit enables an independent channel interrupt upon completion of the data transfer for the chain descriptor. This bit can be set or clear within each chain descriptor. Control of interrupt generation within each descriptor aids in synchronization of the executing software with DMA transfers.

Figure 31 shows two examples of program synchronization. The left shows program synchronization based on individual chain descriptors. Descriptor 1A generated an interrupt to the processor, while descriptor 2A did not because the *Interrupt Enable* bit was clear. The last descriptor *n*A, generated an interrupt to signify the end of the chain has been reached. The right shows an example where the interrupt was generated only on the last descriptor signifying the end of chain.

Figure 31. Synchronizing to Chained Transfers



6.3.6 Appending to The End of a Chain

Once a channel has started processing a chain of DMA descriptors, the application software may need to append a chain descriptor to the current chain without interrupting the transfer in progress. This action is controlled by the Channel Control Register *Chain Resume* bit.

The channel reads the subsequent chain descriptor each time the channel completes the current chain descriptor and the Next Descriptor Address Register is non-zero. The Next Descriptor Address Register always contains the address of the next chain descriptor to be read and the Descriptor Address Register always contains the address of the current chain descriptor.

The procedure for appending chains requires the software to find the last chain descriptor in the current chain and change the Next Descriptor Address in that descriptor to the address of the new chain to be appended. The software then sets the *Chain Resume* bit in the Channel Control Register for the channel. It does not matter when the channel is active or not.

The channel examines the *Chain Resume* bit of the CCR when the channel is idle or upon completion of a chain of DMA transfers. When this bit is set, the channel re-reads the Next Descriptor Address of the current chain descriptor and load it into the Next Descriptor Address Register. The address of the current chain descriptor is contained in the Descriptor Address Register. The channel clears the *Chain Resume* bit and then examine the Next Descriptor Address Register. When the Next Descriptor Address Register is not zero, the channel reads the chain descriptor using this new address and begin a new DMA transfer. When the Next Descriptor Address Register is zero, the channel remains or return to idle and set the end of chain and generates an interrupt when enabled.

There are three cases to consider:

1. The channel completes a DMA transfer and it is not the last descriptor in the chain. In this case, the channel clears the *Chain Resume* bit and reads the next chain descriptor. The appended descriptor is read when the channel reaches the end of the original chain.
2. The channel completes a DMA transfer and it is the last descriptor in the chain. In this case, the channel examines the state of the *Chain Resume* bit. When the bit is set, the channel re-reads the current descriptor to get the address of the appended chain descriptor. When the bit is clear, the channel returns to idle.
3. The channel is idle. In this case, the channel examines the state of the *Chain Resume* bit when the CCR is written. When the bit is set, the channel re-reads the last descriptor from the most-recent chain to get the appended chain descriptor.

6.4 Data Transfers

The 80331 DMA controller is optimized to perform data transfers between the PCI bus and local memory, and between the local memory and the local memory. These transfers are summarized in the following sections.

6.4.1 PCI-to-Local Memory Transfers

PCI-to-Local memory transfers perform read cycles on the PCI bus and place the data into the DMA channel queues. Once data is placed into the queue, the internal bus interface of the DMA channel requests the internal bus and drain the queue by writing the data to the local memory.

DMA Internal Bus transactions that are destined for the external PCI Bus are automatically forwarded through the Outbound ATU to the PCI bus. In this way, the PCI direct addressing programming model defined for the DMA descriptors is preserved.

The DMA controller attempts to perform a memory write on the internal bus once the DMA queue has 1K Bytes of data or the Internal Bus transaction has disconnected. The byte count is set to the full byte count of the DMA and the byte count modified bit is set to 0 for memory write transactions on the internal bus. The write transaction continues on the internal bus until the byte count is satisfied, latency timer expires and **GNT#** is deasserted, or the target initiates a disconnect at an ADB. When the transaction is not complete the DMA restarts a memory write using the modified byte count, the starting address, and the same sequence ID to complete the write sequence.

Note: The Maximum Read Burst size of the DMA on the PCI bus can be throttled in the ATU by bits 3:2 of the PCIXCMD register (see Section 3.10.60, "PCI-X Command Register - PX_CMD" on page 278). The maximum burst length supported by the DMA is 1 Kbytes.

6.4.2 Local Memory to PCI Transfers

Local memory to PCI transfers perform read cycles on the internal bus and place the data into the DMA channel queues. The DMA queue size is 1 Kbyte, thus when the byte count of the descriptor is greater than 1 Kbyte then the DMA is completed in bursts with maximum burst size of 1 Kbyte. Otherwise, the full byte count is used for the byte count. Once data is placed into the queue, the DMA channel requests the internal bus and then drain the queue by writing the data to the PCI bus through the ATU.

DMA Internal Bus transactions that are destined for the external PCI Bus are automatically forwarded through the Outbound ATU to the PCI bus. In this way, the PCI direct addressing programming model defined for the DMA descriptors is preserved.

When operating in PCI mode depending on the Internal Bus transaction's byte count and starting address, the ATU selects between two PCI write command types: Memory Write and Memory Write and Invalidate.

In PCI-X mode, the ATU uses the Memory Write Block command, exclusively.

6.4.3 Local Memory to Local Memory DMA

When bit 6 of the DCR is set, the DMA functions as a Local Memory to Local Memory DMA engine. The transfer takes place from the range of 32 bit addresses defined by the PADR (PCI Address Register -- Lower 32 bits) and the byte count to the range of addresses defined by the LADR (Local Address Register) and the byte count.

Local Memory to Local Memory transfers perform read cycles on the internal bus using PADR based addresses and place the data into the DMA channel queues. The maximum byte count for a DMA transaction is 1 Kbyte, thus when the byte count of the descriptor is greater than 1 Kbyte then the byte count is set to 1 Kbyte. Otherwise, the full byte count is used for the byte count. Once data is placed into the queue, the DMA channel requests the internal bus and then drain the queue by writing the data to the Internal Bus using LADR based addresses.

Local Memory to Local Memory transfers utilizes the Memory Read Block (MRB) and Memory Write Block (MWB) commands on the Internal Bus.

Note: Since the Internal Bus only supports 32 bit addressing, the PCI Upper Address Register (PUADR) is ignored. Because data transfers only occur in one direction in this mode, the command field (bits 3 to 0) of the DCR are also ignored.

6.4.4 Exclusive Access

The DMA Controller does not support exclusive access through the PCI LOCK# signal.

6.5 Data Queues

DMA Ch-0 and Ch-1 each contain three 1 Kbyte, data buffers. These queues temporarily hold data to increase performance of data transfers in both directions.

6.6 Data Alignment

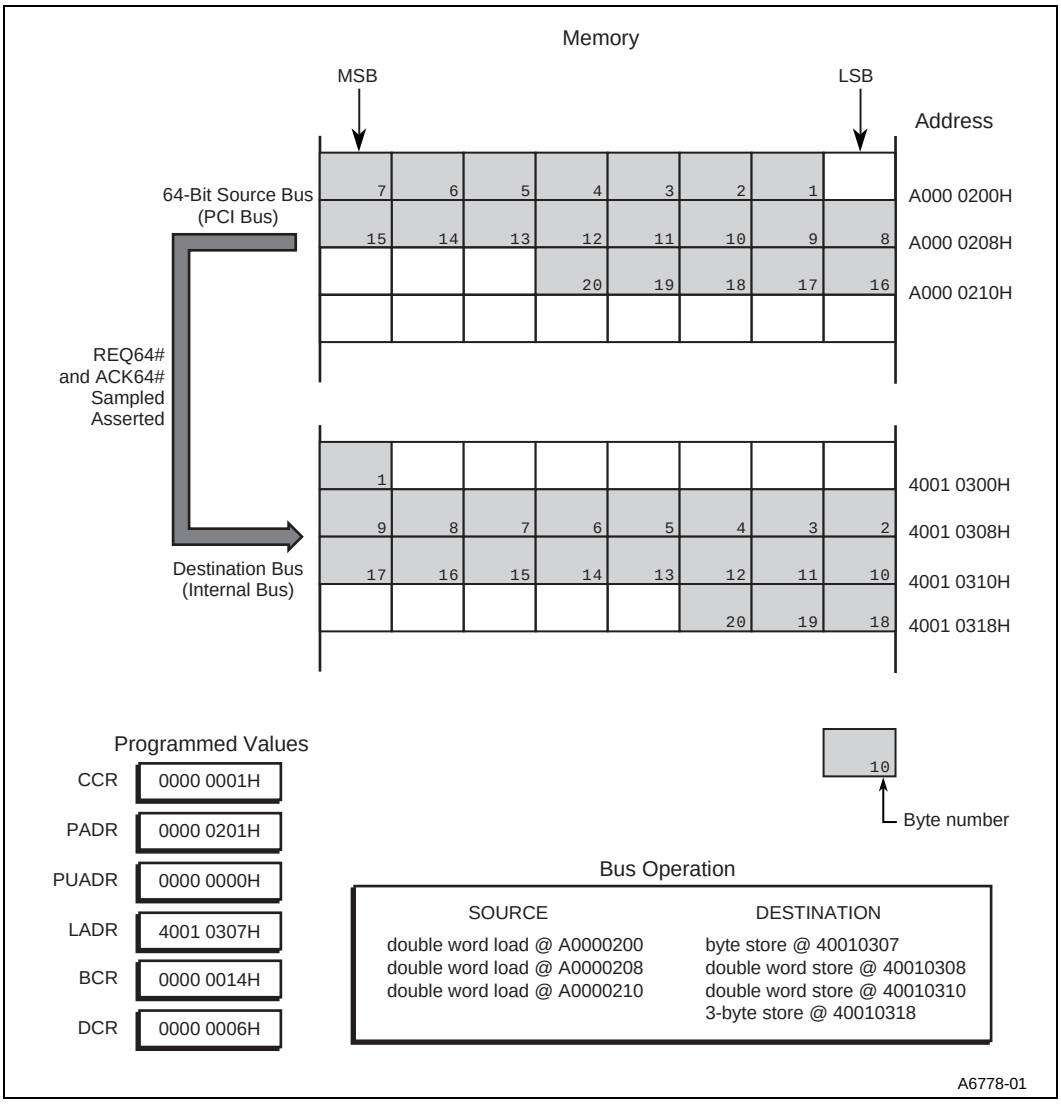
Each channel contains a hardware data alignment unit to support unaligned data transfers between the source and destination busses. The data alignment unit optimizes data transfers to and from 32 and 64-bit memory. The channel reformats data words for the correct bus data width.

Aligned data transfers involve data accesses that fall on natural boundaries. For example; double words are aligned on 8-byte boundaries and words are aligned on 4-byte boundaries. DMA transfers can occur with both the source and destination addresses unaligned.

6.6.1 64-bit Unaligned Data Transfers

Figure 32 illustrates a DMA transfer between unaligned 64-bit source and destination addresses.

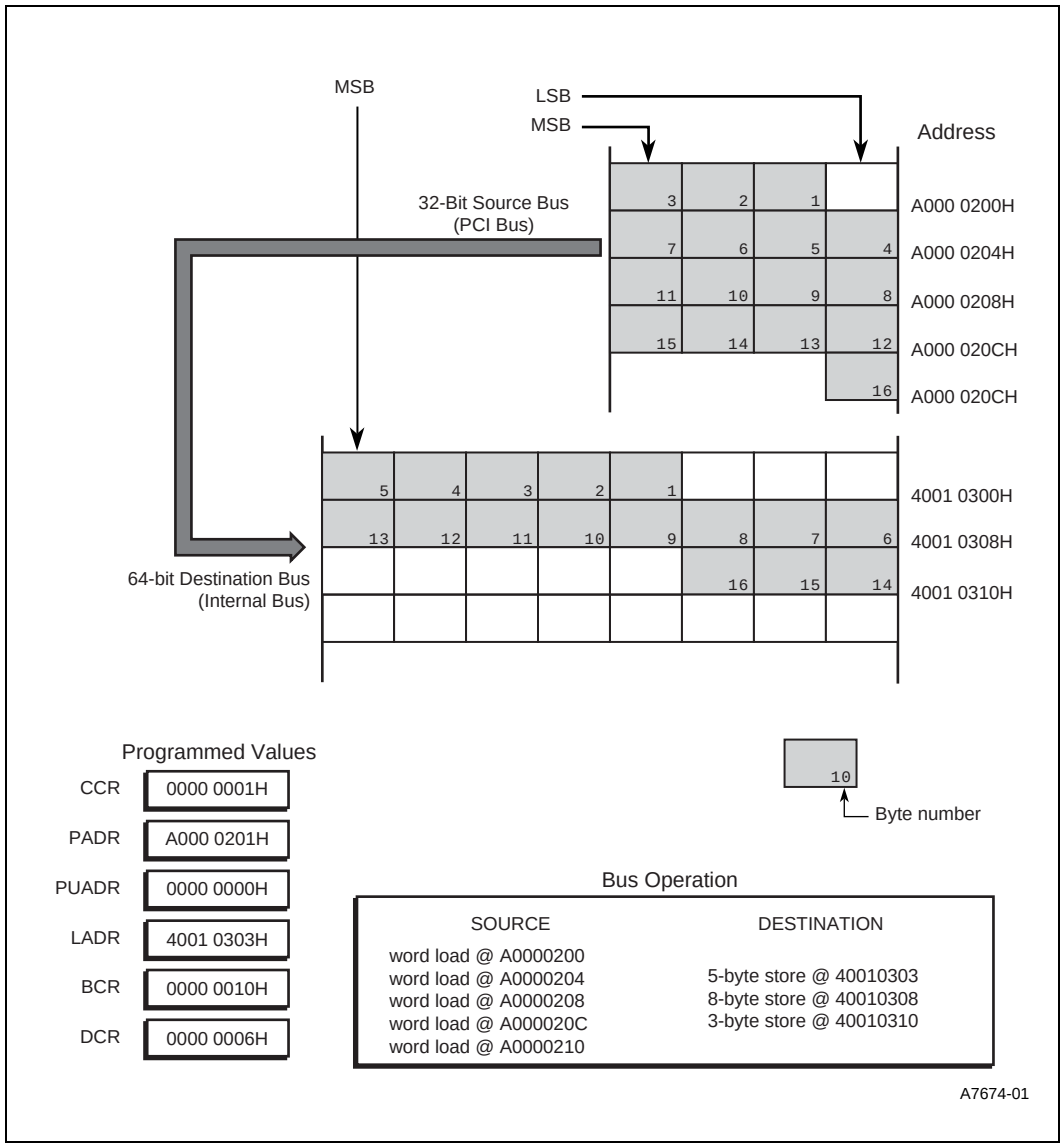
Figure 32. Optimization of an Unaligned DMA



6.6.2 64/32-bit Unaligned Data Transfers

Figure 33 illustrates a DMA transfer between an unaligned 32-bit source address and an unaligned 64-bit destination address.

Figure 33. Optimization of an Unaligned DMA



6.7 CRC Generation

When enabled, the DMA generates a 32-bit CRC based on the programmed data stream and a 32-bit seed. The CRC engine uses the CRC-32C algorithm required by the iSCSI* Specification.

6.7.1 CRC Mode Configuration and Operation

In addition to the normal DMA Descriptor configuration, the following additional steps are required to configure the CRC engine:

1. When writing out the descriptor, bit 8 of the DC is set to enable CRC generation.
2. Word 7 of the descriptor is written with the CRC Address.

When a descriptor is configured to generate CRC, the DMA performs the following steps in addition to the Data Transfer:

1. Prior to the start of the Data Transfer, the DMA loads the 32-bit seed value from the CRC Address.
2. The DMA accumulates the 32-bit CRC as the Data Transfer is occurring. Specifically, the CRC Value is recirculated through the CRC-32C algorithm using the previous CRC Value and the most recent data transferring through the DMA.
   a. CRC Value (n) = CRC-32C (CRC Value (n-1), Data(n)).
3. Following the completion of the Data Transfer, the DMA writes the CRC Value to the CRC Address.
4. Finally, the DMA writes the Descriptor Control Register back to the descriptor with the Transfer Complete bit set.

Note: DCR bit 7 may be used to disable the Data Transfer during a CRC calculation. This provides the ability to generate CRC on a source data block without writing a destination data block.

Note: DCR bit 9 is used to enable chaining of descriptors to calculate CRC across fragmented data source. Bit is clear for first descriptor of chain, then set for subsequent descriptors in the chain for the complete data.

6.7.2 CRC-32C Algorithm

The CRC Engine generates a 32-bit CRC of the programmed data stream using the CRC-32C generator polynomial required by the iSCSI* Specification, (11EDC6F41).

Figure 34. CRC-32C Generator Polynomial

$$1 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{18} + x^{19} + x^{20} + x^{22} + x^{23} + x^{25} + x^{26} + x^{27} + x^{28} + x^{32}$$

Note: The seed value located at the CRC Address (CAD) must be all 1s (FFFF FFFFh) to generate an iSCSI* compatible CRC-32C.

6.8 Channel Priority

The Intel® XScale™ core internal bus arbitration logic determines which internal bus master has access to the internal bus. Each DMA channel has an independent bus Request/Grant signal pair to the internal bus arbitration. Chapter 12, "Intel® 80331 I/O Processor Arbitration Unit" further describes the priority scheme between all the bus masters on the internal bus. Also described is the priority mechanism used between the two DMA channels.

In addition, the internal bus arbitration unit has a Multi-Transaction timer (Section 12.4.3, "Multi-Transaction Timer Register 2 - MTTR2" on page 626) that affects the throughput of a DMA channel. The default value for MTT2 of 152 clocks was chosen to ensure that once an internal bus agent (in this case a DMA channel) is granted the internal bus that it is guaranteed an opportunity to burst data into DDR SDRAM memory. However, when the bus is busy the DMA channel loses grant before the burst is completed. This means that the DMA channel is able to complete only one burst for each arbitration cycle.

Alternatively, the user may wish to increase the value of MTT2 to guarantee that two or more bursts is able to complete within an arbitration cycle.

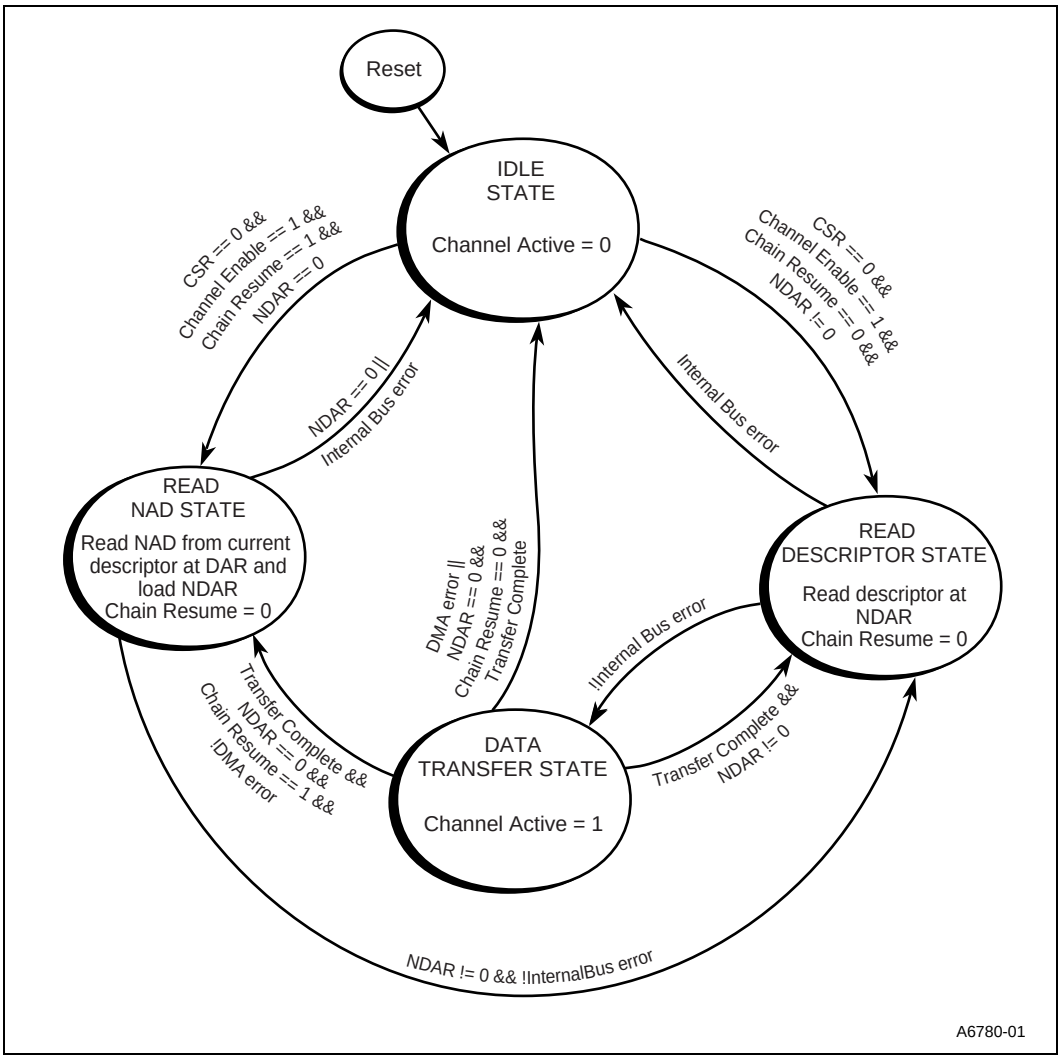
For example, assuming 1 Kbyte bursts and a 64-bit memory subsystem, an MTT2 setting of 192 clocks would be sufficient to support two 1 Kbyte bursts for a given DMA channel in a single arbitration cycle.

Warning: Increasing the MTT2 value may also increase the latency to memory for the Intel® XScale™ core on the average. Before changing the MTT2 value, it's imperative that the overall impact to the performance of the application is considered.

6.9 Programming Model State Diagram

The channel programming model diagram is shown in Figure 35. Error condition states are not shown.

Figure 35. DMA Programming Model State Diagram



6.10 DMA Channel Programming Examples

The software for the DMA channels falls into the following categories:

- Channel initialization
- Start DMA transfer
- Suspend channel

Examples for each of the software is shown in the following sections as pseudo code flow.

6.10.1 Software DMA Controller Initialization

The DMA Controller is designed to have independent control of the interrupts, enables, and control. The initialization consists of virtually no overhead as shown in Figure 36.

Figure 36. Software Example for Channel Initialization

```
CCR0 = 0x0000 0000 ; Disable channel
Call setup_channel
```

6.10.2 Software Start DMA Transfer

The DMA channel control register provides independent control per channel. This provides the greatest flexibility to the applications programmer. The example shown in Figure 38 describes the pseudo code for starting a DMA transfer on channel 0.

Figure 37. Software Example for PCI-to-Local DMA Transfer

```
; Set up descriptor in local memory at address d
d.nad = 0          ; No chaining
d.pad = 0x0000F000 ; Source address of 0000F000H
d.puad = 0         ; DAC is not used
d.lad = 0xB0000000 ; Destination address B0000000H
d.bc = 64          ; byte count of 64
d.dc = 0x0000001E  ; PCI Memory Read Block command
                   ; Interrupt processor after transfer
; Check for inactive channel & no interrupts pending
if (CSR0 != 0) exit; If channel is not ready, exit
; Start transfer
NDAR0 = &d         ; Set up descriptor address
CCR0 = 0x00000001  ; Set Channel Enable bit to start
```

Figure 38. Software Example for Local Memory-to-Local Memory DMA Transfer

```
; Set up descriptor in local memory at address d
d.nad = 0          ; No chaining
d.pad = 0x0000F000 ; Source address of 0000F000H
d.puad = 0         ; DAC is not used
d.lad = 0xB0000000 ; Destination address B0000000H
d.bc = 64          ; byte count of 64
d.dc = 0x0000005E  ; Memory Read Block command, Memory-to-Memory DMA Enabled
                   ; Interrupt processor after transfer
; Check for inactive channel & no interrupts pending
if (CSR0 != 0) exit; If channel is not ready, exit
; Start transfer
NDAR0 = &d         ; Set up descriptor address
CCR0 = 0x00000001  ; Set Channel Enable bit to start
```

6.10.3 Software Suspend Channel

The channel may need to be suspended for various reasons. The channel provides the ability to suspend the state of the channel without losing the current status. The channel resumes DMA operation without requiring the software to save the channel configuration. The example shown in Figure 39 describes the pseudo code for suspending channel 0.

Figure 39. Software Example for Channel Suspend

```
CCR0 = 0x0000 0000 ; Suspend Channel 0

    Channel suspended.....


CCR0 = 0x0000 0001 ; Resume Channel 0
```

6.11 Interrupts

Each channel can generate an interrupt to the Intel® XScale™ core. The *Interrupt Enable* bit in the Descriptor Control Register (DCR) determines whether the channel generates an interrupt upon successful error-free completion of a DMA transfer. Error conditions described in Section 6.12 also generate an interrupt. Each channel has one interrupt output connected to the PCI and Peripheral Interrupt Controller described in Chapter 15, "Interrupt Controller Unit" summarizes the status flags and conditions when interrupts are generated in the Channel Status Register (CSRx).

Table 187. DMA Interrupt Summary

Interrupt Condition	Channel Status Register (CSR) Flags							Interrupt Generated?	
	Active	End of Transfer	End of Chain	PCI Master Abort	PCI Target Abort	Unknown Split Transaction	Internal Bus Error	Interrupt Enabled	Interrupt Disabled
Byte count == 0 && (NDARx != NULL || Resume == 1) (End of Transfer)	1	1	0	0	0	0	0	Y	N
Byte Count == 0 && NDARx == NULL (End of Chain) & resume == 0	0	0	1	0	0	0	0	Y	N
PCI Master-Abort	0	0	0	1	0	0	0	Y	Y
PCI Target-Abort	0	0	0	0	1	0	0	Y	Y
Unknown Split Transaction Error	0	0	0	0	0	1	0	Y	Y
Internal Bus Error	0	0	0	0	0	0	1	Y	Y

Note: End-of-Transfer and End-of-Chain flags is set only when interrupt is enabled. Otherwise, the above flags are always set to 0. End-of-Transfer Interrupt and End-of-Chain Interrupt can only be reported in the CSR when the DMA transfer completed without any reportable errors.The channel shall never report an End-of-Transfer interrupt or End-of-Chain interrupt along with any PCI error conditions. Multiple error conditions may occur and be reported together. Also, because the channel does not stop after reporting the End-of- Transfer Interrupt, internal bus errors may occur before the End-of-Transfer interrupt is acknowledged and cleared.

6.12 Error Conditions

There are four error conditions that may occur during a DMA transfer that are recorded by the channel. All error conditions are reported by setting the appropriate bit in the Channel Status Register (CSR).

The possible error conditions are:

- PCI Master-Abort
- PCI Target-Abort
- Unknown PCI-X Split Transaction Error
- Internal Bus Errors

6.12.1 PCI Errors

- When a PCI Master-Abort occurs during a PCI-to-Local DMA transfer, the channel sets bit 3 in the CSR. Refer to Section 3.7, "ATU Error Conditions" on page 174 for complete details on the ATU error response.
- When a PCI Target-Abort occurs during a PCI-to-Local DMA transfer, the channel sets bit 2 in the CSR. The ATU also records this PCI Bus error condition by setting the appropriate bit in its' status register. Refer to Section 3.7, "ATU Error Conditions" on page 174 for complete details on the ATU error response.
- When an Unknown PCI-X Split transaction error occurs during a PCI-to-Local DMA transfer, the channel sets bit 1 in the CSR. For PCI parity errors, data with incorrect parity is never transferred to the DMA. Refer to Section 3.7, "ATU Error Conditions" on page 174 for complete details on the ATU error response.

Note: Errors (including address/data parity errors) that occur on the PCI bus during Local-to-PCI or PCI-to-Local transfers is logged by the ATU. Refer to Chapter 3, "Address Translation Unit" for complete details.

6.12.2 Internal Bus Errors

- When a Master-Abort occurs during a read of the Chain Descriptor or Next Descriptor Address, the channel sets the Internal Bus Master-abort error flag which is bit 5 in the CSR. Then, the channel loads the registers (when possible) and stop.
- When a Master-Abort occurs during a data transfer, the channel sets the Internal Bus Master-abort error flag which is bit 5 in the CSR. Then, the channel stops operation.
- When a Target-Abort occurs, the DMA stops operation, but the error is recorded by the MCU.

When an error condition occurs, the actions taken are detailed below:

- The channel shall cease data transfers for the current chain descriptor and clear the *Channel Active* flag in the CSR.
- The channel invalidates any data held in the queue and not read any new chain descriptors.
- The channel sets the appropriate error flag in the Channel Status Register. For example; when a PCI Master-Abort occurred during a DMA transfer, the channel sets bit 3 in the CSR.
- The channel also signals an interrupt to the Intel® XScale™ core.
- The channel does not restart a DMA transfer after any error condition. It is the responsibility of the application software to configure the channel to complete any remaining transfers.

Note: Internal Bus errors (**Target-abort only**) that occur while a DMA channel is the master on the Internal Bus are recorded by the MCU and interrupt the core. For correct operation of the DMA channel, user software has to disable the channel before clearing the error condition. Further, the channel needs to be re-enabled by writing a 1 to the CCR channel enable before initiating a new operation.

6.13 Power-up/Default Status

Upon power-up, an external hardware reset, the DMA Registers is initialized to their default values.

6.14 Register Definitions

The DMA controller contains registers for controlling each channel. Each channel has nine memory-mapped control registers for independent operation. In register titles, x is 0 or 1 for channel 0 or 1, respectively.

There is read/write access only to the Channel Control Register, Channel Status Register, and the Next Descriptor Address Register. The remaining registers are read-only and are loaded with new values from the chain descriptor whenever the channel reads a chain descriptor from memory.

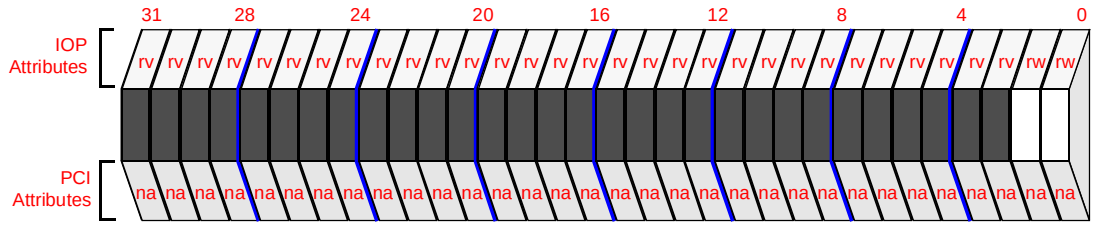
Table 188. DMA Controller Unit Registers

Section, Register Name, Acronym, Page
Section 6.14.1, "Channel Control Register x - CCRx" on page 371
Section 6.14.2, "Channel Status Register x - CSRx" on page 372
Section 6.14.4, "Next Descriptor Address Register x - NDARs" on page 374
Section 6.14.3, "Descriptor Address Register x - DARx" on page 373
Section 6.14.8, "Byte Count Register x - BCRx" on page 378
Section 6.14.5, "PCI Address Register x - PADRx" on page 375
Section 6.14.6, "PCI Upper Address Register x - PUADRx" on page 376
Section 6.14.7, "Local Address Register x - LADRx" on page 377
Section 6.14.9, "Descriptor Control Register x - DCRx" on page 379

6.14.1 Channel Control Register x - CCRx

The Channel Control Register (CCRx) specifies parameters that dictate the overall channel operating environment. The CCRx should be initialized prior to any other DMA register following a system reset. Figure 29 shows the register format for the CCRx. This register can be read or written while the DMA channel is active.

Table 189. Channel Control Register x - CCRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E400H
1	FFFF E440H

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

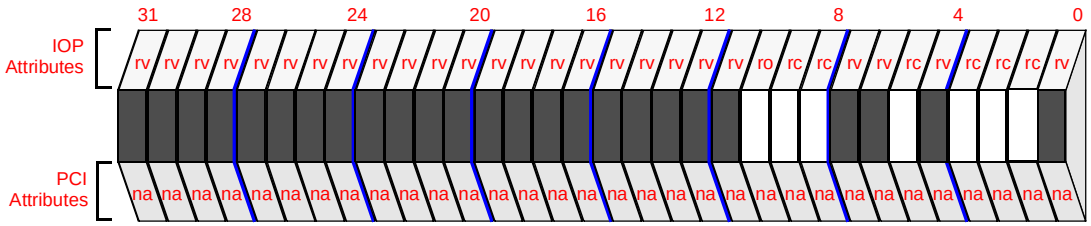
Bit	Default	Description
31:02	0000 0000H	Reserved
01	0_2	Chain Resume - when set, causes the channel to resume chaining by re-reading the current descriptor located at the address in the Descriptor Address Register when the channel is idle (Channel Active bit in the CSR is clear) or when the channel completes a DMA transfer. This bit is cleared by the hardware when either: • The channel completes a DMA transfer and the Next Descriptor Address Register is non-zero. In this case, the channel proceeds to the next descriptor in the chain. • The channel re-reads the chain descriptor located at the address in the Descriptor Address Register and loads the Next Descriptor Address of that descriptor into the Next Descriptor Address Register
00	0_2	Channel Enable - When set, the channel enables DMA transfers. When clear, the channel disables DMA transfers. Clearing this bit once the channel is active suspends the current DMA transfer at the earliest opportunity by halting all internal bus transactions. The PCI interface may continue with the current transfer until the data queue either fills or empties. The channel does not initiate any new DMA transfers when this bit is cleared. Data held in queues remains valid. Setting this bit after the channel is suspended causes the channel to resume the DMA transfer.

6.14.2 Channel Status Register x - CSRx

The Channel Status Register (CSRx) contains status flags that indicate the channel status. This register is typically read by software to examine the source of an interrupt. See Section 6.12 for a description of the error conditions that are reported in the CSRx. See Section 6.11 for a description of interrupts caused by the DMA channel.

When a DMA error occurs, application software must check the status of the Channel Active flag before processing the interrupt. It is possible that the channel may still be active completing outstanding PCI transactions.

Table 190. Channel Status Register x - CSRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E404H
1	FFFF E444H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

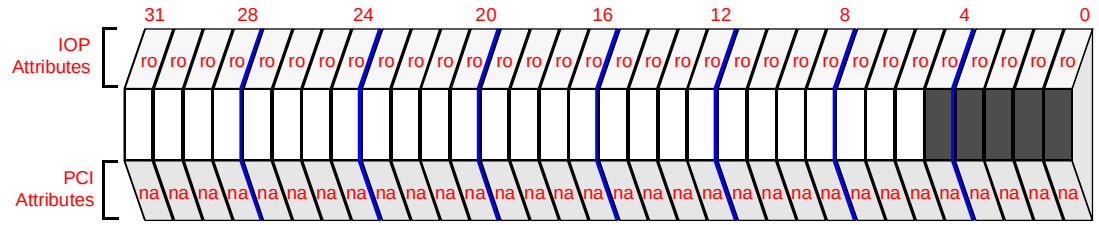
Bit	Default	Description
31:11	000000H	Reserved
10	0_2	Channel Active Flag - indicates the channel is either active (in use) or inactive (available). When set, indicates the channel is in use and actively performing DMA data transfers. When clear, indicates the channel is inactive and available to be configured to transfer data. The channel clears the Channel Active flag when the previously configured DMA transfer completes as a result of: • byte count reached zero and last chain descriptor is encountered and resume bit is clear (NULL value detected for Next Descriptor Address in chain descriptor) • Bus Errors The Channel Active flag is set when a Chain Descriptor is read from memory.
09	0_2	End of Transfer Interrupt Flag - set when the channel has signalled an interrupt to the Intel® XScale™ core after successfully completing an error-free DMA transfer but it is not the last descriptor in a chain.
08	0_2	End of Chain Interrupt Flag - set when the channel has signalled an interrupt to the Intel® XScale™ core after successfully completing an error-free DMA transfer that is the last of a chain.
07:06	0_2	Reserved
05	0_2	Internal Bus Master-Abort Flag - All Master-aborts when the channel is the master on the internal bus is reflected by setting this bit.
04	0_2	Reserved.
03	0_2	PCI Master Abort Flag - set on receiving master abort Split Completion Error Message.
02	0_2	PCI Target Abort Flag - set on receiving target abort Split Completion Error Message.
01	0_2	Unknown PCI-X Split Transaction Error - set on receiving Split Completion Error Message from the external target device.
00	0_2	Reserved

6.14.3 Descriptor Address Register x - DARx

The Descriptor Address Register (DARx) contains the address of the current chain descriptor in 80331 local memory for a DMA transfer. This register is read-only and is loaded when a new chain descriptor is read. Table 191 depicts the Descriptor Address Register.

All chain descriptors are aligned on an eight DWORD boundary.

Table 191. Descriptor Address Register x - DARx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E40CH
1	FFFF E44CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:05	00000000000 00000000000 00000_2	Current Descriptor Address - local memory address of the current chain descriptor that was read by the channel.
04:00	00000_2	Reserved

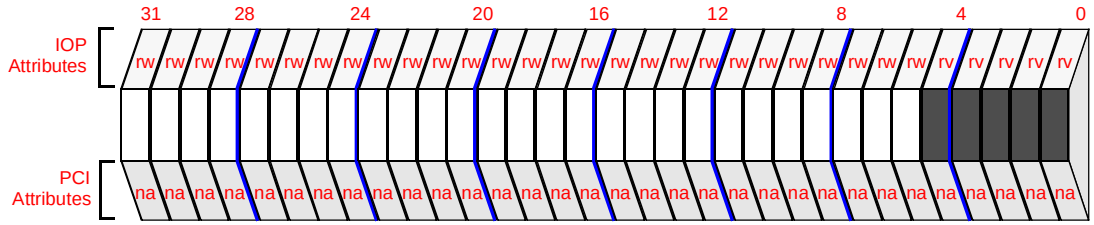
6.14.4 Next Descriptor Address Register x - NDARs

The Next Descriptor Address Register (NDARx) contains the address of the next chain descriptor in 80331 local memory for a DMA transfer. When starting a DMA transfer, this register contains the address of the first chain descriptor. Table 192 depicts the Next Descriptor Address Register.

All chain descriptors are required to be aligned on an eight DWORD boundary. The channel may set bits 04:00 to zero when loading this register.

Note: The *Channel Enable* bit in the CCRx and the *Channel Active* bit in the CSRx must both be clear prior to writing the Next Descriptor Address Register. Writing a value to this register while the channel is active may result in undefined behavior.

Table 192. Next Descriptor Address Register x - NDARs



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E410H
1	FFFF E450H

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:05	00000000000 00000000000 00000_2	Next Descriptor Address - local memory address of the next chain descriptor to be read by the channel.
04:00	00000_2	Reserved

6.14.5 PCI Address Register x - PADRx

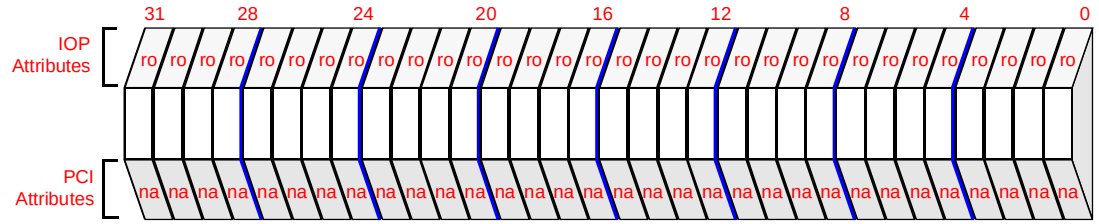
The PCI Address Register (PADRx) contains the 32-bit PCI address for SAC cycles or the lower 32-bit PCI address of a 64-bit PCI address for DAC cycles. This address represents the source or destination of the DMA transfer.

Table 193 shows the PCI Address Register.

The channel uses the full address bus to indicate the starting address. Refer to the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0a for additional information.

Note: The application programmer must not program the channel to transfer data across a 4 Gbyte boundary (i.e., the lower 32-bit address must not increment past the maximum address of FFFF.FFFFH). The channel does not notify the application of this condition.

Table 193. PCI Address Register x - PADRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E414H
1	FFFF E454H

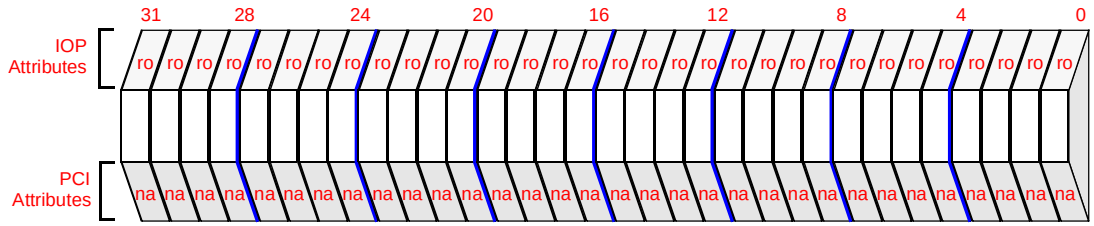
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	PCI Address - is the PCI source/destination address.

6.14.6 PCI Upper Address Register x - PUADRx

The PCI Upper Address Register (PUADRx) contains the upper 32-bits of a 64-bit PCI address. Table 194 shows the register. This register is read-only and is loaded when a chain descriptor is read from memory

Table 194. PCI Upper Address Register x - PU ADRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E418H
1	FFFF E458H

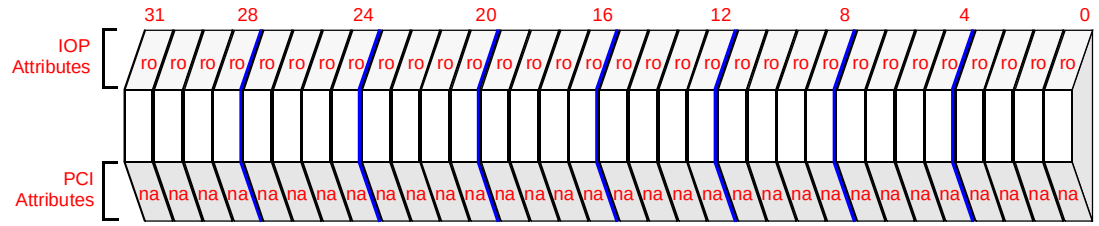
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	PCI Address - is the PCI source/destination upper address.

6.14.7 Local Address Register x - LADRx

The 80331 Local Address Register (LADRx) contains the 32-bit 80331 local address. The 80331 address space is a 32-bit, byte addressable address space. Table 195 shows the Local Address Register. This read-only register is loaded when a chain descriptor is read from memory.

Table 195. Local Address Register x - LADRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E41CH
1	FFFF E45CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	Local Address - the local source/destination address.

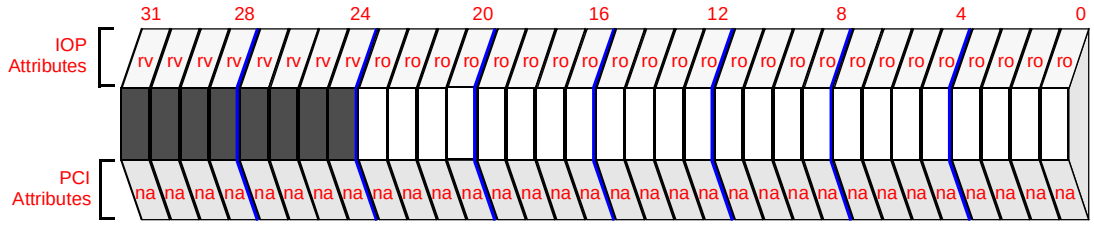
6.14.8 Byte Count Register x - BCRx

The Byte Count Register (BCRx) contains the number of bytes to transfer for a DMA transfer. This is a read-only register that is loaded from the Byte Count word in a chain descriptor. It allows for a maximum DMA transfer of 16 Mbytes. A value of zero is a valid byte count and results in no data words being transferred and no cycles generated on either the PCI bus or the internal bus. This register specifies the amount of data that is moved from the source to the destination.

Table 196 shows the Byte Count Register

Note: The byte count value is not required to be aligned to a DWORD boundary.

Table 196. Byte Count Register x - BCRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E420H
1	FFFF E460H

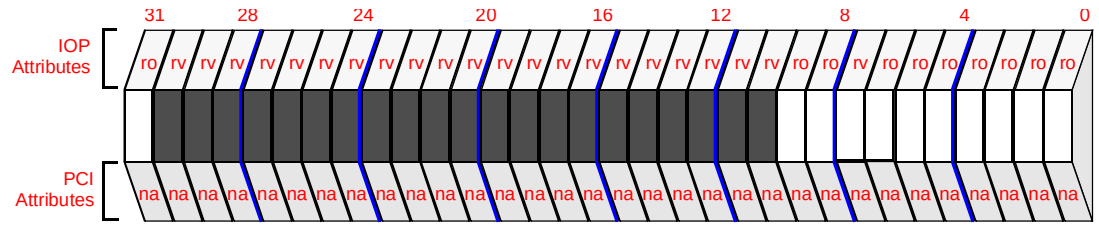
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:24	00H	Reserved
23:00	000000H	Byte Count - is the number of bytes to transfer for a DMA transfer.

6.14.9 Descriptor Control Register x - DCRx

The Descriptor Control Register (DCRx) contains control values for the DMA transfer on a per-chain descriptor basis. This read-only register is loaded whenever a chain descriptor is read from memory. These values may vary from chain descriptor to chain descriptor. Table 197 shows the definition of the Descriptor Control Register.

Table 197. Descriptor Control Register x - DCRx



Channel #	Intel® XScale™ Core internal bus address
0	FFFF E424H
1	FFFF E464H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	CRC Transfer Complete - When the CRC mode is enabled, this bit is set when the DMA completes processing of the current descriptor. **NOTE:** The AA updates this status in memory **only** by updating the Descriptor Control Word of the current descriptor (the sixth word of the descriptor pointed to by the DARx).
30:10	000000H	Reserved
09	0_2	CRC Seed Disable -- When clear, DMA fetches the seed value for the CRC calculation at the CRC Address. When set, DMA does not fetch a seed value, and continue to calculate CRC using the current contents from the last descriptor.
08	0_2	CRC Generation Enable -- When set, the DMA generates the CRC using the CRC-32C algorithm, the transferred data stream, and a seed value located at the CRC Address. When the Data Transfer is complete, the DMA writes the computed CRC value to CRC Address, and then write the DCR to the descriptor pointed to by the DAR with the CRC Transfer Complete bit set (DCR bit 31).
07	0_2	CRC Data Transfer Disable -- When set and the DMA has been enabled to generate CRC (DCR bit 8), the DMA discards the data read from memory after it has been used in the CRC calculation. This provides the DMA with the ability to generate CRC on a source data block without transferring the data block to a destination.
06	0_2	Memory-to-Memory Transfer Enable -- When set, the ATU no longer automatically forwards DMA transactions in the PCI address range defined by the descriptor to the 80331 external PCI bus. These transactions is now claimed via the decode of any of the 32-bit windows on the internal bus. For example, when both the Local Address Register (LADRx) and the PCI Address Register (PADRx) are then programmed to hit the MCUs memory window, local memory to local memory DMA transfers occurs. When clear, the ATU claims and forward any DMA transactions in the PCI address range defined by the descriptor to the 80331 external PCI bus. **NOTE:** All programmed lower 32-bit addresses is direct-mapped to the External PCI bus.
05	0_2	Dual Address Cycle Enable - is included for backward compatibility to previous I/O processor generations, but **has no effect on DMA Channel behavior** in the 80331 . When PUADRx is non-zero, the channel uses Dual Address Cycle (DAC) to transfer a 64-bit address on the PCI bus. When PUADRx equals zero, the channel uses Single Address Cycle (SAC) to transfer a 32-bit address on the PCI bus. For DAC, the PCI Address Register (PADRx) contains the lower 32-bit address used on the first address cycle. The PCI Upper Address Register (PUADRx) contains the upper 32 bits address cycle used on the second address cycle. The upper 32 bit address of a DAC transaction is required to be non-zero.
04	0_2	Interrupt Enable - when set, the channel generates an interrupt to the 80331 upon completion of a DMA transfer. When clear, no interrupt is generated.
03:00	0000_2	PCI Transaction - determines PCI bus transaction type for this DMA transfer. Hardware does not check for reserved or unsupported transaction types.

6.14.9.1 PCI Transactions Support

The Memory Write Block Command on the bus in PCI-X mode performs similarly to the PCI Memory Write and Invalidate command and is fully supported by both channels of the DMA controller. Refer to Section 6.4.2, "Local Memory to PCI Transfers" on page 358 for a complete description of the behavior of the DMA channel during this PCI bus cycle.

DMA Internal Bus transactions that are destined for the external PCI Bus are automatically forwarded through the Outbound ATU to the PCI bus. In this way, the backward compatibility with the previous generation programming model defined for the DMA descriptors is preserved.

Table 198.

DCR[3:0]	PCI Conventional Mode	PCI-X Mode
6H, CH, or EH	MR, MRL, and MRM as appropriate	Memory Read Block is used
7H or FH	Memory Write and Memory Write Invalidate Command as appropriate.	Memory Write Block is use.
all other values	Reserved	

Application Accelerator Unit 7

This chapter describes the integrated Application Accelerator (AA) Unit. The operation modes, setup, external interface, and implementation of the AA unit are detailed in this chapter.

7.1 Overview

The Application Accelerator provides low-latency, high-throughput data transfer capability between the AA unit and Intel® 80331 I/O processor (80331) local memory. It executes data transfers to and from 80331 local memory, check for all-zero result across local memory blocks, perform memory block fills, and provides the necessary programming interface. The Application Accelerator performs the following functions:

- Transfers data (read) from memory controller.
- Performs an optional boolean operation (XOR) on read data.
- Transfers data (write) to memory controller.
- Check for All-zero result across local memory blocks.
- Perform memory block fills.

The AA unit features:

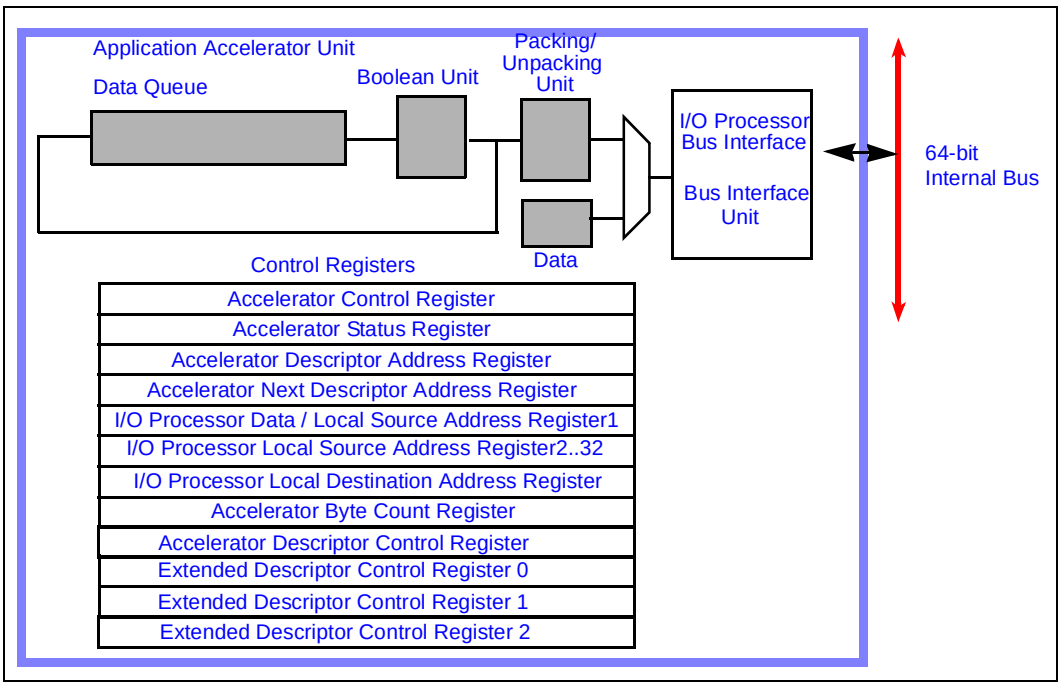
- 1Kbyte/512-byte store queue.
- Utilization of the 80331 memory controller Interface.
- 2^{32} addressing range on the 80331 local memory interface.
- Hardware support for unaligned data transfers for the internal bus.
- Fully programmable from the 80331.
- Support for automatic data chaining for gathering and scattering of data blocks.
- Support for writing a constant value to a memory block (block fill).
- Support for writing descriptor status to local memory.

7.2 Theory of Operation

The Application Accelerator is a master on the internal bus and performs data transfers to and from local memory. It does not interface to the PCI bus. AA uses direct addressing for memory controller.

The AA implements XOR algorithm in hardware. It performs XOR operation on multiple blocks of source (incoming) data and stores result back in 80331 local memory. The source and destination addresses are specified through chain descriptors resident in 80331 local memory. The AA can also check for all-zero result across local memory blocks or fill a memory block with arbitrary data. Figure 40 shows a block diagram of the AA unit. The AA can also perform memory-to-memory transfers of data blocks controlled by 80331 memory controller unit.

Figure 40. Application Accelerator Block Diagram



The Application Accelerator programming interface is accessible from the internal bus through a memory-mapped register interface. Data for the XOR operation is configured by writing the source addresses, destination address, number of bytes to transfer, and various control information into a chain descriptor in local memory. Chain descriptors are described in detail in Section 7.3.2, "Chain Descriptors" on page 384.

The Application Accelerator unit contains a hardware data packing and unpacking unit. This unit enables data transfers from and to unaligned addresses in 80331 local memory. All combinations of unaligned data are supported with the packing and unpacking unit. Data is held internally in the Application Accelerator until ready to be stored back to local memory. This is done using a 1KByte/512Byte holding queue. Data to be written back to 80331 local memory can either be aligned or unaligned.

Each chain descriptor contains the necessary information for initiating an XOR operation on blocks of data specified by the source addresses. The Application Accelerator unit supports chaining. Chain descriptors that specify the source data to be XORed can be linked together in 80331 local memory to form a linked list.

Similar to XOR operations, AA can be programed to compute parity across multiple memory blocks specified by chain descriptors. In addition, AA can also be used to perform memory block fills.

7.3 Hardware-Assist XOR Unit

The Application Accelerator Unit implements the XOR algorithm in hardware. It performs the XOR operation on multiple blocks of source (incoming) data and stores the result back in 80331 local memory.

- The process of reading source data, executing the XOR algorithm, and storing the XOR data hereafter is referred to as *XOR-transfer*.
- The process of reading or writing data hereafter is referred to as *data transfer*.

The source and destination addresses are specified through chain descriptors resident in 80331 local memory.

7.3.1 Data Transfer

All transfers are configured and initiated through a set of memory-mapped registers and one or more chain descriptors located in local memory. A transfer is defined by the source address, destination address, number of bytes to transfer, and control values. These values are loaded in the chain descriptor before a transfer begins. Table 199 describes the registers that need to be configured for any operation.

Table 199. Register Description

Register	Abbreviation	Description
Accelerator Control Register	ACR	Application Accelerator Control Word
Accelerator Status Register	ASR	Application Accelerator Status Word
Accelerator Descriptor Address Register	ADAR	Address of Current Chain Descriptor
Accelerator Next Descriptor Address Register	ANDAR	Address of Next Chain Descriptor
Data / Source Address Register1	D/SAR1	Data to be written or Local memory addresses of source data
Source Address Register 2..8	SAR2.. SAR32	Local memory addresses of source data
Destination Address Register	DAR	Local memory address of destination data
Accelerator Byte Count Register	ABCR	Number of Bytes to transfer
Data Register	DR	Data to be written to the destination
Accelerator Descriptor Control Register	ADCR	Chain Descriptor Control Word

7.3.2 Chain Descriptors

All transfers are controlled by chain descriptors located in local memory. A chain descriptor contains the necessary information to complete one transfer. A single transfer has only one chain descriptor in memory. Chain descriptors can be linked together to form more complex operations.

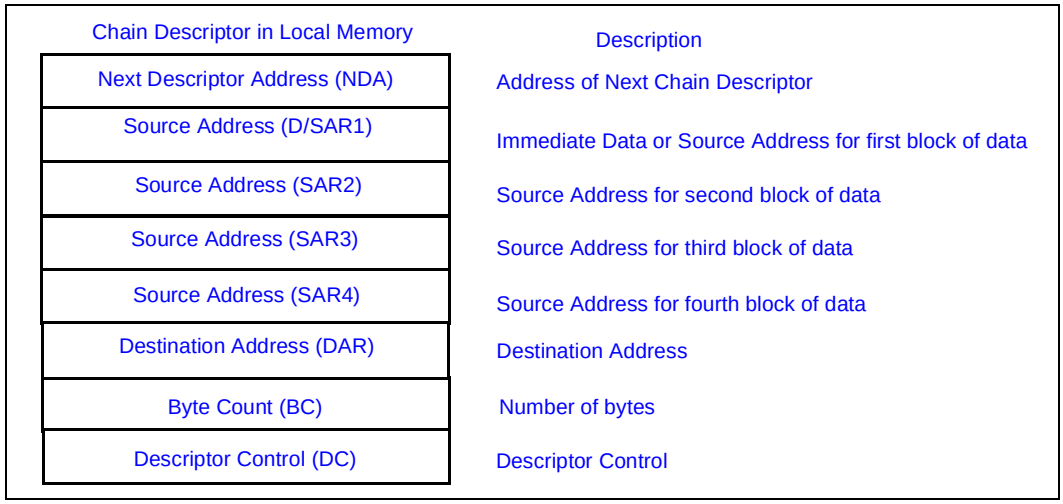
Warning: Chain descriptors can only be located in DDR SDRAM memory in order for the AAU to function properly. Location of the chain descriptors anywhere else (e.g., either on the Peripheral Bus or on PCI) is not supported and the results would be unpredictable.

To perform a transfer, one or more chain descriptors must first be written to 80331 local memory. The words of a descriptor are contiguous in local memory. Descriptors can different sizes, each dependent on the number of sources being addressed by the operation. The sizes supported by the Application Accelerator support four, eight, sixteen and thirty-two sources. The alignment of the descriptor in local memory is dependent on the descriptor size and is defined for each in the following sections. Not all sources must be used in a given descriptor.

7.3.2.1 Four-Source Descriptor Format

Figure 41 shows the format of an individual chain descriptor. This four-source descriptor is the smallest supported descriptor. The four-source descriptor requires eight contiguous words in 80331 local memory and is required to be aligned on an 8-word boundary. All eight words are required.

Figure 41. Chain Descriptor Format



Each word in the chain descriptor is analogous to control register values. Bit definitions for the words in the chain descriptor are the same as for the control registers.

- First word is local memory address of next chain descriptor. A value of zero specifies the end of the chain. This value is loaded into the Accelerator Next Descriptor Address Register. Because chain descriptors must be aligned on an 8-word boundary, the unit may ignore bits 04:00 of this address.
- Second word is address of the first block of data resident in local memory, or immediate data when performing a Memory Block Fill. This value is loaded into the Data / Source Address Register 1.
- Third word is the address of the second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 2.
- Fourth word is the address of the third block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 3.
- Fifth word is the address of the fourth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into the Source Address Register 4.
- Sixth word is the destination address where data is stored in local memory. This address is driven on the internal bus. This value is loaded into the Destination Address Register.
- Seventh word is the Byte Count value. This value specifies the number of bytes of data in the current chain descriptor. This value is loaded into the Accelerator Byte Count Register.
- Eighth word is the Descriptor Control Word. This word configures the Application Accelerator for one operation. This value is loaded into the Accelerator Descriptor Control Register.

There are no data alignment requirements for any source addresses or destination address. However, maximum performance is obtained from aligned transfers, especially small transfers. See Section 7.4.

Refer to Section 7.13 for additional description on the control registers.

7.3.2.2 Eight-Source Descriptor Format

To perform an *XOR-transfer* with up to eight source blocks of data, a special chain descriptor needs to be configured:

- The first part is the four-source descriptor (referred to as the *principal-descriptor*) containing the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains 4, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.

To perform a transfer, both parts (principal and mini-descriptor) must be written to local memory. Figure 42 shows the format of this eight-source descriptor. The eight-source descriptor requires twelve contiguous words in local memory and is required to be aligned on an 16-word boundary. All twelve words are required.

Figure 42. Chain Descriptor Format for Eight Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block

- The first eight words are defined in the four-source descriptor definition. See Section 7.3.2.2, "Eight-Source Descriptor Format" for the principal descriptor definition.
- The ninth word (1st word of mini-descriptor) is the address of the fifth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR5.
- The tenth word (2nd word of mini-descriptor) is the address of the sixth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR6.
- The eleventh word (3rd word of mini-descriptor) is the address of the seventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded SAR7.
- The twelfth word (4th word of mini-descriptor) is the address of the eighth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 8.

7.3.2.3 Sixteen-Source Descriptor Format

To perform an *XOR-transfer* with up to sixteen source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional eight (SAR9 - SAR16) source data blocks and the command/control for these data blocks. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.

To perform a transfer, all three parts (principal descriptor, mini-descriptor and extended-descriptor 0) must be written to local memory. Figure 43 shows the format of this configuration. Every descriptor requires twenty one contiguous words in local memory and is required to be aligned on an 32-word boundary. All twenty one words are required.

Figure 43. Chain Descriptor Format for Sixteen Source Addresses (XOR Function)

Chain Descriptor in Intel® XScale™ Core Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (SAR9)	Source Address for ninth block of data
Source Address (SAR10)	Source Address for tenth block of data
Source Address (SAR11)	Source Address for eleventh block of data
Source Address (SAR12)	Source Address for twelfth block of data
Source Address (SAR13)	Source Address for thirteenth block of data
Source Address (SAR14)	Source Address for fourteenth block of data
Source Address (SAR15)	Source Address for fifteenth block of data
Source Address (SAR16)	Source Address for sixteenth block of data

- The first eight words are defined in the four-source descriptor definition. See Section 7.3.2.1, "Four-Source Descriptor Format" for the principal descriptor definition.
- Words nine through twelve are defined in the eight-source descriptor definition. See Section 7.3.2.1, "Four-Source Descriptor Format" for the principal descriptor definition.
- The thirteenth word (1st word of extended-descriptor 0) is the Extended Descriptor Control Word 0. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 0.
- The fourteenth word (2nd word of extended-descriptor 0) is the address of the ninth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 9.
- The fifteenth word (3rd word of extended-descriptor 0) is the address of the tenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 10.
- The sixteenth word (4th word of extended-descriptor 0) is the address of the eleventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 11.
- The seventeenth word (5th word of extended-descriptor 0) is the address of the twelfth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 12.
- The eighteenth word (6th word of extended-descriptor 0) is the address of the thirteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 13.
- The nineteenth word (7th word of extended-descriptor 0) is the address of the fourteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 14.
- The twentieth word (8th word of extended-descriptor 0) is the address of the fifteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 15.
- The twenty first word (9th word of extended-descriptor 0) is the address of the sixteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 16.

7.3.2.4 Thirty-two-Source Descriptor Format

To perform an *XOR-transfer* with up to thirty two source blocks of data, a special chain descriptor needs to be configured:

- The first part (***principal-descriptor***) contains the address of the first 4 source data blocks along with other information.
- The second part (***mini-descriptor***) contains four, DWORDs containing the address of the additional four (SAR5 - SAR8) source data blocks. The mini-descriptor is written to a contiguous address immediately following the principal descriptor.
- The third part (***extended-descriptor 0***) contains nine, DWORDs containing the address of the additional eight (SAR9 - SAR16) source data blocks and the command/control for these data blocks. The extended-descriptor 0 is written to a contiguous address immediately following the mini descriptor.
- The fourth part (***extended-descriptor 1***) contains nine, DWORDs containing the address of the additional eight (SAR17 - SAR24) source data blocks and the command/control for these data blocks. The extended-descriptor 1 is written to a contiguous address immediately following extended-descriptor 0.
- The fifth part (***extended-descriptor 2***) contains nine, DWORDs containing the address of the additional eight (SAR25 - SAR32) source data blocks and the command/control for these data blocks. The extended-descriptor 2 is written to a contiguous address immediately following extended-descriptor 1.

To perform a transfer, all five parts (principal descriptor, mini-descriptor, extended-descriptor 0, extended-descriptor 1, and extended-descriptor 2) must be written to local memory. Figure 44 shows the format of this configuration. The full descriptor requires thirty nine contiguous words in local memory and is required to be aligned on an 64-word boundary. All thirty nine words are required.

Figure 44. Chain Descriptor Format for Thirty Two Source Addresses (XOR Function)

Chain Descriptor in I/O Processor Memory	Description
Next Descriptor Address (NDA)	Address of Next Chain Descriptor
Source Address (D/SAR1)	Source Address for first block of data
Source Address (SAR2)	Source Address for second block of data
Source Address (SAR3)	Source Address for third block of data
Source Address (SAR4)	Source Address for fourth block of data
Destination Address (DAR)	Destination Address of XOR-ed data
Byte Count (BC)	Number of bytes to XOR
Descriptor Control (DC)	Descriptor Control
Source Address (SAR5)	Source Address for fifth data block
Source Address (SAR6)	Source Address for sixth data block
Source Address (SAR7)	Source Address for seventh data block
Source Address (SAR8)	Source Address for eighth data block
Extended Descriptor Control 0 (EDC0)	Extended Descriptor 0 control
Source Address (SAR9)	Source Address for ninth block of data
Source Address (SAR10)	Source Address for tenth block of data
Source Address (SAR11)	Source Address for eleventh block of data
Source Address (SAR12)	Source Address for twelfth block of data
Source Address (SAR13)	Source Address for thirteenth block of data
Source Address (SAR14)	Source Address for fourteenth block of data
Source Address (SAR15)	Source Address for fifteenth block of data
Source Address (SAR16)	Source Address for sixteenth block of data
Extended Descriptor Control 1 (EDC1)	Extended Descriptor 1 control
Source Address (SAR17)	Source Address for seventeenth block of data
Source Address (SAR18)	Source Address for eighteenth block of data
Source Address (SAR19)	Source Address for nineteenth block of data
Source Address (SAR20)	Source Address for twentieth block of data
Source Address (SAR21)	Source Address for twenty first block of data
Source Address (SAR22)	Source Address for twenty second block of data
Source Address (SAR23)	Source Address for twenty third block of data
Source Address (SAR24)	Source Address for twenty fourth block of data
Extended Descriptor Control 2 (EDC2)	Extended Descriptor 2 control
Source Address (SAR25)	Source Address for twenty fifth block of data
Source Address (SAR26)	Source Address for twenty sixth block of data
Source Address (SAR27)	Source Address for twenty seventh block of data
Source Address (SAR28)	Source Address for twenty eighth block of data
Source Address (SAR29)	Source Address for twenty ninth block of data
Source Address (SAR30)	Source Address for thirtieth block of data
Source Address (SAR31)	Source Address for thirty first block of data
Source Address (SAR32)	Source Address for thirty second block of data

- The first eight words are defined in the Four-source descriptor definition. See Section 7.3.2.1, "Four-Source Descriptor Format" for the principal descriptor definition.
- Words nine through twelve are defined in the Eight-source descriptor definition. See Section 7.3.2.1, "Four-Source Descriptor Format" for the principal descriptor definition.
- Words thirteen through twenty-one are defined in the Sixteen-source descriptor definition. See Section 7.3.2.3, "Sixteen-Source Descriptor Format" for the principal descriptor definition.
- The twenty second word (1st word of extended-descriptor 1) is the Extended Descriptor Control Word 1. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 1.
- The twenty third word (2nd word of extended-descriptor 1) is the address of the seventeenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR17.
- The twenty fourth word (3rd word of extended-descriptor 1) is the address of the eighteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 18.
- The twenty fifth word (4th word of extended-descriptor 1) is the address of the nineteenth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 19.
- The twenty sixth word (5th word of extended-descriptor 1) is the address of the twentieth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 20.
- The twenty seventh word (6th word of extended-descriptor 1) is the address of the twenty first block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 21.
- The twenty eighth word (7th word of extended-descriptor 1) is the address of the twenty second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 22.
- The twenty ninth word (8th word of extended-descriptor 1) is the address of the twenty third block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 23.
- The thirtieth word (9th word of extended-descriptor 1) is the address of the twenty fourth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 24.
- The thirty first word (1st word of extended-descriptor 2) is the Extended Descriptor Control Word 2. This word configures the Application Accelerator for one operation. The value is loaded into the Extended Descriptor Control Register 2.
- The thirty second word (2nd word of extended-descriptor 2) is the address of the twenty fifth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 25.
- The thirty third word (3rd word of extended-descriptor 2) is the address of the twenty sixth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 26.
- The thirty fourth word (4th word of extended-descriptor 2) is the address of the twenty seventh block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 27.

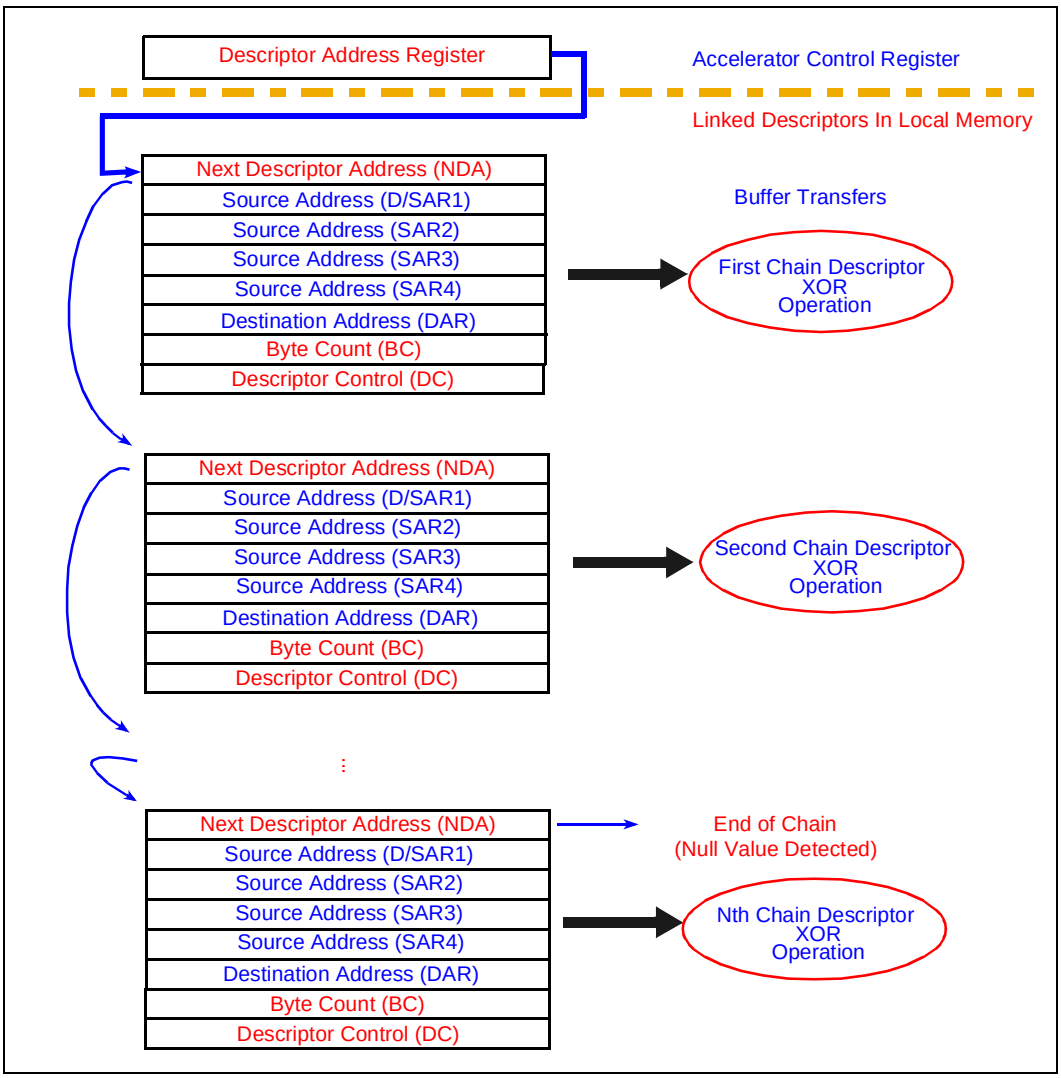
- The thirty fifth word (5th word of extended-descriptor 2) is the address of the twenty eighth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 28.
- The thirty sixth word (6th word of extended-descriptor 2) is the address of the twenty ninth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 29.
- The thirty seventh word (7th word of extended-descriptor 2) is the address of the thirtieth block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 30.
- The thirty eighth word (8th word of extended-descriptor 2) is the address of the thirty first block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 31.
- The thirty ninth word (9th word of extended-descriptor 2) is the address of the thirty second block of data resident in local memory. This address is driven on the internal bus. This value is loaded into SAR 32.

7.3.3 Descriptor Chaining

To perform an AA operation, a series of chain descriptors can be built in local memory to operate on multiple blocks of source data resident in local memory. The result can then be stored back in local memory. An application can build multiple chain descriptors to operate on many blocks of data which have different source addresses within the local memory.

When multiple chain descriptors are built in local memory, the application can link each of these chain descriptors using the Next Descriptor Address in the chain descriptor. This address logically links the chain descriptors together. This allows the application to build a list of transfers which may not require the processor until all transfers are complete. Figure 45 shows an example of a linked-list of transfers using only four-source descriptors specified in external memory.

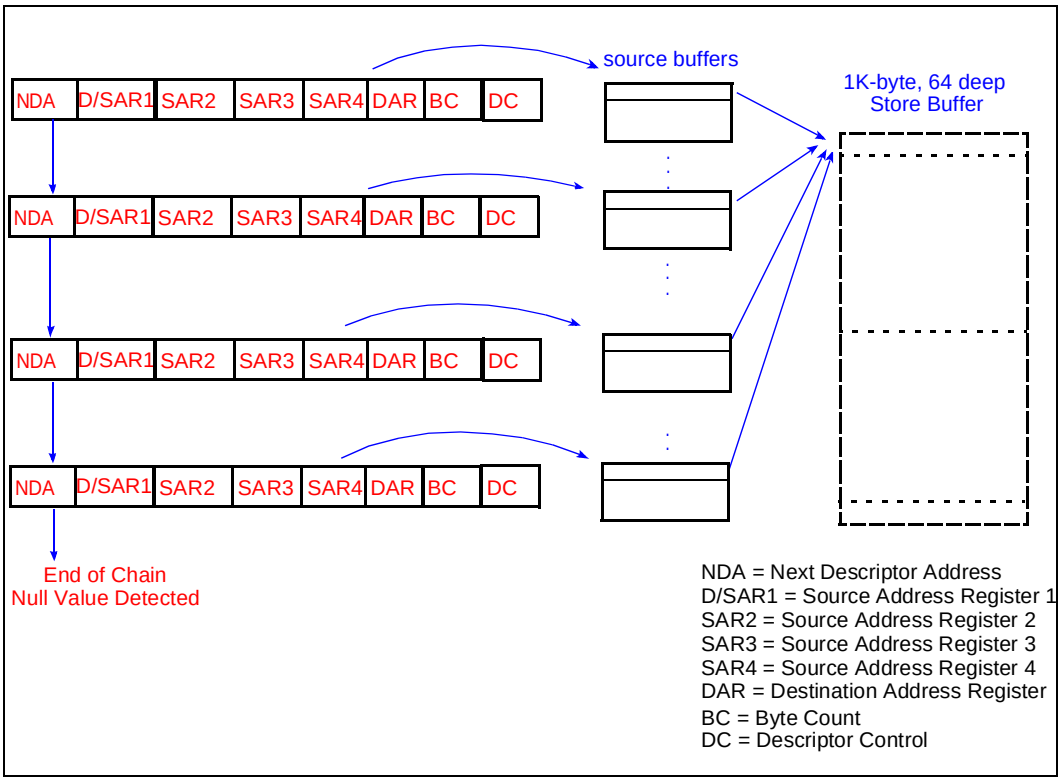
Figure 45. XOR Chaining Operation



7.4 AA Descriptor Processing

An AA operation is initiated by building one or more chain descriptors in Intel® XScale™ core local memory (ARM* architecture compliant). Figure 46 shows the format of a principal descriptor.

Figure 46. Example of Gather Chaining for Four Source Blocks



The following describes the steps for initiating a new AA operation:

1. The AA must be inactive prior to starting an AA operation. This can be checked by software by reading the *Accelerator Active* bit in the Accelerator Status Register. When this bit is clear, the unit is inactive. When this bit is set, the unit is currently active.
2. The ASR must be cleared of all error conditions.
3. The software writes the address of the first chain descriptor to the Accelerator Next Descriptor Address Register (ANDAR).
4. The software sets the *Accelerator Enable* bit in the Accelerator Control Register (ACR). Because this is the start of a new AA operation and not the resumption of a previous operation, the *Chain Resume* bit in the ACR should be clear.
5. The AA starts the AA operation by reading the chain descriptor at the address contained in ANDAR. The AA loads the chain descriptor values into the ADAR and begins data transfer. The Accelerator Descriptor Address Register (ADAR) contains the address of the chain descriptor just read and ANDAR now contains the Next Descriptor Address from the chain descriptor just read.

The last descriptor in the AA chain list has zero in the next descriptor address field specifying the last chain descriptor. A NULL value notifies the AA not to read additional chain descriptors from memory.

Once an AA operation is active, it can be temporarily suspended by clearing the *Accelerator Enable* bit in the ACR. Note that this does not abort the AA operation. The unit resumes the process when the *Accelerator Enable* bit is set.

When descriptors are read from external memory, bus latency and memory speed affect chaining latency. Chaining latency is defined as the time required for the AA to access the next chain descriptor plus the time required to set up the next AA operation.

7.4.1 Scatter Gather Transfers

The Application Accelerator can be used to perform typical scatter gather transfers. This consists of programming the chain descriptors to gather data which may be located in non-contiguous blocks of memory. The chain descriptor specifies the destination location such that once all data has been processed, the data is contiguous in memory. Figure 46 shows how the destination pointers can gather data.

7.4.2 Synchronizing a Program to Chained Operation

Any operation involving the AA can be synchronized to a program executing on the Intel® XScale™ core through the use of processor interrupts. The AA generates an interrupt to the Intel® XScale™ core under certain conditions. They are:

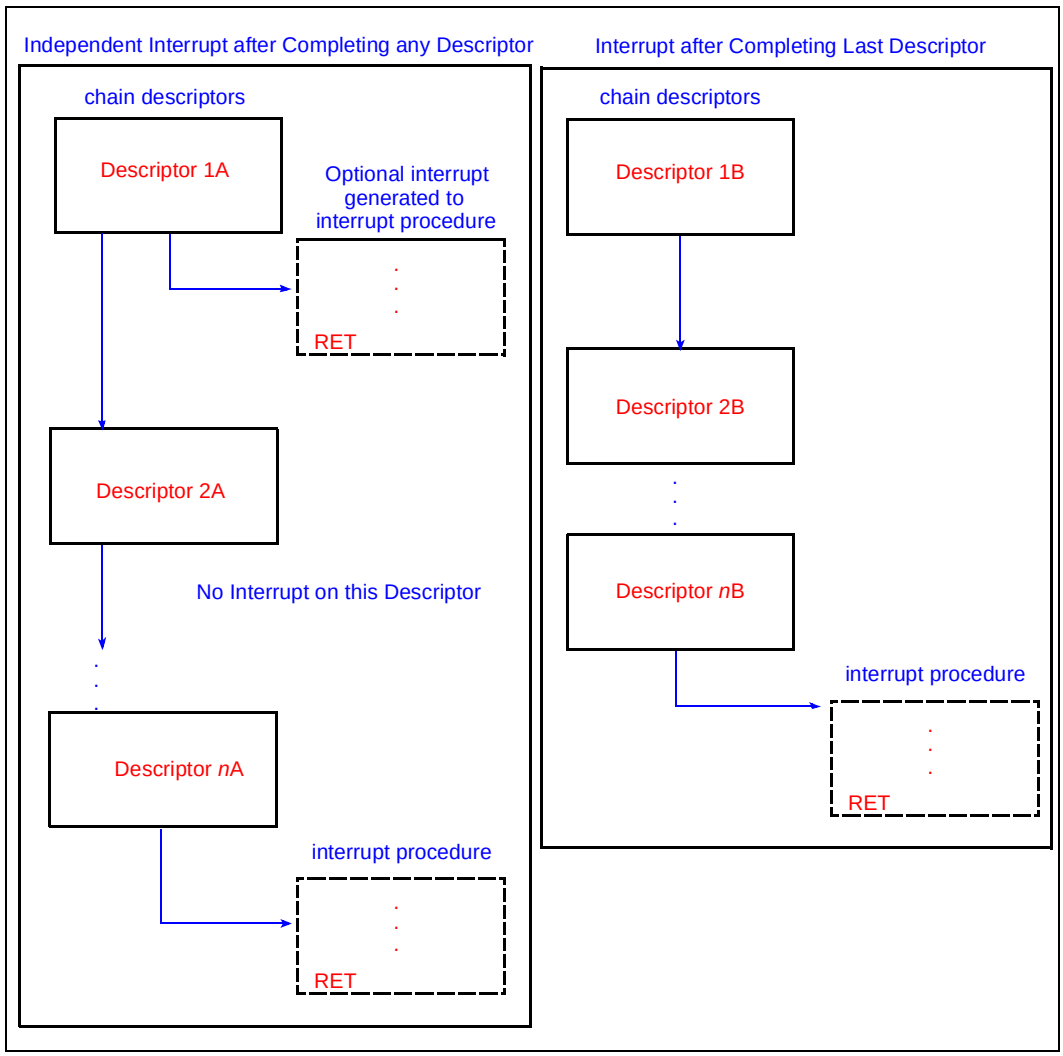
1. [Interrupt and Continue] The AA completes processing a chain descriptor and the Accelerator Next Descriptor Address Register (ANDAR) is non-zero. When the *Interrupt Enable* bit within the Accelerator Descriptor Control Register (ADCR) is set, an interrupt is generated to the Intel® XScale™ core. This interrupt is for synchronization purposes. The AA sets the *End Of Transfer Interrupt* flag in the Accelerator Status Register (ASR). Since it is not the last chain descriptor in the list, the AA starts to process the next chain descriptor without requiring any processor interaction.
2. [End of Chain] The AA completes processing a chain descriptor and the Accelerator Next Descriptor Address Register is zero specifying the end of the chain. When the *Interrupt Enable* bit within the ADCR is set, an interrupt is generated to the Intel® XScale™ core. The AA sets the *End Of Chain Interrupt* flag in the ASR.
3. [Error] An error condition occurs (refer to Section 7.11, "Error Conditions" on page 413 for Application Accelerator error conditions) during a transfer. The AA halts operation on the current chain descriptor and not proceed to the next chain descriptor.

Each chain descriptor can independently set the *Interrupt Enable* bit in the Descriptor Control word. This bit enables an independent interrupt once a chain descriptor is processed. This bit can be set or clear within each chain descriptor. Control of interrupt generation within each descriptor aids in synchronization of the executing software with AA operation.

Figure 47 shows two examples of program synchronization. The left column shows program synchronization based on individual chain descriptors. Descriptor 1A generated an interrupt to the processor, while descriptor 2A did not because the *Interrupt Enable* bit was clear. The last

descriptor *n*A, generated an interrupt to signify the end of the chain has been reached. The right column in Figure 47 shows an example where the interrupt was generated only on the last descriptor signifying the end of chain.

Figure 47. Synchronizing to Chained AA Operation



7.4.3 Appending to The End of a Chain

Once the AA has started processing a chain of descriptors, application software may need to append a chain descriptor to the current chain without interrupting the transfer in progress. The mechanism used for performing this action is controlled by the *Chain Resume* bit in the Accelerator Control Register (ACR).

The AA reads the subsequent chain descriptor each time it completes the current chain descriptor and the Accelerator Next Descriptor Address Register (ANDAR) is non-zero. ANDAR always contains the address of the next chain descriptor to be read and the Accelerator Descriptor Address Register (ADAR) always contains the address of the current chain descriptor.

The procedure for appending chains requires the software to find the last chain descriptor in the current chain and change the Next Descriptor Address in that descriptor to the address of the new chain to be appended. The software then sets the *Chain Resume* bit in the ACR. It does not matter when the unit is active or not.

The AA examines the *Chain Resume* bit of the ACR when the unit is idle or upon completion of a chain of transfers. When this bit is set, the AA re-reads the Next Descriptor Address of the current chain descriptor and load it into ANDAR. The address of the current chain descriptor is contained in ADAR. The AA clears the *Chain Resume* bit and then examines ANDAR. When ANDAR is not zero, the AA reads the chain descriptor using this new address and begin a new operation. When ANDAR is zero, the AA remains or return to idle.

There are three cases to consider:

1. The AA completes an AA operation and it is not the last descriptor in the chain. In this case, the AA clears the *Chain Resume* bit and reads the next chain descriptor. The appended descriptor is read when the AA reaches the end of the original chain.
2. The channel completes an AA transfer and it is the last descriptor in the chain. In this case, the AA examines the state of the *Chain Resume* bit. When the bit is set, the AA re-reads the current descriptor to get the address of the appended chain descriptor. When the bit is clear, the AA returns to idle.
3. The AA is idle. In this case, the AA examines the state of the *Chain Resume* bit when the ACR is written. When the bit is set, the AA re-reads the last descriptor from the most-recent chain to get the appended chain descriptor.

7.5 AA Operations

The AA can be configured on a per descriptor basis through the Descriptor Control Word to perform three distinct operations:

1. In an **XOR operation**, the AA generates a parity data stream in local memory that is comprised of the XOR of up to 32 distinct data streams (i.e., SAR1..32) on a per byte basis. All of the source data streams and the parity data stream can be up to 16 MB long.
2. Perform a **Memory Block Fil**l of up to 16 MB of local memory with a 32-bit constant (DATA/SAR1).
3. With the **Zero Result Buffer Check**, the AA confirms that the XOR of all the bytes of source data results in 00H for the entire bytecount. All the source data streams can be up to 16 MB long. The results of the check is written back to the Descriptor Control Word in local memory.

The following sections describes the three AA operations in detail.

7.5.1 XOR Operation

Figure 48 describes the XOR algorithm implementation. In this illustrative example, there are four blocks of source data to be XOR-ed. The intermediate result is kept by the store queue in the AA before being written back to local memory. The source data is located at addresses A000 0400H, A000 0800H, A000 0C00H and A000 1000H respectively.

All data transfers needed for this operation are controlled by chain descriptors located in local memory. The Application Accelerator as a master on the internal bus initiates data transfer. The algorithm is implemented such that as data is read from local memory, the boolean unit executes the XOR operation on incoming data.

Figure 48. The Bit-wise XOR Algorithm

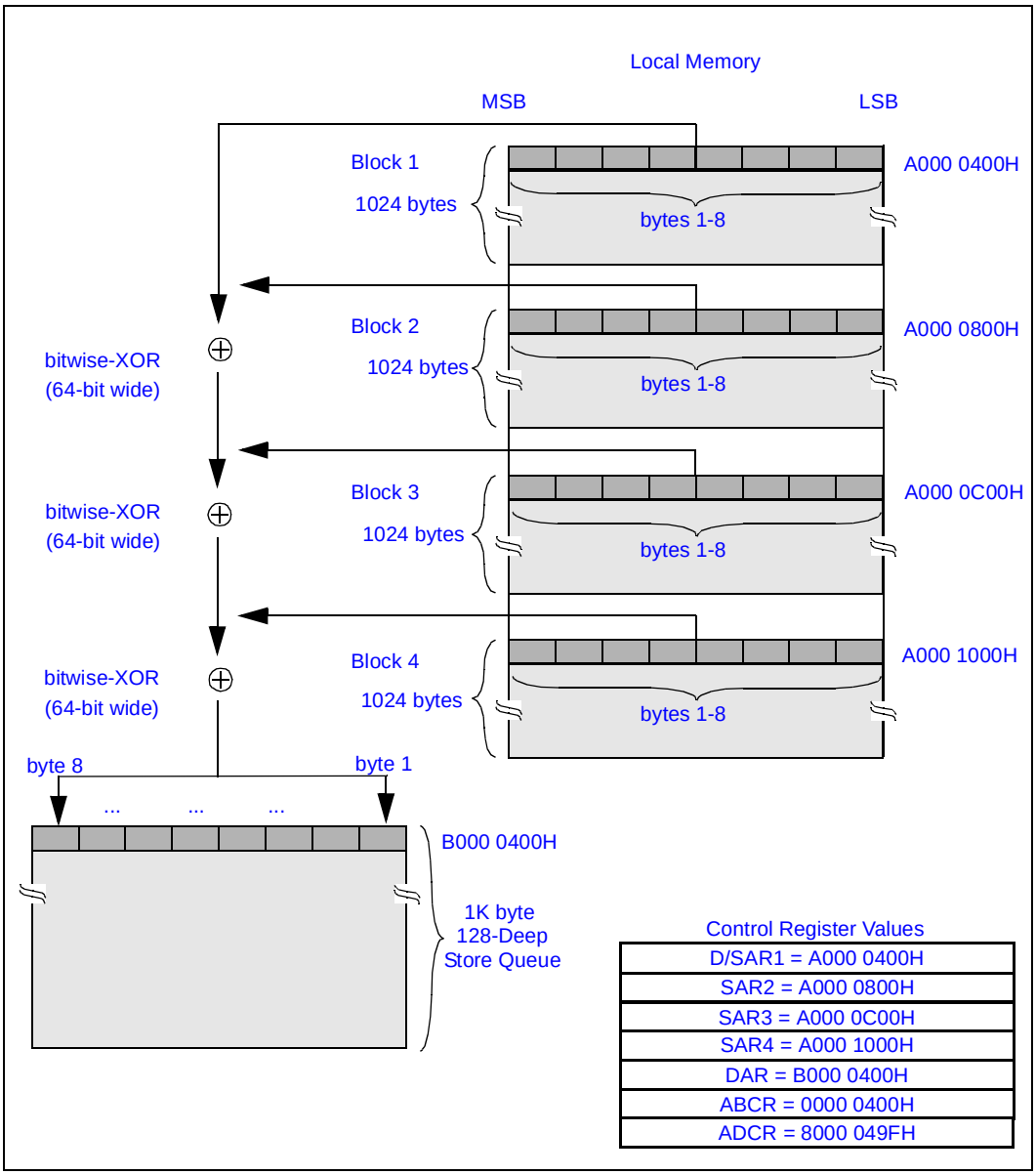
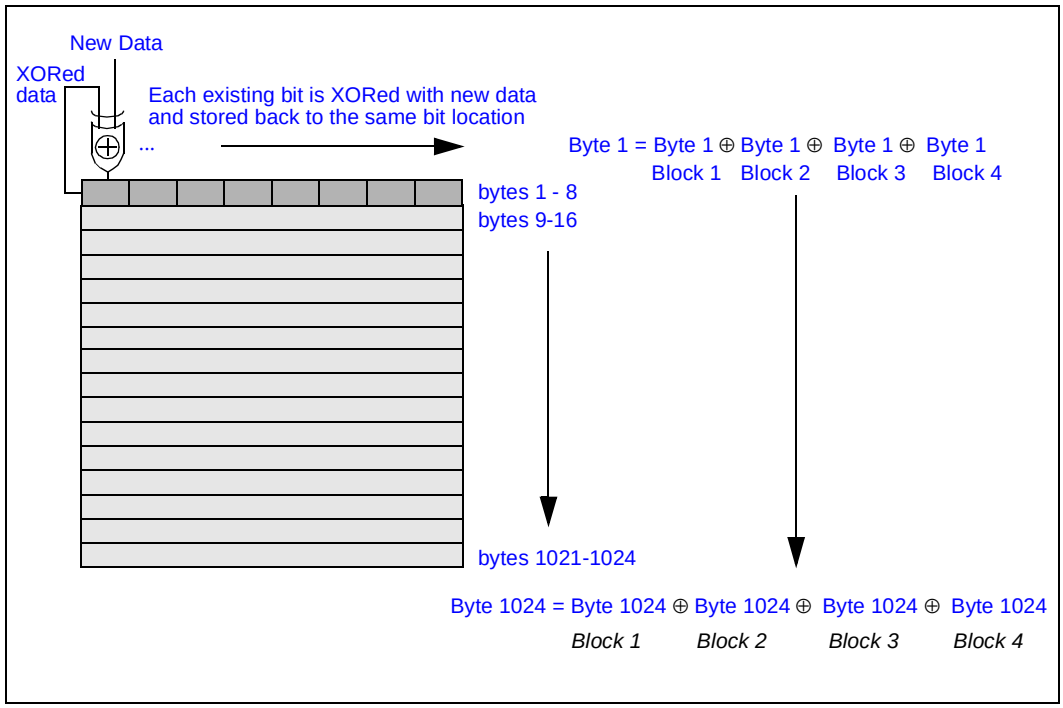


Figure 49. Hardware Assist XOR Unit



The XOR algorithm and methodology followed once a chain descriptor has been configured is detailed below:

1. The Application Accelerator as a master on the bus initiates data transfer from the address pointed at by the First Source Address Register (SAR1). The total number of bytes to *XOR-transfer* is specified by the Byte Count (BC) field in the chain descriptor.
   a. When the Direct Fill command is selected for SAR1, this is designated as the first block of data in the current XOR operation, and the data is transferred directly to the store queue. The number of bytes transferred to the store queue is 1KByte/512Bytes.
   b. When the XOR command is selected for SAR1, the boolean unit performs the XOR operation on the data currently existing in the store queue with the data being transferred from memory (see steps 3-7 for SAR2). This may be done to XOR more than 32 blocks of data together with a byte count of 1KByte or less.

Note: When the Byte Count Register contains a value greater than the buffer size, the AA completes the *XOR-transfer* operation on the first buffer of data obtained from each Source Register (D/SAR1, SAR2- SAR4), then proceeds with the next buffer of data. This process is repeated until the BCR contains a zero value.

2. The Application Accelerator transfers the first eight bytes of data from the address pointed at by the Second Source Address Register (SAR2).
3. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the first eight bytes of data read from D/SAR1 (bytes 1-8) which are stored in the queue and the first eight bytes of data just read from SAR2 (bytes 1-8).
4. The XOR-ed result is transferred to the store queue and stored in the first eight bytes (bytes 1-8) overwriting previously stored data.

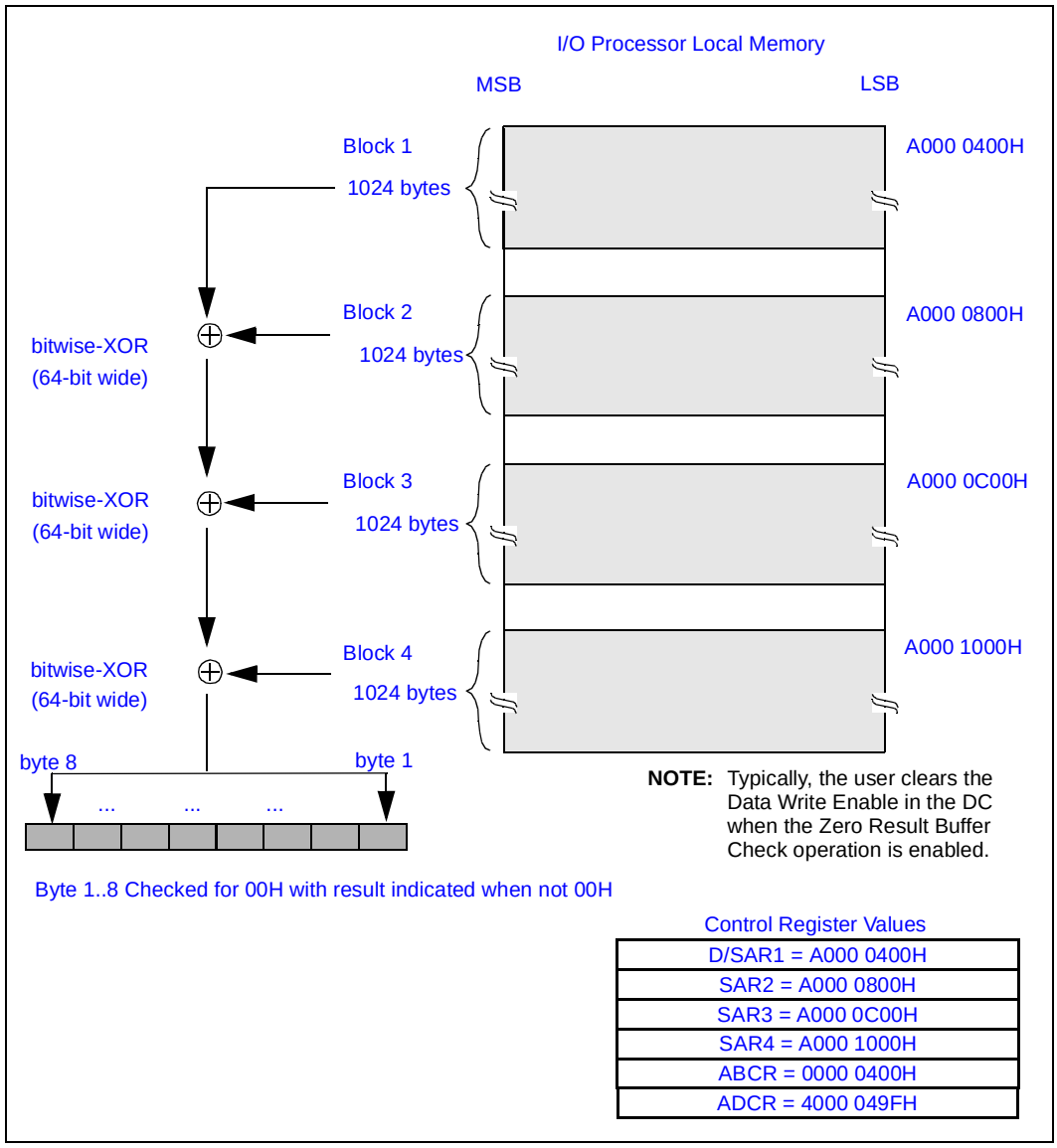
5. The Application Accelerator transfers the next eight bytes of data (bytes 9-16) from address pointed at by the Second Source Address Register (SAR2).
6. The boolean unit performs the bit-wise XOR algorithm on the input operands. The input operands are the next eight bytes of data read from D/SAR1 (bytes 9-16 stored in the queue) and the eight bytes of data read from SAR2 in Step-5.
7. Step-5 and Step-6 (Data transfer and XOR) are repeated until all data pointed at by SAR1 is XOR-ed with the corresponding data pointed at by SAR2. The store queue now contains a buffer full of XOR-ed data, the source addresses for which were specified in SAR1 and SAR2.
8. Steps 1-7 are repeated once again. The first input to the XOR unit is the data held in the store queue and the second input is the data pointed at by SAR3.
9. The above steps are repeated once more. The first input to the XOR unit is the data held in the store queue and the second input is the data pointed at by SAR4.
10. Once Steps 1-9 are completed, the XOR operation is complete for the first full buffer of the current chain descriptor. When the Destination Write Enable Bit in the Accelerator Descriptor Control Register (ADCR) is set, the data in the store queue is written to local memory at the address pointed to by the Destination Address Register (DAR). When the Destination Write Enable Bit in the ADCR is not set, the data is not written to local memory and is held in the queue. Steps 1-9 are repeated until all the bytes of data have undergone the *XOR-transfer* operation.

Note: When the ABCR register contains a value greater than the buffer size and the ADCR.dwe bit is cleared, the AAU only reads the first buffer of data and perform the specified function. It does not read the remaining bytes specified in the ABCR. Further, the AAU proceeds to process the next chain descriptor when it is specified.

7.5.2 Zero Result Buffer Check

The AA can be used to verify parity across memory blocks specified by the SARx registers. As with XOR operations, descriptors are used to specify the memory blocks on which the AA performs the Zero Result Buffer Check. Figure 50 illustrates a Zero Result Buffer Check performed by the AA. After processing all source data, the AA updates the Result Buffer Not Zero bit of the eighth word of the descriptor (ADCR) pointed to by the ADAR in local memory.

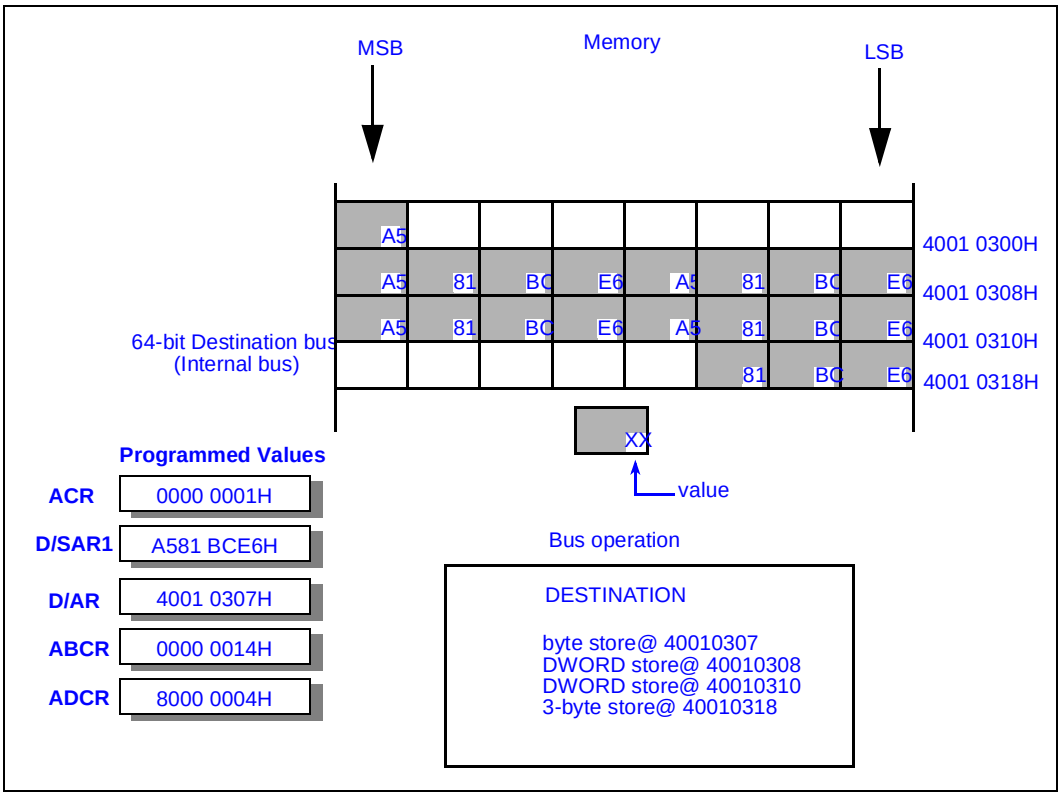
Figure 50. An example of Zero Result Buffer Check



7.5.3 Memory Block Fill Operation

The AA can be used to write a constant value to a memory block in the 80331 local memory. As with XOR operations, descriptors are used to specify the memory blocks to which the AA writes the data contained in the Data / Source Address Register1. All memory block fill operations are controlled by chain descriptors located in the Intel® XScale™ core local memory. Figure 51 illustrates a Block Fill Operation to an arbitrary destination address.

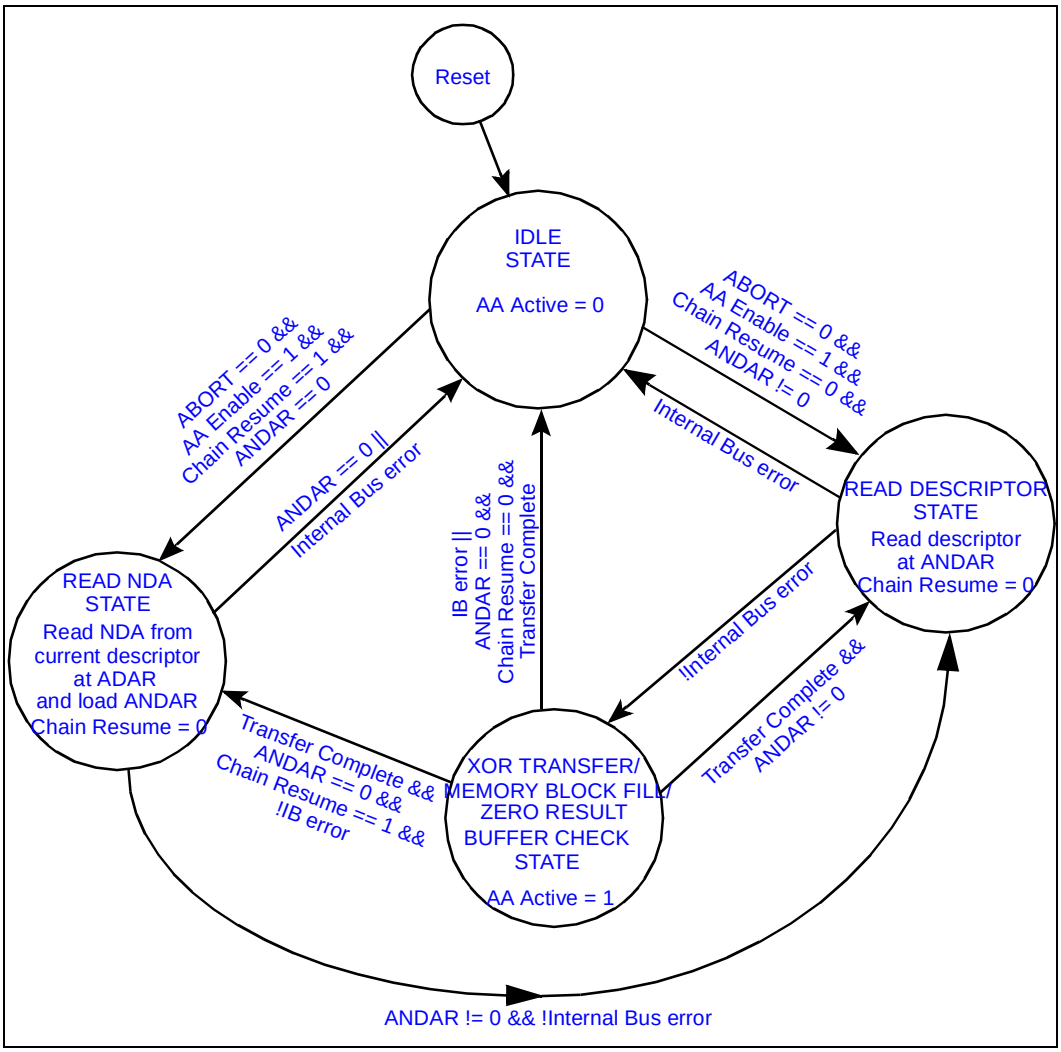
Figure 51. Example of a Memory Block Fill Operation



7.6 Programming Model State Diagram

The AA programming model diagram is shown in Figure 52. Error condition states are not shown.

Figure 52. Application Accelerator Programming Model State Diagram



7.7 Application Accelerator Priority

The internal bus arbitration logic determines which internal bus master has access to the 80331 internal bus. The Application Accelerator has an independent Bus Request/Grant signal pair to the internal bus arbitration logic. Chapter 12, "Intel® 80331 I/O Processor Arbitration Unit" describes in detail the priority scheme between all of the bus masters on the internal bus.

In addition, the internal bus arbitration unit has a Multi-Transaction timer (Section 12.4.3, "Multi-Transaction Timer Register 2 - MTTR2" on page 626) that affects the throughput of the AA. The default value for MTT2 of 152 clocks was chosen to ensure that once an internal bus agent (in this case the AA) is granted the internal bus that it is guaranteed an opportunity to burst data into DDR SDRAM memory. However, when the bus is busy the AA loses grant before the burst is completed. This means that the AA is able to complete only one burst for each arbitration cycle.

Alternatively, the user may wish to increase the value of MTT2 to guarantee that two or more bursts is able to complete within an arbitration cycle.

For example, assuming 1 Kbyte bursts and a 64-bit memory subsystem, an MTT2 setting of 192 clocks would be sufficient to support two 1 Kbyte bursts for the AA in a single arbitration cycle.

Warning: Increasing the MTT2 value may also increase the latency to memory for the Intel® XScale™ core on the average. Before changing the MTT2 value, it's imperative that the overall impact to the performance of the application is considered.

7.8 Packing and Unpacking

The Application Accelerator contains a hardware data packing and unpacking unit to support data transfers between unaligned source and destination addresses. Source and destination addresses can either be unaligned or aligned on natural boundaries. The packing unit optimizes data transfers to and from 32 and 64-bit memory. It reformats data words for the correct bus data width. When the read data needs to be packed or unpacked, the data is held internally and does not need to be re-read.

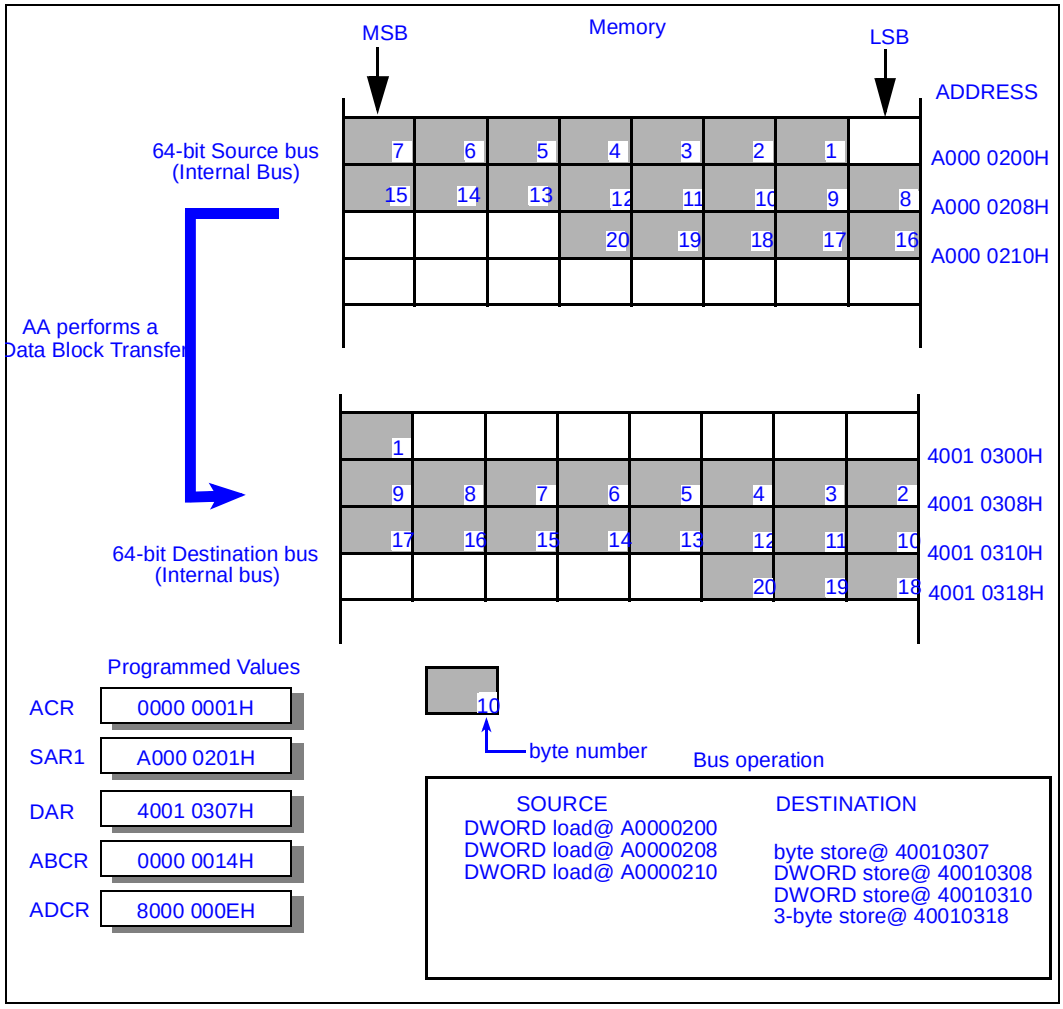
Aligned data transfers fall on natural boundaries. For example; DWORDs are aligned on 8-byte boundaries and words are aligned on 4-byte boundaries. Data transfers take place in two instances:

- The source and destination addresses are both aligned.
- All or some source addresses are unaligned and the destination address is aligned or unaligned.

7.8.1 64-bit Unaligned Data Transfers

Figure 53 illustrates a data transfer between unaligned 64-bit, source and destination addresses.

Figure 53. Optimization of an Unaligned Data Transfer



7.9 Programming the Application Accelerator

The software for Application Accelerator falls into the following categories:

- AA initialization
- Suspend AA
- Appending Descriptors
- Resume AA Operation

An example for each category is shown in the following sections as pseudo code flow.

The AA control register provides independent control each time the AA is configured. This provides the greatest flexibility to the applications programmer.

The most efficient method for operating the AA is to use the *Chain Resume* capability described in Section 7.3.3, "Descriptor Chaining". To use of the *Chain Resume* capability for appending descriptors to chains in normal operation, an initial AA descriptor must be executed. This initialization step is described in Section 7.9.1, "Application Accelerator Initialization". The example AA operations provided later in this section use the *Chain Resume* capability as follows:

- Store Descriptor in Local Memory
- Append Descriptor to Chain
- Resume AA Operation

7.9.1 Application Accelerator Initialization

The AA is designed to have independent control of the interrupts, enables, and control. The initialization consists of virtually no overhead as shown in Figure 54.

Figure 54. Pseudo Code: Application Accelerator Initialization

```
ACR = 0x0000 0000 ; Disable the application accelerator
Call setup_accelerator
```

The following example illustrates how AA initialization S/W prepares the AA for descriptor *Chain Resume* operation. Initializing the AA for chaining requires an initial descriptor be created and executed. This descriptor is then the start of the chain, and future descriptors are appended to this descriptor to create the chain. This descriptor is a NULL descriptor, requiring no source or destination data buffers be allocated. To start an operation, software simply sets the AA Enable bit in the "Accelerator Control Register - ACR" as shown in Figure 55.

Figure 55. Pseudo Code: Application Accelerator Chain Resume Initialization

```
; Set up descriptor in Intel® XScale™ core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0x0000 0000/* Source address of Data Block 1     */
d.SAR2 = 0x0000 0000/* Source address of Data Block 2     */
d.SAR3 = 0x0000 0000/* Source address of Data Block 3     */
d.SAR4 = 0x0000 0000/* Source address of Data Block 4     */
d.DAR  = 0x0000 0000/* Destination address of XOR-ed data */
d.ABCR = 0x0       /* Byte Count of zero */
d.ADCR = 0x000 0000/* Null Descriptor, No Interrupt*/

; Start operation
ANDAR - &d ; Setup descriptor address
ACR = 0x0000 0001 ; Set AA Enable bit
```

7.9.2 Suspend Application Accelerator

The Application Accelerator unit provides the ability to suspend the current state without losing status information. The AA resumes without requiring application software to save the current configuration. The example shown in Figure 56 describes pseudo-code for suspending the ongoing operation and then restarting.

Figure 56. Pseudo Code: Suspend Application Accelerator

```
;Suspend Application Accelerator
ACR = 0x0000 0000  ; Suspend ongoing AA transfer

;Restart Application Accelerator
ACR = 0x0000 0001  ; Restart AA operation
```

7.9.3 Appending Descriptor for XOR Operations

The example shown in Figure 57 describes the pseudo code for initiating an XOR operation with the AA. The examples illustrates appending the XOR operation to an existing chain, and taking advantage of the Chain Resume capability as described

Figure 57. Pseudo Code: XOR Transfer Operation

```
; Set up descriptor in Intel® XScale™ core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Source address of Data Block 1     */
d.SAR2 = 0xA000 0800/* Source address of Data Block 2     */
d.SAR3 = 0xA000 0C00/* Source address of Data Block 3     */
d.SAR4 = 0xA000 1000/* Source address of Data Block 4     */
d.DAR  = 0xB000 0100/* Destination address of XOR-ed data */
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x8000 049F/* Direct fill data from Block 1      */
                   /* XOR with data from Block 2,Block 3 and
                      Block 4                            */
                   /* Store the result and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

7.9.4 Appending Descriptor for Memory Block Fill Operations

The example shown in Figure 58 describes the pseudo code for initiating a Memory Block Fill operation with the AA.

Figure 58. Pseudo Code: Memory Block Fill Operation

```
; Set up descriptor in Intel® XScale™ core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Immediate data used for block write*/
d.DAR  = 0xB000 0100/* Address of the memory block to be written*/
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x8000 0005/* Memory Write Block using data in D/SAR1*/
                   /* Store the result and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

7.9.5 Appending Descriptor for Zero Result Buffer Check

The example shown in Figure 59 describes the pseudo code for initiating an XOR operation with the AA.

Figure 59. Pseudo Code: Zero Result Buffer Check Operation[a]

```
; Set up descriptor in Intel® XScale™ core local memory at address d
d.nda  = 0         /* No chaining */
d.D/SAR1 = 0xA000 0400/* Source address of Data Block 1     */
d.SAR2 = 0xA000 0800/* Source address of Data Block 2     */
d.SAR3 = 0xA000 0C00/* Source address of Data Block 3     */
d.SAR4 = 0xA000 1000/* Source address of Data Block 4     */
d.ABCR = 1024      /* Byte Count of 1024 */
d.ADCR = 0x4000 049F/* Direct fill data from Block 1      */
                   /* XOR with data from Block 2,Block 3 and
                      Block 4                             */
                   /* Check Result, Write Status (ADCR) and interrupt processor */

; Append descriptor to end of last chain at address c
c.nda = d

; Resume AA operation
ACR = 0x00000003   ; Set AA Enable and Resume bits
```

a. Notice that ADCR.dwe is cleared and that the DAR is not programmed. The reasons is that for Zero Result Buffer Check operations, there is no need to write out a destination parity stripe.

7.10 Interrupts

The Application Accelerator can generate an interrupt to the 80331. The *Interrupt Enable* bit in the Accelerator Descriptor Control Register (ADCR.ie) determines whether the AA generates an interrupt upon successful, error-free completion. Error conditions described in Section 7.11 also generate an interrupt. The AA has one interrupt output connected to the PCI and Peripheral Interrupt Controller described in Chapter 15, "Interrupt Controller Unit".

Once the AA is enabled, the AA loads the chain descriptor fields into the respective registers. A special case exists when data write enable is clear, then an interrupt is generated (when enabled) after the descriptor is fetched and processed as defined by the block control fields in the ADCR. Table 200 summarizes the status flags and conditions when interrupts are generated in the Accelerator Status Register (ASR).

Table 200. AA Interrupts

Interrupt Condition	Accelerator Status Register (ASR) Flags				Interrupt Generated?	
	Active	End of Transfer	End of Chain	IB Master Abort	Interrupt Enabled	Interrupt Disabled
(Data Write Enable == 0 || byte count == 0) && (ANDAR != NULL || Resume == 1) (End of Transfer)	1	1	0	0	Y	N
(Data Write Enable == 0 || byte count == 0) && ANDAR == NULL && Resume == 0 (End of Chain)	0	0	1	0	Y	N
IB Master Abort	0	0	0	1	Y	Y
IB Target Abort	0	0	0	0	N	N

Note: End-of-Transfer and End-of-Chain flags is set only when Interrupt Enable is set. When Interrupt Enable is clear, then the above flags are always set to 0. End-of-Transfer Interrupt and End of Chain Interrupt can only be reported in the ASR when the descriptor fetch and processing completed without any reportable errors. However, multiple error conditions may occur and be reported together. Also, because the AA does not stop after reporting the End-of-Transfer interrupt, an IB master-abort error may occur before the End-of-Transfer interrupt is serviced and cleared.

7.11 Error Conditions

Master Aborts that occur during a transfer are recorded by the Application Accelerator.

When an error occurs, the actions taken are detailed below:

- The AA shall cease the ongoing transfer for the current chain descriptor and clear the *Application Accelerator Active* flag in the ASR.
- The AA does not read any new chain descriptors.
- The AA sets the error flag in the Accelerator Status Register. For example; when an IB master-abort occurred during a transfer, the channel sets bit 5 in the ASR.
- The AA signals an interrupt to the Intel® XScale™ core.
- The Application Accelerator does not restart the transfer after an error condition. It is the responsibility of the application software to reconfigure the AA to complete any remaining transfers.

Note: Target-aborts during AAU reads result from multi-bit ECC errors that are recorded by the MCU. Refer to Chapter 8, "Memory Controller" for details on error handling in this instance. For correct operation of the AAU, user software has to disable the AAU before clearing the error condition. Further, the AAU needs to be re-enabled by writing a 1 to the AA Enable bit before initiating a new operation.

7.12 Power-up/Default Status

Upon power-up, an external hardware reset, the Application Accelerator Registers is initialized to their default values.

7.13 Register Definitions

The Application Accelerator Unit contains forty two memory-mapped registers for controlling its operation. There is read/write access only to the Accelerator Control Register, Accelerator Status Register, the Accelerator Next Descriptor Address Register, and the three Extended Descriptor Control Registers. All other registers are read-only and are loaded with new values from the chain descriptor whenever the AA reads a chain descriptor from memory.

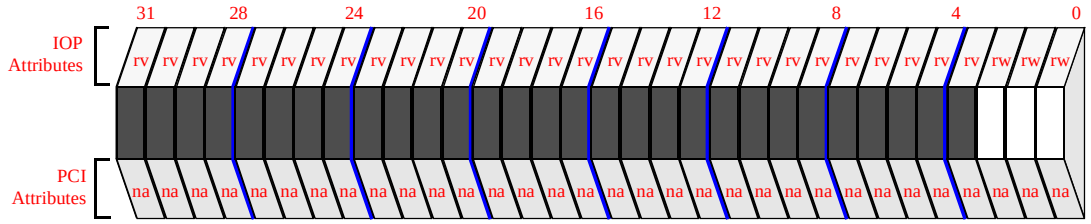
Table 201. Application Accelerator Unit Registers

Section, Register Name - Acronym (page)
Section 7.13.1, "Accelerator Control Register - ACR" on page 415
Section 7.13.2, "Accelerator Status Register - ASR" on page 416
Section 7.13.3, "Accelerator Descriptor Address Register - ADAR" on page 417
Section 7.13.4, "Accelerator Next Descriptor Address Register - ANDAR" on page 418
Section 7.13.5, "Data / Source Address Register1 - SAR1" on page 419
Section 7.13.6, "Source Address Register2..32 - SAR2..32" on page 420
Section 7.13.7, "Destination Address Register - DAR" on page 421
Section 7.13.8, "Accelerator Byte Count Register - ABCR" on page 422
Section 7.13.9, "Accelerator Descriptor Control Register - ADCR" on page 423
Section 7.13.10, "Extended Descriptor Control Register 0 - EDCR0" on page 427
Section 7.13.11, "Extended Descriptor Control Register 1 - EDCR1" on page 430
Section 7.13.12, "Extended Descriptor Control Register 2 - EDCR2" on page 433

7.13.1 Accelerator Control Register - ACR

The Accelerator Control Register (ACR) specifies parameters that dictate the overall operating environment. The ACR should be initialized prior to all other AA registers following a system reset. Table 202 shows the register format. This register can be read or written while the AA is active.

Table 202. Accelerator Control Register - ACR



Internal bus address
FFFF E800H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

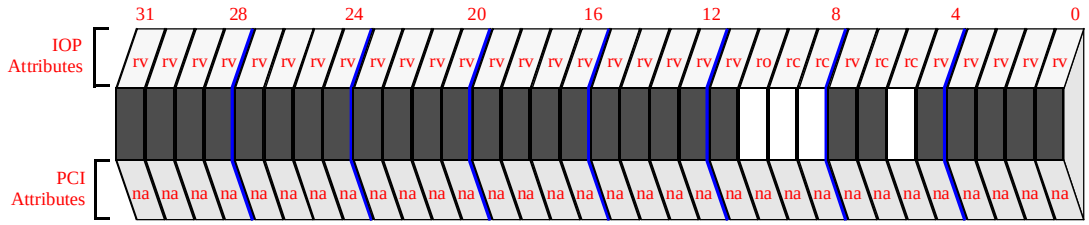
Bit	Default	Description
31:03	0	Reserved
02	0_2	512 Byte Buffer Enable - when set, causes the AA to use only 512 bytes of 1 KB data buffer while processing all descriptors.
01	0_2	Chain Resume - when set, causes the AA to resume chaining by re-reading the current descriptor located at the address in the Accelerator Descriptor Address Register when the AA is idle (AA Active bit in the ASR is clear) or when the AA completes a transfer. This bit is cleared by hardware when either: • The AA completes a transfer and the Accelerator Next Descriptor Address Register is non-zero. In this case, the AA proceeds to the next descriptor in the chain. • The AA re-reads the chain descriptor located at the address in the Accelerator Descriptor Address Register and loads the Next Descriptor Address of that descriptor into the Accelerator Next Descriptor Address Register
00	0_2	AA Enable - When set, the AA enables transfers. When clear, the AA disables any transfer. Clearing this bit when the AA is active suspends the current transfer at the earliest opportunity by halting all internal bus transactions. The AA does not initiate any new transfers when this bit is cleared. Data held in queues remains valid. Setting the bit after the AA is suspended causes the AA to resume the previously ongoing transfer.

7.13.2 Accelerator Status Register - ASR

The Accelerator Status Register (ASR) contains status flags that indicate status. This register is typically read by software to examine the source of an interrupt. See Section 7.11 for a description of the error conditions that are reported in the ASR. See Section 7.10 for a description of interrupts caused by the Application Accelerator.

When an AA error occurs, application software should check the status of Accelerator Active flag before processing the interrupt.

Table 203. Accelerator Status Register - ASR



Internal bus address
FFFF E804H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

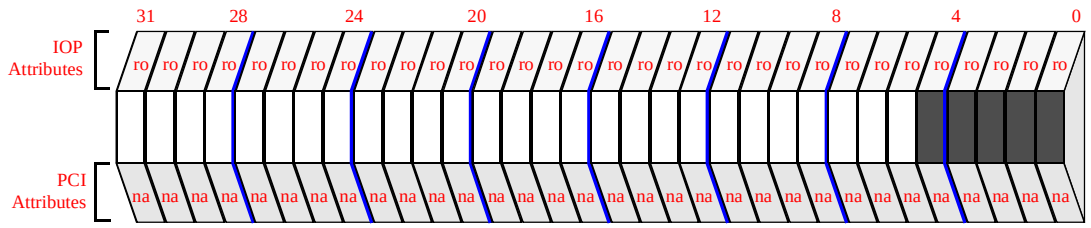
Bit	Default	Description
31:11	000000H	Reserved
10	0_2	Accelerator Active Flag - indicates the AA is either active (in use) or idle (available). When set, indicates the AA is in use and actively performing an operation. When clear, indicates the channel is idle and available to be configured for a new operation. The AA clears the Accelerator Active flag when the previously configured transfer completes as a result of: • byte count reached zero and last chain descriptor is encountered (NULL value detected for Next Descriptor Address in chain descriptor) • Internal Bus Errors • Last chain descriptor is processed (NULL value detected for Next Descriptor Address in chain descriptor) and write enable is zero. The Accelerator Active flag is set once a Chain Descriptor is read from memory.
09	0_2	End of Transfer Interrupt Flag - set when the AA has signalled an interrupt to the 80331 after processing a descriptor but it is not the last descriptor in a chain.
08	0_2	End of Chain Interrupt Flag - set when the channel has signalled an interrupt to the 80331 after processing a descriptor that is the last in a chain.
07:06	0_2	Reserved
05	0_2	This bit is set when a Master-abort occurs during a transaction when the AAU is the master on the internal bus.
04:00	0_2	Reserved

7.13.3 Accelerator Descriptor Address Register - ADAR

The Accelerator Descriptor Address Register (ADAR) contains the address of the current chain descriptor in local memory. This read-only register is loaded when a new chain descriptor is read. Table 204 depicts the Accelerator Descriptor Address Register. Depending on the number of sources, the chain descriptors are required to be aligned on different address boundaries. These include four sources on an eight word address boundary, eight sources on a 16 word address boundary, 16 sources on a 32 word address boundary, and 32 sources on a 64 word address boundary.

Note: In the above paragraph, the term "word" refers to a DWORD.

Table 204. Accelerator Descriptor Address Register - ADAR



Internal bus address
FFFF E808H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:05	00000000000 00000000000 00000_2	Current Descriptor Address - local memory address of the current chain descriptor read by the Application Accelerator.
04:00	00000_2	Reserved

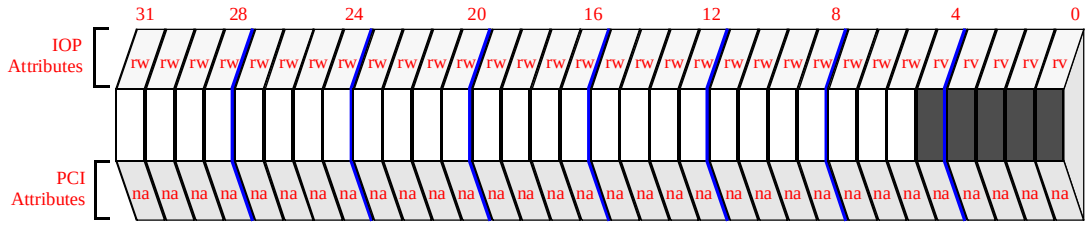
7.13.4 Accelerator Next Descriptor Address Register - ANDAR

The Accelerator Next Descriptor Address Register (ANDAR) contains the address of the next chain descriptor in local memory. When starting a transfer, this register contains the address of the first chain descriptor. Table 205 depicts the Accelerator Next Descriptor Address Register.

All chain descriptors are aligned on an eight DWORD boundary. The AA may set bits 04:00 to zero when loading this register.

Note: The *Accelerator Enable* bit in the ACR and the *Accelerator Active* bit in the ASR must both be clear prior to writing the ANDAR. Writing a value to this register while the AA is active may result in undefined behavior.

Table 205. Accelerator Next Descriptor Address Register - ANDAR



Internal bus address
FFFF E80CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:05	00000000000 00000000000 00000_2	Next Descriptor Address - local memory address of the next chain descriptor to be read by the Application Accelerator.
04:00	00000_2	Reserved

7.13.5 Data / Source Address Register1 - SAR1

The Data / Source Address Register (D/SARx) contains a 32-bit, local memory address or immediate data to be written in case of Memory Block Fill operations. The ADCR register (Table 210) controls the operation performed on data block referenced by this register. The local memory address space is a 32-bit, byte addressable address space.

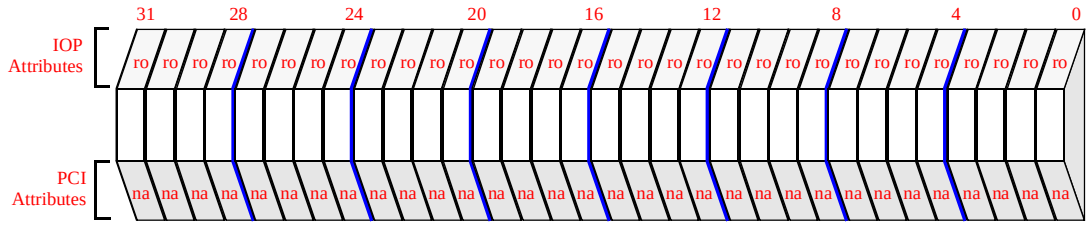
Reading the D/SAR1 register once the AA has started a chain descriptor returns the current source address or immediate data to be written in case of Memory Block Fill operations.

Once an operation is initiated, these registers contain the current source addresses. For example; when the Byte Count is initially 4096 bytes and the AA has completed the operation on the first three 1K-byte data blocks, the value in register SAR1 is the equal to the programmed descriptor value + 3072 (SAR1 + 3072).

During Memory Block Fills the register always contains the data to be written and does not change.

Table 206 shows the Data / Source Address Register1. This read-only register is loaded when a chain descriptor is read from memory.

Table 206. Data / Source Address Register - SAR1



Internal bus address
SAR1 FFFF E810H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	For the XOR command - Local Address - The local source address. For the Memory Block Fill Command - Data to be written to the memory block.

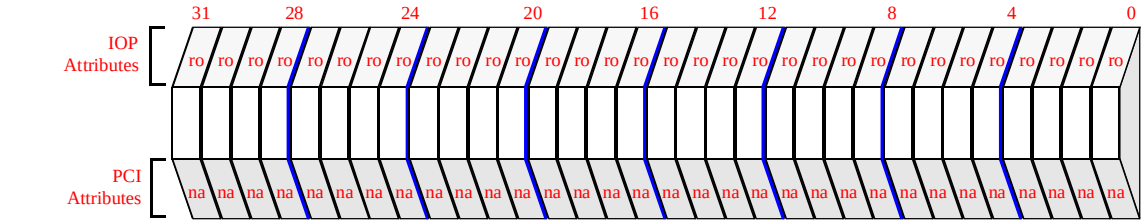
7.13.6 Source Address Register2..32 - SAR2..32

The 80331 Source Address Register2..32 (SAR2..32) contains a 32-bit, local memory address. There are 31 Source Address Registers (SAR2 - SAR32). Each of these registers is loaded with address of blocks of data to be operated upon by the AA. The ADCR, EDCR0, EDCR1, and EDCR2 registers control the operation performed on each data block referenced by the registers (SAR2 - SAR32). The local memory address space is a 32-bit, byte addressable address space.

Reading SARx registers once AA has started a chain descriptor returns the current source addresses. Once an operation is initiated, these registers contain current source addresses. For example; when Byte Count is initially 4096 bytes and AA has completed operation on the first three 1K-byte data blocks, the value in register SARx is the equal to the programmed descriptor value + 3072 (SARx + 3072).

Table 206 shows the Source Address Register2..32. These read-only registers are loaded when a chain descriptor is read from memory.

Table 207. Source Address Register2..32 - SAR2..32



Internal bus address	
SAR2	FFFF E814H
SAR3	FFFF E818H
SAR4	FFFF E81CH
SAR5	FFFF E82CH
SAR6	FFFF E830H
SAR7	FFFF E834H
SAR8	FFFF E838H
SAR9	FFFF E840H
SAR10	FFFF E844H
SAR11	FFFF E848H
SAR12	FFFF E84CH
SAR13	FFFF E850H
SAR14	FFFF E854H
SAR15	FFFF E858H
SAR16	FFFF E85CH
SAR17	FFFF E864H
SAR18	FFFF E868H
SAR19	FFFF E86CH
SAR20	FFFF E870H
SAR21	FFFF E874H
SAR22	FFFF E878H
SAR23	FFFF E87CH
SAR24	FFFF E880H
SAR25	FFFF E888H
SAR26	FFFF E88CH
SAR27	FFFF E890H
SAR28	FFFF E894H
SAR29	FFFF E898H
SAR30	FFFF E89CH
SAR31	FFFF E8A0H
SAR32	FFFF E8A4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	Local Address - The local source address.

7.13.7 Destination Address Register - DAR

The Destination Address Register (DAR) contains a 32-bit, local memory address.

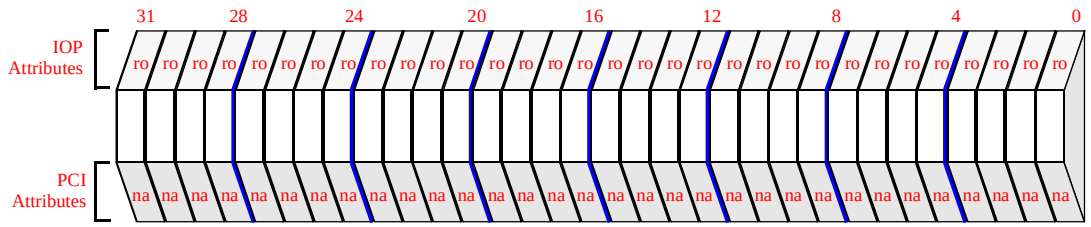
During operations, this address is the destination address in local memory where data is stored. The 80331 local memory address space is a 32-bit, byte addressable address space.

During Memory Block Fill operations, this address points to the memory block to be written with the constant value contained in the D/SAR1 register.

Reading the DAR once the AA has started a chain descriptor returns the current destination address. For example; during an XOR operation when the Byte Count is initially 4096 bytes and the AA has completed the *XOR-transfer* operation on the first three 1K-byte data blocks, the value in the Destination Address Register (DAR) is the equal to the programmed descriptor value + 3072 (DAR + 3072).

Table 208 shows the Destination Address Register. This read-only register is loaded when a chain descriptor is read from memory

Table 208. Destination Address Register - DAR



Internal bus address
FFFF E820H

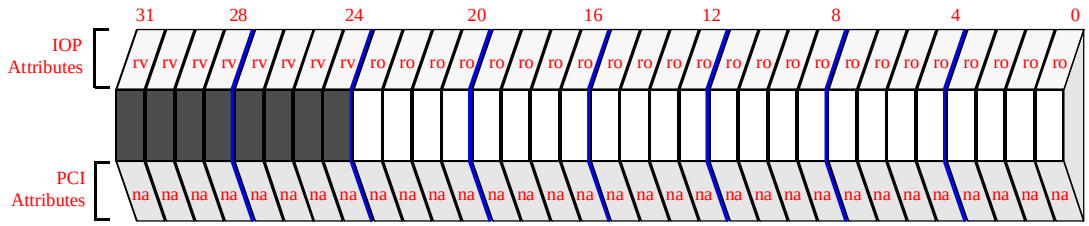
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	Local Address - The local destination address.

7.13.8 Accelerator Byte Count Register - ABCR

The Accelerator Byte Count Register (ABCR) contains the number of bytes to transfer for an operation. This is a read-only register that is loaded from the Byte Count word in a chain descriptor. It allows for a maximum transfer of 16 Mbytes. A value of zero is a valid byte count and results in no read or write cycles being generated to the Memory Controller Unit. No cycles are generated on the internal bus.

Table 209. Accelerator Byte Count Register - ABCR



Internal bus address
FFFF E824H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

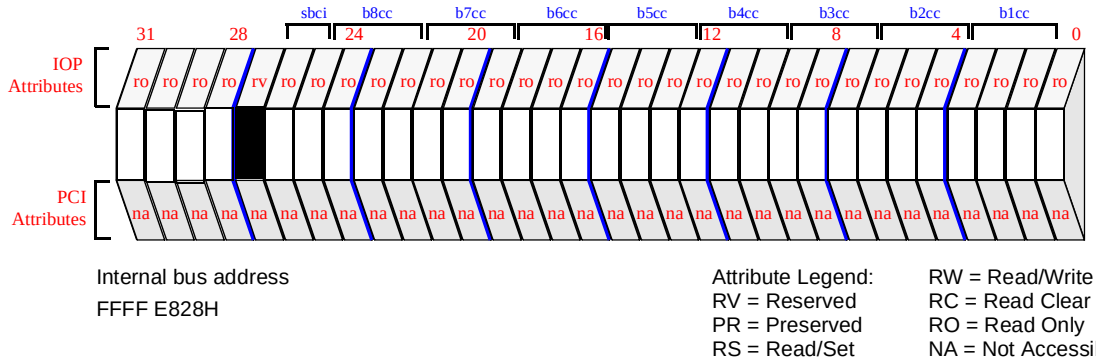
Bit	Default	Description
31:24	00H	Reserved
23:00	000000H	Byte Count - is the number of bytes to transfer for an operation.

Note: Anytime this register is read by the 80331, it contains the number of bytes left to transfer on the internal bus. Note that during an operation valid data may be present in the Application Accelerator store queue. This register is decremented by 1 through 8 for every successful transfer from the store queue to the destination location. During *Memory Block Fills* this register decremented by 1 through 8 for every successful write operation. Table 209 shows the Accelerator Byte Count Register. The byte count value is not required to be aligned to a DWORD boundary (i.e., the byte count value can be a DWORD aligned, short aligned, or byte aligned).

7.13.9 Accelerator Descriptor Control Register - ADCR

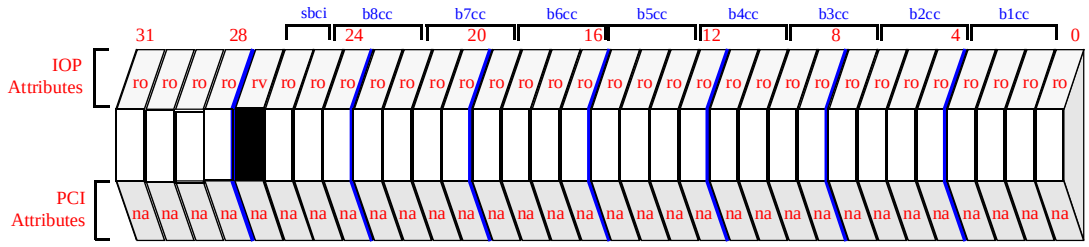
The Accelerator Descriptor Control Register contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor is read from memory. These values may vary from chain descriptor to chain descriptor. The AAU determines whether a mini-descriptor is appended to the end of the current chain descriptor by examining bits 26:25. Table 210 shows the definition of the Accelerator Descriptor Control Register.

Table 210. Accelerator Descriptor Control Register - ADCR (Sheet 1 of 4)



Bit	Default	Description
31	0_2	Destination Write Enable - Determines whether data present in the store queue is written out to local memory. When set, data in the queue is written to the address specified in the Destination Address Register (DAR) after performing the specified operation on data referenced by the SARx registers. When clear, data is held in the queue. **NOTE:** When the ABCR register contains a value greater than the buffer size and this bit is cleared, the AAU only reads the first buffer of bytes and perform the specified function. It does not read the remaining bytes specified in the ABCR. Further, the AAU proceeds to process the next chain descriptor when it is specified.
30	0_2	Zero Result Buffer Check Enable - When this bit is set the AA checks for an all-zero result buffer across the data blocks specified by the SARx registers.
29	0_2	Result Buffer Not Zero- This bit is set when the result buffer computed across the data blocks specified by the SARx registers results in a non-zero value. **NOTE:** The AA updates this status in memory **only** by updating the Descriptor Control Word of the current descriptor (the eighth word of the descriptor pointed to by the ADAR).
28	0_2	Transfer Complete - This bit is set when the AA completes the processing of a descriptor with Zero Result Buffer Check enabled (i.e., bit 30 of the ADCR is set). At the end of the transfer **NOTE:** The AA updates this status in memory **only** by updating the Descriptor Control Word of the current descriptor (the eighth word of the descriptor pointed to by the ADAR).
27	0_2	Reserved
26:25	00	Supplemental Block Control Interpreter - This bit field specifies the number of data blocks on which the operation is executed. 00 0 Blocks - This specifies that no additional data blocks exist. The AA does not read the mini-descriptor to initialize registers SAR5 - SAR8. 01 4 Blocks - This specifies that there are up to 4 additional data blocks. The AA therefore reads the mini-descriptor to initialize registers SAR5 - SAR8. 10 12 Blocks - This specifies that there are up to 12 additional data blocks. The AA therefore reads the mini-descriptor and one extended-descriptor to initialize registers SAR5 - SAR16. 11 28 Blocks - This specifies that there are up to 28 additional data blocks. The AA therefore reads the mini-descriptor and three extended-descriptors to initialize registers SAR5 - SAR32.

Table 210. Accelerator Descriptor Control Register - ADCR (Sheet 2 of 4)

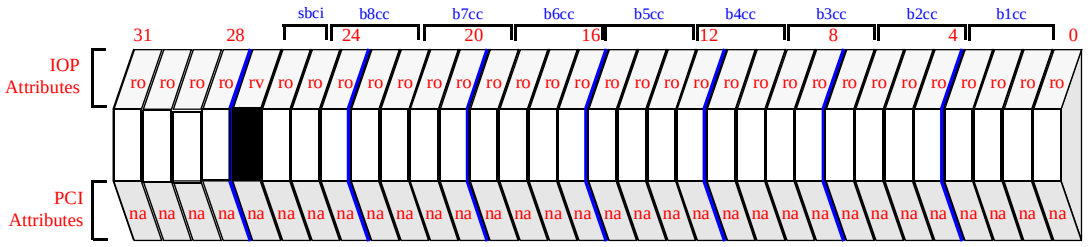


Internal bus address
FFFF E828H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
24:22	0	**Block 8 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR8 register. 000Null command - This implies that Block 8 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 8 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
21:19	0	**Block 7 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR7 register. 000Null command - This implies that Block 7 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 7 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
18:16	0	**Block 6 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR6 register. 000Null command - This implies that Block 6 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 6 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 210. Accelerator Descriptor Control Register - ADCR (Sheet 3 of 4)

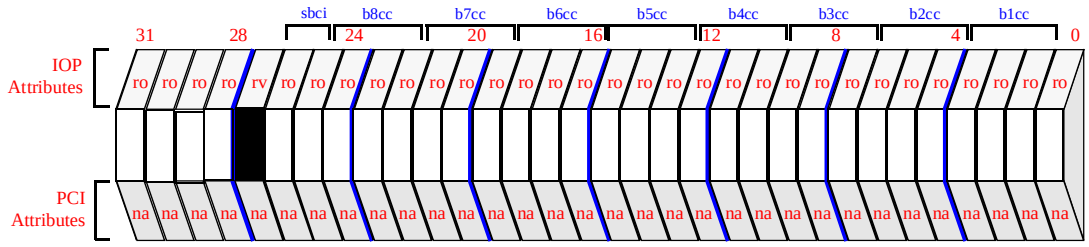


Internal bus address
FFFF E828H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
15:13	0	**Block 5 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR5 register. 000Null command - This implies that Block 5 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 5 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
12:10	0	**Block 4 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR4 register. 000Null command - This implies that Block 4 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 4 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
09:07	0	**Block 3 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR3 register. 000Null command - This implies that Block 3 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 3 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 210. Accelerator Descriptor Control Register - ADCR (Sheet 4 of 4)



Internal bus address
FFFF E828H

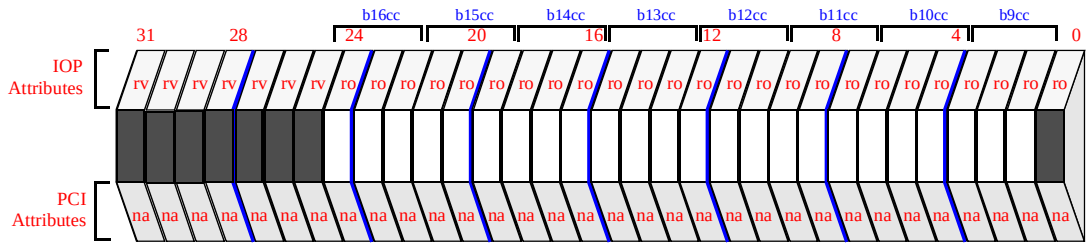
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
06:04	0	**Block 2 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR2 register. 000Null command - This implies that Block 2 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 2 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
03:01	0	**Block 1 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by D/SAR1 register (for XOR command) or with the data contained in the D/SAR1 (for Memory Block Fill command). 000Null command - This implies that Block 1 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 1 Data is transferred to the Application Accelerator to execute the XOR function. 010Memory Block Fill command - This implies that the memory block specified by the DAR is filled with the constant specified by the D/SAR1register. 011Reserved 100Reserved 101Reserved 110Reserved 111Direct Fill - This implies that Block 1 Data is transferred directly from local memory to the store queue.
00	0	**Interrupt Enable** - When set, the Application Accelerator generates an interrupt to the 80331 upon completion of a transfer. When clear, no interrupt is generated.

7.13.10 Extended Descriptor Control Register 0 - EDCR0

The Extended Descriptor Control Register 0 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires a minimum of 16 Source Addresses is read from memory. The values in EDCR0 define the command/control value for SAR16 - SAR9. The AAU determines whether an extended descriptor requiring the use of EDCR0 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 211 shows the definition of the Extended Descriptor Control Register 0.

Table 211. Extended Descriptor Control Register 0 - EDCR0 (Sheet 1 of 3)

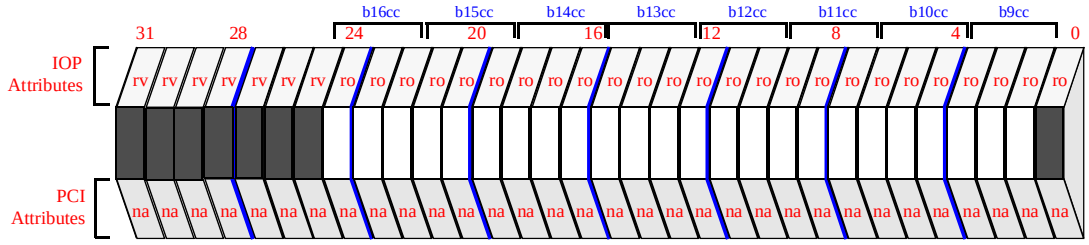


Internal bus address
FFFF E83CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 16 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR16 register. 000Null command - This implies that Block 16 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 16 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
21:19	0	**Block 15 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR15 register. 000Null command - This implies that Block 15 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 15 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 211. Extended Descriptor Control Register 0 - EDCR0 (Sheet 2 of 3)

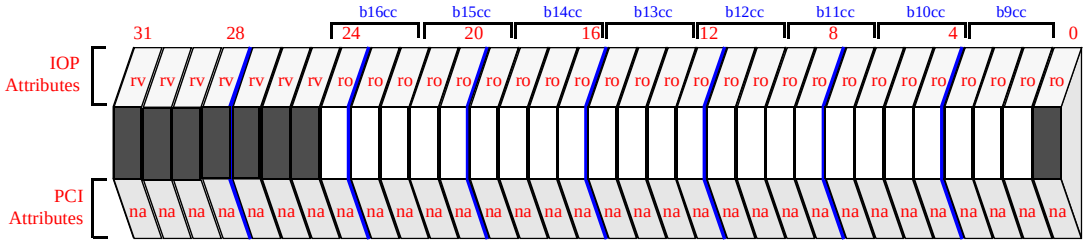


Internal bus address
FFFF E83CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
18:16	0	**Block 14 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR14 register. 000Null command - This implies that Block 14 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 14 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
15:13	0	**Block 13 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR13 register. 000Null command - This implies that Block 13 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 13 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
12:10	0	**Block 12 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR12 register. 000Null command - This implies that Block 12 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 12 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 211. Extended Descriptor Control Register 0 - EDCR0 (Sheet 3 of 3)



Internal bus address
FFFF E83CH

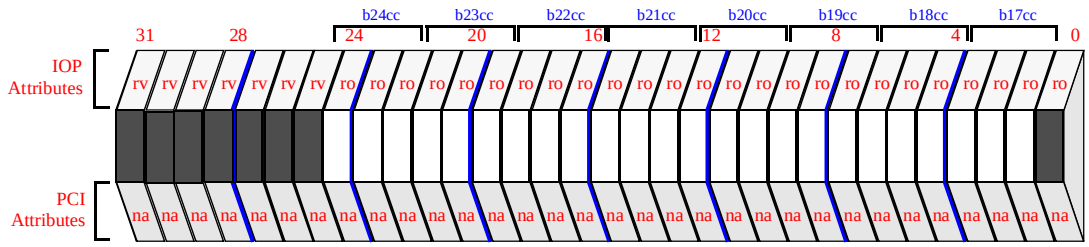
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
09:07	0	**Block 11 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR11 register. 000Null command - This implies that Block 11 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 11 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
06:04	0	**Block 10 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR10 register. 000Null command - This implies that Block 10 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 10 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
03:01	0	**Block 9 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR9 register. 000Null command - This implies that Block 9 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 9 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved.
00	0	Reserved.

7.13.11 Extended Descriptor Control Register 1 - EDCR1

The Extended Descriptor Control Register 1 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires 32 Source Addresses is read from memory. The values in EDCR1 define the command/control value for SAR24 - SAR17. The AAU determines whether an extended descriptor requiring the use of EDCR1 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 212 shows the definition of the Extended Descriptor Control Register 1.

Table 212. Extended Descriptor Control Register 1 - EDCR1 (Sheet 1 of 3)

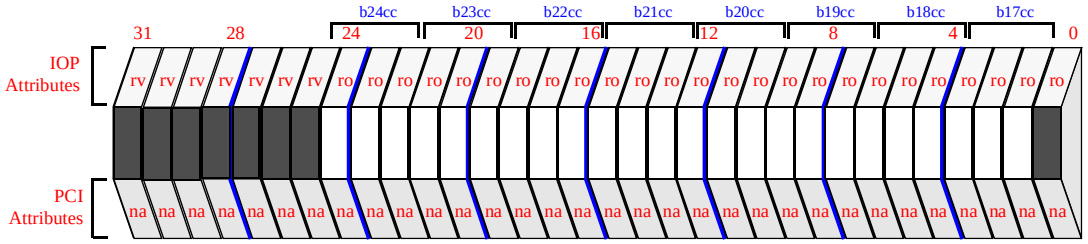


Internal bus address
FFFF E860H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 24 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR24 register. 000Null command - This implies that Block 24 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 24 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
21:19	0	**Block 23 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR23 register. 000Null command - This implies that Block 23 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 23 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 212. Extended Descriptor Control Register 1 - EDCR1 (Sheet 2 of 3)

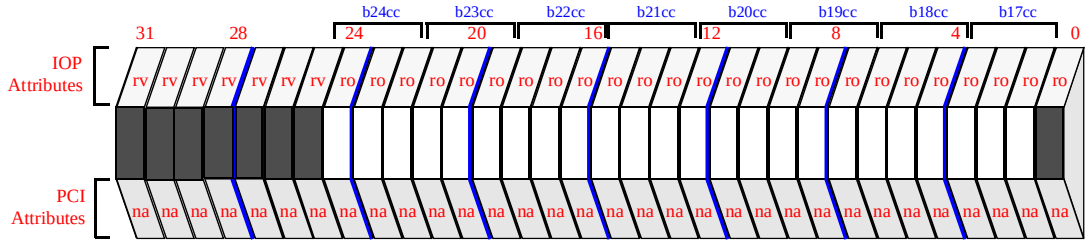


Internal bus address
FFFF E860H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
18:16	0	**Block 22 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR22 register. 000Null command - This implies that Block 22 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 22 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
15:13	0	**Block 21 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR21 register. 000Null command - This implies that Block 21 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 21 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
12:10	0	**Block 20 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR20 register. 000Null command - This implies that Block 20 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 20 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 212. Extended Descriptor Control Register 1 - EDCR1 (Sheet 3 of 3)



Internal bus address
FFFF E860H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

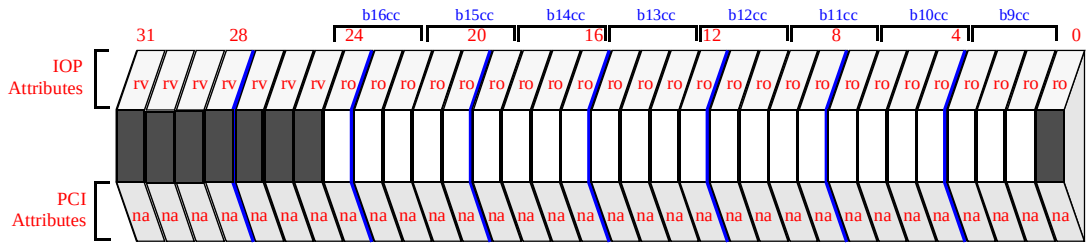
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
09:07	0	**Block 19 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR19 register. 000Null command - This implies that Block 19 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 19 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
06:04	0	**Block 18 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR18 register. 000Null command - This implies that Block 18 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 18 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
03:01	0	**Block 17 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR17 register. 000Null command - This implies that Block 17 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 17 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved.
00	0	Reserved.

7.13.12 Extended Descriptor Control Register 2 - EDCR2

The Extended Descriptor Control Register 2 contains control values for data transfer on a per-chain descriptor basis. This read-only register is loaded when a chain descriptor that requires 32 Source Addresses is read from memory. The values in EDCR2 define the command/control value for SAR32 - SAR25. The AAU determines whether an extended descriptor requiring the use of EDCR2 is appended to the end of the current chain descriptor by examining bits 26:25 of the Accelerator Descriptor Control Register. Table 213 shows the definition of the Extended Descriptor Control Register 2.

Table 213. Extended Descriptor Control Register 2 - EDCR2 (Sheet 1 of 3)

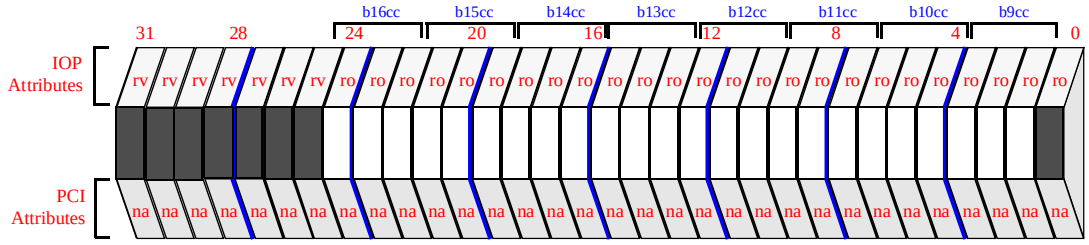


Internal bus address
FFFF E884H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0_2	Reserved
24:22	0	**Block 32 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR32 register. 000Null command - This implies that Block 32 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 32 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
21:19	0	**Block 31 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR31 register. 000Null command - This implies that Block 31 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 31 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 213. Extended Descriptor Control Register 2 - EDCR2 (Sheet 2 of 3)

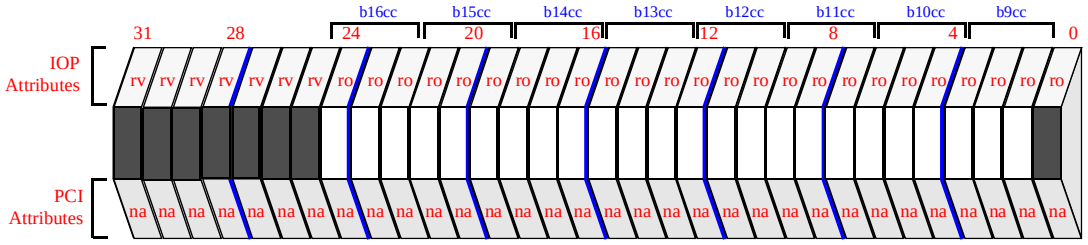


Internal bus address
FFFF E884H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
18:16	0	**Block 30 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR30 register. 000Null command - This implies that Block 30 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 30 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
15:13	0	**Block 29 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR29 register. 000Null command - This implies that Block 29 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 29 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
12:10	0	**Block 28 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR28 register. 000Null command - This implies that Block 28 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 28 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved

Table 213. Extended Descriptor Control Register 2 - EDCR2 (Sheet 3 of 3)



Internal bus address
FFFF E884H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
09:07	0	**Block 27 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR27 register. 000Null command - This implies that Block 27 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 27 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
06:04	0	**Block 26 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR26 register. 000Null command - This implies that Block 26 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 26 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved
03:01	0	**Block 25 Command Control** - This bit field specifies the type of operation to be carried out on the data pointed at by SAR25 register. 000Null command - This implies that Block 25 Data can be disregarded for the current chain descriptor. The Application Accelerator does not transfer data from this block while processing the current chain descriptor. 001XOR command - This implies that Block 25 Data is transferred to the Application Accelerator to execute the XOR function. 010Reserved 011Reserved 100Reserved 101Reserved 110Reserved 111Reserved.
00	0	Reserved.

This Page Intentionally Left Blank

Memory Controller 8

This chapter describes the integrated Memory Controller Unit (MCU). The operating modes, initialization, external interfaces, and implementation are detailed in this chapter.

8.1 Overview

The Intel® 80331 I/O processor (80331) integrates a high performance, multi-ported Memory Controller to provide a direct interface between the 80331 and its local memory subsystem. The Memory Controller supports:

- DDR 333 MHz SDRAM with 128/256/512 Mbit and 1Gbit density supported.
- DDR-II 400 MHz SDRAM with 256/512 Mbit density supported.
- Dedicated port for Intel® XScale™ core to DDR SDRAM.
- Between 64 MBytes and 2 GByte of 64-bit DDR SDRAM (1 GByte for DDR-II SDRAM).
- Between 32 MBytes and 1 GBytes of 32-bit DDR SDRAM for low cost solutions (512 MByte for DDR-II SDRAM).
- Optimized core processor data processing 32-bit region.
- Single-bit error correction, multi-bit detection support (ECC).
- 32-, 40- and 64-, 72-bit wide Memory Interfaces (non-ECC and ECC support).

The DDR SDRAM (DDR and DDR-II SDRAM) interface provides a direct connection to a high bandwidth and reliable memory subsystem. The DDR SDRAM interface consists of a 64-bit wide data path to support up to 3.2 GBytes/sec throughput. An 8-bit Error Correction Code (ECC) across each 64-bit word improves system reliability. The ECC is stored into the DDR 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 banks of DDR SDRAM in the form of one two-bank dual inline memory module (DIMM).

  — The MCU has support for Registered and Unbuffered DDR333 and Registered DDR-II 400 DIMMs.

- The MCU supports a 32-bit SDRAM data interface. This mode enables lower-cost solutions at the cost of system performance.
- The MCU responds to internal bus and core processor memory accesses within its programmed address range and issues the memory request to the DDR SDRAM interface.
- The MCU contains transaction queues for each port enabling pipelining of transactions to the DDR SDRAM for maximum performance.
- For 64-bit ECC memory, a 32-bit memory region can be programmed to operate as 32-bit ECC memory for higher performance core write performance by avoiding Read-Modify-Write (RMW) operation of DDR SDRAM.
- The MCU provides two chip enables to the memory subsystem. These two chip enables service the DDR SDRAM subsystem (one per bank).

8.2 Glossary

This section lists commonly used terms throughout this chapter:

Table 214. Commonly Used Terms

Term	Definition
Bank	A bank is defined as a memory region defined with a base register and a bank size register. Physically, a bank of memory is controlled by a single chip select. A DIMM could be comprised of a single or dual banks.
Column	A column refers to a portion of memory within an DDR SDRAM device. An DDR SDRAM device can be thought of as a grid with rows and columns. Once a row is activated, any column within that row can be accessed multiple times without reactivating the row. Columns are activated with CAS#.
DIMM	A DIMM is an acronym for Dual Inline Memory Module. A DIMM is a physical card comprising multiple DDR SDRAM devices. The card could be populated on one or both sides. A DIMM can be single or dual-bank.
Leaf	DDR SDRAM devices use multiple banks within the device operating in an interleaved mode. The MCU supports 128/256/512 Mbit, 1 Gbit DDR SDRAM and 256/512 Mbit DDR-II SDRAM devices containing four internal banks. An internal bank is defined as a leaf (to avoid confusion with a memory bank).
Page	A page is a row of memory. Once a row is activated, any column within that row can be accessed multiple times without reactivating the row. This is referred to as "keeping the page open." Page size depends on the DDR SDRAM device configuration, and the MCU supports the maximum possible page size (16 Kbytes for 64-bit wide memory and 8 Kbytes for 32-bit wide memory). The MCU breaks up pages across physical memory due to address mapping.
Row	A row refers to a portion of memory within an DDR SDRAM device. An DDR SDRAM device can be thought of as a grid with rows and columns. Once a row is activated, any column within that row can be accessed multiple times without reactivating the row. Rows are activated with RAS#.
Scrubbing	Once an error is detected within the memory array, the MCU must correct the error (if possible) while delivering the data to the initiator. Correcting the memory location is referred to as "scrubbing the array." The MCU relies on software to scrub any errors.
Syndrome	A syndrome is a value which indicates an error in the data read from the memory array. The MCU computes the syndrome with every memory read. Decoding the syndrome indicates: the bit in error for a single-bit error, or a multi-bit error. Table 231 defines the syndrome decoding.

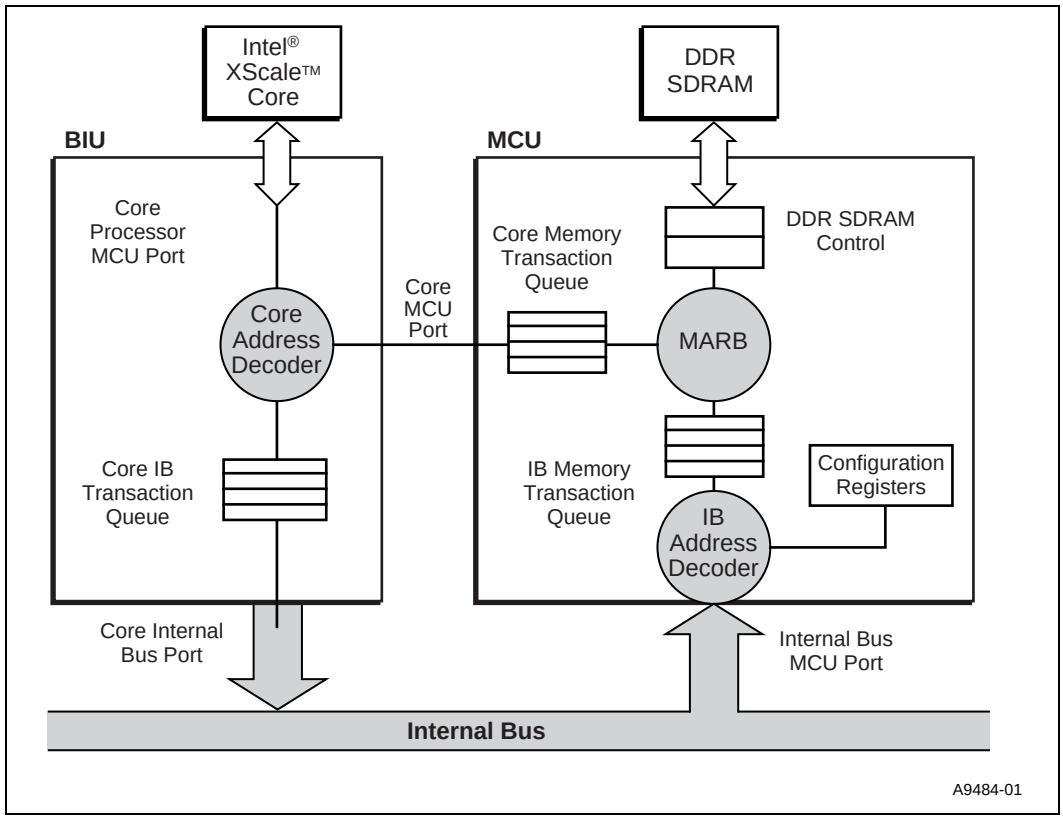
8.3 Theory of Operation

The 80331 memory controller translates the internal bus and core processor transactions into the protocol supported by the DDR SDRAM memory subsystem.

8.3.1 Functional Blocks

Note: The memory controller and Intel® XScale™ core processor bus interface unit (BIU) logically comprises the blocks illustrated in Figure 60. The memory controller is a multiported unit, supporting an inbound path for both the BIU and Internal Bus (IB) to the DDR SDRAM.

Figure 60. Memory Controller Block Diagram



8.3.1.1 Transaction Ports

Note: The MCU provides two transaction ports for DDR SDRAM access. They are:

8.3.1.1.1 Core Processor Port

The Core Processor Port provides a direct connection between the 80331 core processor's bus interface and the Memory Controller. This Core Processor Port allows core transactions targeting the DDR SDRAM to pass directly to the DDR SDRAM.

8.3.1.1.2 Internal Bus Port

The Internal Bus Port provides the connection to the DDR SDRAM from the Internal Bus. All peripheral unit transactions targeting the DDR SDRAM are claimed by this port.

8.3.1.2 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 SDRAM memory space and Peripheral Memory-Mapped Register (PMMR) space.

8.3.1.2.1 DDR SDRAM Memory Space

The DDR SDRAM memory space is defined with the DDR SDRAM Base Address Register (SDBR) and the DDR SDRAM Boundary Registers (SBR0, SBR1, S32SR). The transaction is intended for a DDR SDRAM bank if the address is between the base register (SDBR) and between the boundaries programmed with SBR0, SBR1 and S32SR as defined in Section 8.3.3.2, "DDR SDRAM Addressing" on page 452.

Note: For DDR SDRAM Bank 0 or Bank 1 overlapping PMMR space (FFFF E000H to FFFF FFFFH), DDR SDRAM is not accessible, and PMMRs are addressed. The read, write, and reserved characteristics of PMMR space is applicable as defined in Chapter 17, "Peripheral Registers".

8.3.1.2.2 Memory-Mapped Register Space

The MCU PMMR memory space is FFFF E500H to FFFF E5FFH and FFFF F500H to FFFF F5FFH. The registers are detailed in Section 8.7, "Register Definitions" on page 496.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.

8.3.1.2.3 Core Processor Port Address Decode

Note: The address decode block for the Core Processor transactions resides in the BIU. The Core MCU port therefore does not require any decode, and will process any transaction received from the BIU via the Core MCU Port.

8.3.1.2.4 Internal Bus Port Address Decode

Internal Bus transactions are decoded to determine if they address the DDR SDRAM Memory Space or MCU MMR Space. If the transaction addresses either of these two spaces, the transaction is claimed by the Internal Bus Port. If the transaction addresses the DDR 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.

8.3.1.3 Memory Transaction Queues

There are two transaction queues for transactions which address the MCU DDR Memory Space. One transaction queue for Core Processor transactions and one transaction queue for Internal Bus transactions.

8.3.1.3.1 Core Processor Memory Transaction Queue (CMTQ)

The CMTQ stores memory transactions from the core which have not been processed by the Memory Controller. The CMTQ supports 8 Core Processor read transactions up to 32-Bytes each, which equals the maximum number of outstanding transaction the Core Processor Bus Controller can support. The CMTQ also supports 8 Core Processor posted write transactions up to 16-Bytes each.

The number of CMTQ write transactions is selectable between 8 and 2, where the lower number may find benefit in some applications by throttling back on the core processor, and thereby 'encouraging' coalescing in the Core Processor Bus Controller.

8.3.1.3.2 Internal Bus Memory Transaction Queue (IBMTQ)

Note: The IBMTQ stores memory transactions from the Internal Bus that address the DDR SDRAM Memory Space. The IBMTQ can hold 8 outstanding Internal Bus read transaction requests. Of the 8 read transactions in the IBMTQ, 4 can be processed at a time, each with up to 1KBytes of read data. The other 4 read transactions are simply stored in the IBMTQ. The IBMTQ also supports 4 posted write transactions up to 1KBytes each.

8.3.1.4 Configuration Registers

Note: The Configuration Registers block contains all of the memory-mapped registers listed in Section 8.7, "Register Definitions" on page 496. These registers define the memory subsystem connected to the 80331. The status registers indicate the current MCU status.

8.3.1.5 Refresh Counter

The Refresh Counter block keeps track of when the DDR SDRAM devices need to be refreshed. The refresh interval is programmed in the "Frequency Register - RFR" on page 512. Once the 12-bit refresh counter reaches the programmed interval, the DDR SDRAM state machine issues a refresh command to the DDR SDRAM devices. If a transaction is currently in progress, the DDR SDRAM State Machine waits for the completion of the transaction to issue the refresh cycle. See Section 8.3.3.12, "DDR SDRAM Refresh Cycle" on page 474 for more details.

Note: If the memory controller is completing transactions from the core processor memory transaction queue (CMTQ) when the refresh counter expires, the MARB will complete the CMTQ tenure based on the "MCU Port Transaction Count Register - MPTCR" on page 510, before any refresh cycles are issued.

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.

8.3.1.6 Memory Controller Arbiter (MARB)

The MARB determines what transaction to issue next to the DDR SDRAM Control Unit. The MARB is configurable to adjust the selection of transactions from the MCU ports and the refresh counter. The MARB selects transactions from all memory transaction queues based on a programmable arbitration scheme as described in Section 8.3.2, "MCU Arbitration and Configuration" on page 445.

8.3.1.7 DDR SDRAM Control Block

The DDR SDRAM Control Block contains all functionality to process the DDR SDRAM data accesses per the transactions issued by the MARB. To process a transaction the DDR SDRAM Control Block employs several sub-blocks. The sub-blocks include the Page Control block, DDR SDRAM State Machine and Pipeline Queues, and Error Correction Logic.

8.3.1.7.1 Page Control Block

The Page Control Block records and maintains the open DDR SDRAM pages. The MCU can keep a maximum of eight pages open simultaneously (4 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 8.3.3.5, "Page Hit/Miss Determination" on page 458.

8.3.1.7.2 DDR 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 SDRAM state machine pipelines DDR SDRAM memory operations for several clocks.

8.3.1.7.3 Error Correction Logic

The Error Correction Logic generates the ECC code for DDR 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 which bit 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 8.3.4, "Error Correction and Detection" on page 476 for more details.

8.3.2 MCU Arbitration and Configuration

The order transactions are processed by the DDR SDRAM Control block is determined by the Memory Controller Arbiter (MARB). The MARB selects the next transaction from the memory transaction queues and refresh control based on the settings programmed in the MARB configuration registers ("MCU Port Transaction Count Register - MPTCR" on page 510, "MCU Preemption Control Register - MPCR" on page 511). These registers enable the MCU to be tuned for optimal performance for the design.

8.3.2.1 MCU Port Priority

Note: The Memory controller for the 80331 has only two inbound ports for memory transactions. Given this, there is no priority programming required, and the MARB will effectively toggle tenure between the two ports when both ports' memory transaction queues have pending transactions.

8.3.2.2 MCU Port Transaction Count

The MCU Port Transaction Count Register (MPTCR) defines the number of transactions the MARB will select from a given port during the corresponding port's tenure. The intent is that ports with frequent small transactions (core processor port) be allowed to complete multiple transactions during a tenure, while ports with less frequent large transactions (IB port) be limited to a small number of transactions (possibly just 1).

8.3.2.3 Core Processor Port Preemption

In order to maximize core processor performance, the MARB can be programmed to interrupt active transactions from other memory transaction queues, in order to process transactions from the Core Processor. By preempting an active transaction, the MARB guarantees a maximum number of data cycles getting processed for the current transaction, therefore reducing the possible latency the Core Processor might see for memory accesses. Core processor preemption is enabled through the MARB Preemption Control Register (MPCR).

Note: Preemption control applies to all inbound MCU ports based on pending core transactions. Once the core processor transaction queue is 'activated' by the MARB, a standard tenure will take place. At the completion of the Core Processor transaction queue tenure, the MARB resumes the previously active transaction. The previously active transaction may be interrupted again, provided the conditions for preemption occur again prior to the completion of that transaction. No other effect on the MARB processing will occur.

8.3.2.4 Core Port Transaction Ordering

The Core port maintains order of Intel® XScale™ core requests addressing the DDR SDRAM. Coherency between the core port and the IB port is maintained by the MCU as described in Section 8.3.2.6, "MCU Port Coherency" below.

8.3.2.5 IB Port Ordering

The IB port implements ordering of like transactions where reads do not pass reads, and writes do not pass writes. Write transactions are allowed to pass read transactions. Read transactions are processed by the MCU as described in Section 8.3.2.6, "MCU Port Coherency" below.

8.3.2.6 MCU Port Coherency

With the queueing of DDR SDRAM transactions in multiple ports, coherency of memory must be maintained. The MARB maintains memory coherency by ensuring all writes to a given memory address are completed before any read to the same address is processed. This address comparison is done with a 1 KByte granularity.

The highest priority Read transaction is compared to all pending write transactions from both memory transaction queues.If a write transaction is pending for the same memory location (1 KByte granularity), the write is allowed to complete first, before the read transaction is processed. Also, to maintain ordering rules, all write transactions preceding the 'incoherent write' are also processed.

8.3.3 DDR SDRAM Memory Support

The 80331 memory controller supports one or two banks of DDR SDRAM. DDR SDRAM allows zero data-to-data wait-state operation. DDR SDRAM offers an extremely wide range of configuration options emerging from the SDRAMs internal interleaving and bursting capabilities.

The MCU supports both 32-bit and 64-bit data bus width memory implementations (with and without ECC). The data bus width is controlled by the DDR SDRAM Control Register. In addition, a 64-bit data bus width DDR SDRAM implementation can be configured to operate as if a region is 32-bits wide, providing higher performance when the core processor is processing data by eliminating any RMW cycle required for 4-Byte store ECC generation.

The MCU supports DDR SDRAM burst length of four. 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.

8.3.3.1 DDR SDRAM Interface

The DDR SDRAM interface signals generated by the memory controller unit are divided into two sections. The first section lists those signals for DDR SDRAM operation, and the second list those additional signals supporting DDR-II SDRAM.

The MCU SDRAM interface provides a flexible mix of combinations including:

Table 215. DDR SDRAM Memory Configuration Options

Data Bus Width	ECC Enabled	Maximum Throughput(1)
64 bit	Yes	3200 Mbyte/s
64 bit	No	3200 Mbyte/s
32 bit	Yes	1600 Mbyte/s
32 bit	No	1600 Mbyte/s

NOTE:
1. Based on DDR-II 400MHz SDRAM

Table 216 shows the DDR SDRAM interface signals.

Table 216. DDR SDRAM Interface Signals

Pin Name	Description
M_CK[2:0]	*DDR SDRAM Clock Out* - Three (positive lines) of the output clocks driven to the Unbuffered DIMMs supported by the 80331.Registered DIMMs will only use **M_CK[0]** which will drive the input to the on-DIMM PLL. Section 8.3.6, "DDR SDRAM Clocking" on page 487 describes the DDR SDRAM clocking strategy for both Unbuffered and Registered DIMMs.
M_CK[2:0]#	*DDR SDRAM Clock Out* - Three (negative lines) of the output clocks driven to the Unbuffered DIMMs supported by the 80331.Registered DIMMs will only use **M_CK[0]#** which will drive the input to the on-DIMM PLL. Section 8.3.6, "DDR SDRAM Clocking" on page 487 describes the DDR SDRAM clocking strategy for both Unbuffered and Registered DIMMs.
M_RST#	*DDR Registered DIMM Reset* - Used to re-initialize registered DIMM during a **P_RST#** assertion or whenever internal bus reset bit is asserted in the PCSR. PCI Configuration and Status Register - PCSR describes the internal bus reset sequence.
CKE[1:0]	*Clock enables* - One clock after **CKE[1:0]** is de-asserted, data is latched on **DQ[63:0]** and **CB[7:0]**. Burst counters within DDR SDRAM device are not incremented. De-asserting this signal places the DDR SDRAM in self-refresh mode. For normal operation, **CKE[1:0]** must be asserted.
CS[1:0]#	*Chip Select* - Must be asserted for all transactions to the DDR SDRAM device. One per bank.
BA[1:0]	*DDR SDRAM Bank Selects* - Controls which of the internal DDR SDRAM banks to read or write. **BA[1:0]** are used for all technology types supported.
MA[10]	*Address bit 10* - If high during a read or write command, auto-precharge occurs after the command. During a **row-activate** command, this bit is part of the address (Section 8.3.3.2). Auto-precharge is **not** supported by the 80331.
MA[13:0]	*Address bits 12 through 0* - Indicates the row or column to access depending on the state of RAS# **and** CAS# (Section 8.3.3.2).
DQ[63:0]	*Data Bus* - 64-bit wide data bus.
CB[7:0]	*ECC Bus* - 8-bit error correction code which accompanies the data on **DQ[63:0].**
DM[8:0]	*Data Bus Mask* - Controls the DDR SDRAM data input buffers. Asserting **WE#** causes the data on **DQ[63:0]** and **CB[7:0]** to be written into the DDR SDRAM devices. **DM[8:0]** controls this operation on a per byte basis.
DQS[8:0]	*Data Strobes* - Strobes that accompany the data to be read or written from the DDR SDRAM devices. Data is sampled on the negative and positive edges of these strobes.
WE#	*Write Strobe* - Defines whether or not the current operation by the DDR SDRAM is to be a read or a write.
RAS#	*Row Address Strobe* - Indicates that the current address on **MA[13:0]** is the row.
CAS#	*Column Address Strobe* - Indicates that the current address on **MA[13:0]** is the column.
VREF	*SDRAM Voltage Reference* - is used to supply the reference voltage to the differential inputs of the memory controller pins.

Table 217 shows the DDR-II SDRAM interface signals.

Table 217. DDR-II SDRAM Interface Signals

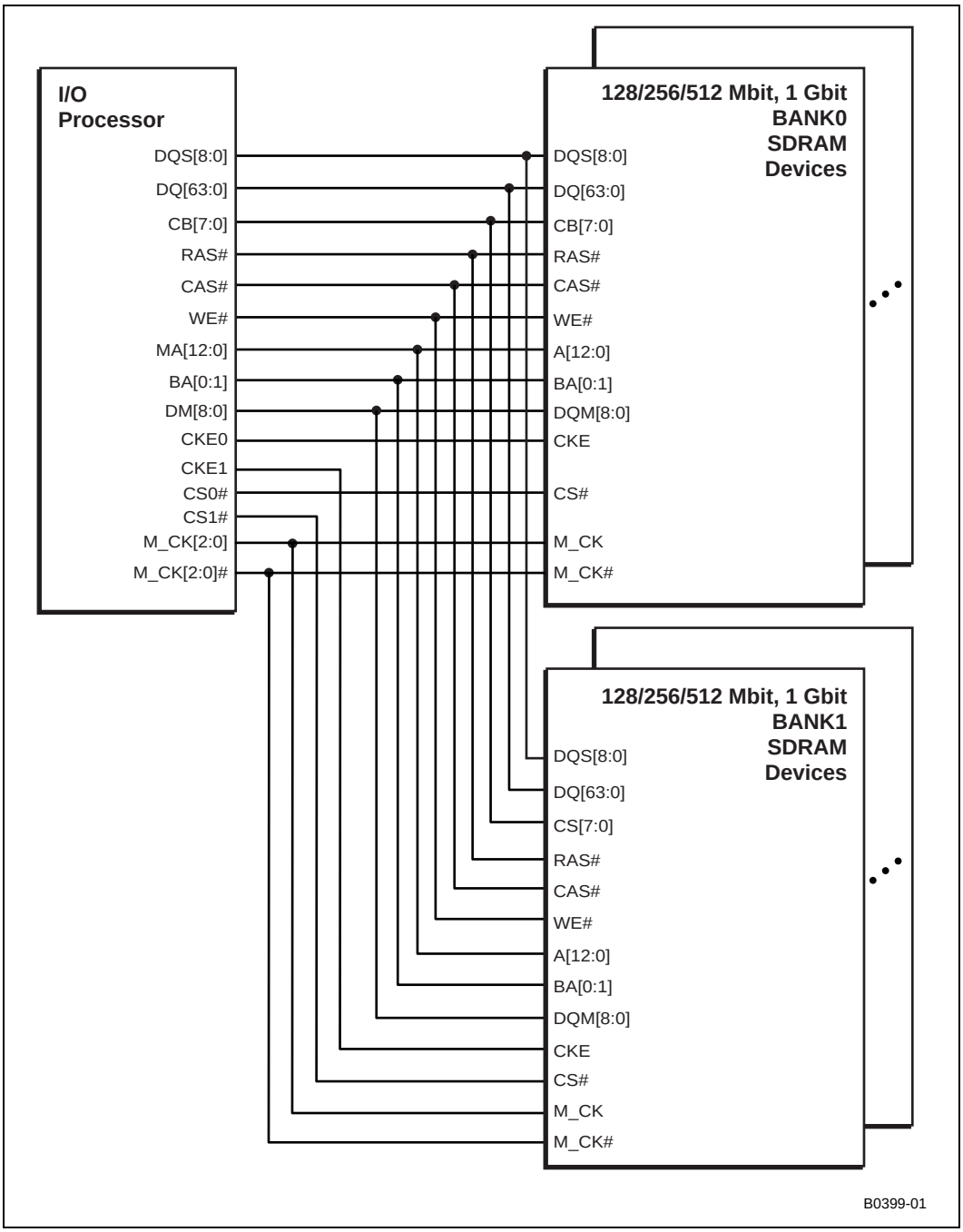
Pin Name	Description
DQS[8:0]#	*Data Strobes Differential*- Strobes that accompany the data to be read or written from the DDR SDRAM devices. Data is sampled on the negative and positive edges of these strobes.
DDRRES[2:1]	**COMPENSATION FOR DDR OCD** (analog) DDR2 mode only
ODT[1:0]	On Die Termination signals control - turns on SDRAM termination during writes

Utilizing the DDR SDRAM chip selects **CS[1:0]#** and internal bank selects **BA[1:0]**, the MCU keeps a maximum of eight pages open simultaneously. The number of available pages depends on the memory subsystem population. A single bank allows four pages and two banks allow eight pages.

Open pages allow optimal performance when a read or write occurs to an open page. Multiple open pages allow multiple memory segments to be open simultaneously and is well-suited for the 80331 system environment. The MCUs paging algorithm is detailed in Section 8.3.3.5, "Page Hit/Miss Determination" on page 458. The waveforms illustrating the performance issues are in Section 8.3.3.10, "DDR SDRAM Read Cycle" on page 468 and Section 8.3.3.11, "DDR SDRAM Write Cycle" on page 471.

Figure 61 illustrates how two banks of DDR SDRAM would interface with the 80331 through the MCU.

Figure 61. Dual-Bank DDR SDRAM Memory Subsystem



8.3.3.2 DDR SDRAM Addressing

MCU supports memory subsystem ranging from 64 Mbytes-2 Gbyte. An ECC or non-ECC system may be implemented using x8, or x16 devices and 32-bit or 64-bit data bus widths. Required information to configure the MCU are: row address size, column address size and size of DIMM module which can be read via the DIMM Serial Presence Detect (SPD).

Table 218 illustrates supported DDR SDRAM configurations. This table reflects the bank size associated with the respective row and column address size. The Bank size reflects 64-bit data bus width. The table also specifies the Page Size corresponding to each column address size.

Table 218. Supported DDR SDRAM Bank and Page Sizes[a]

Row Address Size (bits)	Column Address Size (bits)		
	9	10	11
12	64MB	128MB	**n/a**
13	128MB	256MB	512MB
14	n/a	512MB	**1GB**
Page Size	4KB	8KB	16KB

a. Table indicates 64-bit wide memory subsystem sizes. For 32-bit wide memory, memory subsystem and page size are half of the size indicated.

The supported SDRAM devices comprise four internal leaves. The MCU controls the leaf selects within the SDRAM by toggling **BA[0]** and **BA[1]**.

Table 219. Bank Address Decode

Bank Size[a]	Leaf Select	
	BA[1]	BA[0]
64M	**I_AD[25]**	**I_AD[24]**
128M	**I_AD[26]**	**I_AD[25]**
256M	**I_AD[27]**	**I_AD[26]**
512M	**I_AD[28]**	**I_AD[27]**
1G	**I_AD[29]**	**I_AD[28]**

a. Table indicates 64-bit wide memory bank sizes. For 32-bit wide memory, bank size is half of the size indicated.

Two DDR SDRAM chip enables (**CS[1:0]#**) support DDR SDRAM memory subsystem consisting of two banks. Base address for two contiguous banks are programmed in DDR SDRAM Base Register (SDBR) and must be aligned to a 32 Mbyte boundary. Size of each DDR SDRAM bank is programmed with DDR SDRAM boundary registers (SBR0 and SBR1). Bank 0 is configurable into multiple regions. By default bank 0 is a single 64-bit region. For higher core data processing performance, a 32-bit region can be defined within Bank 0 with the DDR SDRAM 32-bit Region Size Register (S32SR).

Table 220 is used to determine which DDR SDRAM address decode is used to program "SDRAM Boundary Register 0 - SBR0" and "SDRAM Boundary Register 1 - SBR1".

Table 220. 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**

Table 221, Table 222 and Table 223 illustrate the address decodes listed in Table 220. These tables show how the internal address is mapped to the **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 220, the unused address lines are still driven with the internal bus address shown.

Table 221. DDR SDRAM Address Decode #1

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]	V[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 228 for more details.
2. For the Leaf Selects, see Section 8.3.3.2.

DDR SDRAM Address translation #2 presents **I_AD[24]** as bit 12 of the row address instead of **I_AD[25]** as in Address decode #1.

Table 222. DDR SDRAM Address Decode #2

MA[12:0]	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]	V[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 228 for more details.
2. For the Leaf Selects, see Section 8.3.3.2.

DDR SDRAM Address decode #3 presents **I_AD[26]** as bit 13 of the row address instead of **I_AD[27]** as in Address decode #1.

Table 223. DDR SDRAM Address Decode #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]	V[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 228 for more details.
2. For the Leaf Selects, see Section 8.3.3.2.

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.

8.3.3.3 DDR SDRAM Configuration

Table 224. DDR SDRAM Address Register Summary

DDR SDRAM Address Register	Definition
DDR SDRAM Base Register (SDBR)	Lowest address for DDR SDRAM memory space aligned to a 32-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[30:25].
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.
DDR SDRAM 32-Bit Size Register (S32SR)	The size for the memory space to operate as 32-bit memory (in MBs). S32SR must be less than, or equal to 1/2 of bank 0 size. Ignored with a 32-bit data bus width.

Note: DDR SDRAM memory space must be aligned to a 32 Mbyte boundary and must ***never*** cross a 2 Gbyte boundary.

Figure 62. DDR SDRAM 64-bit Memory Address Map

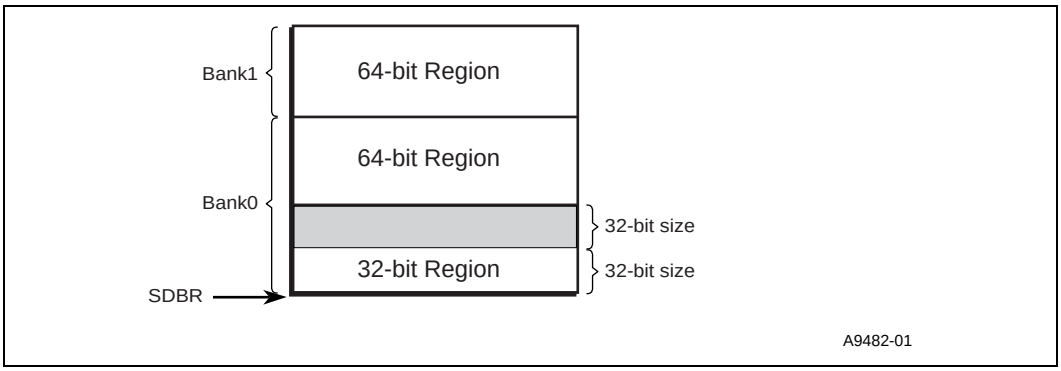


Figure 63. FDDR SDRAM 64-bit Physical Map

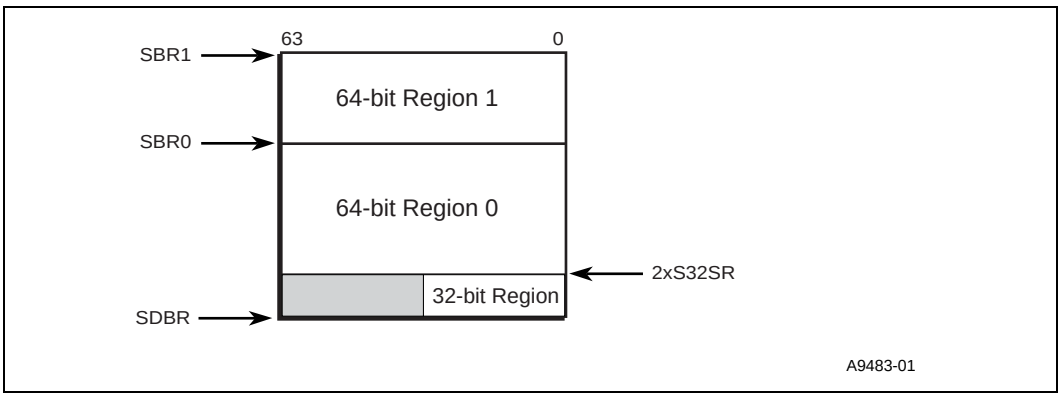


Figure 62 illustrates the Intel® 80331 I/O processor DDR SDRAM 64-bit Memory Address Map, and Figure 63 illustrates the Physical map of the 64-bit DDR SDRAM. The size of each region is configured separately. The 32-bit region is only applicable when the 80331 is connected to 64-bit DDR SDRAM memory. With 32-bit DDR SDRAM attached to 80331, all DDR SDRAM memory space behaves as 32-bit DDR SDRAM and the value in S32SR is ignored.

When the 32-bit region defined by S32SR is greater than zero, bank 0's address space is split into three regions as illustrated in Figure 62. The lowest addressable region operates as a 32-bit region of size defined in the S32SR. A second region, contiguous with the 32-bit region, and equal in size, is defined as invalid for the DDR SDRAM memory space. This invalid region reflects the other half of the 64-bit DDR SDRAM which is not used in the 32-bit region. Transactions which address this region will result in an error and interrupt to the Intel® XScale™ core. The third region is contiguous with the second and is the remainder of bank 0 address space, and is a 64-bit region.

The base register defines the upper seven address bits of the DDR SDRAM memory space. The boundary registers define the address limits for each DDR SDRAM bank in 32 Mbyte granularity. Table 225 defines the conditions which must be satisfied to activate a DDR SDRAM memory bank.

Table 225. Address Decoding for DDR SDRAM Memory Banks

Condition	DDR SDRAM Bank Selected
AD[31] is not equal to the SDBR[31]	None
AD[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the SDBR[30:25] AD[30:25] is less than the value in SBR0	Bank 0
AD[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the value in SBR0 AD[30:25] is less than the value in SBR1	Bank 1
AD[31] is equal to the SDBR[31] AD[30:25] is greater than or equal to the value in SBR1	None

Table 226 shows the correct programming values for the DDR SDRAM Bank Size encoding.

Table 226. Programming Codes for the DDR SDRAM Bank Size

Bank Size	Code	Bank Size	Code
Empty	00H	**256MB**	08H
32MB	01H	**512MB**	10H
64MB	02H	**1GB**	20H
128MB	04H		

These Bank Size Codes are used in the calculation of the DDR SDRAM boundary registers programming values. Equation 5 and Equation 6 show the required equations to calculate the programming values for the DDR SDRAM boundary registers.

Equation 5. Programming Value for DDR SDRAM Boundary Register 0 (SBR0[6:0])

SBR0[6:0] = Bank 0 Size Code+ SDBR[30:25]

Equation 6. Programming Value for DDR SDRAM Boundary Register1 (SBR1[6:0])

SBR1[6:0] = Bank 1 Size Code + SBR0[6:0]

Table 227 shows the correct programming values for the 32-bit DDR SDRAM Size Register.

Table 227. 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	32M	020H
1M	001H	64M	040H
2M	002H	128M	080H
4M	004H	256M	100H
8M	008H	512M	200H
16M	010H	reserved	all other values

Example 2. Address Register Programming Example 1

The user wants to program the DDR SDRAM memory space to begin at B000 0000H. Bank 0 is 64 Mbytes and Bank 1 is 128 Mbytes yielding in a total memory of 192 Mbytes. There is no 32-bit memory region. The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	B000 0000H
32-Bit Region Top (end 32-bit region)	n/a
Invalid Region Top (start 64-bit region)	n/a
Bank 0 Top	B3FF FFFFH
Bank 1 Top	BBFF FFFFH

The registers would be programmed as follows:

Bank 0 Size = 64 MB, code = 000010_2
Bank 1 Size = 128 MB, code = 000100_2
SDBR = B000 0000H, SDBR[30:25] = 011000_2
SBR0[6:0]= 011010_2 = 1AH
SBR1[6:0]= 011110_2 = 1EH
S32SR[29:20] = 0H, S32SR = 0H

Example 3. Address Register Programming Example 2

The user wants to program the DDR SDRAM memory space to begin at A000 0000H. Bank 0 is 64 Mbytes and Bank 1 is un-populated. There is a 4 MB 32-bit region. The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	A000 0000H
32-Bit Region Top (end 32-bit region)	A03F FFFFH
Invalid Region Top (start 64-bit region)	A07F FFFFH
Bank 0 Top	A3FF FFFFH
Bank 1 Top	n/a

The registers would be programmed as follows:

Bank 0 Size = 64 MB, code = 000010_2
Bank 1 Size = empty, code = 000000_2
SDBR = A000 0000H, SDBR[30:25] = 000000_2
SBR0[6:0] = 000010_2 = 02H
SBR1[6:0] = 000010_2 = 02H
S32SR[29:20] = 004H, S32SR = 0040 0000H

Example 4. Address Register Programming Example 3

The user wants to program the DDR SDRAM memory space to begin at C000 0000H. Bank 0 is 512 Mbytes and Bank 1 is 512 Mbytes yielding a total memory of 1 Gbyte. The 32-bit memory region is 256 Mbytes (Bank 0 is entirely 32-bit region). The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	C000 0000H
32-Bit Region Top (end 32-bit region)	CFFF FFFFH
Invalid Region Top (start 64-bit region)	DFFF FFFFH
Bank 0 Top	DFFF FFFFH
Bank 1 Top	FFFF DFFFH*

Note: Bank 1 Top is limited by PMMR space overlap at FFFF E000H in this example.

The registers would be programmed as follows:

Bank 0 Size = 512 MB, code = 010000_2
Bank 1 Size = 512 MB, code = 010000_2
SDBR = C000 0000H, SDBR[30:25] = 010000_2
SBR0[6:0] = 100000_2 = 40H
SBR1[6:0] = 110000_2 = 50H
S32SR[29:20] = 100H, S32SR = 1000 0000H

Example 5. Address Register Programming Example 4

The user wants to program the DDR SDRAM memory space to begin at 0000 0000H. Bank 0 is 1 Gbyte and Bank 1 is 1 Gbyte yielding a total memory of 2 Gbytes. There is a 128MB 32-bit region. THis configuration also requires that the ATU Outbound Direct Addressing window be disabled before configuring the MCU for 2 Gbytes. The memory space summary is:

Memory Space Limit	Address
DDR SDRAM Base	0000 0000H
32-Bit Region Top (end 32-bit region)	01FF FFFFH
Invalid Region Top (start 64-bit region)	03FF FFFFH
Bank 0 Top	3FFF FFFFH
Bank 1 Top	7FFF FFFFH

The registers would be programmed as follows:

Bank 0 Size = 1 GB, code = 100000_2
Bank 1 Size = 1 GB, code = 100000_2
SDBR =0000 0000H, SDBR[30:25] = 000000_2
SBR0[6:0] = 0100000_2 = 20H
SBR1[6:0] = 1000000_2 = 40H
S32SR[29:20] = 080H, S32SR = 0800 0000H

8.3.3.4 32-bit Data Bus Width

Using 64 Mbit SDRAMs, a 64-bit data bus yields a minimum memory size of 64 Mbytes. To address cost-sensitive applications requiring less than 64 Mbytes of local memory, the MCU supports a 32-bit data bus. While 32-bit mode decreases the minimum memory size to 32 Mbytes, the bus throughput also reduces by half.

The MCU does not support switching between 32-bit data bus width and 64-bit data bus width. The data bus width is selected by bit 2 of the SDCR (see Section 8.7.2, "SDRAM Control Register 0 - SDCR0" on page 498). The default is 64-bit bus width.

Reducing the data bus width by half also reduces the page size by half. The page sizes listed in Section 8.3.3.2 are listed for a 64-bit data bus, and must therefore are one half for a 32-bit data bus. The MCU disconnects from the internal bus if the page is crossed during a burst read or write.

8.3.3.5 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 Section 8.3.3.2.

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 (**CS[1:0]#**) and leaf selects (**BA[1:0]**) determine which page address to compare.

If the current transaction misses the open page selected then the MCU closes the open page pointed to by **CS[1:0]#** and **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 **CS[1:0]#** and **BA[1:0]** so it may be compared for future transactions. See Table 221, Table 222and Table 223 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 does not need to open the page (assert RAS#) 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 74. 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 does not need to issue the command again, but simply drive column address for an open page.

If the refresh timer expires and the MCU issues an **auto-refresh** command, all pages are closedFigure 73 illustrates the performance benefit of a read hit versus a read miss in Figure 76. Figure 74 illustrates the performance benefit of a write hit versus a write miss in Figure 79.

Figure 64. Logical Memory Image of a DDR SDRAM Memory Subsystem

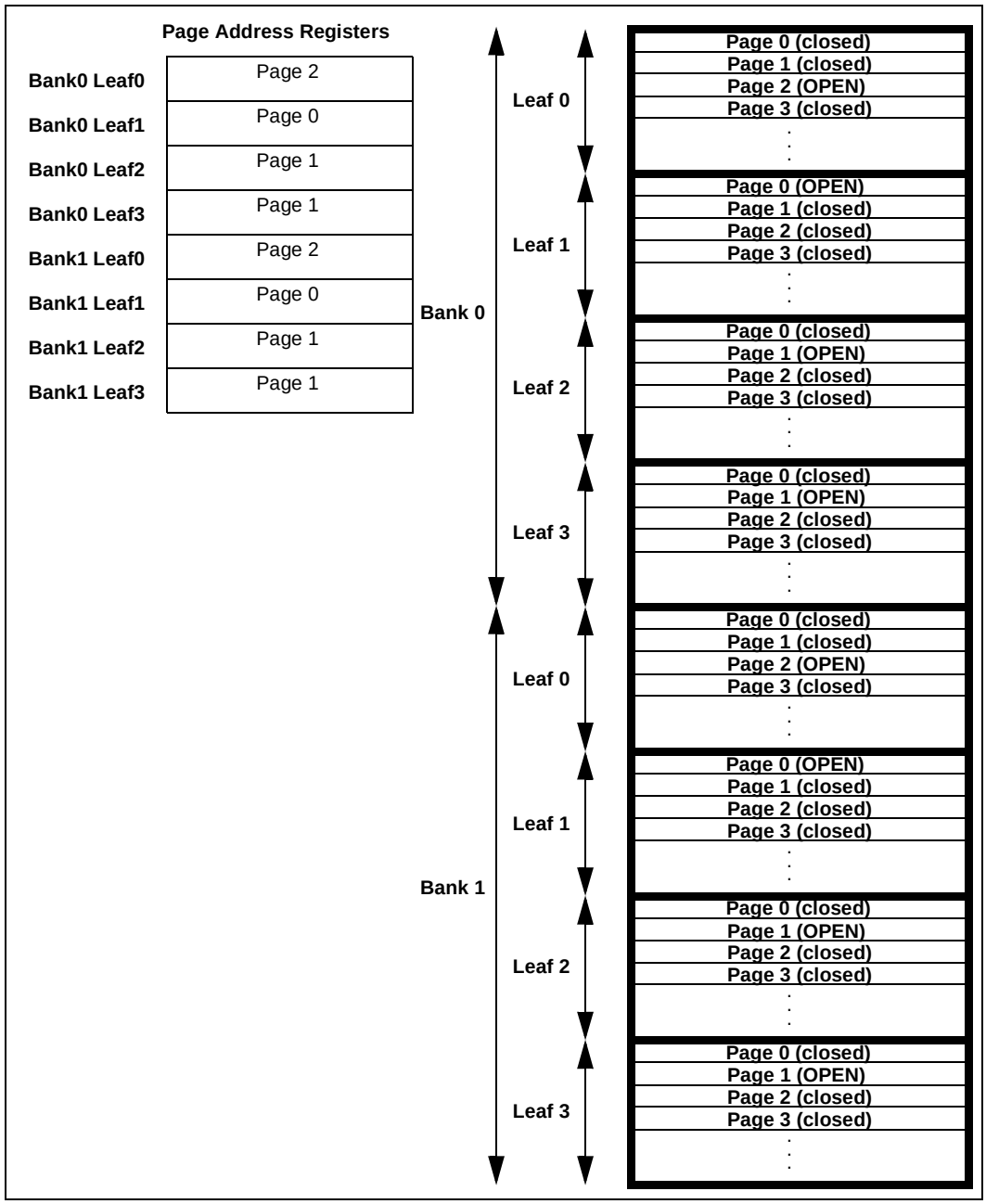


Figure 64 illustrates how the logical memory image is partitioned with respect to open and closed pages. If the above image represents a 128 Mbyte DDR SDRAM memory size, each bank is 64 Mbytes and each leaf is 16 Mbytes.

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 Section 8.3.3.2. The programmer can optimize DDR SDRAM transactions by partitioning code and data across the leaf boundaries to maximize the number of page hits.

8.3.3.6 On DIMM Termination

The memory controller supports On DIMM Termination (ODT) when controlling DDR-II SDRAM technology. ODT eliminates the need for on board termination resisters, thereby reducing the implementation cost and area. Control for enabling ODT is performed in Section 8.7.2, "SDRAM Control Register 0 - SDCR0" on page 498.

8.3.3.7 DDR SDRAM Commands

The MCU issues specific commands to the DDR SDRAM devices by encoding them on the **CS[1:0]#**, RAS#, CAS#, and **WE#** inputs. Table 228 lists all of the DDR SDRAM commands understood by DDR SDRAM devices. The MCU supports a subset of these commands.

Table 228. DDR SDRAM Commands

Command[a,b]	Conditions					Comments
	SCE#	RAS#	CAS#	SWE#	Other	
NOP	0	1	1	1		No Operation
Mode Register Set	0	0	0	0	**BA[0]** = Sel[c] **BA[1]** = 0	Load the (Extended) Mode Register from **MA[13:0]**
Row Activate	0	0	1	1	**BA[1:0]** = Leaf	Activate a row specified on **MA[13:0]**
Read	0	1	0	1	**BA[1:0]** = Leaf **MA[10]** = 0	Column burst read Column address on **MA[13:0]**
Read w/ Auto-Precharge	0	1	0	1	**BA[1:0]** = Leaf **MA[10]** = 1	Column burst read with row precharge at the end of the transfer
Write	0	1	0	0	**BA[1:0]** = Leaf **MA[10]** = 0	Column burst write Column address on **MA[13:0]**
Write w/ Auto-Precharge	0	1	0	0	**BA[1:0]** = Leaf **MA[10]** = 1	Column burst write with row precharge at the end of the transfer
Precharge	0	0	1	0	**BA[1:0]** = Leaf **MA[10]** = 0	Precharge a single leaf
Precharge All	0	0	1	0	**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 while **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.

a. This table copied from New DRAM Technologies by Steven Przybylski.
b. Shaded boxes indicate commands not supported by 80331. They are included for completeness.
c. During a Mode Register Set command, **BA[1:0]** = 00_2 selects the Mode Register, **BA[1:0]** = 01_2 selects the Extended Mode Register, all others are served.

DDR SDRAM commands are synchronous to the clock so the MCU sets up the above conditions prior to the **M_CK[2:0]** rising edge.

8.3.3.8 DDR SDRAM Initialization

Since DDR SDRAM devices contain a controller within the device, the MCU must initialize them specifically. Upon the deassertion of **I_RST#**, software initializes the DDR SDRAM devices with the sequence illustrated with Figure 68:

1. The MCU applies the clock (**M_CK[2:0]**) at power up along with system power (clock frequency unknown).
2. The MCU must stabilize **M_CK[2:0]** within 100 µs after power stabilizes.
3. The MCU holds all the control inputs inactive (RAS#, CAS#, **WE#**, **SCE[1:0]# = 1**), places all data outputs and strobes in the High-Z state (**DQS[8:0]**, **DQ[71:0]**), and deasserts **CKE[1:0]** for a minimum of 200 us after supply voltage reaches the desired level. Asserting **P_RST#** achieves this state.
4. Software disables the refresh counter by setting the RFR to zero.
5. Software issues one **NOP** cycle after the 200 us device deselect. A **NOP** is accomplished by setting the SDIR to 0011_2. The MCU asserts **CKE[1:0]** with the NOP.
6. Software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0010_2.
7. Software issues an **extended-mode-register-set** command to enable the DLL by writing 0100_2 to the SDIR. The MCU supports the following DDR 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 80331.

Figure 65. Supported DDR SDRAM Extended Mode Register Settings

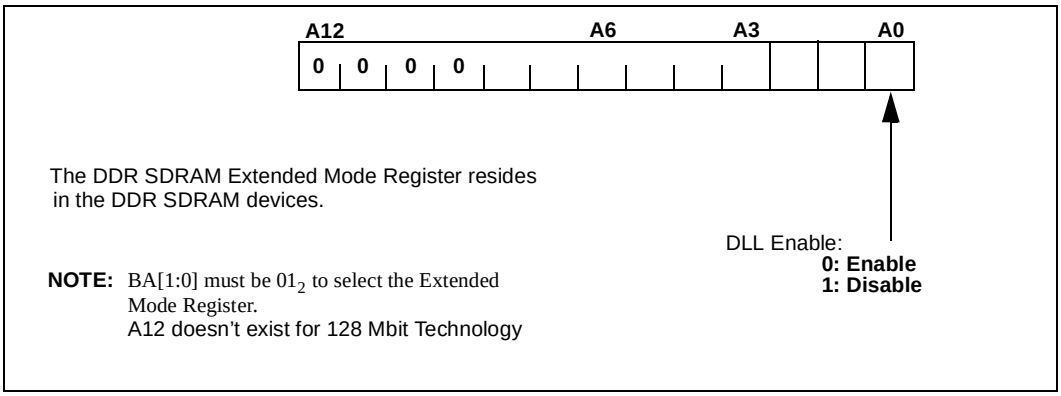
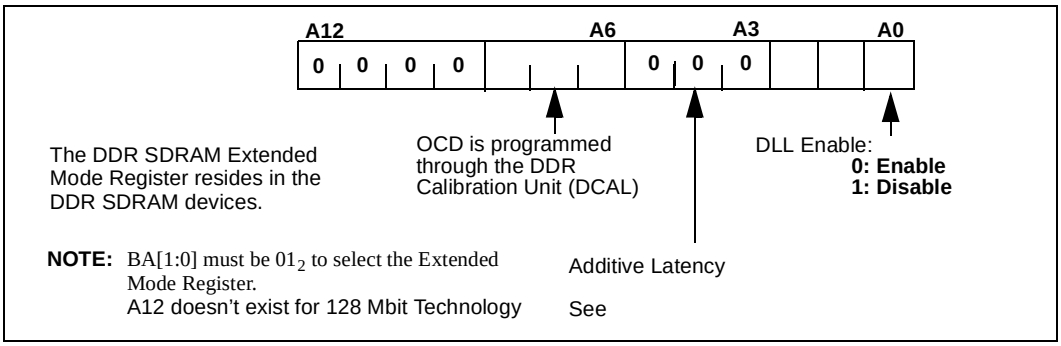
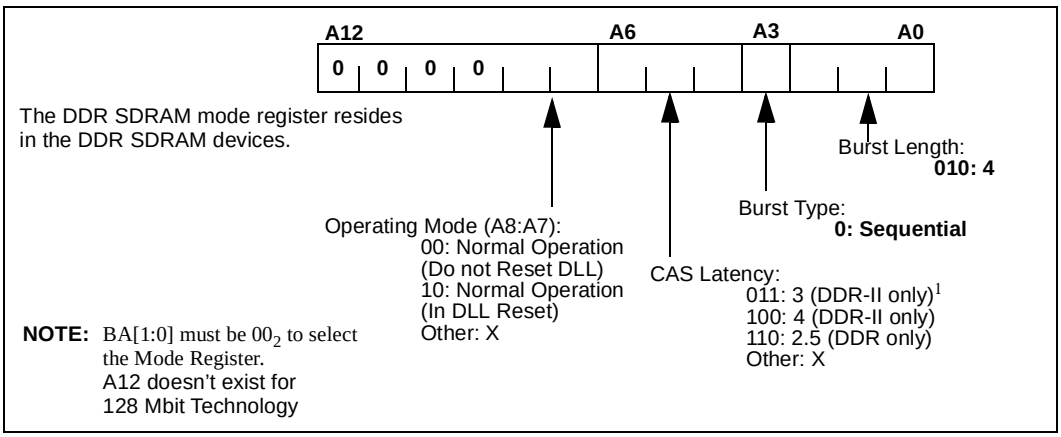


Figure 66. Supported DDR-II SDRAM Extended Mode Register Settings



8. 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 DDR SDRAM mode parameters:
   a. CAS Latency (tCAS) = two and one-half for DDR, or three[1] or four for DDR-II based on the programmed setting in Section 8.7.2, "SDRAM Control Register 0 - SDCR0"
   b. Burst Type = Sequential
   c. Burst Length (BL) = four

Figure 67. Supported DDR SDRAM Mode Register Settings

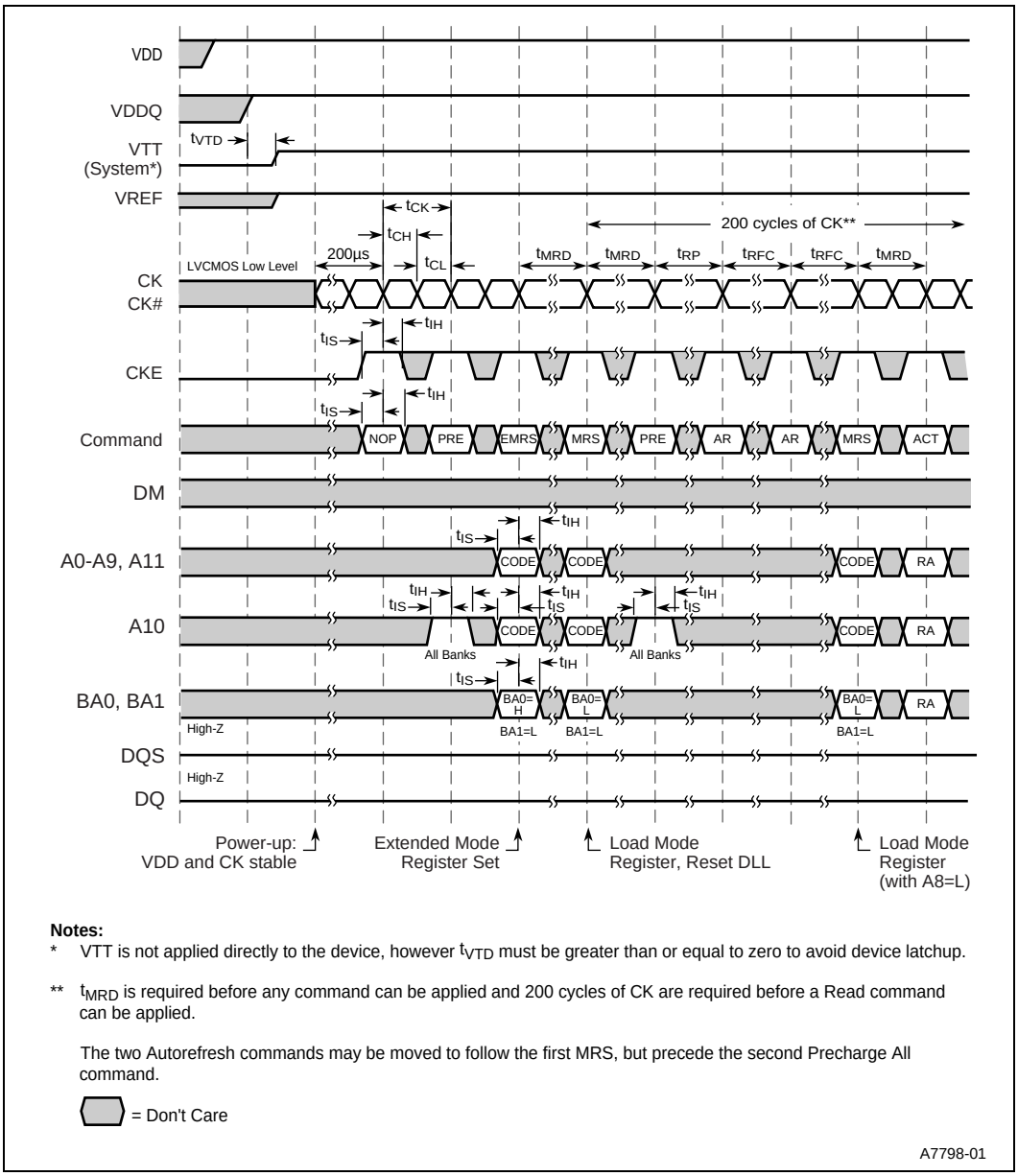


9. After waiting T_{mrd} cycles, software issues a **precharge-all** command to the DDR SDRAM interface by setting the SDIR to 0010_2.
10. After waiting T_{rp} cycles, software provides two **auto-refresh** cycles. An **auto-refresh** cycle is accomplished by setting the SDIR to 0110_2. Software must ensure at least T_{rfc} cycles between each **auto-refresh** command.
11. 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.
12. The MCU may issue a **row-activate** command T_{mrd} cycles after the **mode-register-set** command.
13. Software re-enables the refresh counter by setting the RFR to the required value.

1. CAS latency of 3 is not supported.

The waveform in Figure 68 illustrates the DDR SDRAM initialization sequence.

Figure 68. 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

A7798-01

If the DDR SDRAM subsystem implements ECC (see Section 8.3.4, "Error Correction and Detection" on page 476), then initialization software should initialize the entire memory array with the 80331. It is important that every memory location has a valid ECC byte. The AAU includes a memory block fill mode which can be used to fill the memory array with a constant (Section 7.5.3, "Memory Block Fill Operation" on page 404), thereby initializing the associated ECC bytes in the process. A small portion of memory must be initialized by the Intel® XScale™ core processor software to store the AAU descriptor. If the memory array is not initialized, the BIU may attempt to read memory locations beyond the specified word(s). In this case, the MCU will report an ECC error even though software did not specifically request the un-initialized data.

8.3.3.9 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 8.3.3.8, "DDR SDRAM Initialization" on page 461) 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 to 80331. 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 will NOT interact properly with the DDR SDRAM until the SDCR[1:0] registers have been programmed.

The duration between valid commands must be programmed by the user before the DDRSM can interact properly with the DDR. These parameters programmed or used to in calculations are listed in Table 229, and are defined by either JEDEC and/or within this document in the timing diagrams and equations in this section. See Section 8.7.2, "SDRAM Control Register 0 - SDCR0" on page 498 and Section 8.7.3, "SDRAM Control Register 1 - SDCR1" on page 500 for exact format of these registers.

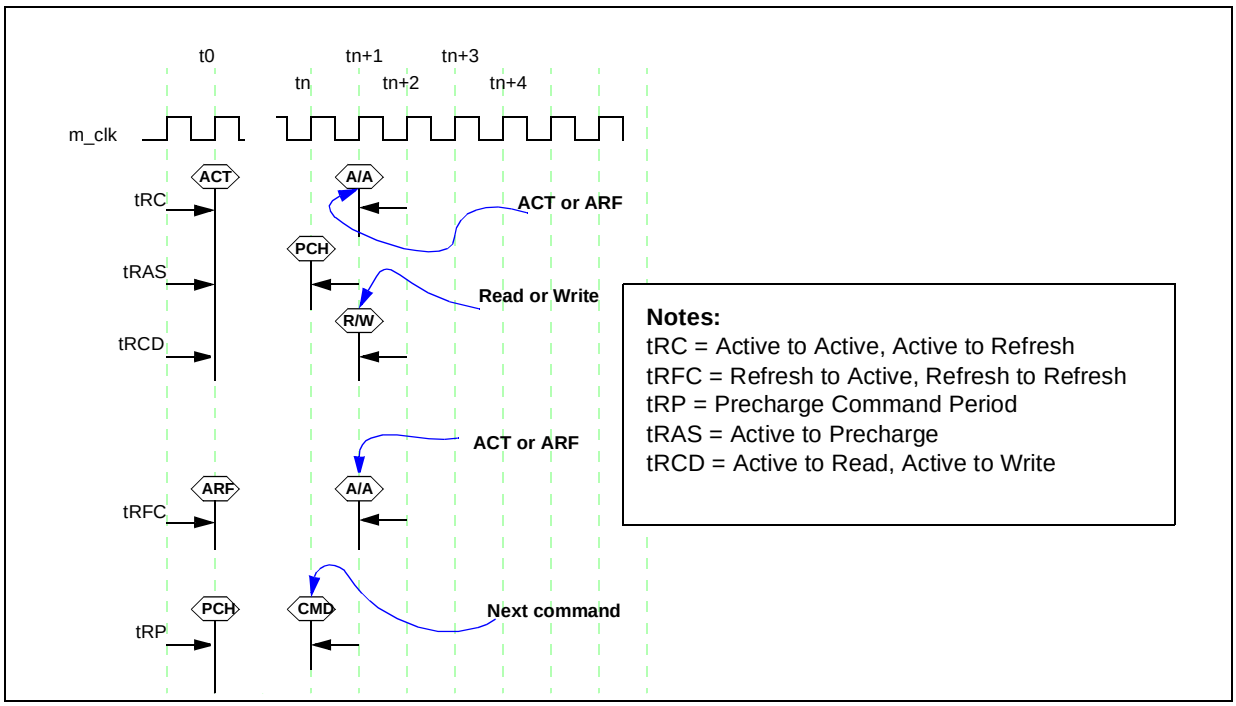
Table 229. SDCR[1:0] Timing Parameters Summary

Parameter	Source	Use	Parameter	Source	Use
tBL	JEDEC	Equation 7, Equation 8, Equation 9	tRTCMD	Equation 7	SDCR1[30:28]
tCAS	JEDEC	Equation 7, Equation 8, Equation 9, Equation 10 Figure 66 and SDCR0[9:8]	tRTW	Equation 7	SDCR1[22:20]
tEDP[a]	80331	Equation 7 and SDCR0[17:16]	tWL	Equation 10 and JEDEC	Equation 8, Equation 9, Equation 11 Figure 66
tRAS	JEDEC	SDCR0[31:28]	tWDL	Equation 11	SDCR0[13:12]
tRC	JEDEC	SDCR1[8:4]	tWR	JEDEC	Equation 9 and SDCR1[11:9]
tRCD	JEDEC	SDCR0[22:20]	tWTR	JEDEC	Equation 8
tRFC	JEDEC	SDCR1[16:12]	tWTRD	Equation 8	SDCR1[2:0]
tRP	JEDEC	SDCR0[26:24]	tWTCMD	Equation 9	SDCR1[26:24]

a. tEDP default value is 1 (01_2) and must be programmed to a value of 2 (10_2) in the SDCR0.

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*, June 200 and *JEDEC DDR - II SDRAM Specification*, September 2002. These parameters, and their relationship to each other, are outlined in Figure 69. Please see the JEDEC specification for the timing parameters specific to the DDR device that is to be implemented in the system.

Figure 69. MCU Active, Precharge, Refresh Command Timing Diagram



The timing parameters for DDR reads are defined in Figure 70. Both Read to Write (80331 *tRTW*) and Read to Command (80331 *tRTCMD*) are defined the same. Both parameters take into account CAS latency (**JEDEC**: *tCAS*) and Burst Length (**JEDEC**: *BL*).

Note: Burst Length is fixed at four for 80331

To program tRTW and tRTCMD, the user should use the following equation:

Equation 7. tRTW = tRTCMD = tCAS + (BL/2) + tREG + (tEDP[1] -1) = tCAS + 1+ tREG + tEDP[1]

Where tREG equals:

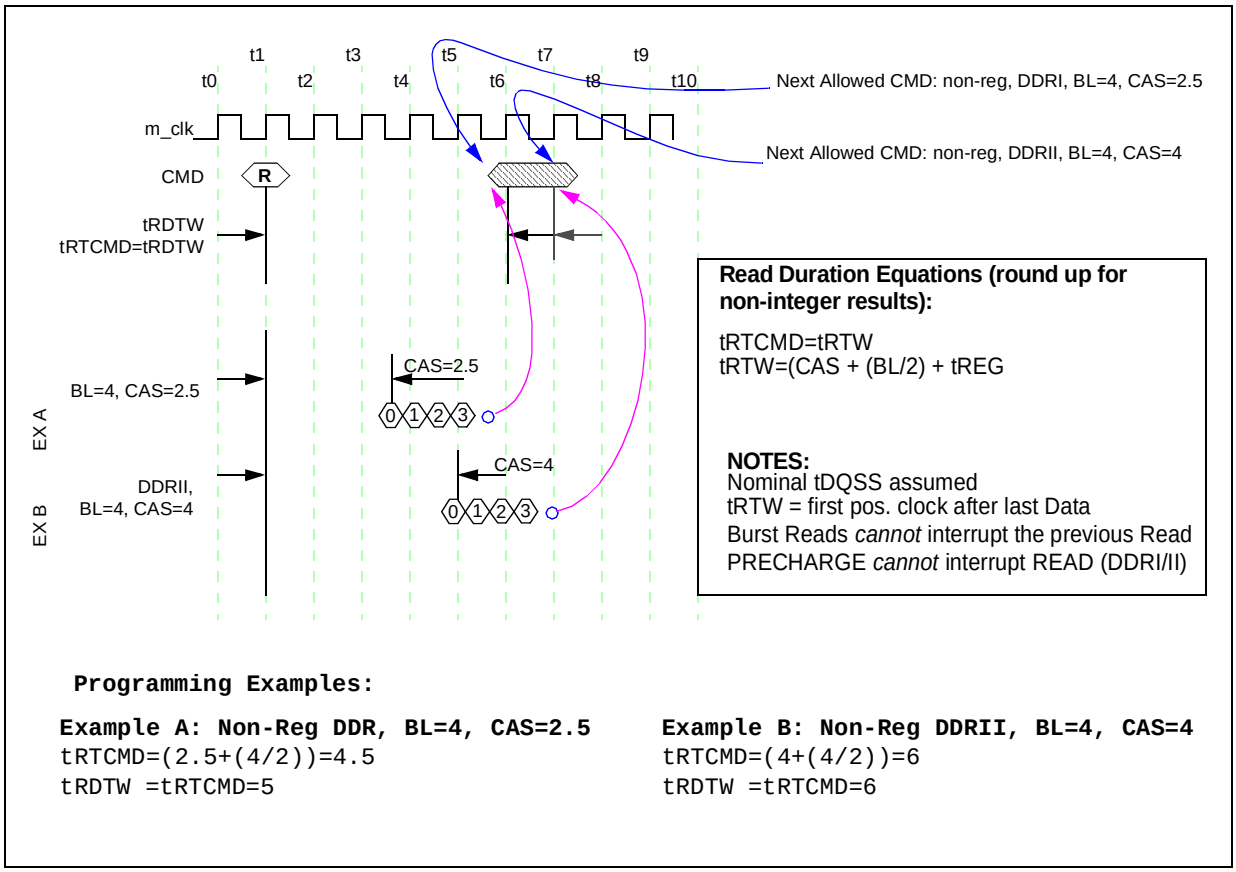
— 1 = Registered SDRAM

— 0 = Unbuffered SDRAM

The MCU allows for back-to-back reads, so long as they are to a currently open page.

1. tEDP default value is 1 (01_2) and must be programmed to a value of 2 (10_2) in the SDCR0.

Figure 70. MCU DDR Read Command to Next Command Timing Diagram



The timing parameters for DDR writes are defined in Figure 71. Both Write to Read (80331: *tWTRD*) and Write to Command (80331: *tWTCMD*) 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**: *tWR*) and Write to Read (**JEDEC**: *tWTR*).

Note: Burst Length is fixed at four for 80331

To program tWTRD and tWTCMD, the user should use the following equations:

Equation 8. tWTRD = tWL + (BL/2) +tWTR + tREG = tCAS + 1 +tWTR + tREG

Equation 9. tWTCMD = tWL + (BL/2) +tWR + tREG = tCAS +1 +tWR + tREG

Where the following is used for tWL:

Equation 10. tWL = AL + tCAS - 1 = 0 + tCAS - 1 = tCAS - 1

and where BL=4, AL=0 and tREG equals:

— 1 for Registered SDRAM

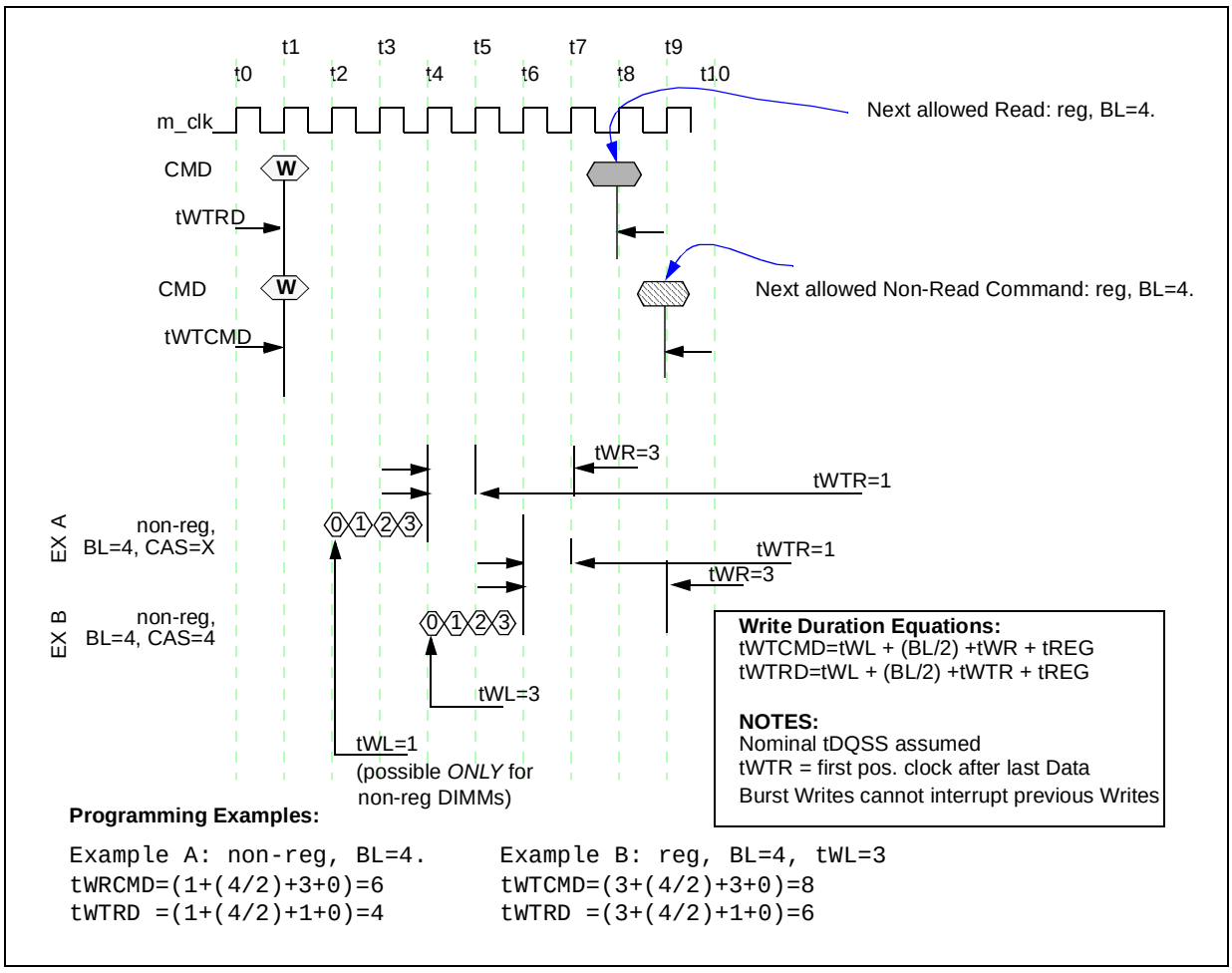
— 0 for Unbuffered SDRAM

To program tWDL, the user should use the following equation:

Equation 11. tWDL = tWL - 1 = tCAS - 1 - 1 = tCAS - 2

The MCU allows for back-to-back Writes, so long as they are to an open page.

Figure 71. MCU DDR Write Command to Next Command Timing Diagrams



8.3.3.10 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 73. 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 which 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 72 for random read memory transactions.

Figure 72. DDR SDRAM Pipelined Reads

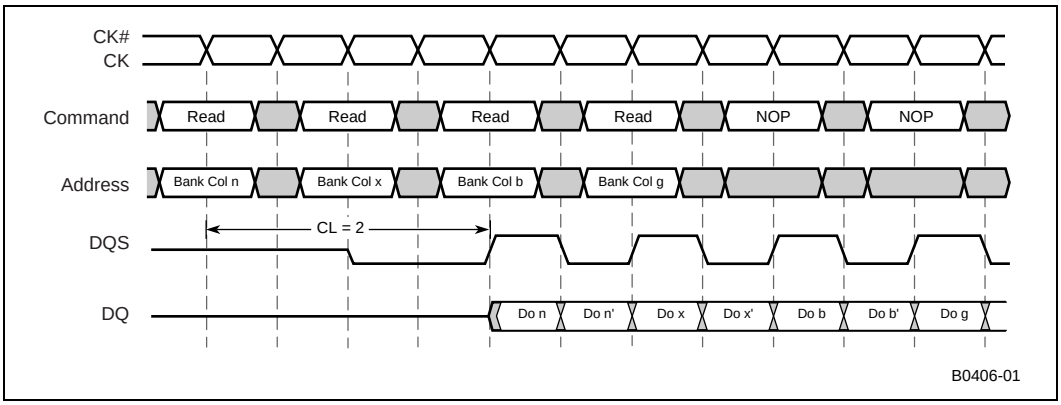
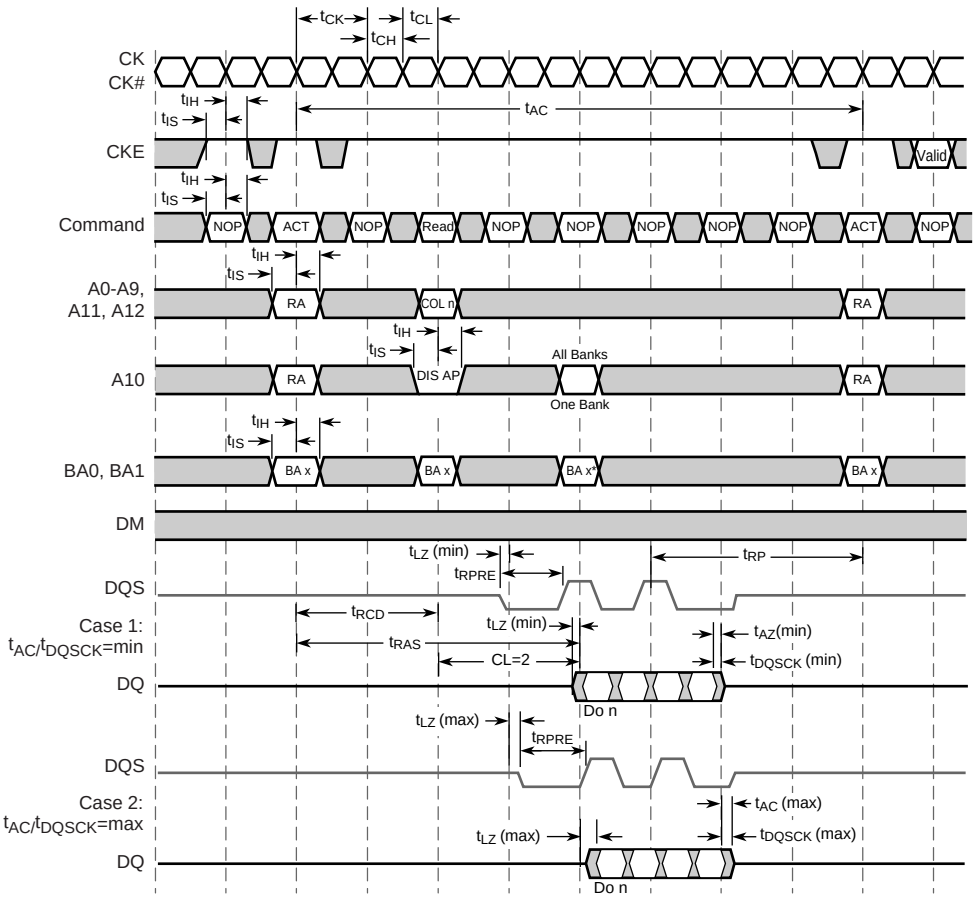


Figure 73. DDR SDRAM Read, 36 Bytes, ECC Enabled, BL=4



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, SBR1, and S32SR 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 needs to be closed before the read can be issued to the new page. Refer to Section 8.3.3.5, "Page Hit/Miss Determination" on page 458 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. (Not depicted in Figure 68).
   — 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 RAS#, de-asserts **WE#**, and drives the row address on **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 CAS#, de-asserts **WE#**, and places the column address on **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 73.
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 8.3.4, "Error Correction and Detection" on page 476 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.

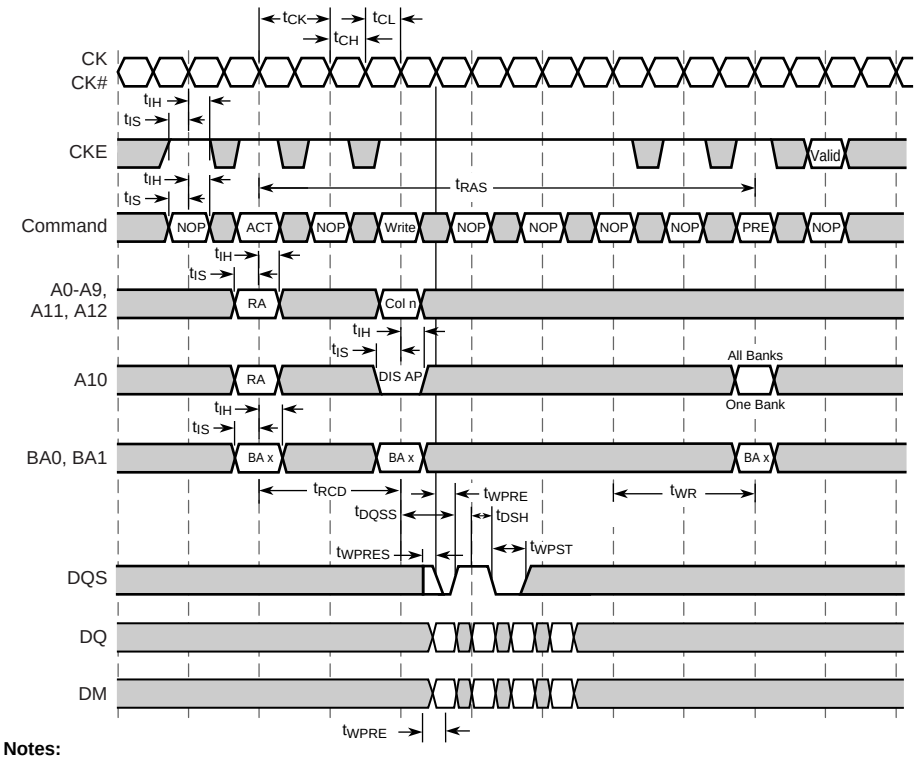
8.3.3.11 DDR SDRAM Write Cycle

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. 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 8.3.4, "Error Correction and Detection" on page 476 explains the ECC algorithm in more detail.

For a page hit, the MCU does not need to open the page (assert RAS#) 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 74. For a page hit, the two cycles required for row activation are saved resulting in lower first word write latency.

Figure 74. DDR SDRAM Write, 36 Bytes, ECC Enabled, BL=4



Notes:
t_{DQSS} = min
DI n = Data in for column n.
3 subsequent elements of data in are applied in the programmed order following DI n.
DIS AP = Disable Auto Precharge
* = Don't care if A10 is High at this point.
PRE = Precharge; ACT = Active; RA = Row Address
NOP commands are shown for ease of illustration; other valid commands may be possible at these times.
= Don't Care

B0407-01

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, SBR1, and S32SR 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.

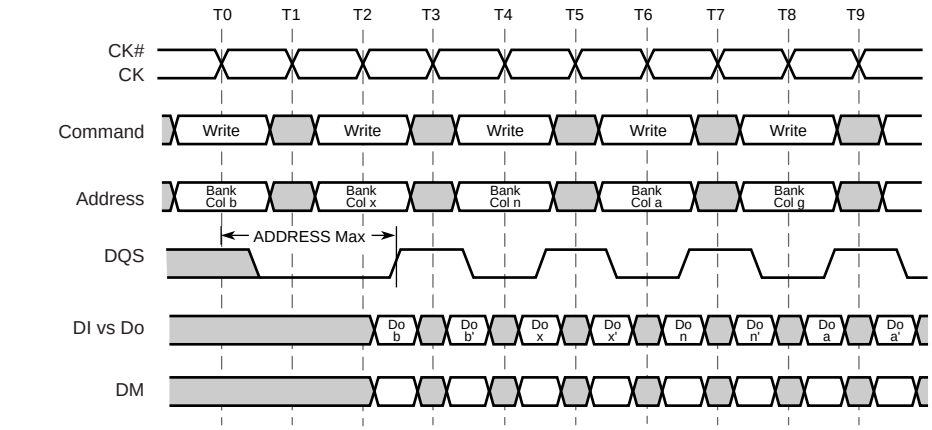
A write that misses the open page encounters a miss penalty because the currently open page needs to be closed before the write can be issued to the new page. Refer to Section 8.3.3.5, "Page Hit/Miss Determination" on page 458 for the paging algorithm details. If a page hit occurs, steps 2-3 are skipped 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 RAS#, de-asserts **WE#**, and drives the row address on **MA[13:0]**.
5. After T_{rcd} cycles in the case of a page miss, the DDR SDRAM Control Block asserts CAS#, asserts **WE#**, and places the column address on **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 an 8 byte boundary (4 byte for 32-bit data bus width or 32-bit region), the DDR SDRAM Control Block will perform a read-modify-write of the entire 8 byte aligned quad-word (4 byte aligned double-word for 32-bit data bus width or 32-bit region) and incorporate the new data while regenerating ECC.

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 which 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 75) for random read memory transactions.

Figure 75. DDR SDRAM Pipelined Writes



Notes:
- DI b, etc. = Data In for column b, etc.
- DI b', etc. = the next Data in following DI b, etc. according to the programmed burst order
- Burst Length = 4 is shown.
- Write command may be to any bank and may be in the same or different devices.

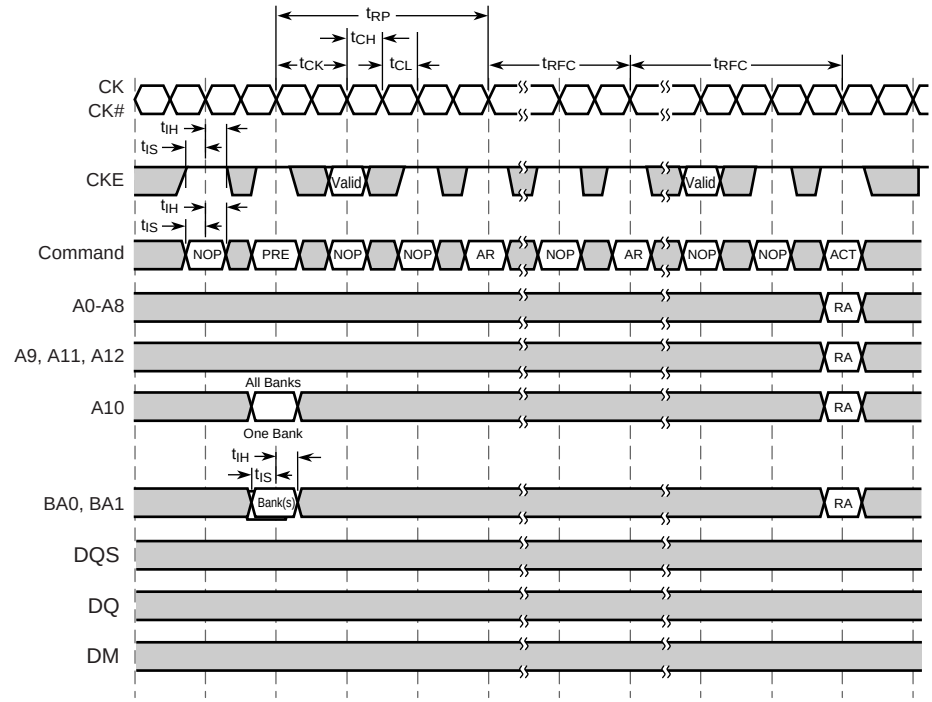
= Don't Care

B0408-01

8.3.3.12 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. The waveform in Figure 76 illustrates the case where the refresh timer expires while the memory bus is not busy.

Figure 76. 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 complete 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.
- After T_{rfc} cycles, the DDR SDRAM Control Block can service a new transaction or another refresh cycle.

It is recommended that the RFR ("Frequency Register - RFR" on page 512) is programmed with the value to achieve 7.8 us, 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 230 can provides for these two typical values.

Table 230. Typical Refresh Frequency Register Values

DDR Speed	7.8 µs Value	15.6 µs Value
333 MHz	A00H	1400H
400 MHz	C00H	1800H

The longest possible internal bus transaction is writing a 1 Kbyte burst where each data cycle results in a read-modify-write due to partial writes (see Section 8.3.4.2, "ECC Generation for Partial Writes" on page 478). The longest possible CMTQ tenure is 16 transactions where each of the transaction are page misses and partial writes. Such periods potentially require queueing two refresh cycles.

8.3.4 Error Correction and Detection

The MCU is capable of correcting any single bit errors and detecting any double bit errors in the 80331 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, which simply detects single bit errors, error correction requires an additional 8-bit code word for the 64-bit (32-bit) datum. This means that a memory must have the additional 8-bit error correction code (**CB[7:0]**) per 64-bit (32-bit) datum (**DQ[63:0]**) resulting in a 72-bit (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 the 32-bit region with 64-bit memory, or with 32-bit wide memory, the 80331 will zero extend the 32-bit datum to a 64-bit datum in order to generate, check and correct ECC. This means that a 32-bit datum memory with ECC will result in a 40-bit wide memory since an 8-bit error correction code is still required.

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 80331, 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.

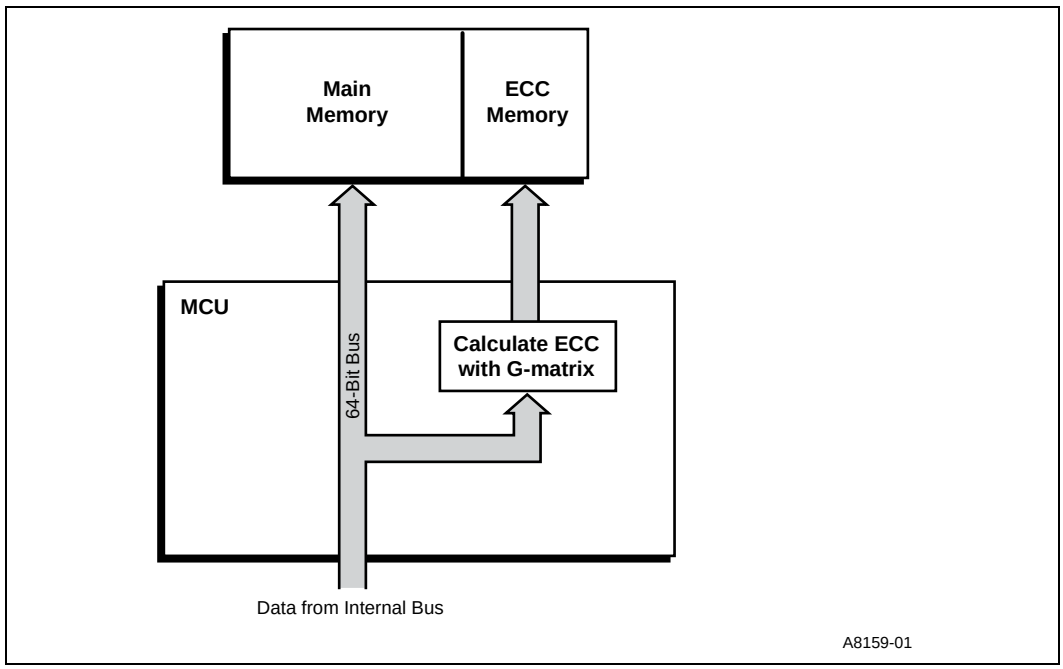
8.3.4.1 ECC Generation

For write operations, the MCU generates the error correction code which is written along with the data. This section describes the operation of the DDR SDRAM Control Block for ECC generation in a 64-bit wide memory and 64-bit region. The same principles apply for 32-bit wide memory and in the 32-bit region in 64-bit wide memory, however the MCU will generate 8-bit wide ECC by zero extending the data to 64-bits The algorithm for a write transaction is:

```
if data to write is 64 bits wide
    Generate the ECC_with the G-matrix
    Write the new data and ECC
    else {Partial Write}
    Read entire 64-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 77 shows how the data logically flows through the ECC hardware for a write transaction.

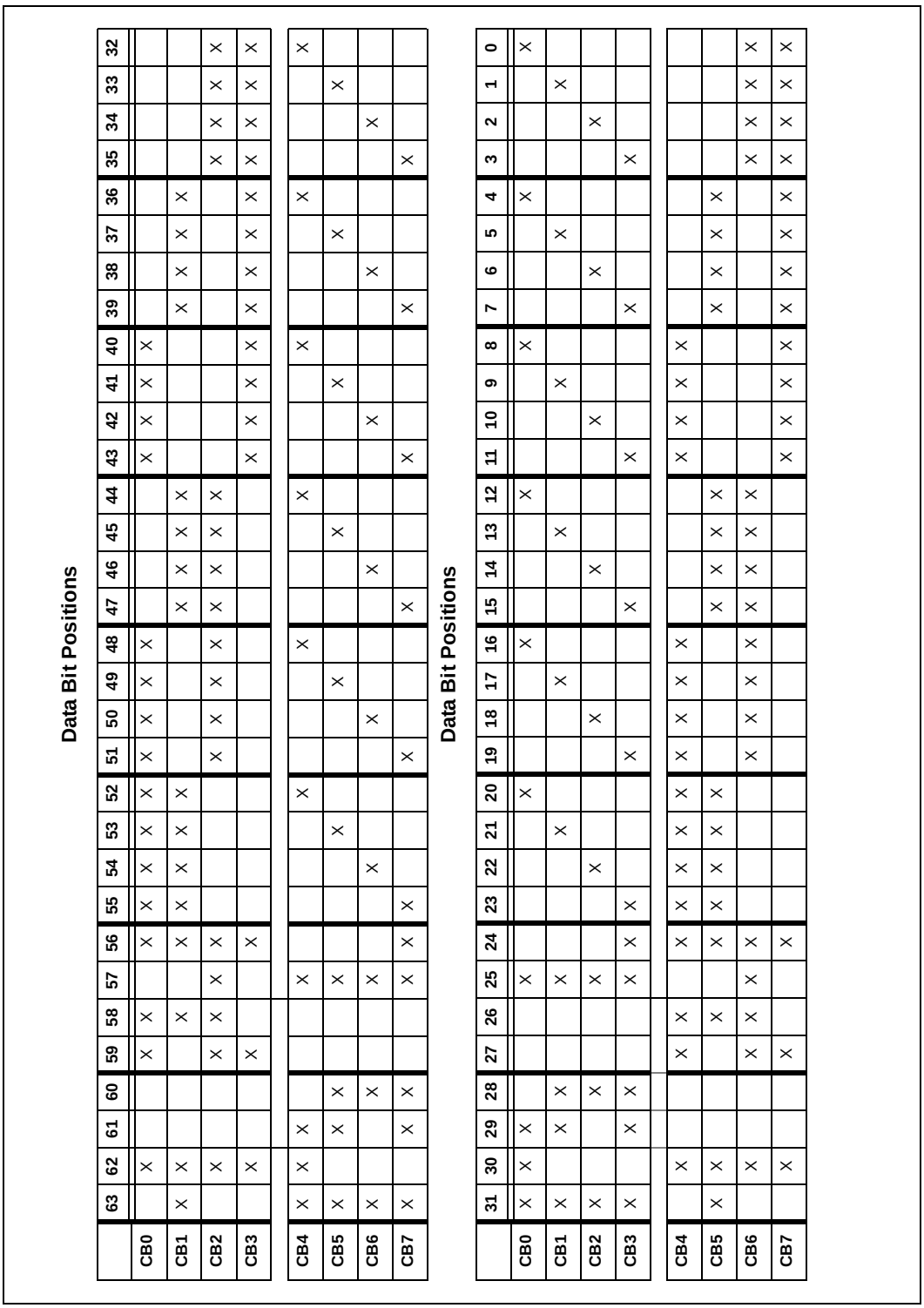
Figure 77. ECC Write Flow



The G-Matrix in Figure 78 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 which data bits of **AD[63:0]** needs to be XORed together to form the ECC bit. The resulting ECC bits are driven on **CB[7:0]**.

8.3.4.2 ECC Generation for Partial Writes

Figure 78. Intel® 80331 I/O Processor G-Matrix (generates the ECC)



If the memory transaction writes less than the data bus width programmed in the SDCR, 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:

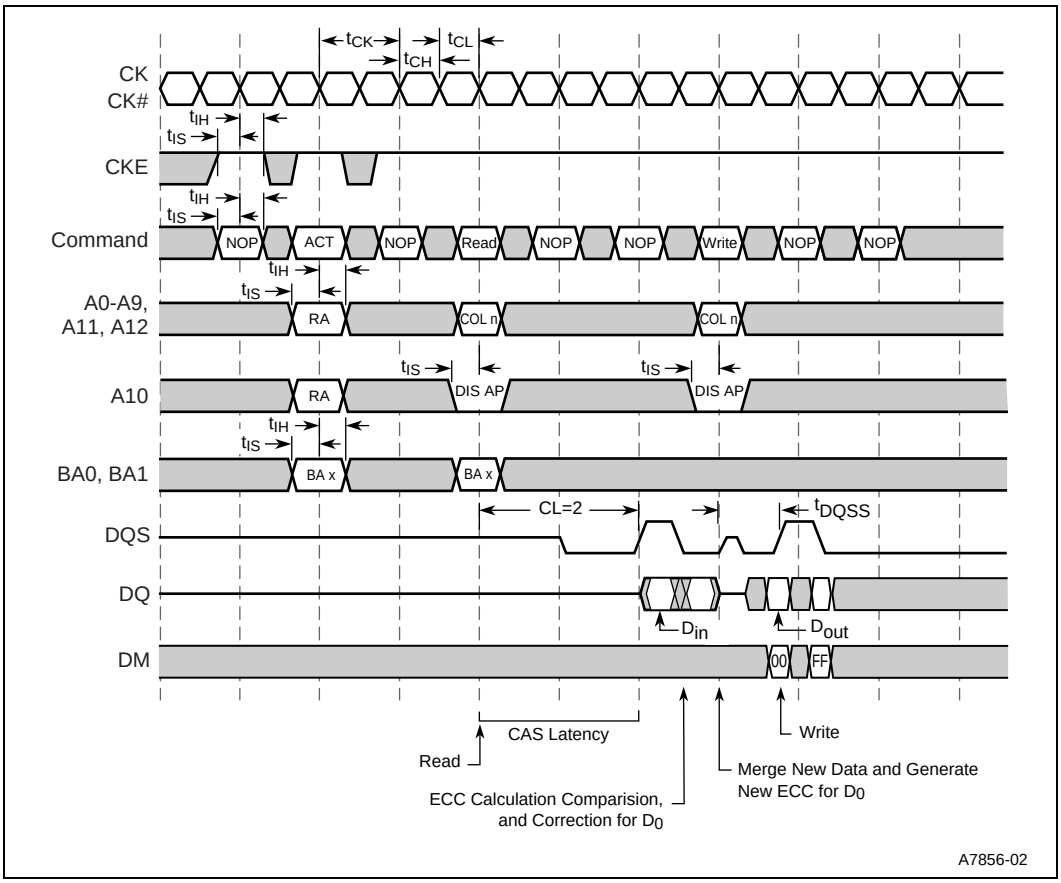
1. Issues a 64-bit (32-bit) read.
2. Modifies the value with the new portion to be written.
3. Calculates the ECC on the modified value.
4. Writes the 64-bit (32-bit) value and ECC.

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 8.5, "Interrupts/Error Conditions" on page 493.

Figure 79 shows an example where the data of a write is less than 64-bits wide. The waveform illustrates how the DDR SDRAM Control Block issues a read-modify-write cycle for the data (D_1).

Note: In 32-bit wide memory and in the 32-bit region in 64-bit wide memory, the DDR SDRAM Control Block will still generate 8-bit wide ECC by zero extending the data to 64-bits. A partial write is a write of less then 4-Bytes.

Figure 79. Sub 64-bit DDR SDRAM Write (D_0)



8.3.4.3 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 64-bit data and 8-bit ECC
Compute the syndrome by passing the 64-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 231 shows how the MCU decodes the syndrome for DDR SDRAM read cycles.

Table 231. Syndrome Decoding

Error Type	Symptom
None	The syndrome is 0000 0000.
Single-Bit	Use the H-Matrix in Figure 81 to determine which bit the MCU will invert 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 80 shows how the data flows through the ECC hardware for a read transaction.

Figure 80. ECC Read Data Flow

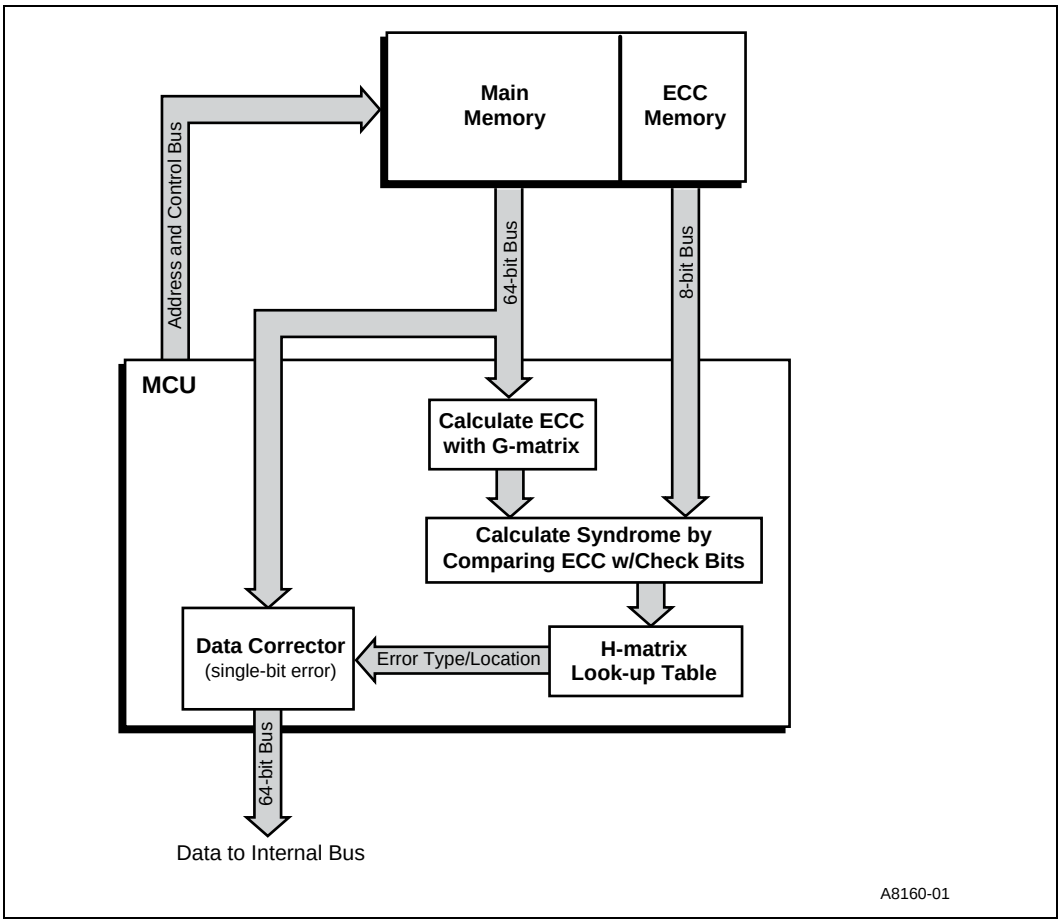
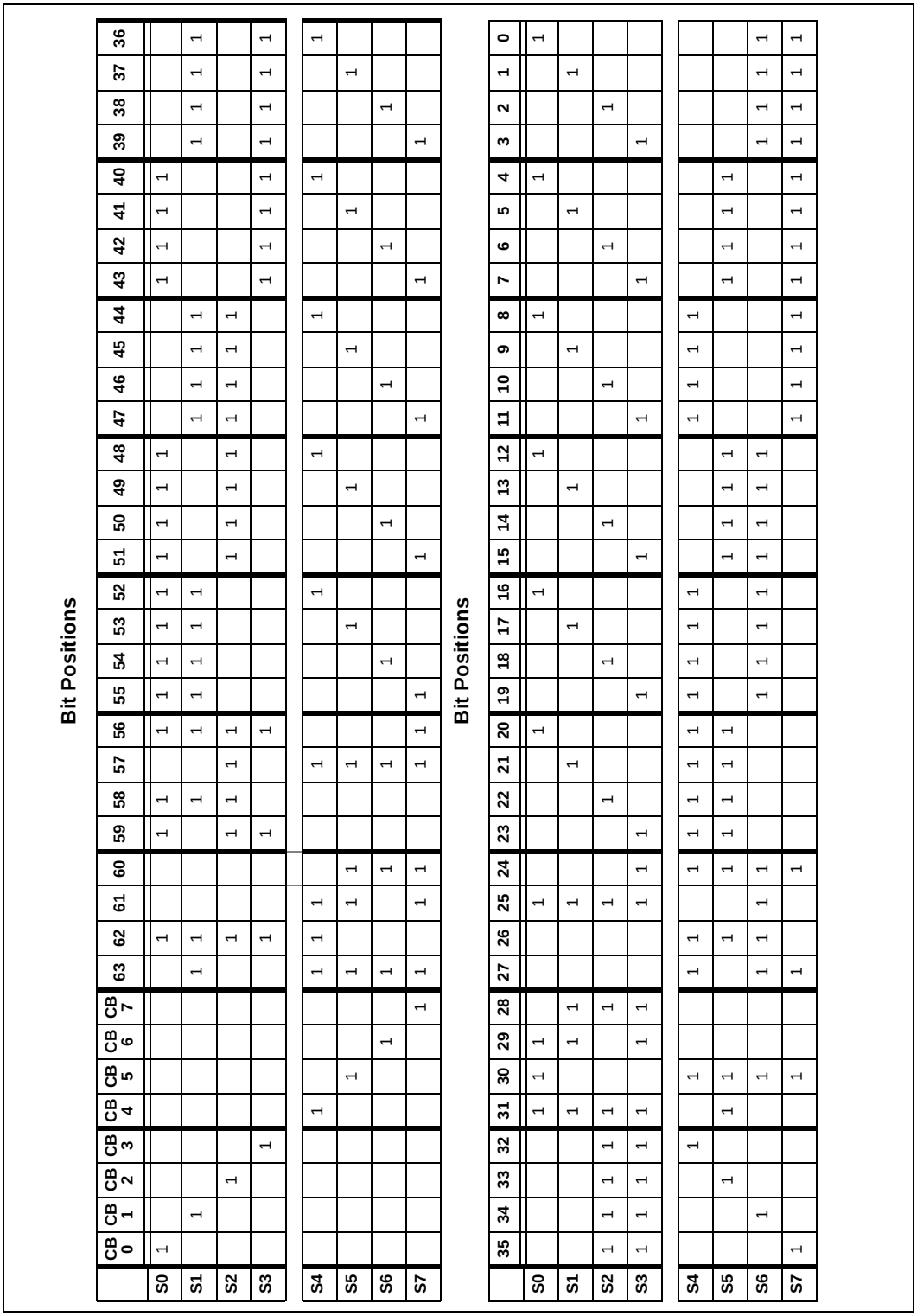


Figure 81 illustrates the H-Matrix used for decoding the syndrome. For single-bit errors, the H-Matrix indicates the bit that contains the error and consequently, which bit to fix.

Figure 81. Intel® 80331 I/O Processor H-Matrix (indicates the single-bit error location)



Referring to Figure 80, the syndrome bits are created by XORing the data bits as indicated by the appropriate row of the G-Matrix in Figure 78 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 **CB[0]**). The MCU performs eight such XOR operations (one per syndrome bit).

If decoding the syndrome indicates multi-bit error (see Table 231), the transaction results in a target-abort 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 81). For example, if the syndrome was 1100 0001, the error is with bit 0 of **DQ[63:0]**. The MCU inverts bit 0 before driving the data on **AD[63:0]**.

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 8.5, "Interrupts/Error Conditions" on page 493.

Note: In 32-bit wide memory and in the 32-bit region in 64-bit wide memory, the DDR SDRAM Control Block will still generate 8-bit wide ECC by zero extending the data to 64-bits. A partial write is a write of less then 4-Bytes.

8.3.4.4 Scrubbing

Fixing the data error in memory is called scrubbing. The 80331 relies on Intel® XScale™ core 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 core. The core 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 will have 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 either DWORD of the 64-bit memory space (QWORD aligned location) 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 in order to fix the single-bit error, a second error will be 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 will automatically scrub the QWORD aligned location if it contains a single-bit ECC error

Multi-bit errors cannot be fixed by the H-Matrix.

8.3.4.4.1 ECC Example Using the H-Matrix

Assume the core writes 1234 5678 9ABC DEF0H to the SDRAM memory space. The Core 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 5678 9ABC DEF0H on **AD[63:0]**.

During the next CMTQ tenure, this transaction is processes and the DDR SDRAM Control Block receives the data and must calculate the ECC code.

Using the G-Matrix in Figure 78, the DDR SDRAM Control Block creates each check bit by XORing the appropriate bits in the row. Using 1234 5678 9ABC DEF0H, the ECC code generated is D2H. This code is written with the data to the SDRAM devices on **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 core issues a read transaction which 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 1234 5678 9AB**E** DEF0H on **DQ[63:0]**, the DDR SDRAM Control Block calculates the syndrome with the G-Matrix in Figure 78. 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. The syndrome is created during a memory read by XORing the 8-bit value generated by XORing appropriate data bits (**DQ[63:0]**) indicated by the G-Matrix with the check bits (**CB[7:0]**).

Referring to Table 231, if the syndrome is non-zero and matches a value in the H-Matrix, there is a single-bit error that can be fixed. A syndrome of 52H matches a value in the H-Matrix (see Figure 81) which indicates that bit 17 has an error. The DDR SDRAM Control Block inverts bit 17 prior to returning the corrected data on **AD[63:0]**. The MCU returns 1234 5678 9ABC DEF0H on **AD[63:0]**.

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] which 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.

8.3.4.5 ECC Disabled

If software disables ECC, the MCU does generate the ECC byte for writes, but does not check the ECC byte for reads. For writes, the MCU will not perform the Read and Modify steps normally performed for a sub 64-bit write as described in Section 8.3.4.2, "ECC Generation for Partial Writes" on page 478. When the Intel® XScale™ core writes 0's to memory, with ECC disabled, the MCU will pad all sub 64-bit writes with zeros, calculate ECC, and store the ECC value along with the write data.

This mode can be used to initialize ECC and memory from the Intel® XScale™ core. By writing the entire 64-bits of data (two stores by the core), all of the data will be zero, and the ECC value will be updated (twice) with a valid code for zero data.

Note: For faster initialization of large memory space, the Block Fill function of the Application Accelerator should be used.

8.3.4.6 ECC Testing

Section 8.3.4.4, "Scrubbing" on page 485 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 Section 8.5, "Interrupts/Error Conditions" on page 493).

8.3.5 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 FFFF E000H to FFFF 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, and the size of the 32-bit region in S32SR if desired.

While it is not recommended, the two ranges could overlap. In the case of a memory space overlap, refer to Table 232 for the priority rules.

Table 232. Overlapping Address Priorities

Priority	Address Region
Highest	Memory Mapped Register Address Space
Lowest	DDR SDRAM Address Space

8.3.6 DDR SDRAM Clocking

The MCU provides 6 clocks, three positive (**M_CK[2:0]**) and three negative (**M_CK[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*, June 2000 requires 6 clocks to distribute the loading across eighteen x8 DDR SDRAM components.

These metrics do not measure performance impact of preemption capability.

8.4 Power Failure Mode

The 80331 is an I/O processor used in server applications including networking and storage. Specifically, the storage applications supported utilize the 80331 as the IOP for a SCSI RAID disk subsystem and the local memory is used for disk caching. The local memory is used for the temporary storage of disk writes which greatly improves disk performance.

While the host assumes all written data is stored on the non-volatile disk subsystem, the IOP must ensure that eventually all the data in the disk cache is actually stored onto disk.

The power supply could fail to provide power to the I/O subsystem in the case of a power outage or a failed power supply. It is imperative that the cached data within the IOP's local memory is not lost. If power fails, the local memory subsystem must remain powered with a battery backup and some agent must continue to refresh at the appropriate interval specified by the memory component datasheet.

This section defines a mechanism with which the 80331 memory controller ensures that the data within local memory is not lost during a power failure.

8.4.1 Theory of Operation

DDR SDRAM technology provides a simple way of enabling data preservation through the **self-refresh** command. This command is issued by the memory controller and the DDR SDRAM will refresh itself autonomously with internal logic and timers. The **self-refresh** command is defined in Table 228.

The DDR SDRAM device will remain in self-refresh mode as long as:

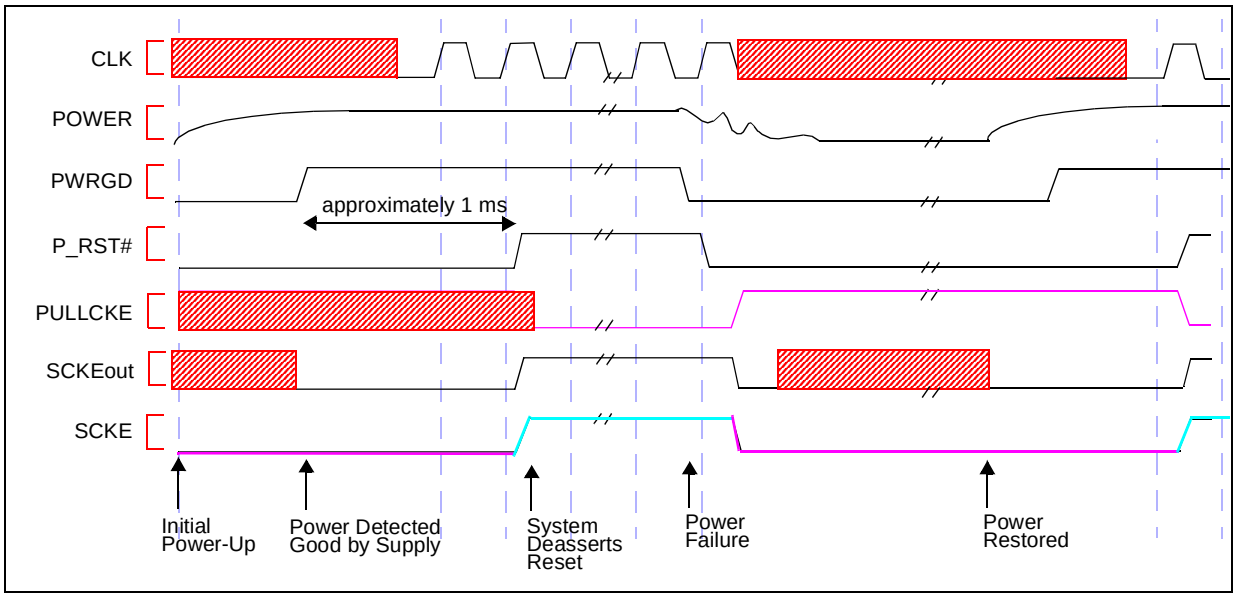
- The device continues to be powered.
- **CKE** is held low until the memory controller is ready to control the DDR SDRAM once again.

Power to the DDR SDRAM subsystem is ensured with an adequate battery backup and a reliable method for switching between system power and battery power. The memory controller is responsible for deasserting **CKE[1:0]** when issuing the **self-refresh** command but while power gradually drops, **CKE[1:0] must** remain deasserted regardless of the state of V_{cc} powering the 80331.

8.4.2 Power Failure Sequence

Figure 82 illustrates the sequence of events during a power failure as defined by *PCI Local Bus Specification*, Revision 2.2.

Figure 82. Power Failure Sequence



8.4.2.1 Power Failure Impact on the System

Upon initial power-up a power supply provides the appropriate voltage to the system. The voltage level will increase at a rate that is dependent on the type of power supply used and the components in the system. These variables are not certain, so the power supply often provides a signal called **PWRGD** which indicates the time when the voltage has reached a reliable level. The power supply deasserts **PWRGD** if the voltage level drops below a certain minimum threshold.

PCI Local Bus Specification, Revision 2.2 indicates that once **PWRGD** is deasserted, the PCI reset pin (**P_RST#**) is asserted in order to float the output buffers. In the specification T_{fail} is defined as the time when **P_RST#** is asserted in response to the power rail going out of specification. T_{fail} is the minimum of:

- 500 ns from either power rail going out of specification (exceeding specified tolerances by more than 500mV)
- 100 ns from the 5V rail falling below the 3.3V rail by more than 300mV

8.4.2.2 System Assumptions

This proposal makes specific assumptions about the system's behavior during a power failure. If the below assumptions are not guaranteed, it is the vendor's responsibility to ensure them.

1. **P_RST#** is asserted to the 80331 when there is at least 2 us of reliable power remaining. This is required so that the memory controller can execute it's power-failure state machine in response to the assertion of **P_RST#**.

8.4.3 Memory Controller Response to P_RST#

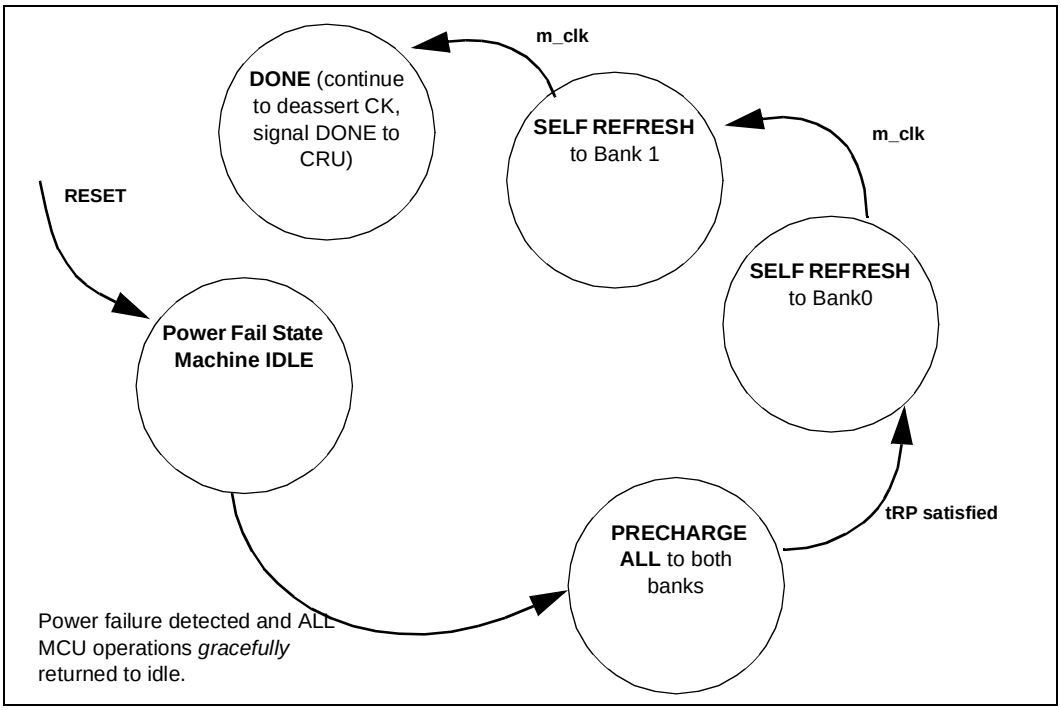
If **PWRDELAY** is asserted indicating that an initial power up sequence has completed, the memory controller assumes a power failure condition whenever **P_RST#** is asserted. **P_RST#** assertion following an initial power up sequence results in the following sequence of events:

1. The Clock/Reset Unit (CRU) will request that the MCU run the power fail sequence depicted in Figure 84 on page 491. The MCU will take the following steps to execute the power fail sequence:
   a. Gracefully terminate the current transaction.
   b. Deactivate all DDR SDRAM leaves with the **precharge-all** command.
   c. After Trp, the MCU will issue a **self-refresh** command to the DDR SDRAM devices one bank at a time and continue to deassert **CKE[1:0].**
2. The MCU will notify the CRU that the power fail sequence has completed on the memory bus.
3. The CRU will then assert **I_RST#** to reinitialize all internal bus agents including the MCU.

Note that **I_RST#** is asserted in response to the assertion of **P_RST#**. If **P_RST#** indicates a true power failure (i.e., **PWRDELAY** is asserted), then battery-backup power is supplied to the DDR SDRAM array.

Refer to Figure 83 on page 490 for a high-level state machine representation illustrating the memory controller's behavior during a power failure condition.

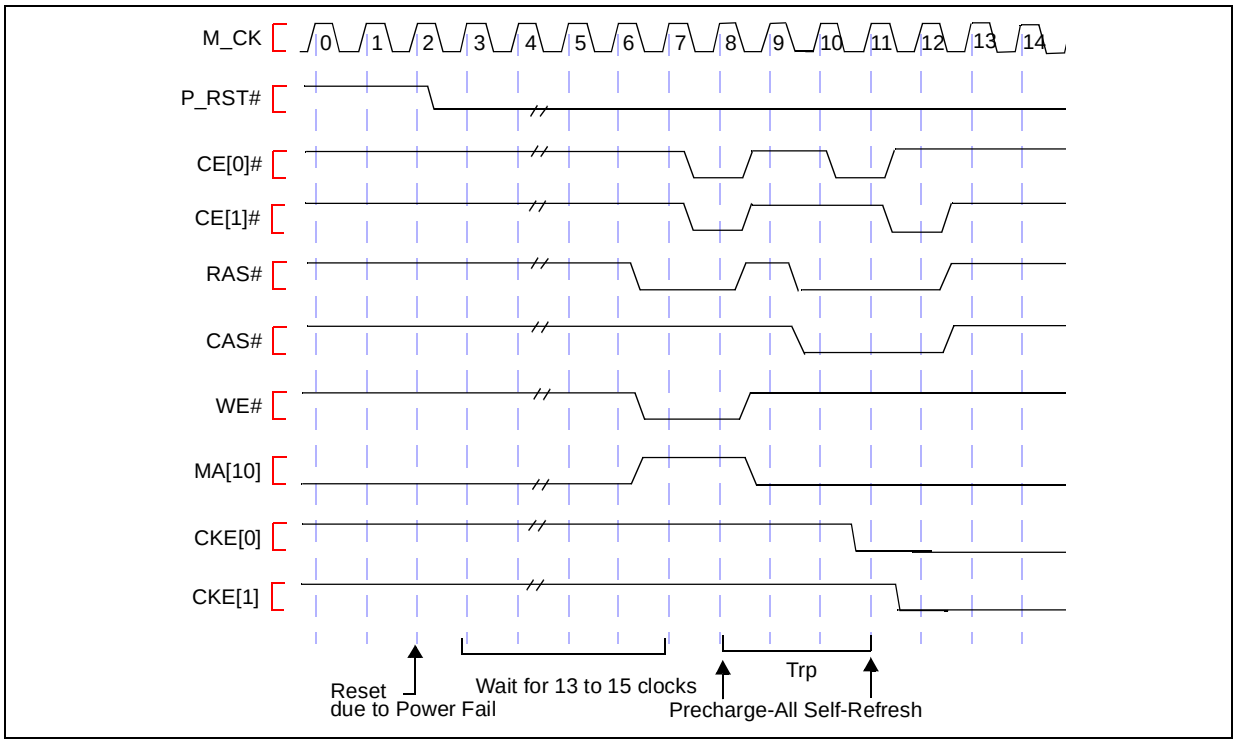
Figure 83. Power Failure State Machine



Note: Following the request from the CRU to run the power fail sequence, any data that is in the MCUs 1024 byte posted write buffer will be discarded.

Figure 84 on page 491 illustrates the DDR SDRAM waveforms after the assertion of **P_RST#** during a true power failure.

Figure 84. Power Failure Sequence



CKE[1:0] must be held low throughout the power-down period. The memory controller drives it low initially with the **self-refresh** command, but an external pull-down is required to continually drive it low when the 80331 loses power. External logic ensures that **CKE[1:0]** is held low after the memory controller initially deasserts it. Likewise, the external logic must stop driving **CKE[1:0]** low once **P_RST#** is deasserted by the system. Figure 84 shows one example of the external logic required for power failure mode.

Due to the high loading on **CKE** and the requirement of PC200 DDR operation, the memory controller must drive two copies to the DDR SDRAM DIMM. The board layout will distribute the two **CKE[1:0]** signals between the two DDR SDRAM banks equally.

As long as the DDR SDRAM memory subsystem is powered with a battery source and **CKE[1:0]** is held low, the DDR SDRAM preserves its memory image.

When power is restored, the system asserts **P_RST#** to the 80331. While the 80331 is reset, **CKE[1:0]** is held low by memory controller. After **P_RST#** is deasserted (and subsequently, **I_RST#** is deasserted), the 80331 must be re-initialized to reset the DDR SDRAM memory subsystem operating parameters. The first step of DDR SDRAM initialization sequence re-asserts **CKE[1:0]** to ones and the memory controller resumes refreshing. DDR SDRAM initialization sequence does not affect memory contents. For more details about the DDR SDRAM initialization sequence, refer to Section 8.3.3.8, "DDR SDRAM Initialization" on page 461.

Note: The power failure mechanism in the memory controller is not responsible for maintaining the 80331 state. The purpose of this mechanism is to maintain the memory so that any data cached in the local memory can be flushed once power is restored. Any data queued within the 80331 components (MCU, ATU, DMAs, etc.) will be lost.

8.4.3.1 External Logic Required for Power Failure

8.4.3.1.1 Assertion of P_RST# During Power Failure

The 80331 **P_RST#** input signal must be asserted if system power goes below 3.0V.

8.4.3.1.2 Distinguishing Between a Power Up and a Power Failure P_RST#Assertion

The 80331 provides a dedicated input pin, **PWRDELAY** that will be used to distinguish between and initial power up and a power failure assertion of **P_RST#**. This signal should be driven by external circuitry that will assert **PWRDELAY** following the initial deassertion of **P_RST#**. **PWRDELAY** will not be deasserted until power has truly failed.

This circuitry could consist of a capacitor that is charged up through a FET enabled via deassertion of **P_RST#**. The only discharge path available to the capacitor would be through a Zener diode connected to VCC, thus **PWRDELAY** would be deasserted only when power has truly failed.

The Clock Reset Unit will **not** request the MCU to run the power failure sequence if **P_RST#** is asserted while **PWRDELAY** is deasserted.

8.4.3.2 P_RST# Usage Versus I_RST#

The memory controller logic is initialized with the assertion of the Internal Bus Reset signal **I_RST#**. It is important to note that the external logic required for power failure mode uses **P_RST#** since **I_RST#** is not available. It is possible to assert **I_RST#** with either **P_RST#** or software using the PCI Configuration and Status Register (see Table 125, "PCI Configuration and Status Register - PCSR" on page 253 located in the ATUs MMR space. If software resets the internal bus, the external logic will not function (but it is not required to anyway).

8.5 Interrupts/Error Conditions

The MCU has two conditions which require intervention from the Intel® XScale™ core. If a single-bit error is detected during a read cycle, the MCU can correct the data returned but software needs to 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 233 shows how the MCU responds to error conditions.

Table 233. MCU Error Response

Error Type	MCU Action[a]
Single-Bit during a read or write	Fix Error (if ECC error correction enabled in the SDCR)
Multi-bit during a read	Target Abort the Internal Bus transaction 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.

a. The ECC Enable bit in the SDCR needs to 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 Table 233 actions.

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 Section 8.3.4.1, "ECC Generation" on page 477 for details.

8.5.1 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 core 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 8.3.4.4, "Scrubbing" on page 485).

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 core needs to 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 core.
- 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 233.

8.5.2 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] which 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 **DQ[63:0]** so the contents of memory after the read-modify-write cycle will be 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 233.

It is interrupt handler responsibility to decide how to handle this error condition and clear the MCISR.

8.6 Reset Conditions

Once **I_RST#** is deasserted and 200 us have passed, software must issue the initialization sequence defined in Section 8.3.3.8, "DDR SDRAM Initialization" on page 461. 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.

While **I_RST#** is asserted, the MCU initializes its MMR registers to the states defined in Section 8.7, "Register Definitions" on page 496.

Note: The operation of any memory transactions are not guaranteed when **P_RST#** is asserted.

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 Section 8.3.4.1, "ECC Generation" on page 477 for details.

8.7 Register Definitions

A series of configuration registers control the MCU. Software can determine the status of the MCU by reading the status registers. Table 234 lists all of the MCU registers which are detailed further in proceeding sections.

Note: 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.

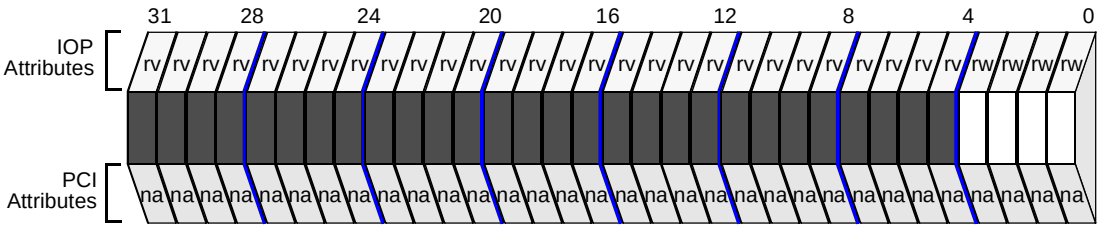
Table 234. Memory Controller Register

Section, Register Name - Acronym (Page)
Section 8.7.1, "SDRAM Initialization Register - SDIR" on page 497
Section 8.7.2, "SDRAM Control Register 0 - SDCR0" on page 498
Section 8.7.3, "SDRAM Control Register 1 - SDCR1" on page 500
Section 8.7.4, "SDRAM Base Register - SDBR" on page 501
Section 8.7.5, "SDRAM Boundary Register 0 - SBR0" on page 502
Section 8.7.6, "SDRAM Boundary Register 1 - SBR1" on page 503
Section 8.7.7, "SDRAM 32-bit Region Size Register - S32SR" on page 504
Section 8.7.8, "ECC Control Register - ECCR" on page 505
Section 8.7.9, "ECC Log Registers - ELOG0, ELOG1" on page 506
Section 8.7.10, "ECC Address Registers - ECAR0, ECAR1" on page 507
Section 8.7.11, "ECC Test Register - ECTST" on page 508
Section 8.7.12, "Memory Controller Interrupt Status Register - MCISR" on page 509
Section 8.7.13, "MCU Port Transaction Count Register - MPTCR" on page 510
Section 8.7.14, "MCU Preemption Control Register - MPCR" on page 511
Section 8.7.15, "Frequency Register - RFR" on page 512
Section 8.7.16, "DCAL Control and Status Register - DCALCSR" on page 513
Section 8.7.17, "DCAL Address Register - DCALADDR" on page 515
Section 8.7.18, "DCAL Data Registers 17:0 - DCALDATA[17:0]" on page 516
Section 8.7.19, "Receive Enable Delay Register - RCVDLY" on page 521
Section 8.7.20, "Slave Low Mix 0 - SLVLMIX0" on page 522
Section 8.7.21, "Slave Low Mix 1 - SLVLMIX1" on page 523
Section 8.7.22, "Slave High Mix 0 - SLVHMIX0" on page 524
Section 8.7.23, "Slave High Mix 1 - SLVHMIX1" on page 525
Section 8.7.24, "Slave Length - SLVLEN" on page 526
Section 8.7.25, "Master Mix - MASTMIX" on page 527
Section 8.7.26, "Master Length - MASTLEN" on page 528
Section 8.7.27, "DDR Drive Strength Status Register - DDRDSSR" on page 529
Section 8.7.28, "DDR Drive Strength Control Register - DDRDSCR" on page 530
Section 8.7.29, "DDR Miscellaneous Pad Control Register - DDRMPCR" on page 531

8.7.1 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 Section 8.3.3.8, "DDR SDRAM Initialization" on page 461).

Table 235. DDR SDRAM Initialization Register - SDIR



Intel® XScale™ Core Local Bus Address
FFFF E500H

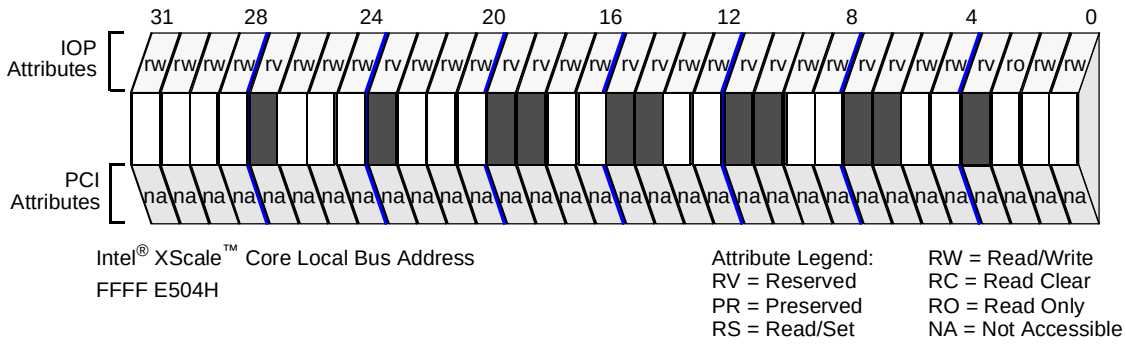
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	0	Reserved
03:00	1111_2	**Special DDR SDRAM Command**: These bits are used for DDR SDRAM initialization. See Section 8.3.3.8, "DDR SDRAM Initialization" on page 461 for details. While not in the initialization sequence, these bits should be set to $1xxx_2$. For details on the exact DDR SDRAM commands, refer to Table 228, "DDR SDRAM Commands" on page 460. • 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-Register Set (EMRS) Command** where DLL is enabled on the DDR SDRAM devices. • 0101_2 - Extended **Mode-Register Set (EMRS) Command** where DLL is disabled on the DDR SDRAM devices. • 0110_2 - **Auto-Refresh Command**: The MCU issues one refresh sequence (i.e. precharge-all followed by **auto-refresh** command) to the DDR SDRAM devices. • Others - **Normal DDR SDRAM Operation**

8.7.2 SDRAM Control Register 0 - SDCR0

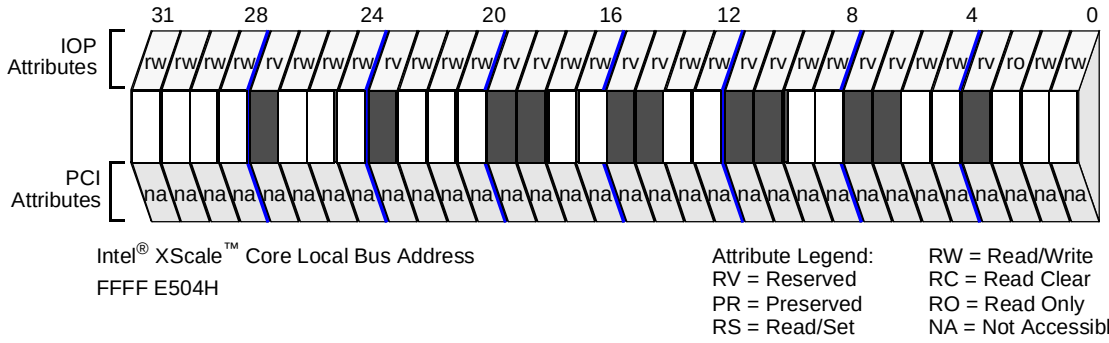
The SDRAM Control Registers (SDCR[1:0]) are responsible for programming the operation of the DDR SDRAM state machines. The SDCR0 specifies the DIMM type, data bus width, and some SDRAM timing parameters required by the DDR SDRAM state machine as defined in Section 8.3.3.8, "DDR SDRAM Initialization" on page 461 and Section 8.3.3.9, "DDR SDRAM Mode Programming" on page 464. The remaining SDRAM timing parameters required by the DDR SDRAM state machine are set in SDCR1.

Table 236. DDR SDRAM Control Register 0 - SDCR0 (Sheet 1 of 2)



Bit	Default	Description
31:28	0H	**tRAS:** Active to Precharge duration in MCLK periods
27	0_2	Reserved
26:24	000_2	**tRP:** Precharge Command Period in MCLK periods
23	0_2	Reserved
22:20	000_2	**tRCD:** Active to Read, Active to Write Period in MCLK periods
19:18	00_2	Reserved
17:16	01_2	**tEDP:** Data Path Latency in MCLK periods This field should be programmed to 10_2.
15:14	00_2	Reserved
13:12	00_2	tWDL: Write Data Latency in MCLK periods used by the Memory Controller state machine. See Equation 11. • 00 = 0 MCLK periods (tCAS = 2.5) - (DDR-II Type only) • 01 = 1 MCLK period (tCAS = 3) - (DDR-II Type only) - *not supported* • 10 = 2 MCLK periods (tCAS = 4) - (DDR-II Type only) • 11 = reserved
11:10	00_2	Reserved
09:08	00_2	**tCAS: CAS Latency:** Indicates the CAS Latency used by the Memory Controller state machine. • 00 = reserved • 01 = 2.5 MCLK periods (DDR-I Type only) • 10 = 3 MCLK periods (DDR-II Type only) - *not supported* • 11 = 4 MCLK periods (DDR-II Type only)
07:06	00_2	Reserved.

Table 236. DDR SDRAM Control Register 0 - SDCR0 (Sheet 2 of 2)

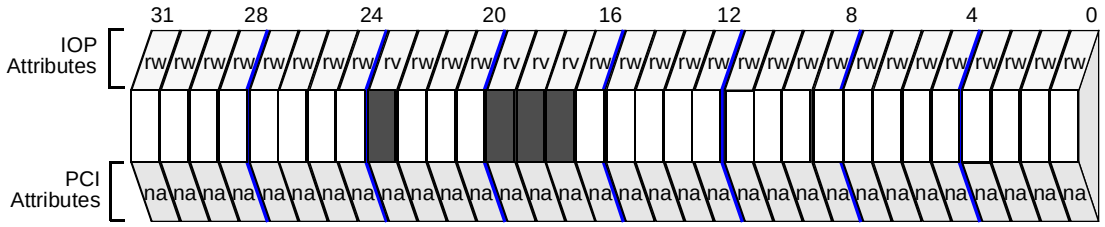


Bit	Default	Description
05:04	00_2	ODT Termination Value: Determines the termination value of the On Die Termination for both Banks (controlled by **ODT[1:0]**). Applies to DDR-II SDRAM memory type only. • 00 Disabled • 01 75 ohm • 10 150 ohm • 11 reserved
03	0_2	Reserved
02	Varies with external state of **MEM_TYPE** at PCI bus reset	**DDR Type:** Identifies the selected DDR generation of SDRAM based on the **MEM_TYPE** reset strap. 0 = DDR-II (supported speed of 400 MHz) - **MEM_TYPE** Deasserted. 1 = DDR (supported speed of 333 MHz) - **MEM_TYPE** Asserted.
01	0_2	**Data Bus Width:** Indicates the width of the data bus. See Section 8.3.3.4, "32-bit Data Bus Width" on page 458. 0 = 64 bits 1 = 32 bits
00	0_2	**DIMM Type:** Selects unbuffered or registered DIMM operating modes for the MCU. 0 = Unbuffered* 1 = Registered **NOTE:** Unbuffered DDR SDRAM memory subsystems will use the Unbuffered mode.

8.7.3 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 8.3.3.8, "DDR SDRAM Initialization" on page 461 and Section 8.3.3.9, "DDR SDRAM Mode Programming" on page 464. The SDCR1 specifies the remaining SDRAM timing parameters required by the DDR SDRAM state machine not specified in SDCR0.

Table 237. DDR SDRAM Control Register 1 - SDCR1



Intel® XScale™ Core Local Bus Address
FFFF E508H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	**DQS# Disable:** Controls the behavior of the strobes as well as the configuration of the DIMM. 0 = DQS# Enabled for Differential operation, EMRS bit 10 will be programmed as zero. 1 = DQS# Disabled for Singled-ended operation, EMRS bit 10 will be programmed as one.
30:28	000_2	**tRTCMD:** Read-to-Command (non-Read) turnaround period in MCLK periods. See Equation 7
27:24	0H	**tWTCMD:** Write-to-Command (non-Read) turnaround period in MCLK periods. See Equation 9
23	0_2	Reserved
22:20	000_2	**tRTW:** Read-to-Write turnaround period in MCLK periods. See Equation 7 **NOTE:** a programmed value of 000_2 represents a decimal value of eight (8).
19:17	000_2	Reserved
16:12	$0\ 0000_2$	**tRFC:** Refresh-to-Active and Refresh-to-Refresh period in MCLK periods
11:09	000_2	**tWR:** Write Recovery time in MCLK periods. • 000 = 0 - for DDR333 • 010 = 3 - for DDR-II 400 (encoding per JEDEC spec) all other values reserved
08:04	$0\ 0000_2$	**tRC:** Active-to-Active and Active-to-Refresh period in MCLK periods.
03:00	0H	**tWTRD:** Write-to-Read turnaround period in MCLK periods. See Equation 8

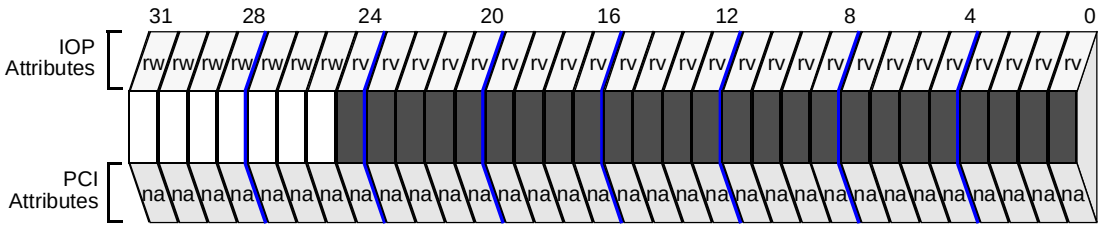
8.7.4 SDRAM Base Register - SDBR

This register indicates the beginning of SDRAM space. See Section 8.3.3.2, "DDR SDRAM Addressing" on page 452 for usage details. There can be 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 2 Gbyte boundary.

Note: This register should be read back after being written, before the Intel® XScale™ core performs transactions which address the DDR SDRAM.

Table 238. SDRAM Base Register - SDBR



Intel® XScale™ Core Local Bus Address
FFFF E50CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:25	0	**SDRAM Base Address:** These bits define the upper six bits of the DDR SDRAM base address.
24:00	0	Reserved

8.7.5 SDRAM Boundary Register 0 - SBR0

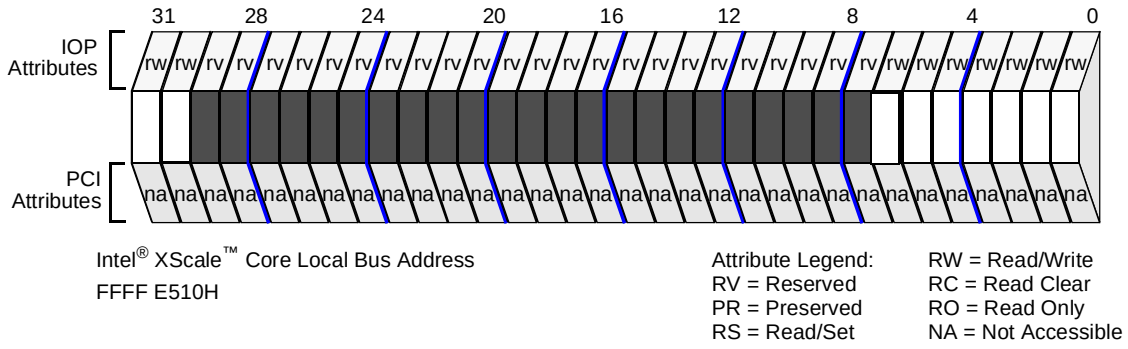
This register indicates the upper boundary of SDRAM bank 0 and its memory technology. If bank 0 is unpopulated, SBR0[6:0] is programmed either with all zeros or a number equal to the value in SDBR[30:25]. See Section 8.3.3.2, "DDR SDRAM Addressing" on page 452 for more details and programming examples.

Bank 0 may have multiple regions based on the value in "SDRAM 32-bit Region Size Register - S32SR" on page 504. See Section 8.3.3.2, "DDR SDRAM Addressing" on page 452 for more details

Note: DDR SDRAM memory space must ***never*** cross a 2 Gbyte boundary.

Note: This register should be read back after being written, before the Intel® XScale™ core performs transactions which address the DDR SDRAM.

Table 239. SDRAM Boundary Register 0 - SBR0



Bit	Default	Description
31:30	00_2	**SDRAM Address Translation:** Based on Table 220, "DDR SDRAM Address Decode Summary" on page 452. • 00 DDR SDRAM Address Translation #1 • 10 DDR SDRAM Address Translation #2 • 11 DDR SDRAM Address Translation #3 • 01 Reserved
30:07	000 0000H	Reserved
06:00	00000_2	**SDRAM Boundary:** Defines the upper limit of SDRAM bank 0.

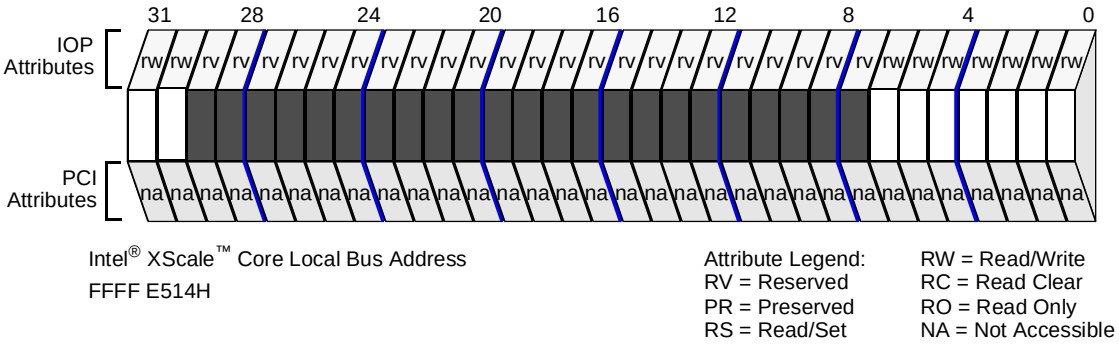
8.7.6 SDRAM Boundary Register 1 - SBR1

This register indicates the upper boundary of SDRAM bank 1 and its memory technology. If bank 1 is unpopulated, SBR1[6:0] is programmed either with all zeroes or a value equal to SBR0[6:0]. If bank 1 is populated, SBR1[6:0] must be programmed greater than or equal to SBR0[6:0]. See Section 8.3.3.2, "DDR SDRAM Addressing" on page 452 for more details and programming examples.

Note: DDR SDRAM Memory Space must never cross a 2 Gbyte boundary.

Note: This register should be read back after being written, before the Intel® XScale™ core performs transactions which address the DDR SDRAM.

Table 240. SDRAM Boundary Register - SBR1



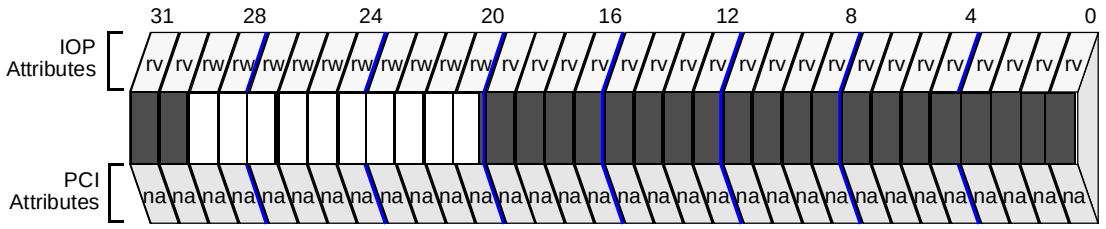
Bit	Default	Description
31:30	00_2	**SDRAM Address Translation:** Based on Table 220, "DDR SDRAM Address Decode Summary" on page 452. • 00 DDR SDRAM Address Translation #1 • 10 DDR SDRAM Address Translation #2 • 11 DDR SDRAM Address Translation #3 • 01 Reserved
30:07	0000 000H	Reserved
06:00	000000_2	**SDRAM Boundary:** Defines the upper limit of DDR SDRAM bank 1.

8.7.7 SDRAM 32-bit Region Size Register - S32SR

.Defines the size of the 32-bit region located at the base of SDRAM Bank 0.This register must be programmed with a size that is less than or equal to one half of the size of DDR SDRAM Bank 0. Sizes are limited to binary sizes with a minimum size of 1MB. Also, the DDR SDRAM type must be 64-bit, as defined by "SDRAM Control Register 0 - SDCR0" on page 498. See also "SDRAM Boundary Register 0 - SBR0" on page 502 and Section 8.3.3.2, "DDR SDRAM Addressing" on page 452.

This register should be read back after being written, before the Intel® XScale™ core performs transactions which address the DDR SDRAM.

Table 241. DDR SDRAM 32-bit Region Size Register - S32SR



Intel® XScale™ Core Local Bus Address
FFFF E518H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:30	00_2	Reserved
29:20	000H	32-bit Region Size - indicates the size of the 32-bit region at the base of Bank0 when a 64-bit data bus width is implemented. This size is also the size of the invalid region adjacent to the 32-bit region. 00 0000 0000 = no 32-bit region defined 00 0000 0001 - 1MB 00 0000 0010 - 2MB 00 0000 0100 - 4MB 00 0000 1000 - 8MB 00 0001 0000 - 16MB 00 0010 0000 - 32MB 00 0100 0000 - 64MB 00 1000 0000 - 128MB 01 0000 0000 - 256MB 10 0000 0000 - 512MB all other values are reserved

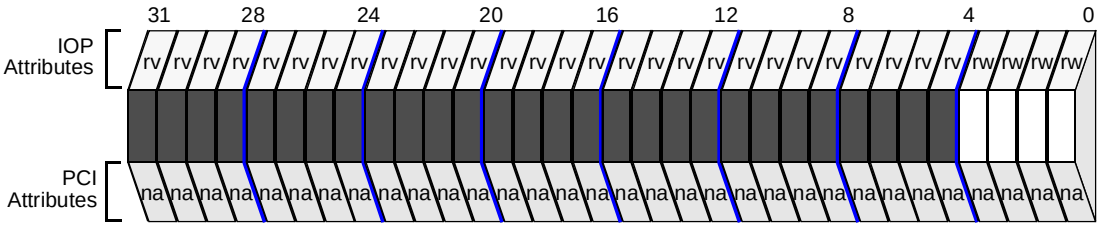
8.7.8 ECC Control Register - ECCR

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

For more details, see Section 8.3.4, "Error Correction and Detection" on page 476 and Section 8.5, "Interrupts/Error Conditions" on page 493.

Table 242. ECC Control Register - ECCR



Intel® XScale™ Core Local Bus Address
FFFF E51CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	000 0000H	Reserved
03	0_2	**ECC Enabled:** Enables ECC Read Modify Write sequence for ECC calculation and generation during sub 64-bit writes. See Section 8.3.4.5, "ECC Disabled". 0 = ECC Disabled (mode for Intel® XScale™ core ECC scrub) 1 = ECC Enabled (normal operation)
02	0_2	**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
01	0_2	**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
00	0_2	**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

8.7.9 ECC Log Registers - ELOG0, ELOG1

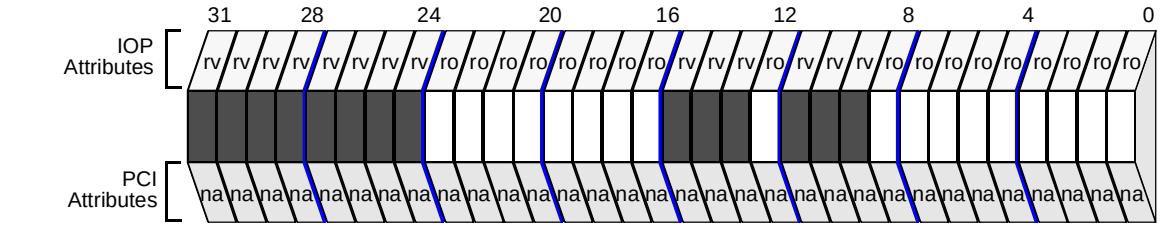
The ECC Log Registers are responsible for logging the error types detected on the local memory bus. Two errors can be detected and logged. The error type is logged (single-bit or multi-bit) along with the syndrome that indicated the error. For a single-bit error, software can read this syndrome and determine which bit had the error in order to perform scrubbing. For a multi-bit error, the syndrome will not match an entry in the H-Matrix and thus, is uncorrectable (see Table 231, "Syndrome Decoding" on page 481).

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 Section 8.3.4, "Error Correction and Detection" on page 476.

AR: MARK - close with Marc on need of syndrome

Table 243. ECC Log Registers - ELOG0, ELOG1



Error #	Intel® XScale™ Core Local Bus Address
0	FFFF E520H
1	FFFF E524H

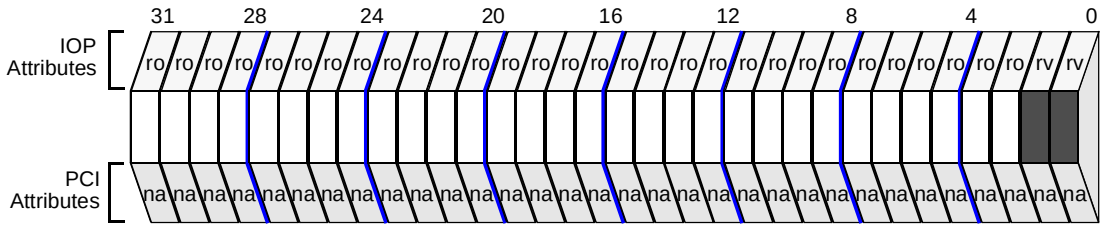
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:19	0	Reserved
23:16	00H	**ECC Error Requester: Indicates the requester of the logged error.** **Internal Bus Device ID (23:19) / Internal Bus Function (18:16) / Internal Bus Requester** 00010 / 000 / **Intel® XScale™ core BIU** 01010 / 001 / ATU 01101 / 000 / DMA Channel 0 **01101 / 001 / DMA Channel 1** **11010 / 000 / Application Accelerator** **All Other Codes Are Reserved** Note: Intel® XScale™ core Internal Bus Device ID and Internal Bus Function is used for both core transactions directed to the MCU via the Core MCU port or the IB port of the BIU.
15:13	000_2	Reserved
12	0_2	**Read or Write**: Indicates if the error occurred during a read or write transaction. 0 = Read error 1 = Write Error
11:09	000_2	Reserved
08	0_2	**ECC Error Type:** Indicates the type of error that occurred at this address. 0 = Single Bit Error 1 = Multi-Bit Error
07:00	00H	**Syndrome:** Holds the syndrome value that indicated the error.

8.7.10 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 can be detected and logged. The software knows which DDR SDRAM address had the error by reading these registers and decoding the syndrome in the log registers. For error details, see Section 8.3.4, "Error Correction and Detection" on page 476).

Table 244. ECC Address Registers - ECAR0, ECAR1



Error #	Intel® XScale™ Core Local Bus Address
0	FFFF E528H
1	FFFF E52CH

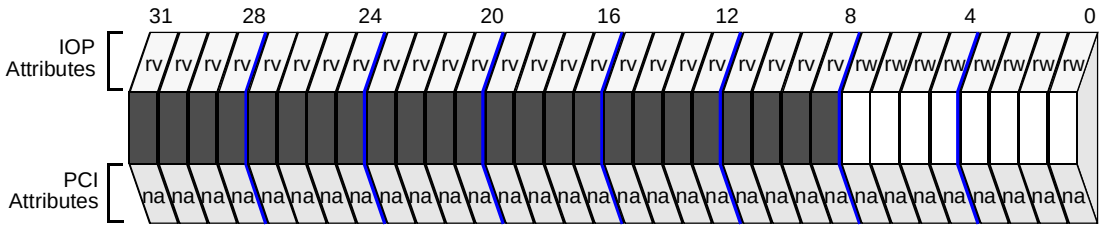
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:02	0	Error Address: Stores the upper 30 bits of the address that resulted in a single bit or multi-bit error.
01:00	000_2	Reserved

8.7.11 ECC Test Register - ECTST

This register allows testing between the ECC logic and the memory subsystem (Section 8.3.4.6, "ECC Testing" on page 486). 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.

Table 245. ECC Test Register - ECTST



Intel® XScale™ Core Local Bus Address
FFFF E530H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:08	00 0000H	Reserved
07:00	00H	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 Section 8.3.4.6, "ECC Testing" on page 486.

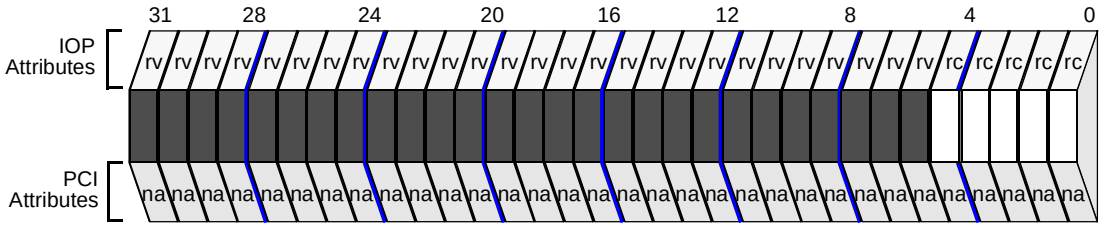
8.7.12 Memory Controller Interrupt Status Register - MCISR

Setting the MCISR asserts an interrupt to the core. Upon an interrupt, the Intel® XScale™ core 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 2:0 are read/clear bits which means that to clear them, software must write a one to these bits.

Table 246. Memory Controller Interrupt Status Register - MCISR



Intel® XScale™ Core Local Bus Address
FFFF E534H

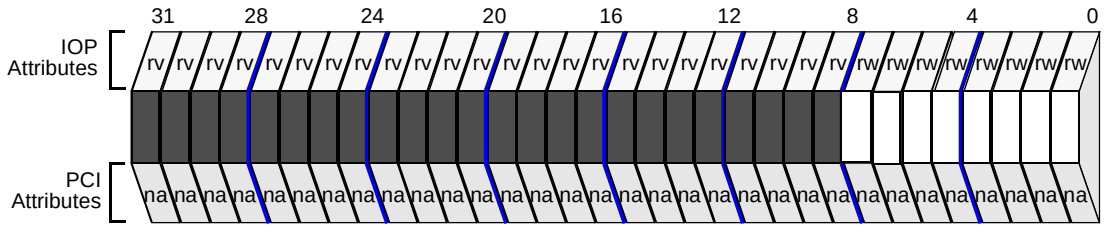
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:05	0000 000H	Reserved
04	0_2	**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
03	0_2	**Address Region Error:** Indicates that a transaction to the invalid address region created with a 32-bit Region. (see Section 8.3.3.2, "DDR SDRAM Addressing" on page 452) 0 = No error detected 1 = Error detected
02	0_2	**ECC Error N:** Indicates that the MCU detected an ECC error while MCISR[1] and MCISR[0] are both set. 0 = No error detected 1 = Error detected
01	0_2	**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
00	0_2	**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

8.7.13 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 a different port for DDR SDRAM transactions. This register along with the "MCU Preemption Control Register - MPCR" on page 511 used to optimize the memory controller operation.

Table 247. MCU Port Transaction Count Register - MPTCR



Intel® XScale™ Core Local Bus Address
FFFF E53CH

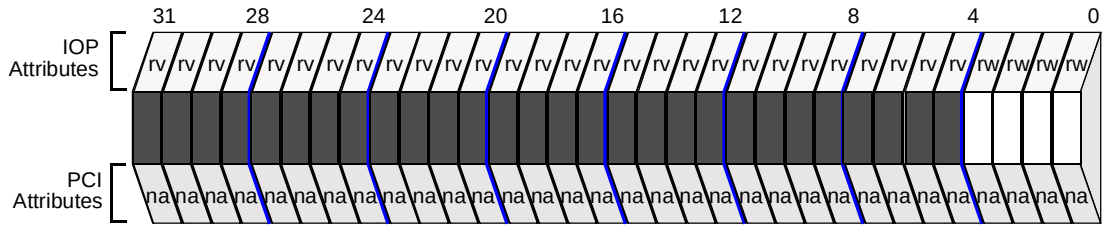
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:08	0	Reserved
07:04	1H	IB 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
03:00	CH	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

8.7.14 MCU Preemption Control Register - MPCR

Enables Preemption of IB port transaction when a core processor transaction is pending.

Table 248. MCU Preemption Control Register - MPCR



Intel® XScale™ Core Local Bus Address
FFFF E540H

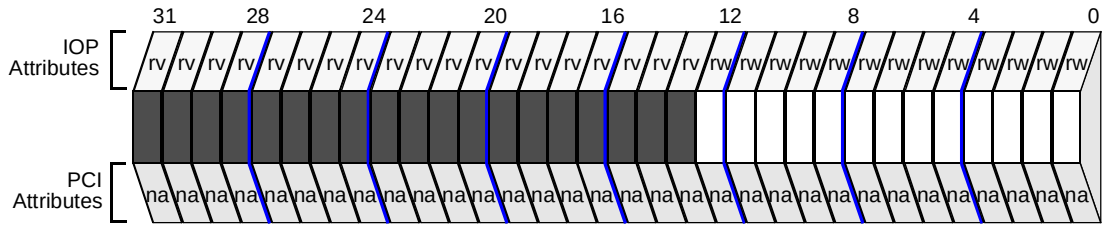
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	0	Reserved
03:00	0H	**Preemption Data Phase Count:** Specifies the number of DDR data bursts that will complete before the current transactions will be preempted. The count is based from the beginning of the transaction. When a core transaction port request is detected after the count has been exceeded by the current transaction, the transaction will be preempted at the next burst length boundary. The current transaction must be a non-Core transaction. 0H = Disabled 4H = Enabled for 4 bursts (16 data phases for Burst Length=4, or 128Bytes for 64-bit data bus) all other values are Reserved

8.7.15 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 Section 230, "Typical Refresh Frequency Register Values" on page 475 for recommended programmed values.

Table 249. Refresh Frequency Register - RFR



Intel® XScale™ Core Local Bus Address
FFFF E548H

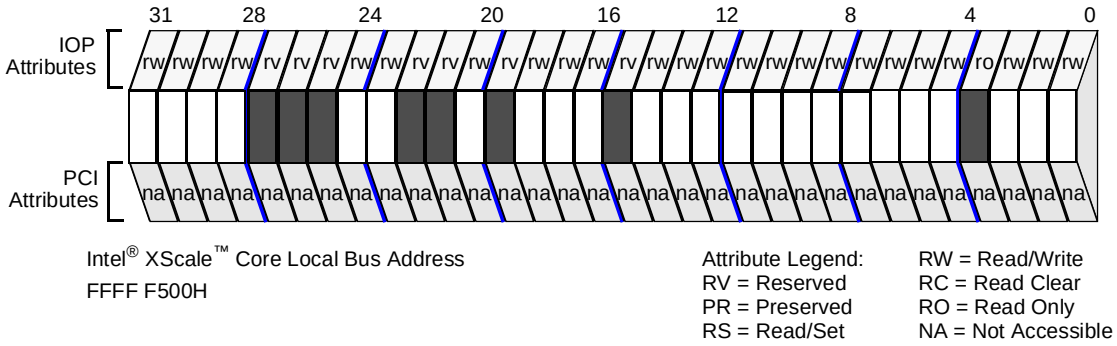
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:13	0	Reserved
12:00	000H	**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 Section 8.3.1.5, "Refresh Counter" on page 443. **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.

8.7.16 DCAL Control and Status Register - DCALCSR

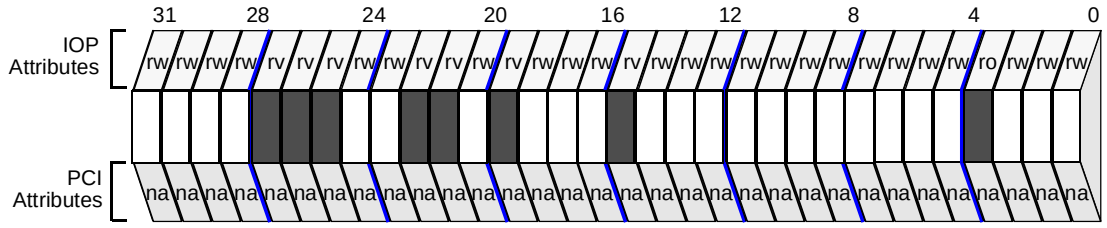
This 32 bit register controls and shows status for the DCAL operation.

Table 250. DCAL Control and Status Register - DCALCSR (Sheet 1 of 2)



Bit	Default	Description
31	0	When written: 0 = no action 1 = Start operation defined in this register When read: 0 = Operation completed 1 = Operation in progress
30:28	000	Pass Fail Indicators 000 - Pass x01 - SDRAM Access Denied x10 - Unpopulated Row Select x11 - Unsupported Opcode 1xx - Operation Completed with a Failure
27:25	000	Reserved
24	1	SDRAM I/F Select 0 = Not Selected - will result in Fail Indicator 001 in bits 30:28 1 = Selected
23	0	Operation Mode: applicable to the Receive Enable Calibration and DQS Calibration operations. 0 = One Pass - uses value in operation modifier field 1 = All Passes - cycles through all possible encodings
22:21	00	Reserved
20	0	Row Select (Chip Select) 0 = CS0# selected 1 = CS1# selected
19	0	Reserved 1 =

Table 250. DCAL Control and Status Register - DCALCSR (Sheet 2 of 2)



Intel® XScale™ Core Local Bus Address
FFFF F500H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

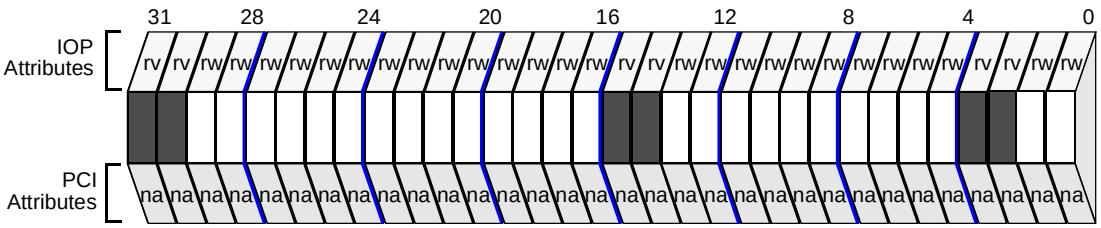
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
18:16	000	Fixed Data Pattern Selection: Only applicable to DQS Calibration operation. • 000 - F -> 0 -> F -> 0 • 001 - 0 -> F -> 0 -> F • 010 - A -> 5 -> A -> 5 • 011 - 5 -> A -> 5 -> A • 100 - C -> 3 -> C -> 3 • 101 - 3 -> C -> 3 -> C • 110 - 9 -> 6 -> 9 -> 6 • 111 - 6 -> 9 -> 6 -> 9
15	0	Reserved
14:4	000H	Opcode Modifiers EMRS OCD Calibration: • 14 - Reserved • 13:4 - Number of clock cycles to wait before collecting OCD Drive(0)/Drive(1) Samples. Receive Enable • 14:10 - Reserved • 9:4 - Receive Enable Delay - contain a 6-bit receive enable delay. Applicable for Single Pass operation only DQS Calibration • 14:12 - DLL Slave Length: used to set the coarse DLL delay adjustment, used in both single pass and all-pass modes • 11:8 - DLL Slave Mix: used to set the fine DLL delay adjustment. Applicable for Single Pass operation only. • 7:4 - Reserved 0 = Physical Address used for operation 1 = Logical Address used for operation
3	0	Reserved
2:0	0H	Opcode • 011 - EMRS OCD Calibration: requires the BA field (001b) and OCD Drive[0] or Drive[1] command is selected (MA[9:7] = 010b or 001b) in the DCALADDR register. • 100 – Receive Enable Calibration • 101 – DQS Calibration • All other values are reserved

8.7.17 DCAL Address Register - DCALADDR

This 32 bit register supplies address for the DCAL operation selected in Section 8.7.16, "DCAL Control and Status Register - DCALCSR" on page 513. The opcodes use this register to specify the row, column and bank address that should be driven to the DIMMs with the command.

Table 251. DCAL Address Register - DCALADDR



Intel® XScale™ Core Local Bus Address
FFFF F504H

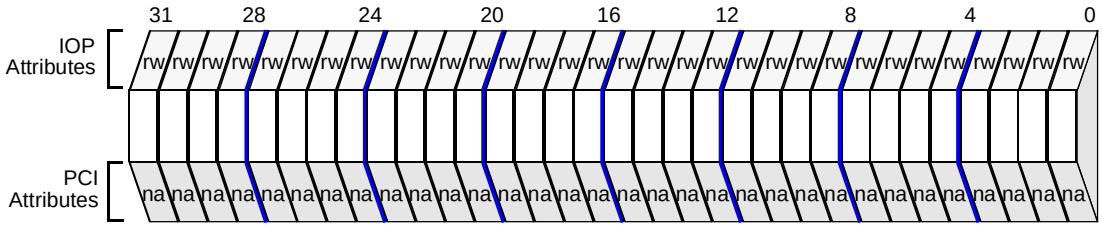
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:30	00	Reserved
29:16	0000H	Row Address: 14-bit address driven by the MCU on MA[13:0] pins for the activate (row) command for the Receive Enable Calibration and DQS Calibration operations and for the EMRS OCD Command.
15:14	00	Reserved
13:4	000H	Column Address: Used to construct the 14-bit address to be driven on the MA[13:0] pins for the read/write (column) command for the Receive Enable Calibration and DQS Calibration operations and for the EMRS OCD Calibration Command. Note, the MA[13, 1:0] pins are always driven to 0.
3:2	00	Reserved
1:0	00	Bank Address: 2-bit Bank address to be driven on the BA[1:0] pins for all operations.

8.7.18 DCAL Data Registers 17:0 - DCALDATA[17:0]

These ten 32-bit registers are used to support the different opcodes. The definition of these bits as with the address register changes depending on the operation.

Table 252. DCAL Data Registers 17-0 - DCALDATA[17:0]



DCALDATA	Intel® XScale™ Core Local Bus
0	FFFF F508H
1	FFFF F50CH
2	FFFF F510H
3	FFFF F514H
4	FFFF F518H
5	FFFF F51CH
6	FFFF F520H
7	FFFF F524H
8	FFFF F528H
9	FFFF F52CH
10	FFFF F530H
11	FFFF F534H
12	FFFF F538H
13	FFFF F53CH
14	FFFF F540H
15	FFFF F544H
16	FFFF F548H
17	FFFF F54CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0000 0000H	DCAL Data definition based on OPCODE

8.7.18.1 Opcode: EMRS OCD Adjust/Drive Commands

The DCAL Data Registers consist of Adjust and Drive fields for each DQS line. The fields are identical for each DQS line and are defined in Table 253. The location of the fields within the DCAL Data Registers is provided in Table 254 following the field definitions.

Adjust: A 4-bit value stored as 0000 $D_{T3}D_{T2}D_{T1}D_{T0}$ within the byte (MSB to LSB), which is the 4-bit data burst for the OCD adjust command as documented in the latest version of the JEDEC DDR-II Component specification.

Note: The bits are reversed, bit 0 in left hand column.

Table 253. OCD Adjust Field Encoding

4-bit burst codes inputs to all DQs				Operation	
DT0	DT1	DT2	DT3	Pull-up driver strength	Pull-down driver strength
0	0	0	0	NOP	NOP
0	0	0	1	Increase by 1 step	NOP
0	0	1	0	Decrease by 1 step	NOP
0	1	0	0	NOP	Increase by 1 step
1	0	0	0	NOP	Decrease by 1 step
0	1	0	1	Increase by 1 step	Increase by 1 step
0	1	1	0	Decrease by 1 step	Increase by 1 step
1	0	0	1	Increase by 1 step	Decrease by 1 step
1	0	1	0	Decrease by 1 step	Decrease by 1 step
Other combinations				Reserved	

Drive: An 8-bit value stored as $D_{1S1}D_{1S2}D_{1S3}D_{0S4}D_{0S1}D_{0S2}D_{0S3}D_{0S4}$ within the byte. The D_{1SX} values represent the 4 samples collected during the EMRS OCD Drive[1] operation and the D_{0SX} values represent the 4 samples collected during the EMRS OCD Drive[0] operation. After running the Drive[1] and Drive[0] commands, this 8-bit value should be 0xFF to guarantee that the drive strengths are sufficient. If Drive[1] samples do not equal 0xF, then the pull-up driver strength needs to be increased by (at least) one step with the OCD adjust command. If the Drive[0] samples do not equal 0xF, then the pull-down driver strength needs to be increased by (at lease) one step with the OCD adjust command.

Table 254 defines the usage of the DCAL Data Registers for the OCD Adjust/Drive fields.

Note: The Adjust fields must be duplicated per DQS as indicated. The Drive fields are also duplicated, and either field (odd or even byte) can be read for the corresponding DQS value.

Table 254. OCD Definition of DCALDATA[17:0] Registers

DCALDATA Register	Bits[31:24] - Byte3	Bits[23:16] - Byte2	Bits[15:8] - Byte1	Bits[7:0] - Byte0
17	Reserved		DQS8 - Adjust	DQS8 - Adjust
16			DQS8 - Drive	DQS8 - Drive
15:12	Reserved			
11	DQS7 - Adjust	DQS7 - Adjust	DQS5 - Adjust	DQS5 - Adjust
10	DQS3 - Adjust	DQS3 - Adjust	DQS1 - Adjust	DQS1 - Adjust
9	DQS6 - Adjust	DQS6 - Adjust	DQS4 - Adjust	DQS4 - Adjust
8	DQS2 - Adjust	DQS2 - Adjust	DQS0 - Adjust	DQS0 - Adjust
7:4	Reserved			
3	DQS7 - Drive	DQS7 - Drive	DQS5 - Drive	DQS5 - Drive
2	DQS3 - Drive	DQS3 - Drive	DQS1 - Drive	DQS1 - Drive
1	DQS6 - Drive	DQS6 - Drive	DQS4 - Drive	DQS4 - Drive
0	DQS2 - Drive	DQS2 - Drive	DQS0 - Drive	DQS0 - Drive

8.7.18.2 Opcode: Receive Enable Calibration

The DCAL Data Registers consist of Write pointer and Vector fields for each DQS line. The fields are identical for each DQS line and are defined below. The location of the fields within the DCAL Data Registers is provided following the field definitions.

Write Pointers (wrptr): A 5-bit value stored as 000x xxxx within the byte. The write pointers should be one-hot encoded value. The legal values are 0x01, 0x02, 0x04, 0x08, and 0x10.

Sampled High/Low Vectors: A 64-bit value stored in 2 consecutive registers that contain 1-bit per pass of the receive enable operation. Bit 0 represents the results from pass 0, bit 1 represents the results from pass 1, and so forth. By comparing the sampled high vector to the sampled low vector, the rising/falling edges of DQS can be located. In general, the pattern in each vector should repeat every 8 bits, except where the preamble is located. Legal values: bit x of one (or both) of the sampled high and low vectors should be equal to zero (both should never be equal to one). In other words, a bit-wise AND of the 2 vectors should result in all zeros.

Table 255. Receive Enable Calibration of DCALDATA[9] Register

Bits	Description
31:24	Expected write Pointer when DQS is sampled high. This is when the 1 cycle wide rcvcal pulse ANDed with the DQS signal produces a two small pulses that causes the write pointers to advance twice. The initial value of the write pointer is 0x01, so this value should be programmed to 0x04.
23:16	Expected write Pointer when DQS is sampled low. This is when the 1 cycle wide rcvcal pulse ANDed with the DQS signal produces a single pulse that causes the write pointers to advance once. The initial value of the write pointer is 0x01, so this value should be programmed to 0x02.
15:8	Reserved
7:0	DRAM I/F ORed wrptr: All of the channel A write pointers are logically ORed to produce this value, which is then compared to the expected sampled high/low write pointers to produce the sampled high/low vectors stored in DCALDATA[8:0]

Table 256 defines the usage of the DCAL Data Registers for the Receive Enable fields.

Table 256. Receive Enable Calibration Definition of DCALDATA[17:0] Registers

DCALDATA Register	Bits[31:24] - Byte3	Bits[23:16] - Byte2	Bits[15:8] - Byte1	Bits[7:0] - Byte0
17	Expected Write Pointer High Sample	Expected Write Pointer Low Sample	Reserved	DRAM I/F ORed wrptr
16	reserved			DQS8 - wrptr
15:12	Reserved			
11	reserved	DQS7 - wrptr	reserved	DQS6 - wrptr
10		DQS5 - wrptr		DQS4 - wrptr
9		DQS3 - wrptr		DQS2 - wrptr
8		DQS1 - wrptr		DQS0 - wrptr
7:4	Reserved			
3	Cumulative "Sampled High" Vector for passes (63:32			
2	Cumulative "Sampled High" Vector for passes (31:0)			
1	Cumulative "Sampled Low" Vector for passes (63:32)			
0	Cumulative "Sampled Low" Vector for passes (31:0)			

8.7.18.3 Opcode: DQS Calibration

The DCAL Data Registers consist of 16-bit field for each DQS line in DQS Calibration mode. The fields are identical for each DQS line and are defined in Table 257. The location of the fields within the DCAL Data Registers is provided following the field definitions in Table 258.

Note: Each nibble may have a different left and right edge value, but the calibration will be done across the byte.

Table 257. DQS Calibration Field Encodings for DCALDATA[17:0] Registers

Bits	Description
15:14	Right Edge Detection Indicator 11 - right edge found
13:8	Right edge slavelen[2:1] and slavemix[3:0] values
7:6	Left Edge Detection Indicator 11 - left edge found
5:0	Left edge slavelen[2:1] and slavemix[3:0] values

Table 258 defines the usage of the DCAL Data Registers for the Receive Enable fields.

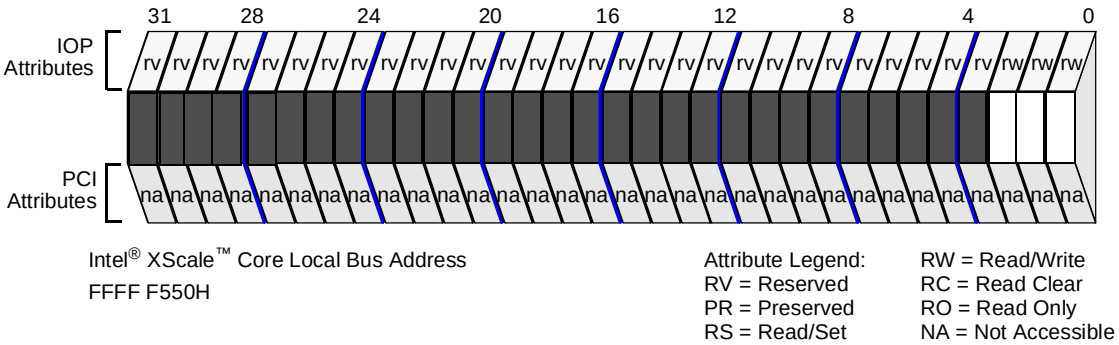
Table 258. DQS Definition of DCALDATA[17:0] Registers

DCALDATA Register	Bits[31:24] - Byte3	Bits[23:16] - Byte2	Bits[15:8] - Byte1	Bits[7:0] - Byte0
17	reserved		DQS8 - High Nibble	
16			DQS8 - Low Nibble	
15:12	Reserved			
11	DQS7 - High Nibble		DQS5 - High Nibble	
10	DQS3 - High Nibble		DQS1 - High Nibble	
9	DQS6 - High Nibble		DQS4 - High Nibble	
8	DQS2 - High Nibble		DQS0 - High Nibble	
7-4	Reserved			
3	DQS7 - Low Nibble		DQS5 - Low Nibble	
2	DQS3 - Low Nibble		DQS1 - Low Nibble	
1	DQS6 - Low Nibble		DQS4 - Low Nibble	
0	DQS2 - Low Nibble		DQS0 - Low Nibble	

8.7.19 Receive ENABLE Delay Register - RCVDLY

This 32 bit register consists of a packed 3-bit field for DQS Receive Enable Calibration.

Table 259. Receive Enabled Delay Register - RCVDLY

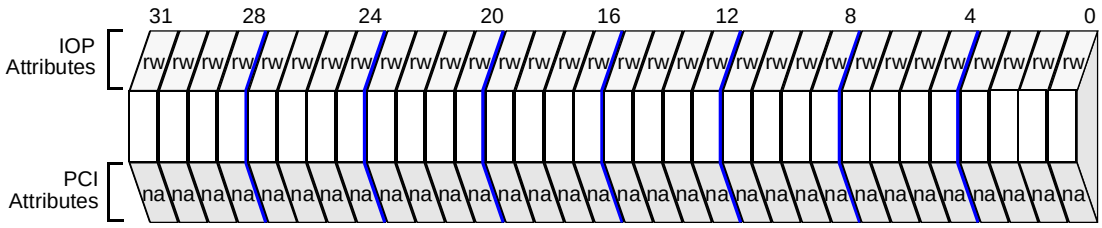


Bit	Default	Description
31:03	0000 0000H	Reserved.
2:0	101_2	Receive Enable Delay Value. This field should be programmed as follows • 010_2 - DDR-II • 101_2 - DDR-I (default)

8.7.20 Slave Low Mix 0 - SLVLMIX0

This 32 bit register consists of the first 8 of 9 packed 4-bit fields for dynamic DQS DLL delay for the low nibble.

Table 260. Slave Low Mix 0 - SLVLMIX0



Intel® XScale™ Core Local Bus Address
FFFF F554H

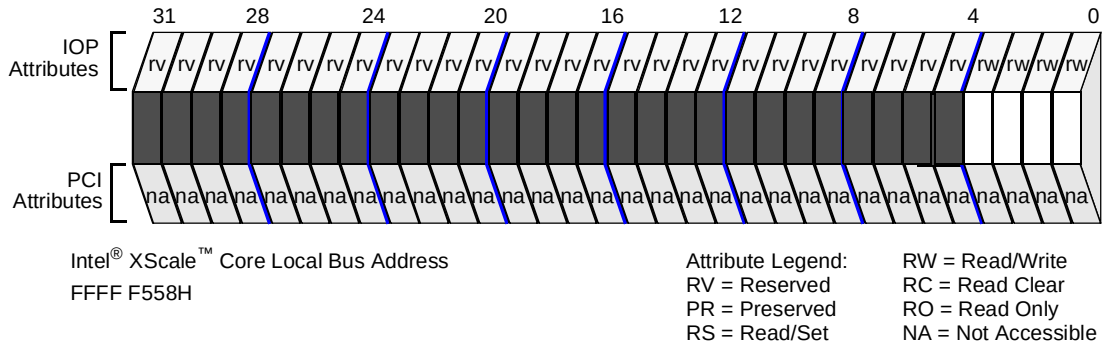
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	3H	Dynamic DQS DLL delay value for DQS7.
27:24	3H	Dynamic DQS DLL delay value for DQS6.
23:20	3H	Dynamic DQS DLL delay value for DQS5.
19:16	3H	Dynamic DQS DLL delay value for DQS4.
15:12	3H	Dynamic DQS DLL delay value for DQS3.
11:8	3H	Dynamic DQS DLL delay value for DQS2.
7:4	3H	Dynamic DQS DLL delay value for DQS1.
3:0	3H	Dynamic DQS DLL delay value for DQS0.

8.7.21 Slave Low Mix 1 - SLVLMIX1

This 32 bit register consists of the last 4-bit field for dynamic DQS DLL delay for the low nibble.

Table 261. Slave Low Mix 1 - SLVLMIX1

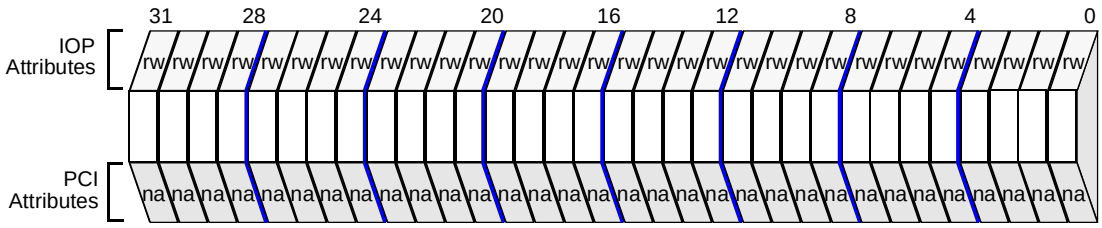


Bit	Default	Description
31:4	000 0000H	Reserved.
3:0	3H	Dynamic DQS DLL delay value for DQS8.

8.7.22 Slave High Mix 0 - SLVHMIX0

This 32 bit register consists of the first 8 of 9 packed 4-bit fields for dynamic DQS DLL delay for the high nibble.

Table 262. Slave High Mix 0 - SLVHMIX0



Intel® XScale™ Core Local Bus Address
FFFF F55CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

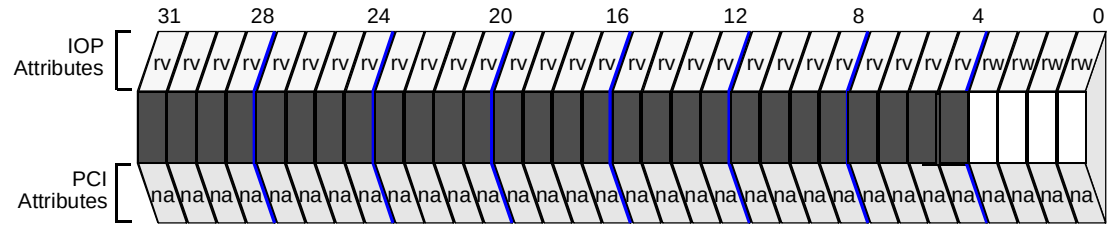
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	3H	Dynamic DQS DLL delay value for DQS7.
27:24	3H	Dynamic DQS DLL delay value for DQS6.
23:20	3H	Dynamic DQS DLL delay value for DQS5.
19:16	3H	Dynamic DQS DLL delay value for DQS4.
15:12	3H	Dynamic DQS DLL delay value for DQS3.
11:8	3H	Dynamic DQS DLL delay value for DQS2.
7:4	3H	Dynamic DQS DLL delay value for DQS1.
3:0	3H	Dynamic DQS DLL delay value for DQS0.

8.7.23 Slave High Mix 1 - SLVHMIX1

This 32 bit register consists of the last 4-bit field for dynamic DQS DLL delay for the high nibble.

Table 263. Slave High Mix 1 - SLVHMIX1



Intel® XScale™ Core Local Bus Address
FFFF F560H

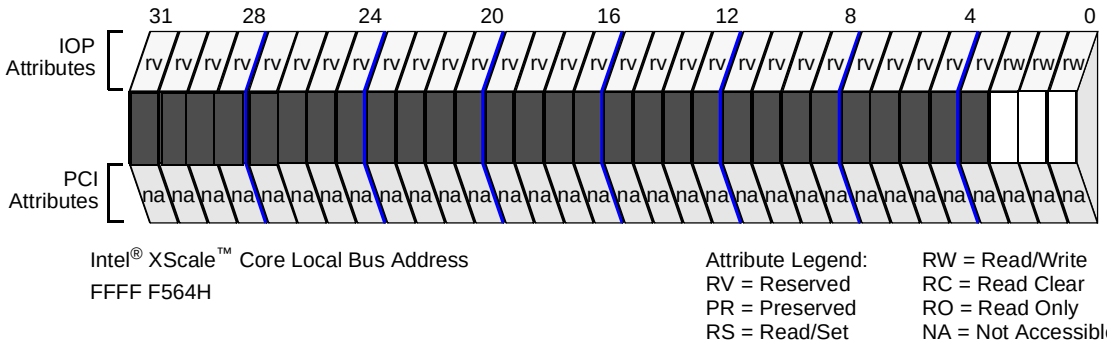
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:4	000 0000H	Reserved.
3:0	3H	Dynamic DQS DLL delay value for DQS8.

8.7.24 Slave Length - SLVLEN

This 32 bit register consists of a packed 3-bit field for Static DQS Slave DLL length.

Table 264. Slave Length - SLVLEN

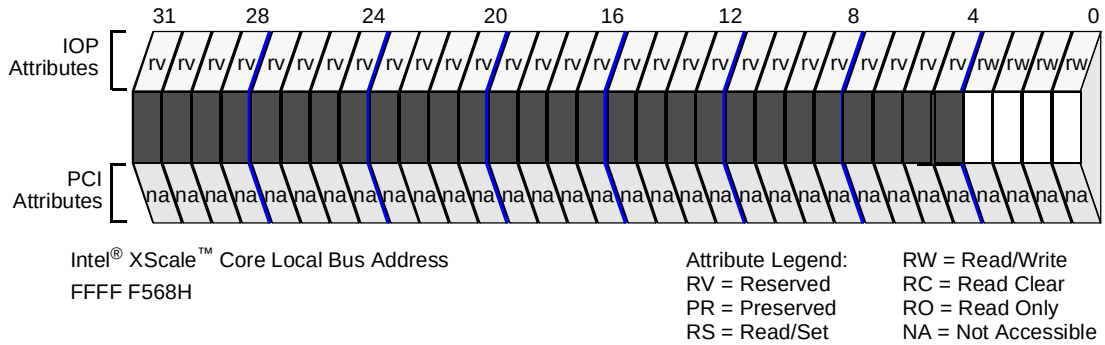


Bit	Default	Description
31:3	0000 0000H	Reserved.
2:0	110_2	Static DQS slave DLL length.

8.7.25 Master Mix - MASTMIX

This 32-bit register consists of a 4-bit field for master DQS DLL delay loop mixer control.

Table 265. Master Mix - MASTMIX

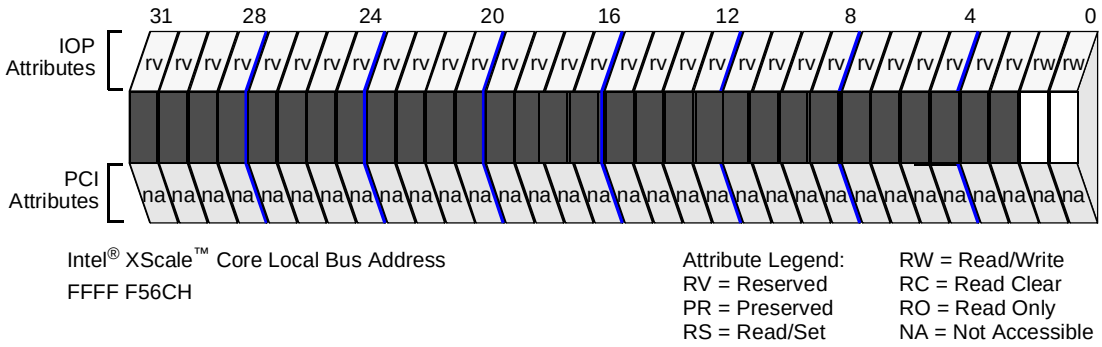


Bit	Default	Description
31:4	000 0000H	Reserved.
3:0	3H	Master DQS DLL delay loop mixer.

8.7.26 Master Length - MASTLEN

This 32 bit register consists of the nine 2-bit field for master DQS DLL length control.

Table 266. Master Length- MASTLEN



Bit	Default	Description
31:02	0000 0000H	Reserved.
1:0	10_2	Master DQS DLL length control.

8.7.27 DDR Drive Strength Status Register - DDRDSSR

This 32-bit register consists of a 4-bit field for drive strength value.

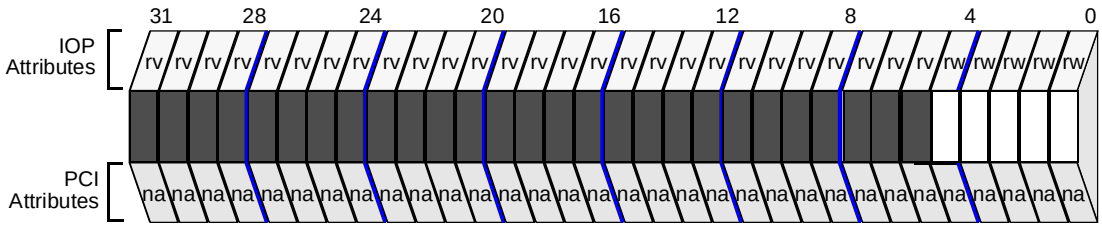
Table 267. DDR Drive Strength Status Register - DDRDSSR

Bit	Default	Description
31:4	000 0000H	Reserved.
3:0	unknown	DDR Drive Strength Value: Value is based on I/O hardware and board loading characteristics. (and may not equal the DDR Drive Strength Hint value from the DDRDSCR)

8.7.28 DDR Drive Strength Control Register - DDRDSCR

This 32-bit register consists of a 5-bit field to override the DDR pad automatic drive strength control mechanism.

Table 268. DDR Drive Strength Control Register - DDRDSCR



Intel® XScale™ Core Local Bus Address
FFFF F574H

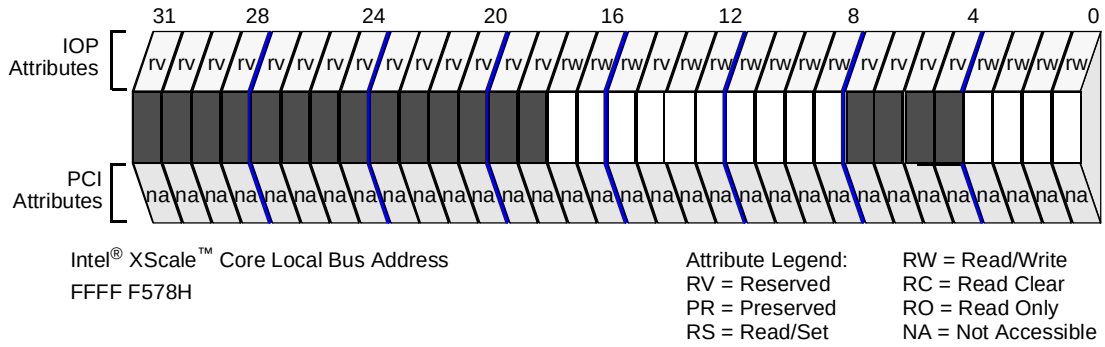
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:5	000 0000H	Reserved.
Overdrive Enable = 0		
4	0_2	Overdrive Enable 0 = Disabled: automatic pad control used for drive strength value. 1 = Enabled: bits 3:0 used for pad control
3:0	1001_2	Drive Strength Value Hint: Value used by DDR pad control drive strength.
Overdrive Enable = 1		
4	0_2	Overdrive Enable 0 = Disabled: automatic pad control used for drive strength value. 1 = Enabled: bits 3:0 used for pad control
3	n/a	Pad Control Selection. (Valid when Overdrive Enable is set) 0 = Pad control uses Drive Strength Overdrive Value (bits 2:0). 1 = Locks current drive strength value determined from automatic pad control (i.e. freezes automatic control). Value reflected in DDRDSSR.
2:0	n/a	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.

8.7.29 DDR Miscellaneous Pad Control Register - DDRMPCR

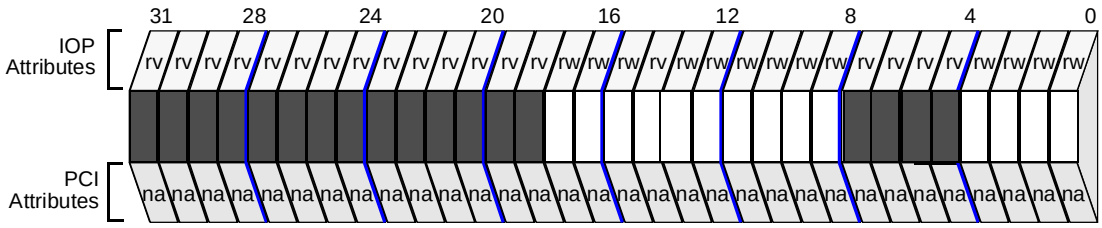
This 32-bit register contains miscellaneous control signals for the DDR pads

Table 269. DDR Miscellaneous Pad Control Register - DDRMPCR



Bit	Default	Description
31:18	0000H	Reserved
17:16	10	read pointer delay: Determines the read pointer delay for DCAL, which is based on DIMM topology and technology. Command Clocks per Frequency DDR-I 333 DDR-II 400 • 00 0 (0 ns) 0 (0 ns) • 01 1 (6 ns) 1 (5 ns) • 10 2 (12 ns) 2 (10 ns) • 11 3 (18ns) 3 (15 ns)
15	0	Fast slew rate control: allows extended range on slew rate: Set high for DDR-II, low for DDR-I
14	0	Half gain control: Select for DQS differential amplifier gain. Set 1 to cut gain in half for differential strobe mode associated with DDR-II.
13	0_2	Leg test mode: When set, Places output buffer strength control in a test mode which will allow testing of each leg of a buffer driver independently.This can be done by placing the DDR pads in slew rate override mode (DDRDSCR register). Each value of drive strength override (bits 2:0 in DDRDSCR) will enable an individual driver leg in each DDR pad.
12	0_2	VOX Start: Back-up override for the voltage output crossing control loop.
11	0_2	Slew Rate Override: Slew rate override adjustment for analog validation
10	0_2	DLL Bypass: Bypasses DLL for calibration on DDR bus
9	0_2	OCD Load Enable: Calibration buffer load placed on incoming signals to perform calibration. Required for OCD calibration when DDR-II ODT mode is in use

Table 269. DDR Miscellaneous Pad Control Register - DDRMPCR



Intel® XScale™ Core Local Bus Address
FFFF F578H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
8	0_2	Analog Validation Mode (behavior TBD)
7:4	0000_2	Reserved.
3:0	0000_2	VREF Select: Controls the VREF selection/generation mechanism for DDR pads. VREF calibration is a required BIOS function 0 External VREF from bump (default) 1 Internal VREF - 250mV 2 Internal VREF - 200mV 3 Internal VREF - 150mV 4 Internal VREF - 100mV 5 Internal VREF - 50mV 6 Internal VREF - 10mV 7 Internal precision VREF 8 Internal VREF + 10mV 9 Internal VREF + 50mV 10 Internal VREF + 100mV 11 Internal VREF + 150mV 12 Internal VREF + 200mV 13 Internal VREF + 250mV 14 Pull-down Calibration Mode 15 Pull-up Calibration Mode

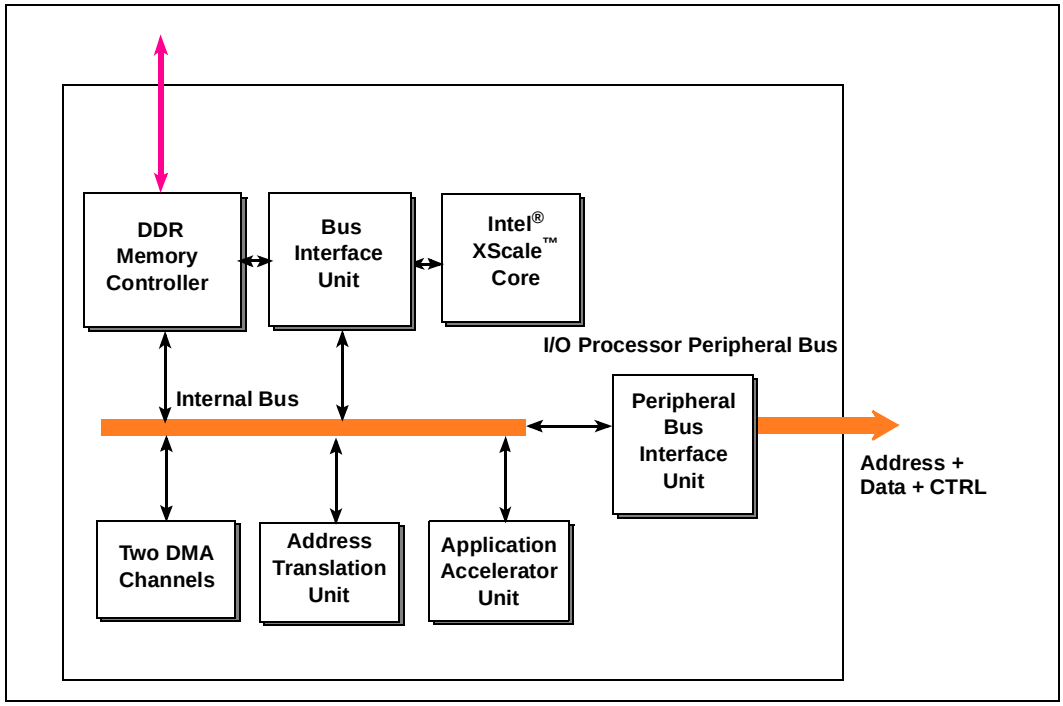
Peripheral Bus Interface Unit 9

This chapter describes the Peripheral Bus Interface Unit (PBI) of the Intel® 80331 I/O processor (80331). It explains the following:

- Peripheral Bus signals, which consist of address/data, control/status.
- Peripheral Bus Read, and write transactions.
- Peripheral Bus configuration and Flash Memory Support.
- Support for Intel® XScale™ core (ARM* architecture compliant) boot from the PCI Bus.
- Registers.

This chapter also serves as a starting point for the hardware designer when interfacing typical flash components to the 80331 Peripheral Bus.

Figure 85. The Peripheral Bus Interface Unit



9.1 Overview

The Peripheral Bus Interface Unit (PBI) is a data communication path to the flash memory components and peripherals of a 80331 hardware system. The PBI allows the processor to read and write data to these supported flash components and other peripherals. To perform these tasks at high bandwidth, the bus features a burst transfer capability which allows successive 8- or 16-bit data transfers.

The peripheral bus is controlled by the on-chip bus masters: the Intel® XScale™ core, the ATU, AAU and DMA units.

The address/data path is multiplexed for economy, and the bus width is programmable to 8-, and 16-bit widths. The PBI performs the necessary packing and unpacking of bytes to communicate properly across the 80331 Internal Bus.

The PBI unit includes two chip enables.The PBI chip enables activate the appropriate peripheral device when the address falls within one of the PBI's two programmable address ranges. Both address ranges incorporate functionality that optimizes an interface for Flash Memory devices.

9.2 Peripheral Bus Signals

Bus signals consist of two groups: address/data, and control/status.

9.2.1 Address/Data Signal Definitions

The address/data signal group consists of 26 lines. 16 of these signals multiplex within the processor to serve a dual purpose. During and address cycle (T_A), the processor drives **A[22:16]** and **AD[15:0]** with the address of the bus access. At all other times, the **AD[15:0]** lines are defined to contain data. **A[2:0]** are demultiplexed address pins providing incrementing byte addresses during burst cycles.

9.2.2 Control/Status Signal Definitions

The control/status signals control peripheral device enables and direction. All output control/status signals are three-state.

The PBI pulses **ALE** (address latch enable) active high for one clock during T_A to latch the multiplexed address on **AD[15:2]** in external address latches.

A peripheral read may be either non-burst or burst. A non-burst read ends after one data transfer to a single location.

When the data bus is configured for 16 bits, demultiplexed address bits **A[2:1]** are used to burst across up to four short-words. For an 8-bit data bus, demultiplexed address bits **A[1:0]** are used to burst across up to four bytes.

The Output Enable, **POE#**, is used for burst or non-burst read accesses to a peripheral device and is asserted during the $T_A/T_W/T_D$ states.

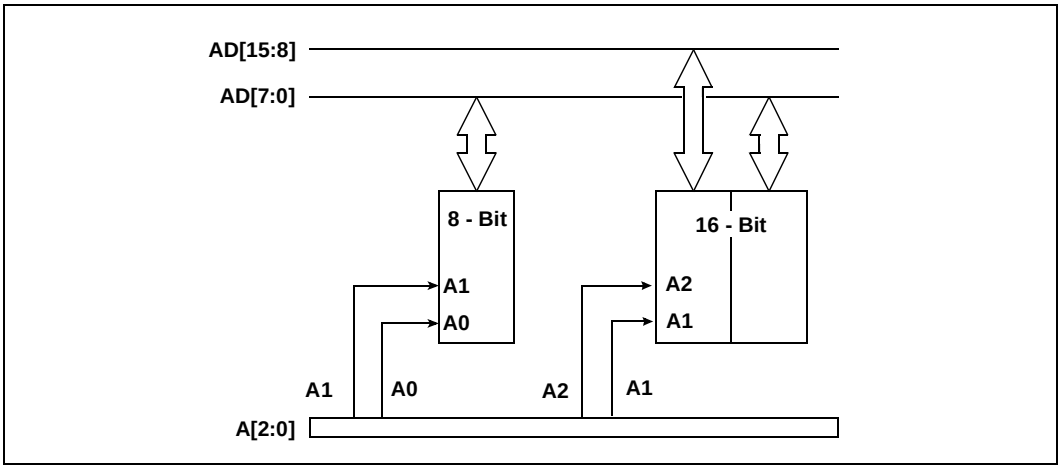
The Write Enable, **PWE#**, is used for non-burst write accesses to a peripheral device and is asserted during the T_W/T_D states.

Note: Burst write accesses to Flash Devices are not supported.

9.2.3 Bus Width

Each address range's attributes are programmed in the PBIs boundary registers. The PBI allows an 8-, or 16-bit data bus width for each range. The PBI places 8- and 16-bit data on low-order data signals, simplifying the interface to narrow bus external devices. As shown in Figure 86, 8-bit data is placed on lines **AD[7:0]**; 16-bit data is placed on lines **AD[15:0]**.

Figure 86. Data Width and Low Order Address Lines



The user needs to wire up the flash memories in a manner consistent with the programmed bus width:

- 8-bit region: **A[1:0]** provide the demultiplexed byte address for a read burst.
- 16-bit region: **A[2:1]** provide the demultiplexed short-word address for a read burst.

During initialization, bus width is selected for each of the two address ranges in the Peripheral Base Address Registers (PBBAR0 - PBBAR1). In addition, the PBBAR0-PBBAR5 can be used to configure these ranges as Peripheral Windows and to set a Wait state profile.

The PBI drives determinate values on all address/data signals during T_W/T_D write operation states. For an 8-bit bus, the PBI continues to drive address on unused data signals **AD[15:8]**.

9.2.4 Detailed Signal Descriptions

Bus signal descriptions are detailed in Table 271.

Table 270. Bus Signal Descriptions

NAME	DESCRIPTION
A[22:16]	**ADDRESS BUS 22:16** carries a demultiplexed version of address bits **A[22:16]**. During address (T_a), wait state (T_w) and data cycles (T_d) cycles, **A[22:16]** represents the upper 7 address bits for the current access. **A[22:16]** allows the PBI interface to address up to 8 MBytes per peripheral device.
AD[15:0]	**ADDRESS / DATA BUS** carries 16-bit physical addresses and 8-, or 16-bit data to and from memory. During an address (T_a) cycle, bits 2-15 contain a physical word address (bits 0-1 indicate SIZE; see below). During a data (T_d) cycle, bits 0-7, or 0-15 contain read or write data, depending on the corresponding bus width. During write operations to 8-bit wide memory regions, the PBI drives unused bus pins high or low. SIZE, which comprises bits 0-1 of the AD lines during a T_a cycle specifies the number of data transfers during the bus transaction. AD1 AD0 0 0 1 Transfer 0 1 2 Transfers 1 0 3 Transfers 1 1 4 Transfers
A[2:0]	**ADDRESS BUS 2:0** carries a demultiplexed version of bits 2:0 of the **AD[15:0]** bus. During a bursted read data (T_d) cycle, **A[2:0]** represents the current byte address in the bursted transaction. **A[2:1]** are used for an 16-bit wide peripheral while A[1:0] are used for an 8-bit wide peripheral.
ALE	**ADDRESS LATCH ENABLE** indicates the transfer of a physical address. ALE is asserted during a T_a cycle and deasserted before the beginning of the T_d state.
POE#	**PERIPHERAL OUTPUT ENABLE** specifies, during a T_a cycle, whether the operation is a write (1) or read (0). It is latched on-chip and remains valid during T_d cycles. This signal is used as an OUTPUT ENABLE signal (OE#) for Peripheral Devices.
PCE[1:0]#	**PERIPHERAL CHIP ENABLES 1:0** specify, during a T_a cycle, which of the two Memory Address Ranges are associated with the current bus access. It remains valid during T_d cycles
PWE#	**PERIPHERAL WRITE ENABLE** indicates to a peripheral device whether or not to use the data on the AD15:0 bus to write the addressed space. It is low during T_w cycles and deasserts during the T_d cycle for a write; it is high during T_a and T_w/T_d cycles for a read.

9.2.5 Flash Memory Support

PBI peripheral bus interface supports 8-, or 16- bit Flash devices.

The PBI provides programmable wait state functionality for peripheral memory windows.

Note: Potentially, programmable wait state functionality could be connected to any peripheral device that has a deterministic wait state profile. However, data valid and turn-around times would need to fit within parameters provided by programmable wait state profiles to support Flash devices.

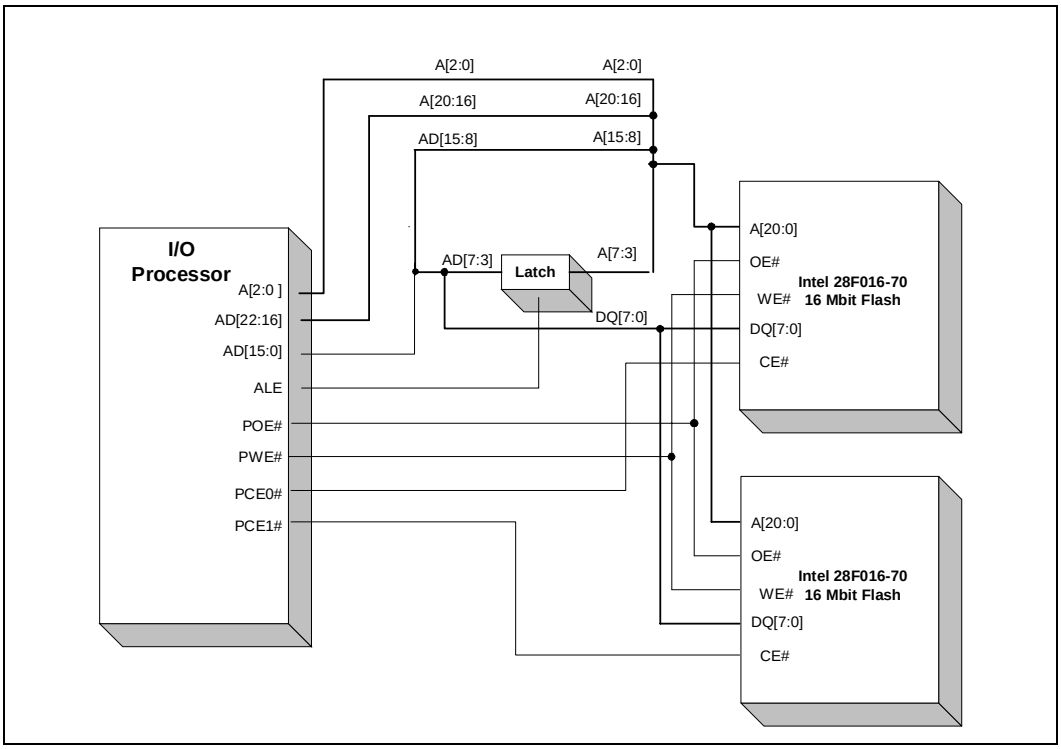
Any write transactions issued to a Flash address space window must always represent a single flash bus data cycle (**strb, strh**).

The peripheral chip enables, **PCE[1:0]#**, activate the appropriate Peripheral window when the address falls within one of the Peripheral address ranges.

Note: By default, bank 0 is enabled with the maximum number of Address-to-Data and Recovery Wait states. The width of the interface can be strapped for either 8-bit wide Flash or 16-bit wide flash. Thus, **PCE0#** is the Peripheral Bus chip enable to be used for booting purposes.

Figure 87 on page 538 illustrates how two 8-bit Flash devices would interface with the 80331 through the PBI Interface.

Figure 87. Four Mbyte Flash Memory System[a]



a. 16-bit wide flash devices would require two latches.

9.2.5.1 Flash Read Cycle

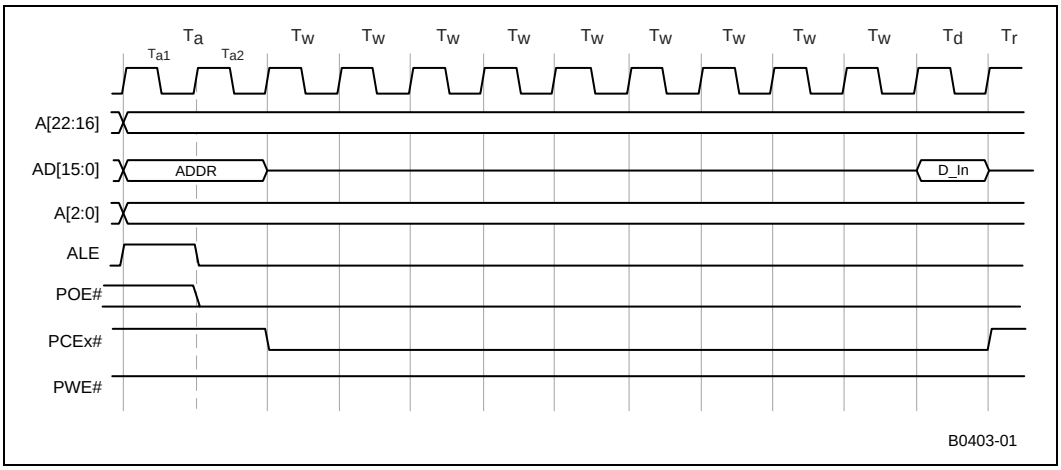
Reading a Flash device involves driving the address, output enable, and chip enable. Depending on the speed of the Flash device, the data returns several cycles later.

The definition of address-to-data wait states are the number of cycles between the assertion of **PCE[1:0]#**, and the arrival of data from the Flash device on **AD[7:0]** (8-bit Flash). The definition of recovery wait states are the number of cycles between the data arrival on **AD[7:0]** and the address for the next Peripheral transaction.

Address-to-data and recovery wait states are programmed in PBBAR0 and PBBAR1 and are identical for reads and writes. Since the read wait state requirement is typically greater, the write wait state requirement is guaranteed to be met.

Refer to Figure 88 illustrates a read cycle for a 120 ns Flash device.

Figure 88. 120 ns Flash Read Cycle



Refer to Table 271 for the programmable address-to data and recovery wait states. These numbers are based on a 66 MHz internal clock for the PBI interface.

Table 271. Flash Wait State Profile Programming[a]

Flash Speed	Address-to-Data Wait States	Recovery Wait States
<= 55 ns	4	0
<= 115 ns	8	2
<= 150 ns	10	2

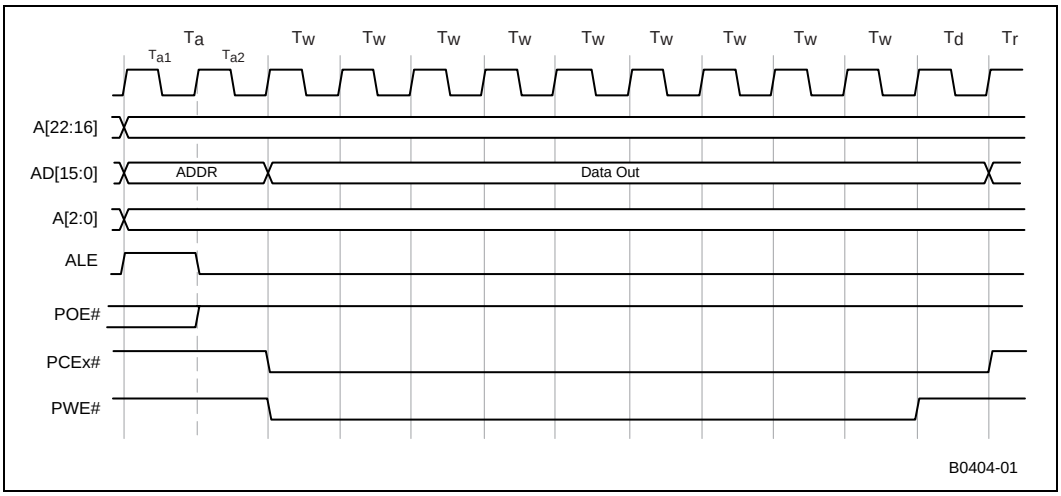
a. Each Wait State Represents 15 ns.

9.2.5.2 Flash Write Cycle

Address-to-data and recovery wait states for reads and writes are identical and programmed in FBBAR0 and FBBAR1. Refer to Table 271 for the programmable address-to data wait states. However, Any write transactions issued to a Peripheral address space window must always represent a single peripheral bus data cycle (**strb, strh**) depending on the bus width selected in PBBAR0 - PBBAR1. Bursting is not supported to a Flash device since the Flash device has no write buffers to support bursting data.

Figure 89 illustrates a write cycle to a 120 ns Flash device.

Figure 89. 120 ns Flash Write Cycle



9.3 Intel® XScale™ Core PCI Memory Boot Support

When PBI Window 0 is disabled (Section 9.4.4, "PBI Limit Register 0 - PBLR0" on page 546) and the Intel® XScale™ core PCI Bus Boot Enable in the PBCR (Section 9.4.1, "PBI Control Register - PBCR" on page 543) is set, the PBI provides the ability for the Intel® XScale™ core to boot (Section 15.3.3, "Exception Priorities and Vectors" on page 678) from PMMR register space. When the mode is enabled, three registers is available at the Internal Bus address 0000 0000H, 0000 0020H, and 0000 0024H, otherwise these registers are accessible only from their normal PMMR register addresses FFFF E6C0H, FFFF E6E0H, and FFFF E6E4H, respectively.

To boot from PCI memory, the user needs to configure the 80331 power-on-reset straps to hold the Intel® XScale™ core in reset (PCSR bit 1 is set) and to allow a host processor to configure the 80331 PCI interface (PCSR bit 2 is clear) (see Table 125, "PCI Configuration and Status Register - PCSR" on page 253 for details).

Following the host configuration of the 80331 PCI interface, the user can boot using code that resides on the PCI bus (typically host memory) through an Outbound Memory window. When the Intel® XScale™ core is released from reset (PCSR bit 1 is cleared), the PBI provides facilities for the core to execute a short jump or a long jump from the reset vector to any of the Outbound Memory windows including the direct addressing window.

Example 6. Memory-less Boot through by Primary ATU Outbound Memory Window 0

1. Disable or Remap PBI Memory Window 0 which is the default window used to boot from Flash. Write to PBLR0 to disable the PBI Memory Window 0. Remap PBI Memory Window 0 to an address above 0x3FH by writing to PBBAR0.
2. Set the Intel® XScale™ core PCI Bus Boot Enable bit (PBCR bit 3).
3. Write a short branch into PMBR0 (Intel® XScale™ core Reset exception vector). Typically, PMBR0 is written with a short branch to 0000 0020H (PMBR1) (see Section 9.4.7, "PBI Memory-less Boot Registers 2:0 - PMBR[2:0]" on page 549).
4. Write a long branch into the PMBR1/2 registers. PMBR1 holds the branch instruction while PMBR2 represents the 32-bit branch vector.
5. Write the 64 Mbyte aligned Host Memory Address where the Intel® XScale™ core boot code resides into the OMWTVR0 register (Section 3.10.34, "Outbound Memory Window Translate Value Register 0 - OMWTVR0" on page 247).
6. When the Host Memory Address resides above the 4 Gbyte address boundary, write the upper 32 bits of the Host Memory Address into the OUMWTVR0 register (Section 3.10.35, "Outbound Upper 32-bit Memory Window Translate Value Register 0 - OUMWTVR0" on page 248).
7. Release the Intel® XScale™ core to boot from host memory by clearing the core processor reset bit in the PCSR (PCSR bit 1).

Note: The Long Branch at 0000 0020H (PMBR1/2) should branch to the 64 Mbytes of address space that is claimed by the ATUs Primary Outbound Memory Window 0 (8000 0000H to 83FF FFFFH). With the MMU enabled following boot-up, the exception vector's physical addresses can be moved from the address range 0000 0000H to 0000 001CH to an address range contained in the 64 Mbytes of Primary Outbound Memory Window 0. Typically, the exception vectors is locked into the cache following boot-up.

9.4 Register Definitions

A series of configuration registers control the PBI. Software can determine the status of the PBI by reading the status register. Table 272 lists all of the PBI registers which are detailed further in proceeding sections.

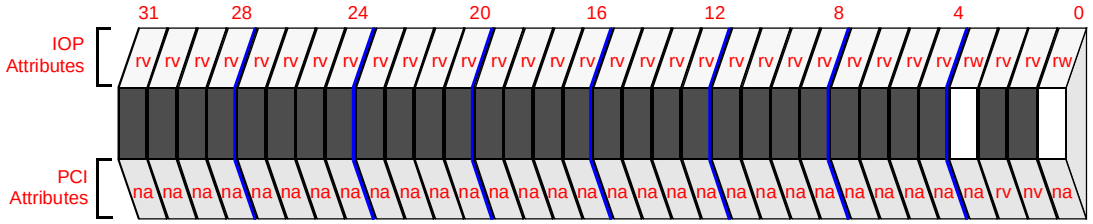
Table 272. Peripheral Bus Interface Register

Section, Register Name - Acronym (Page)
Section 9.4.1, "PBI Control Register - PBCR" on page 543
Section 9.4.3, "PBI Base Address Register 0 - PBBAR0" on page 545
Section 9.4.4, "PBI Limit Register 0 - PBLR0" on page 546
Section 9.4.5, "PBI Base Address Register 1 - PBBAR1" on page 547
Section 9.4.6, "PBI Limit Register 1 - PBLR1" on page 548
Section 9.4.7, "PBI Memory-less Boot Registers 2:0 - PMBR[2:0]" on page 549
Section 9.4.8, "PBI Drive Strength Control Register - PBDSCR" on page 550

9.4.1 PBI Control Register - PBCR

The PBI Control Register (PBCR) is responsible for enabling the operation of the PBI state machines.

Table 273. PBI Control Register - PBCR



Intel® XScale™ Core Local Bus Address
FFFF E680H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	0	Reserved
03	0	**Intel® XScale™ Core PCI Bus Boot Enable:** When set, this bit enables access to the PBI Memory-less boot registers **PMBR[2:0]** at locations 0000 0000H, 0000 0020H, and 0000 0024H, when PBI Memory Window 0 is disabled. Please see **Section 9.3, "Intel® XScale™ Core PCI Memory Boot Support" on page 541** for more details on the PCI Memory boot sequence.
02:01	10	Reserved (bit 2 reads as '1')
00	1_2	**PBI Enable:** When set, this bit enables the PBI unit bus interface 0 = Peripheral Bus Disabled 1 = Peripheral Bus Enabled.

9.4.2 Determining Block Sizes for Memory Windows

The memory window size can be determined by writing ones to the appropriate upper bits of the limit register. The binary-weighted value of the first non-zero bit set in the limit register indicates the size of the memory window. Table 274 describes the relationship between limit register values and the byte sizes of the memory window.

Table 274. Memory Block Size Limit Register Value

Limit Register Value[a]	Size (in Bytes)	Limit Register Value	Size (in Bytes)
FFFFFFF0H	16	FFF00000H	1 M
FFFFFFE0H	32	FFE00000H	2 M
FFFFFFC0H	64	FFC00000H	4 M
FFFFFF80H	128	FF800000H	8 M
FFFFFF00H	256	FF000000H	Address Window Closed.
FFFFFE00H	512	FE000000H	
FFFFFC00H	1K	FC000000H	
FFFFF800H	2K	F8000000H	
FFFFF000H	4K	F0000000H	
FFFFE000H	8K	E0000000H	
FFFFC000H	16K	C0000000H	
FFFF8000H	32K	80000000H	
FFFF0000H	64K	00000000H	
FFFE0000H	128K		
FFFC0000H	256K		
FFF80000H	512K		

a. Shaded Limit Register Values and Memory Block Sizes are not supported by the PBI.

As an example, assume that FFF0 0004H is written to the PBI Limit Register 0 (FBLR0). Scanning upwards starting at bit 12, bit 20 is the first one bit found. The binary-weighted value of this bit is 1,048,576, indicating a 1 Mbyte of memory window.

When programming the Base and Limit Registers for a memory window, the Base Address always needs to be aligned the size of the memory window set in a limit register. For a 1 Mbyte memory window, only bit 20 through bit 31 of the base address from the PBI Base Address Register 0 (FBBAR0) is relevant to the PBI when decoding Memory Window 0.

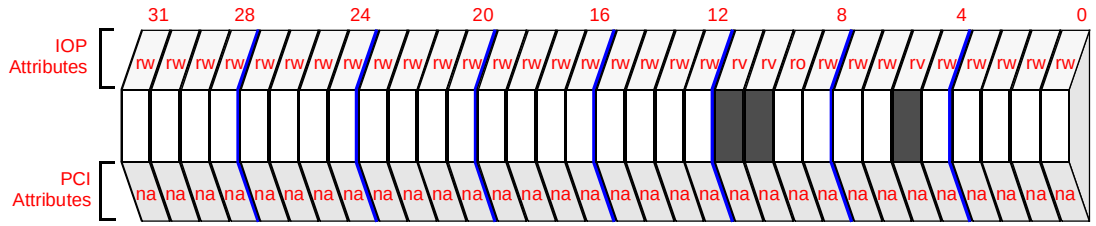
Warning: A given PBI Base (PBBAR0-PBBAR1) and Limit (PBLR0-PBLR1) register pair should not be modified during the time there is activity on the peripheral bus associated with that particular peripheral memory window. For instance, following boot-up, code executing from Peripheral Memory Window 0 may be used to modify the PBI Base and Limit registers for Peripheral Memory Window 1, but **not** for Peripheral Memory Window 0.

9.4.3 PBI Base Address Register 0 - PBBAR0

The PBI Base Address Register 0 (PBBAR0) defines the block of memory addresses where PBI Memory Window 0 begins. The PBBAR0 defines the base address and describes the required memory block size; see Section 9.4.2, "Determining Block Sizes for Memory Windows" on page 544. The selected base address needs to be naturally aligned to the granularity of the memory block size. For instance, when a 64 Kbyte memory window size is selected, the base address needs to be 64 Kbyte address aligned (i.e., bits 15:12 of the base address are required to be 000b).

Bits 1:0 define the PBI bus width for PBI Memory Window 0.

Table 275. PBI Base Address Register 0 - PBBAR0



Intel® XScale™ Core Local Bus Address
FFFF E688H

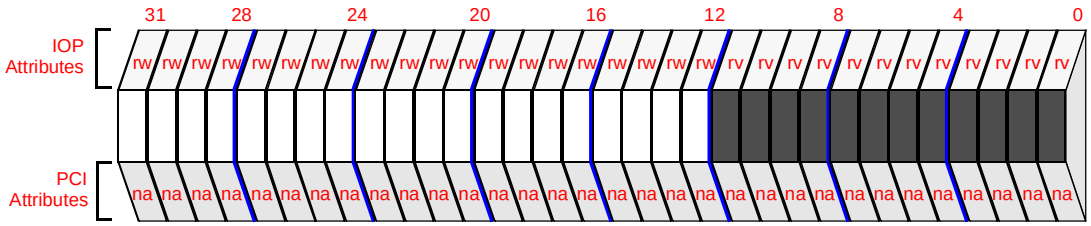
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	**Memory Window 0 Base Address:** These bits define the actual location the PBI responds to for accesses to Memory Window 0.
11:10	00_2	Reserved.
09	1_2	Read-Only = 1_2.
08:06	111_2	**Recovery Cycle Wait States:** Defines the number of recovery cycle wait states for the Peripheral window. 000 - 1 Recovery wait state 001 - 4 Recovery wait states 010 - 8 Recovery wait states 011 - 12 Recovery wait states 100 - 16 Recovery wait states Others (Default) - 20 Recovery wait states
05	0_2	Reserved
04:02	111_2	**Address-to-Data Wait States:** Defines the number of address-to-data wait states for the Peripheral window during a read or write transaction. 000 - 4 Address-to-Data wait states 001 - 8 Address-to-Data wait states 010 - 12 Address-to-Data wait states 011 - 16 Address-to-Data wait states Others (Default) - 20 Address-to-Data wait states
01:00	Varies with external state of **P_BOOT16#,** during **P_RST#**	**Bus Width:** Defines the bus width for this Memory Window: 00 - 8 bits wide 01 - 16 bits wide 10 - Reserved 11 - Reserved **NOTE: When P_BOOT16# is asserted, the default bus width is 16 bits wide (01), else the default bus width is 8 bits wide (00).**

9.4.4 PBI Limit Register 0 - PBLR0

The 80331 limit register's (PBLR0) programmed value must be naturally aligned with the base address register's (PBBAR0) programmed value. The limit register is used as a mask when the address decode for memory window 0 is performed.

Table 276. PBI Limit Register 0 - PBLR0



Intel® XScale™ Core Local Bus Address
FFFF E68CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

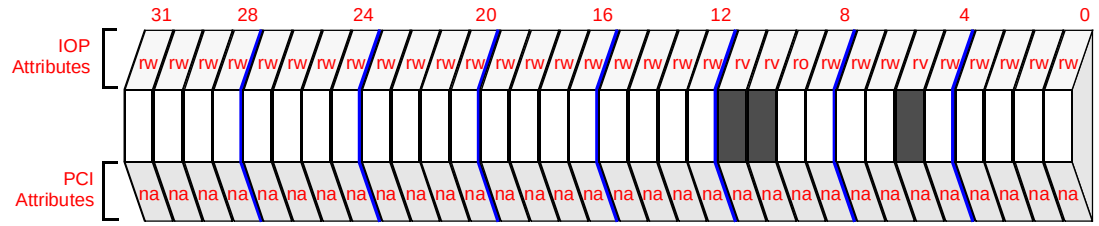
Bit	Default	Description
31:12	FF800H	**Memory Window 0 Limit:** This value determines the memory block size required for the Memory Window 0. Defaults to an 8MB Peripheral Window
11:00	000H	Reserved

9.4.5 PBI Base Address Register 1 - PBBAR1

The PBI Base Address Register 1 (PBBAR1) defines the block of memory addresses where PBI Memory Window 1 begins. The PBBAR1 defines the base address and describes the required memory block size; see Section 9.4.2, "Determining Block Sizes for Memory Windows" on page 544. The selected base address needs to be naturally aligned to the granularity of the memory block size. For instance, when a 64 Kbyte memory window size is selected, the base address needs to be 64 Kbyte address aligned (i.e., bits 15:12 of the base address are required to be 000b).

Bits 1:0 define the PBI bus width for PBI Memory Window 1.

Table 277. PBI Base Address Register 1- PBBAR1



Intel® XScale™ Core Local Bus Address
FFFF E690H

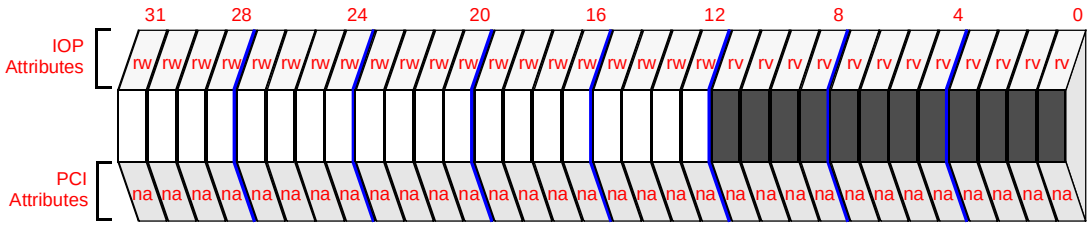
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	**Memory Window 1 Base Address:** These bits define the actual location the PBI responds to for accesses to Memory Window 1.
11:10	00_2	Reserved.
09	1_2	Read-Only = 1_2
08:06	111_2	**Recovery Cycle Wait States:** Defines the number of recovery cycle wait states for the Peripheral window. 000 - 1 Recovery wait state 001 - 4 Recovery wait states 010 - 8 Recovery wait states 011 - 12 Recovery wait states 100 - 16 Recovery wait states Others (Default) - 20 Recovery wait states
05	0_2	Reserved
04:02	111_2	**Address-to-Data Wait States:** Defines the number of address-to-data wait states for the Peripheral window during a read or write transaction. 000 - 4 Address-to-Data wait states 001 - 8 Address-to-Data wait states 010 - 12 Address-to-Data wait states 011 - 16 Address-to-Data wait states Others (Default) - 20 Address-to-Data wait states
01:00	00_2	**Bus Width:** Defines the bus width for this Memory Window: 00 - 8 bits wide 01 - 16 bits wide 10 - Reserved 11 - Reserved

9.4.6 PBI Limit Register 1 - PBLR1

The 80331 limit register's (PBLR1) programmed value must be naturally aligned with the base address register's (PBBAR1) programmed value. The limit register is used as a mask when the address decode for memory window 1 is performed.

Table 278. PBI Limit Register 1 - PBLR1



Intel® XScale™ Core Local Bus Address
FFFF E694H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	00000H	**Memory Window 1 Limit:** This value determines the memory block size required for the Memory Window 1.
11:00	000H	Reserved

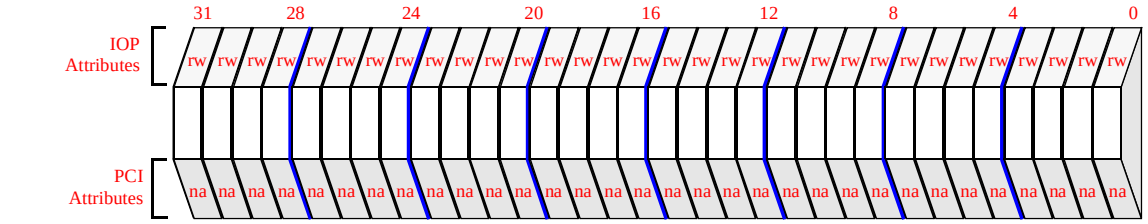
9.4.7 PBI Memory-less Boot Registers 2:0 - PMBR[2:0]

The three PBI Memory-less Boot Registers are used to hold the Intel® XScale™ core Reset exception vector and a long branch during a PCI Memory Boot sequence.

These three registers is made accessible at the addresses 0000 0000H, 0000 0020H, and 0000 0024H, when the 80331 performs a PCI Memory Boot sequence (see Section 9.3, "Intel® XScale™ Core PCI Memory Boot Support" on page 541 for more details).

Table 279 shows the PBI Memory-less Boot Registers 2:0.

Table 279. PBI Memory-less Boot Registers 2:0 - PMBR[2:0]



Internal bus address

PMBR0	FFFF E6C0H
PMBR1	FFFF E6E0H
PMBR2	FFFF E6E4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

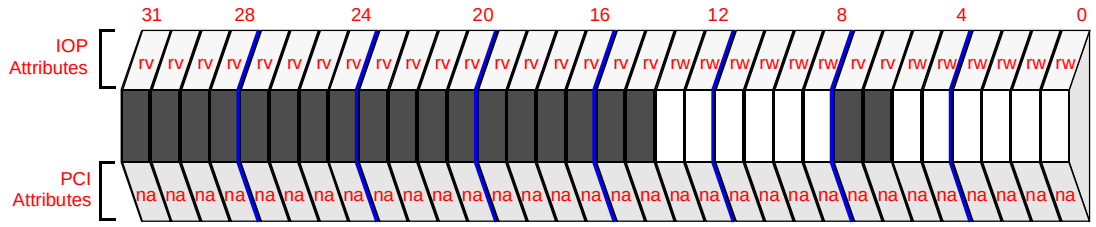
Bit	Default	Description
31:00	00000000H	Branch Instruction - 32-bit Intel® XScale™ core exception vector. PMBR0 (Reset Vector) - Short Branch to address 0000 0020H PMBR[2:1] - Long Branch to an Outbound Memory Window

9.4.8 PBI Drive Strength Control Register - PBDSCR

The PBI drive strength control register is used to manually control the slew rate and drive strength of the peripheral bus interface, the **TDO** pin, and the **GPIO[7:0]** pins.

Note: By default, the user is **not** required to program this register. This register should not be programmed to a different value without consulting the *Intel® 80331 I/O Processor Datasheet* where the appropriate values are specified.

Table 280. PBI Drive Strength Control Register - PBDSCR



Intel® XScale™ Core Local Bus Address
FFFF F580H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:14	0	Reserved
13:08	3FH	**Pull-Down Drive Strength:** This field programs the pull-up drive strength for the peripheral bus interface and the **GPIO[7:0]** pins.
07:06	00	Reserved
05:00	3FH	**Pull-Up Drive Strength:** This field programs the pull-down drive strength for the peripheral bus interface and the **GPIO[7:0]** pins.

I²C Bus Interface Units 10

This chapter describes the two I²C (Inter-Integrated Circuit) bus interface units, including the operation modes and setup. Throughout this manual, these peripherals are referred to as the I²C units.

10.1 Overview

Both I²C Bus Interface Units allow the Intel® 80331 I/O processor (80331) to serve as a master and slave device residing on the I²C bus. The I²C bus is a serial bus developed by Philips Corporation consisting of a two-pin interface. **SDA** is the data pin for input and output functions and **SCL** is the clock pin for reference and control of the I²C bus.

The I²C bus allows the 80331 to interface to other I²C peripherals and microcontrollers for system management functions. The serial bus requires a minimum of hardware for an economical system to relay status and reliability information on the 80331 subsystem to an external device.

The I²C Bus Interface Unit is a peripheral device that resides on a 80331 internal bus. Data is transmitted to and received from the I²C bus via a buffered interface. Control and status information is relayed through a set of memory-mapped registers. Refer to the I²C - *Bus Specification*, version 2.1 for complete details on I²C bus operation.

10.2 I2C Interface

Table 281. I2C Interface Pins

Signal	Description
SCL0	I²C Clock: Provides synchronous operation of I²C bus zero.
SDA0	I²C Data: Is used for data transfer and arbitration of I²C bus zero.
SCL1	I²C Clock: Provides synchronous operation of I²C bus one.
SDA1	I²C Data: Is used for data transfer and arbitration of I²C bus one.
Total	4

10.3 Theory of Operation

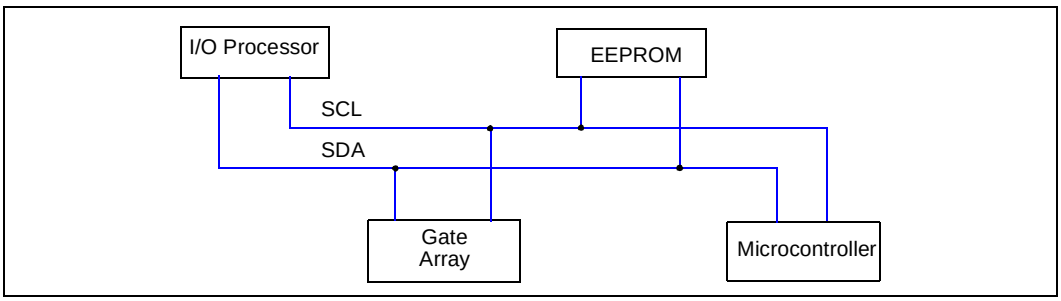
The I²C bus defines a serial protocol for passing information between agents on the I²C bus using only a two pin interface. The interface consists of a Serial Data/Address (**SDA**) line and a Serial Clock Line (**SCL**). Each device on the I²C bus is recognized by a unique 7-bit address and can operate as a transmitter or as a receiver. In addition to transmitter and receiver, the I²C bus uses the concept of master and slave. Table 282 lists the I²C device types.

Table 282. I²C Bus Definitions

I²C Device	Definition
Transmitter	Sends data to the I²C bus.
Receiver	Receives data from the I²C bus.
Master	Initiates a transfer, generates the clock signal, and terminates the transactions.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, when more than one master simultaneously tries to control the bus, only one is allowed. This procedure ensures that messages are not corrupted.

As an example of I²C bus operation, consider the case of the 80331 acting as a master on the bus (see Figure 90). The 80331, as a master, addresses an EEPROM as a slave to receive data. The 80331 is a master-transmitter and the EEPROM is a slave-receiver. When the 80331 reads data, the 80331 is a master-receiver and the EEPROM is a slave-transmitter. In both cases, the master generates the clock, initiates the transaction and terminates it.

Figure 90. I²C Bus Configuration Example



The I²C bus allows for a multi-master system, which means more than one device can initiate data transfers at the same time. To support this feature, the I²C bus arbitration relies on the wired-AND connection of all I²C interfaces to the I²C bus. Two masters can drive the bus simultaneously provided they are driving identical data. The first master to drive **SDA** high while another master drives **SDA** low loses the arbitration. The **SCL** line consists of a synchronized combination of clocks generated by the masters using the wired-AND connection to the **SCL** line.

The I²C bus serial operation uses an open-drain wired-AND bus structure, which allows multiple devices to drive the bus lines and to communicate status about events such as arbitration, wait states, error conditions and so on. For example, when a master drives the clock (**SCL**) line during a data transfer, it transfers a bit on every instance that the clock is high. When the slave is unable to accept or drive data at the rate that the master is requesting, the slave can hold the clock line low between the high states to insert a wait interval. The master's clock can only be altered by a slow slave peripheral keeping the clock line low or by another master during arbitration.

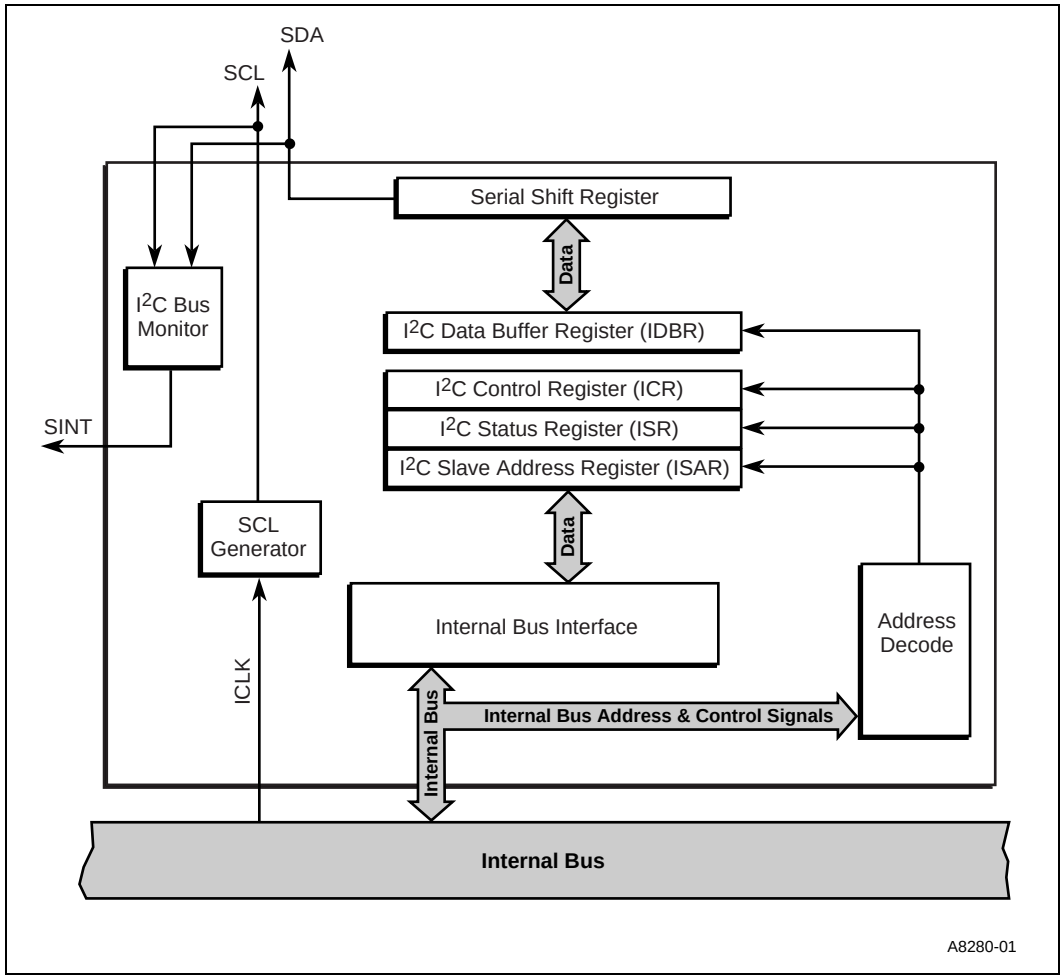
I²C transactions are either initiated by the 80331 as a master or are received by the processor as a slave. Both conditions may result in the processor doing reads, writes, or both to the I²C bus.

10.3.1 Operational Blocks

The I2C Bus Interface Unit is a slave peripheral device that is connected to the internal bus. The 80331 interrupt mechanism can be used for notifying the 80331 that there is activity on the I2C bus. Polling can be also be used instead of interrupts, although it would be very cumbersome. Figure 91 shows a block diagram of the I2C Bus Interface Unit and its interface to the internal bus.

The I2C Bus Interface Unit consists of the two wire interface to the I2C bus, an 8-bit buffer for passing data to and from the 80331, a set of control and status registers, and a shift register for parallel/serial conversions.

Figure 91. I2C Bus Interface Unit Block Diagram



The I2C interrupts are signalled through a single pin which provides a level sensitive interrupt to the 80331 interrupt control unit. The I2C Bus Interface Unit can cause and interrupt when a buffer is full, buffer empty, slave address detected, arbitration lost, or bus error condition occurs. All interrupt conditions must be cleared explicitly by software. See Section 10.9.2, "I2C Status Register x - ISRx" on page 578 for details.

The I2C Data Buffer Register (IDBR) is an 8-bit data buffer that receives a byte of data from the shift register interface of the I2C bus on one side and parallel data from the 80331 internal bus on the other side. The serial shift register is not user accessible.

The control and status registers are located in the I2C memory-mapped address space (FFFF F680H to FFFF F694H). The registers and their function are defined in Section 10.9.

The I2C Bus Interface Unit supports fast mode operation of 400 Kbits/sec. Fast mode logic levels, formats, and capacitive loading, and protocols are exactly the same as the 100 Kbits/sec standard mode. Because the data setup and hold times differ between the fast and standard mode, the I2C is designed to meet the slower, standard mode requirements for these two specifications. Refer to the I2C Bus Specification for details.

10.3.2 I2C Bus Interface Modes

The I2C Bus Interface Unit can be in different modes of operation to accomplish a transfer. Table 283 summarizes the different modes.

Table 283. Modes of Operation

Mode	Definition
Master - Transmit	• I2C Bus Interface Unit acts as a master. • Used for a write operation. • I2C Bus Interface Unit sends the data. • I2C Bus Interface Unit is responsible for clocking. • Slave device is in slave-receive mode
Master - Receive	• I2C Bus Interface Unit acts as a master. • Used for a read operation. • I2C Bus Interface Unit receives the data. • I2C Bus Interface Unit is responsible for clocking. • Slave device is in slave-transmit mode
Slave - Transmit	• I2C Bus Interface Unit acts as a slave. • Used for a read (master) operation. • I2C Bus Interface Unit sends the data. • Master device is in master-receive mode.
Slave - Receive (default)	• I2C Bus Interface Unit acts as a slave. • Used for a write (master) operation. • I2C Bus Interface Unit receives the data. • Master device is in master-transmit mode.

While the I2C Bus Interface Unit is in idle mode (neither receiving or transmitting serial data), the unit defaults to Slave-Receive mode. This allows the interface to monitor the bus and receive any slave addresses that might be intended for the 80331.

When the I2C Bus Interface Unit receives an address that matches the 7-bit address found in the I2C Slave Address Register (ISAR) or the General Call Address (00H), the interface either remains in Slave-Receive mode or transitions to Slave-Transmit mode. This is determined by the Read/Write (R/W#) bit (the least significant bit of the byte containing the slave address). When the R/W# bit is low, the master initiating the transaction intends to do a write and the I2C Bus Interface Unit remains in Slave-Receive mode. When the R/W# is high, the initiating master wants to read data and the slave transitions to Slave-Transmit mode. Slave operation is further defined in Section 10.4.6, "Slave Operations" on page 568.

When the 80331 wants to initiate a read or write on the I2C bus, the I2C Bus Interface Unit transitions from the default Slave-Receive mode to Master-Transmit mode. When the 80331 wants to write data, the interface remains in Master-Transmit mode after the address transfer has completed. (see Section 10.3.3.1, "START Condition" on page 557) for START information). When the 80331 wants to read data, the I2C Bus Interface Unit transmits the start address, then transition to Master-Receive mode. Master operation is further defined in Section 10.4.5, "Master Operations" on page 564.

10.3.3 Start and Stop Bus States

The I²C bus defines a transaction START and a transaction STOP bus state that are used at the beginning and end of the transfer of one to an unlimited number of bytes on the bus.

The 80331 uses the START and STOP bits in the I²C Control Register (ICR) to:

- initiate an additional byte transfer
- initiate a START condition on the I²C bus
- enable Data Chaining (repeated START)
- initiate a STOP condition on the I²C bus

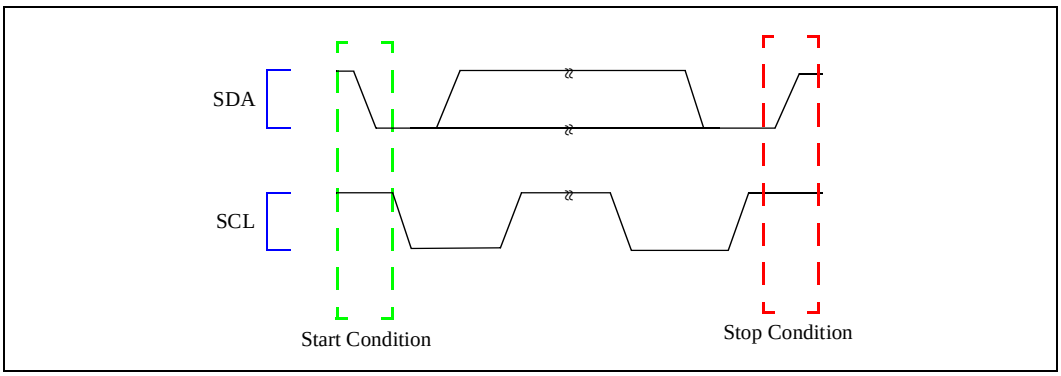
Table 284 summarizes the definition of the START and STOP bits in the ICR.

Table 284. START and STOP Bit Definitions

STOP bit	START bit	Condition	Notes
0	0	No START or STOP	• No START or STOP condition is sent by the I²C Bus Interface Unit. This is used when multiple data bytes need to be transferred.
0	1	START Condition and Repeated START	• The I²C Bus Interface Unit sends a START condition and transmit the contents of the 8 bit IDBR after the START. The IDBR must contain the 7-bit address and the R/W# bit before a START is initiated. • For a repeated start, the IDBR contents contains the target slave address and the R/W# bit. This enables multiple transfers to different slaves without giving up the bus. • The interface stays in Master-Transmit mode when a write is used or transition to master-receive mode when a read is requested.
1	X	STOP Condition	• In Master-Transmit mode, the I²C Bus Interface Unit transmits the 8-bit IDBR and then send a STOP on the I²C bus. • In Master-Receive mode, the Ack/Nack Control bit in the ICR must be changed to a negative Ack (see Section 10.4.3). The I²C Bus Interface Unit writes the Nack bit (Ack/Nack Control bit must be 1), receive the data byte in the IDBR, then send a STOP on the I²C bus.

Figure 92 shows the relationship between the **SDA** and **SCL** lines for a START and STOP condition.

Figure 92. Start and Stop Conditions



10.3.3.1 START Condition

The START condition (bits 1:0 of the ICR set to 01_2) initiates a master transaction or repeated START. Software must load the target slave address and the R/W# bit in the IDBR (see Section 10.9.4, "I²C Data Buffer Register x - IDBRx" on page 581) before setting the START ICR bit. The START and the IDBR contents are transmitted on the I²C bus when the ICR Transfer Byte bit is set. The I²C bus stays in master-transmit mode when a write is requested or enters master-receive mode when a read is requested. For a repeated start (a change in read or write or a change in the target slave address), the IDBR contains the updated target slave address and the R/W# bit. A repeated start enables multiple transfers to different slaves without giving up the bus.

The START condition is not cleared by the I²C unit. When arbitration is lost while initiating a START, the I²C unit may re-attempt the START when the bus becomes free. See Section 10.4.4, "Arbitration" on page 562 for details on how the I²C unit functions under those circumstances.

10.3.3.2 No START or STOP Condition

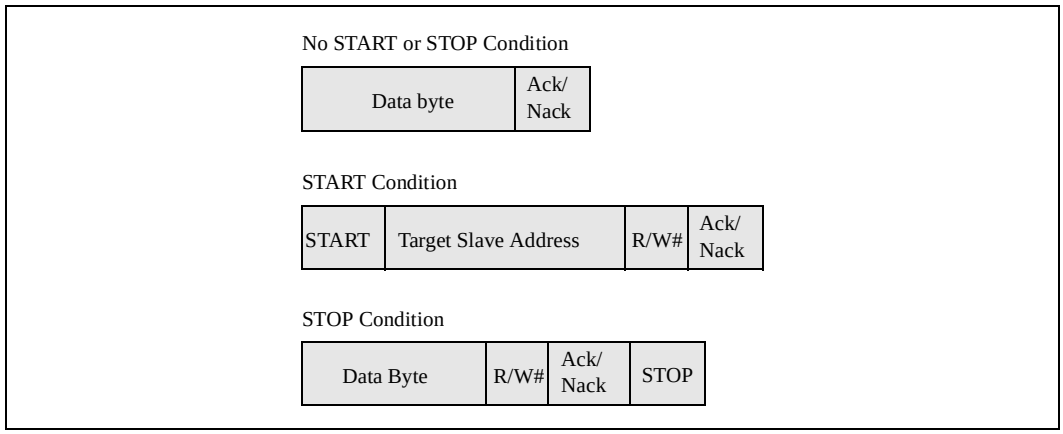
No START or STOP condition (bits 1:0 of the ICR set to 00_2) is used in master-transmit mode while the 80331 is transmitting multiple data bytes (see Figure 92). Software writes the data byte, sets the IDBR Transmit Empty bit in the ISR (and interrupt when enabled), and clears the Transfer Byte bit in the ICR. The software then writes a new byte to the IDBR and sets the Transfer Byte ICR bit, which initiates the new byte transmission. This continues until the software sets the START or STOP bit. The START and STOP bits in the ICR are not automatically cleared by the I²C unit after the transmission of a START, STOP or repeated START.

After each byte transfer (including the Ack/Nack bit) the I²C unit holds the **SCL** line low (inserting wait states) until the Transfer Byte bit in the ICR is set. This action notifies the I²C unit to release the **SCL** line and allow the next information transfer to proceed.

10.3.3.3 STOP Condition

The STOP condition (bits 1:0 of the ICR set to 10_2) terminates a data transfer. In master-transmit mode, the STOP bit and the Transfer Byte bit in the ICR must be set to initiate the last byte transfer (see Figure 92). In master-receive mode, to initiate the last transfer the 80331 must set the Ack/Nack bit, the STOP bit, and the Transfer Byte bit in the ICR. Software must clear the STOP condition after it is transmitted.

Figure 93. START and STOP Conditions



10.4 I²C Bus Operation

The I²C Bus Interface Unit transfers in 1 byte increments. A data transfer on the I²C bus always follows the sequence:

1) START
2) 7-bit Slave Address
3) R/W# Bit
4) Acknowledge Pulse
5) 8 Bits of Data
6) Ack/Nack Pulse
7) Repeat of Step 5 and 6 for Required Number of Bytes
8) Repeated START (Repeat Step 1) or STOP

10.4.1 Serial Clock Line (SCL) Generation

The 80331 I²C unit is required to generate the I²C clock output when in master mode (either receive or transmit). **SCL** clock generation is accomplished through the use of the Fast Mode Enable bit, which is programmed at initialization. The following equation is used to determine the **SCL** transition period:

Equation 12. SCL Transition Period

```
SCL Transition Period = (30 ns) * (167 - (Fast Mode Enable * 125))
```

10.4.2 Data and Addressing Management

Data and slave addressing is managed via the I²C Data Buffer Register (IDBR) and the I²C Slave Address Register (ISAR). The IDBR (see Section 10.9.4, "I²C Data Buffer Register x - IDBRx" on page 581) contains data or a slave address and R/W# bit. The ISAR contains the 80331 programmable slave address. Data coming into the I²C unit is received into the IDBR after a full byte is received and acknowledged. To transmit data, the processor writes to the IDBR, and the I²C unit passes this onto the serial bus when the Transfer Byte bit in the ICR is set. See Section 10.9.1, "I²C Control Register x - ICRx" on page 576.

When the I²C unit is in transmit mode (master or slave):

1. Software writes data to the IDBR over the internal bus. This initiates a master transaction or sends the next data byte, after the IDBR Transmit Empty bit is sent.
2. The I²C unit transmits the data from the IDBR when the Transmit Empty bit in the ICR is set.
3. When enabled, an IDBR Transmit Empty interrupt is signalled when a byte is transferred on the I²C bus and the acknowledge cycle is complete.
4. When the I²C bus is ready to transfer the next byte before the processor has written the IDBR (and a STOP condition is not in place), the I²C unit inserts wait states until the processor writes a new value into the IDBR and sets the ICR Transfer Byte bit.

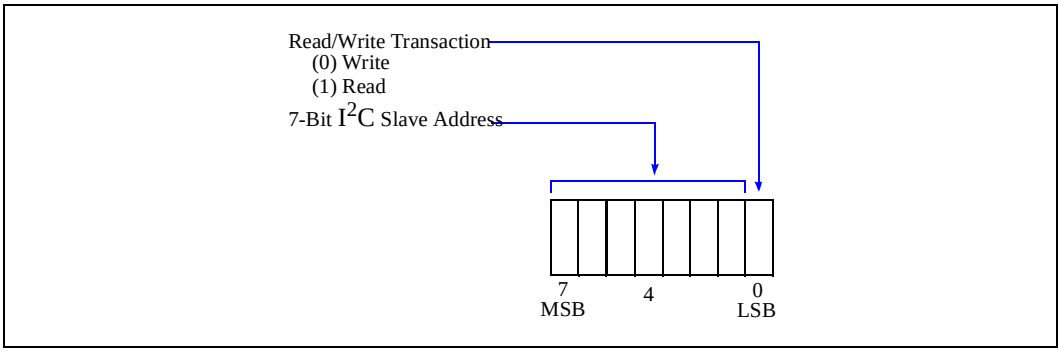
When the I²C unit is in receive mode (master or slave):

1. The processor reads the IDBR data over the internal bus after the IDBR Receive Full interrupt is signalled.
2. The I²C unit transfers data from the shift register to the IDBR after the Ack cycle completes.
3. The I²C unit inserts wait states until the IDBR is read. Refer to Section 10.4.3, "I²C Acknowledge" on page 561 for acknowledge pulse information in receiver mode.
4. After the Intel® XScale™ core (ARM* architecture compliant) reads the IDBR, the I²C unit writes the ICR's Ack/Nack Control bit and the Transfer Byte bit, allowing the next byte transfer to proceed.

10.4.2.1 Addressing a Slave Device

As a master device, the I²C unit must compose and send the first byte of a transaction. This byte consists of the slave address for the intended device and a R/W# bit for transaction definition. The slave address and the R/W# bit are written to the IDBR (see Figure 94).

Figure 94. Data Format of First Byte in Master Transaction



The first byte transmission must be followed by an Ack pulse from the addressed slave. When the transaction is a write, the I²C unit remains in master-transmit mode and the addressed slave device stays in slave-receive mode. When the transaction is a read, the I²C unit transitions to master-receive mode immediately following the Ack and the addressed slave device transitions to slave-transmit mode. When a Nack is returned, the I²C unit aborts the transaction by automatically sending a STOP and setting the ISR bus error bit.

When the I²C unit is enabled and idle (no bus activity), it stays in slave-receive mode and monitors the I²C bus for a START signal. Upon detecting a START pulse, the I²C unit reads the first seven bits and compares them to those in the I²C Slave Address Register (ISAR) and the general call address (00H). When the bits match those of the ISAR register, the I²C unit reads the eighth bit (R/W# bit) and transmits an Ack pulse. The I²C unit either remains in slave-receive mode (R/W# = 0) or transitions to slave-transmit mode (R/W# = 1). See Section 10.4.7, "General Call Address" on page 570 for actions when a general call address is detected.

10.4.3 I2C Acknowledge

Every I2C byte transfer must be accompanied by an acknowledge pulse, which is always generated by the receiver (master or slave). The transmitter must release the **SDA** line for the receiver to transmit the acknowledge pulse (see Figure 95).

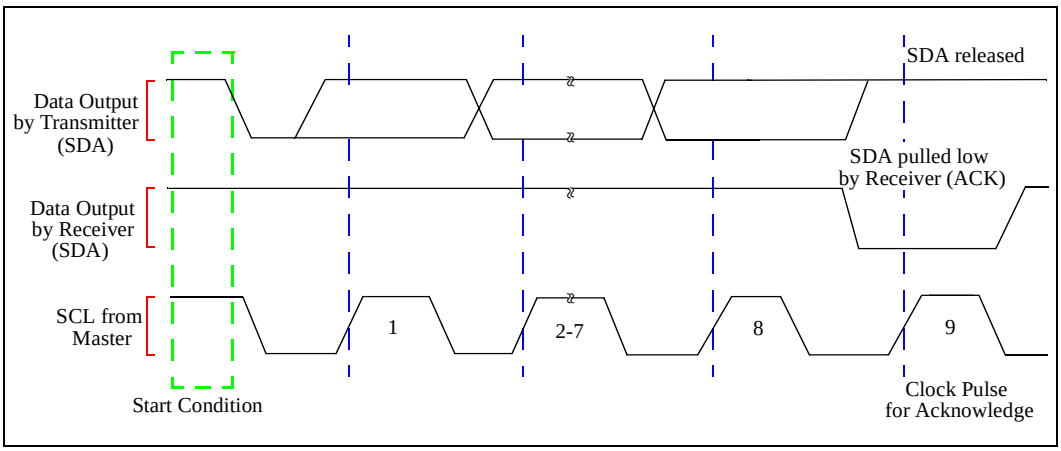
In master-transmit mode, when the target slave receiver device cannot generate the acknowledge pulse, the **SDA** line remains high. This lack of acknowledge (Nack) causes the I2C unit to set the bus error detected bit in the ISR and generate the associated interrupt (when enabled). The I2C unit aborts the transaction by generating a STOP automatically.

In master-receive mode, the I2C unit signals the slave-transmitter to stop sending data by using the negative acknowledge (Nack). The Ack/Nack bit value driven by the I2C bus is controlled by the Ack/Nack bit in the ICR. The bus error detected bit in the ISR is not set for a master-receive mode Nack (as required by the I2C bus protocol). The I2C unit automatically transmits the Ack pulse, based on the Ack/Nack ICR bit, after receiving each byte from the serial bus. Before receiving the last byte, software must set the Ack/Nack Control bit to Nack. Nack is then sent after the next byte is received to indicate the last byte.

In slave mode, the I2C unit automatically acknowledges its own slave address, independent of the Ack/Nack bit setting in the ICR. As a slave-receiver, an Ack response is automatically given to a data byte, independent of the Ack/Nack bit setting in the ICR. The I2C unit sends the Ack value after receiving the eighth data bit of the byte.

In slave-transmit mode, receiving a Nack from the master indicates the last byte is transferred. The master then sends either a STOP or repeated START. The ISR's unit busy bit (2) remains set until a STOP or repeated START is received.

Figure 95. Acknowledge on the I2C Bus



10.4.4 Arbitration

Arbitration on the I²C bus is required due to the multi-master capabilities of the I²C bus. Arbitration is used when two or more masters simultaneously generate a START condition within the minimum I²C hold time of the START condition.

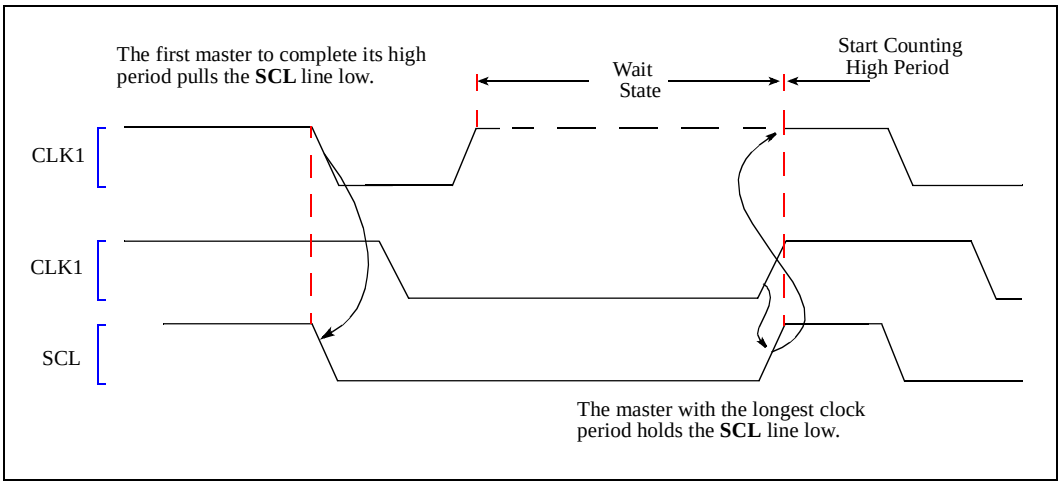
Arbitration can continue for a long period. When the address bit and the R/W# are the same, the arbitration moves to the data. Due to the wired-AND nature of the I²C bus, no data is lost when both (or all) masters are outputting the same bus states. When the address, the R/W# bit, or the data are different, the master which outputted the high state (master's data is different from **SDA**) loses arbitration and shut its data drivers off. When losing arbitration, the I²C Bus Interface Unit shuts off the **SDA** or **SCL** drivers for the remainder of the byte transfer, set the Arbitration Loss Detected bit, then return to idle (Slave-Receive) mode.

10.4.4.1 SCL Arbitration

Each master on the I²C bus generates its own clock on the **SCL** line for data transfers. With masters generating their own clocks, clocks with different frequencies may be connected to the **SCL** line. Since data is valid when the clock is in the high period, a defined clock synchronization procedure is needed during bit-by-bit arbitration.

Clock synchronization is accomplished by using the wired-AND connection of the I²C interfaces to the **SCL** line. When a master's clock transitions from high to low, this causes the master to hold down the **SCL** line for its associated period (see Figure 96). The low to high transition of the clock may not change when another master has not completed its period. Therefore, the master with the longest low period holds down the **SCL** line. Masters with shorter periods are held in a high wait-state during this time. Once the master with the longest period completes, the **SCL** line transitions to the high state, masters with the shorter periods can continue the data cycle.

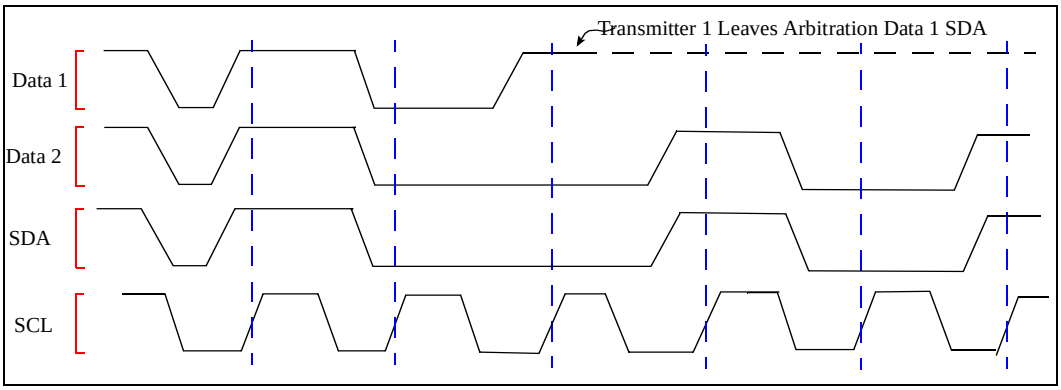
Figure 96. Clock Synchronization During the Arbitration Procedure



10.4.4.2 SDA Arbitration

Arbitration on the **SDA** line can continue for a long period, starting with address and R/W# bits and continuing with data bits. Figure 97 shows the arbitration procedure for two masters (more than two may be involved depending on how many masters are connected to the bus). When the address and R/W# are the same, arbitration moves to the data. Due to the wired-AND nature of the I²C bus, no data is lost when both (or all) masters are outputting the same bus states. When address, R/W#, or data is different, the master that output the first low data bit loses arbitration and shuts its data drivers off. When the I²C unit loses arbitration, it shuts off the **SDA** or **SCL** drivers for remainder of byte transfer, sets arbitration loss detected ISR bit, then returns to idle (Slave-Receive) mode.

Figure 97. Arbitration Procedure of Two Masters



When the I²C unit loses arbitration during transmission of the seven address bits and the 80331 is not being addressed as a slave device, the I²C unit re-sends the address when the I²C bus becomes free. This is possible because the IDBR and ICR registers are not overwritten when arbitration is lost.

When the arbitration loss is to due to another bus master addressing the 80331 as a slave device, the I²C unit switches to slave-receive mode and the original data in the I²C data buffer register is overwritten. Software is responsible for clearing the start and re-initiating the master transaction at a later time.

Note: Software must not allow the I²C unit to write to its own slave address. This can cause the I²C bus to enter an indeterminate state.

Boundary conditions exist for arbitration when an arbitration process is in progress and a repeated START or STOP condition is transmitted on the I²C bus. To prevent errors, the I²C unit, acting as a master, provides for the following sequences:

- No arbitration takes place between a repeated START condition and a data bit
- No arbitration takes place between a data bit and a STOP condition
- No arbitration takes place between a repeated START condition and a STOP condition

These situations arise only when different masters write the same data to the same target slave simultaneously and arbitration is not resolved after the first data byte transfer.

Note: Typically, software is responsible for ensuring arbitration is lost soon after the transaction begins. For example, the protocol might insist that all masters transmit their I²C address as the first data byte of any transaction ensuring arbitration is ended. A restart is then sent to begin a valid data transfer (the slave can then discard the master's address).

10.4.5 Master Operations

When software initiates a read or write on the I2C bus, the I2C unit transitions from the default slave-receive mode to master-transmit mode. The start pulse is sent followed by the 7-bit slave address and the R/W# bit. After the master receives an acknowledge, the I2C unit has the option of two master modes:

- Master-Transmit — The 80331 writes data
- Master-Receive — The 80331 reads data

The 80331 initiates a master transaction by writing to the ICR register. Data is read and written from the I2C unit through the memory-mapped registers.

Table 285 describes the I2C Bus Interface Unit responsibilities as a master device.

Table 285. Master Transactions (Sheet 1 of 3)

I2C Master Action	Mode of Operation	Definition
Generate clock output	Master-transmit Master-receive	• The master always drives the **SCL** line. • The **SCL** Enable bit must be set. • The Unit Enable bit must be set.
Write target slave address to IDBR	Master-transmit Master-receive	• The Intel® XScale™ core writes to IDBR bits 7-1 before a START condition is enabled. • First 7 bits sent on bus after START. • See Section 10.3.3.
Write R/W# Bit to IDBR	Master-transmit Master-receive	• The Intel® XScale™ core writes to the least significant IDBR bit with the target slave address. • When low, the master remains a master-transmitter. When high, the master transitions to a master-receiver. • See Section 10.4.2.
Signal START Condition	Master-transmit Master-receive	• See "Generate clock output" above. • Performed after the target slave address and the R/W# bit are in the IDBR. • Intel® XScale™ core sets the START bit. • Intel® XScale™ core sets the Transfer Byte bit which initiates the start condition. • See Section 10.3.3.
Initiate first data byte transfer	Master-transmit Master-receive	• Intel® XScale™ core writes byte to IDBR • I2C Bus Interface Unit transmits the byte when the Transfer Byte bit is set. • I2C Bus Interface Unit clears the Transfer Byte bit and sets the IDBR Transmit Empty bit when the transfer is complete.

Table 285. Master Transactions (Sheet 2 of 3)

I²C Master Action	Mode of Operation	Definition
Arbitrate for I²C Bus	Master-transmit Master-receive	• When two or more masters signal a start within the same clock period, arbitration must occur. • The I²C Bus Interface Unit arbitrates for as long as necessary. Arbitration takes place during slave address, R/W# bit, and data transmission and continues until all but one master loses the bus. No data is lost during arbitration. • When the I²C Bus Interface Unit loses arbitration, it sets the Arbitration Loss Detect ISR bit after byte transfer is complete and transition to slave-receive (default) mode. • When I²C Bus Interface Unit loses arbitration while attempting to send the target address byte, the I²C Bus Interface Unit attempts to resend it when the bus becomes free. • The system designer must ensure the boundary conditions described in Section 10.4 do not occur.
Write one data byte to the IDBR	Master-transmit only	• Data transmit mode of I²C master operation. • Occurs when the IDBR Transmit Empty ISR bit is set and the Transfer Byte bit is clear. When enabled, the IDBR Transmit Empty Interrupt is signalled to the Intel® XScale™ core. • Intel® XScale™ core writes 1 data byte to the IDBR, set the appropriate START/STOP bit combination, and then set the Transfer Byte bit to send the data. Eight bits are written on the serial bus followed by a STOP when requested.
Wait for Acknowledge from slave-receiver	Master-transmit only	• As a master-transmitter, the I²C Bus Interface Unit generates the clock for the acknowledge pulse. The I²C Bus Interface Unit is responsible for releasing the **SDA** line to allow slave-receiver Ack transmission. • See Section 10.4.3.
Read one byte of I²C Data from the IDBR	Master-receive only	• Data receive mode of I²C master operation. • Eight bits are read from the serial bus, collected in the shift register then transferred to the IDBR after the Ack/Nack bit is read. • The Intel® XScale™ core reads the IDBR when the IDBR Receive Full bit is set and the Transfer Byte bit is clear. When enabled, a IDBR Receive Full Interrupt is signalled to the Intel® XScale™ core processor. • When the IDBR is read, when the Ack/Nack Status is clear (indicating Ack), the Intel® XScale™ core writes the Ack/Nack Control bit and set the Transfer Byte bit to initiate the next byte read. • When the Ack/Nack Status bit is set (indicating Nack), Transfer Byte bit is clear, STOP bit in the ICR is set, and Unit Busy bit in the ISR is set, then the last data byte has been read into the IDBR and the I²C Bus Interface Unit is sending the STOP. • When the Ack/Nack Status bit is set (indicating Nack), Transfer Byte bit is clear, but the STOP bit is clear, then the Intel® XScale™ core has two options: 1. set the START bit, write a new target address to the IDBR, and set the Transfer Byte bit which sends a repeated start condition, 2. set the Master Abort bit and leave the Transfer Byte clear which sends a STOP only.

Table 285. Master Transactions (Sheet 3 of 3)

I²C Master Action	Mode of Operation	Definition
Transmit Acknowledge to slave-transmitter	Master-receive only	• As a master-receiver, the I²C Bus Interface Unit generates the clock for the acknowledge pulse. The I²C Bus Interface Unit is also responsible for driving the **SDA** line during the Ack cycle. • When the next data byte is to be the last transaction, the Intel® XScale™ core sets the Ack/Nack Control bit for Nack generation. • See Section 10.4.3.
Generate a Repeated START to chain I²C transactions	Master-transmit Master-receive	• When data chaining is desired, a repeated START condition is used instead of a STOP condition. • This occurs after the last data byte of a transaction has been written to the bus. • The Intel® XScale™ core writes the next target slave address and the R/W# bit to the IDBR, set the START bit, and set the Transfer Byte bit. • See Section 10.3.3.
Generate a STOP	Master-transmit Master-receive	• Generated after the Intel® XScale™ core writes the last data byte on the bus. • Intel® XScale™ core generates a STOP condition by setting the STOP bit in the ICR. • See Section 10.3.3.

When the 80331 needs to read data, the I²C unit transitions from slave-receive mode to master-transmit mode to transmit the start address and immediately following the ACK pulse transitions to master-receive mode to wait for the reception of the read data from the slave device (see Figure 98). It is also possible to have multiple transactions during an I²C operation such as transitioning from master-receive to master-transmit through a repeated start or Data Chaining (see Figure 99). Figure 100 shows the wave forms of **SDA** and **SCL** for a complete data transfer.

Figure 98. Master-Receiver Read from Slave-Transmitter

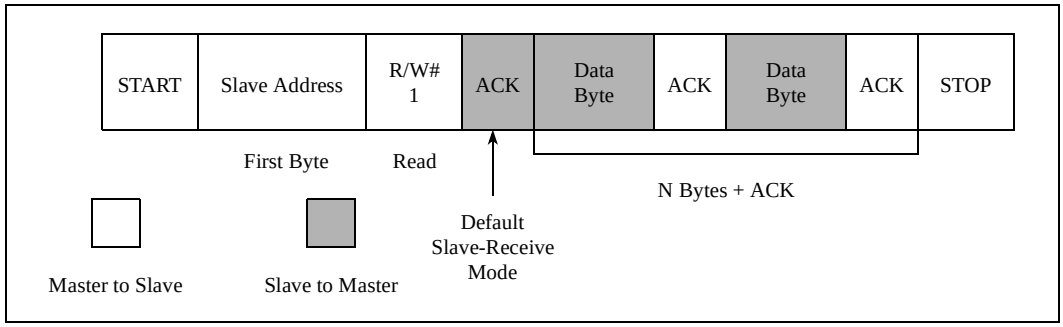


Figure 99. Master-Receiver Read from Slave-Transmitter / Repeated Start / Master-Transmitter Write to Slave-Receiver

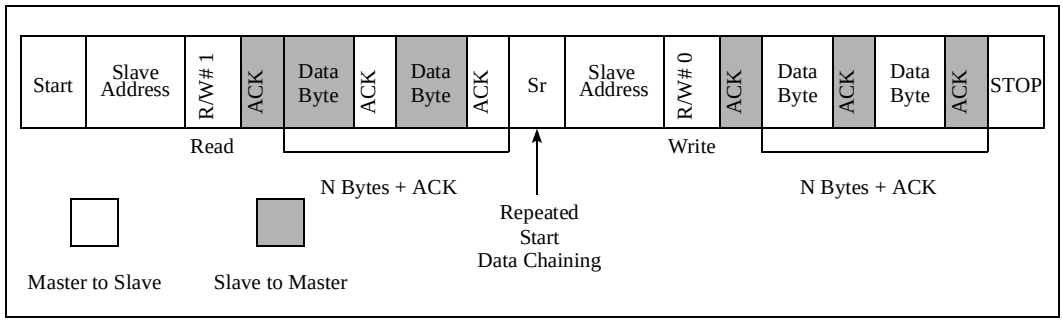
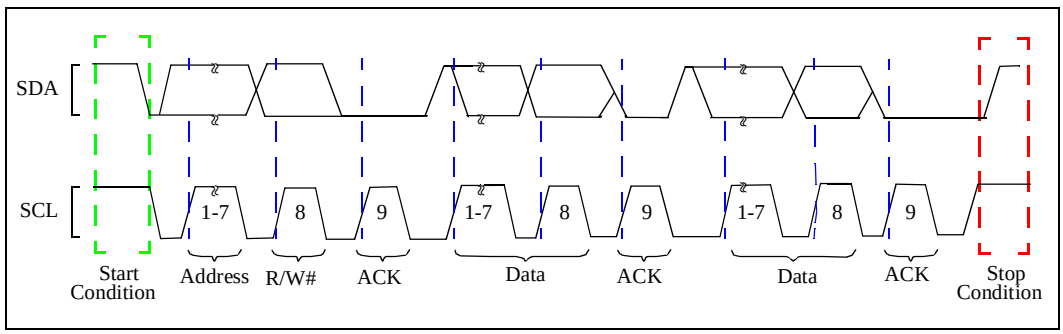


Figure 100. A Complete Data Transfer



10.4.6 Slave Operations

Table 286 describes the I²C Bus Interface Unit's responsibilities as a slave device.

Table 286. Slave Transactions

I²C Slave Action	Mode of Operation	Definition
Slave-receive (default mode)	Slave-receive only	• I²C Bus Interface Unit monitors all slave address transactions. • The I²C Bus Interface Unit Enable bit must be set. • I²C Bus Interface Unit monitors bus for START conditions. When a START is detected, the interface reads the first 8 bits and compares the most significant 7 bits with the 7 bit I²C Slave Address Register and the General Call address (00H). When there is a match, the I²C Bus Interface Unit sends an Ack. • When the first 8 bits are all zero's, this is a general call address. When the General Call Disable bit is clear, both the General Call Address Detected bit and the Slave Mode Operation bit in the ISR is set. See Section 10.4.7. • When the 8th bit of the first byte (R/W# bit) is low, the I²C Bus Interface Unit stays in slave-receive mode and the Slave Mode Operation bit is cleared. When the R/W# bit is high, the I²C Bus Interface Unit transitions to slave-transmit mode and the Slave Mode Operation bit is set.
Setting the Slave Address Detected bit	Slave-receive Slave-transmit	• Indicates the interface has detected an I²C operation that addresses the 80331 (this includes general call address). The Intel® XScale™ core can distinguish an ISAR match from a General Call by reading the General Call Address Detected bit. • An interrupt is signalled (when enabled) after the matching slave address is received and acknowledged.
Read one byte of I²C Data from the IDBR	Slave-receive only	• Data receive mode of I²C slave operation. • Eight bits are read from the serial bus into the shift register. When a full byte has been received and the Ack/Nack bit has completed, the byte is transferred from the shift register to the IDBR. • Occurs when the IDBR Receive Full bit in the ISR is set and the Transfer Byte bit is clear. When enabled, the IDBR Receive Full Interrupt is signalled to the Intel® XScale™ core. • Intel® XScale™ core reads 1 data byte from the IDBR. When the IDBR is read, the Intel® XScale™ core writes the desired Ack/Nack Control bit and set the Transfer Byte bit. This causes the I²C Bus Interface Unit to stop inserting wait states and let the master transmitter write the next piece of information.
Transmit Acknowledge to master-transmitter	Slave-receive only	• As a slave-receiver, the I²C Bus Interface Unit is responsible for pulling the **SDA** line low to generate the Ack pulse during the high **SCL** period. • The Ack/Nack Control bit controls the Ack data the I²C Bus Interface Unit drives. See Section 10.4.3.
Write one byte of I²C data to the IDBR	Slave-transmit only	• Data transmit mode of I²C slave operation. • Occurs when the IDBR Transmit Empty bit is set and the Transfer Byte bit is clear. When enabled, the IDBR Transmit Empty Interrupt is signalled to the Intel® XScale™ core. • Intel® XScale™ core writes a data byte to the IDBR and set the Transfer Byte bit to initiate the transfer.
Wait for Acknowledge from master-receiver	Slave-transmit only	• As a slave-transmitter, the I²C Bus Interface Unit is responsible for releasing the **SDA** line to allow the master-receiver to pull the line low for the Ack. • See Section 10.4.3.

Figure 101 through Figure 103 are examples of I²C transactions. These show the relationships between master and slave devices.

Figure 101. Master-Transmitter Write to Slave-Receiver

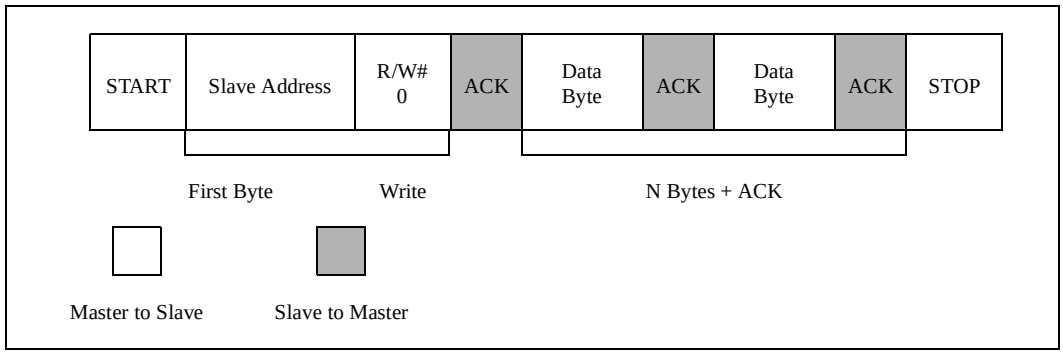


Figure 102. Master-Receiver Read to Slave-Transmitter

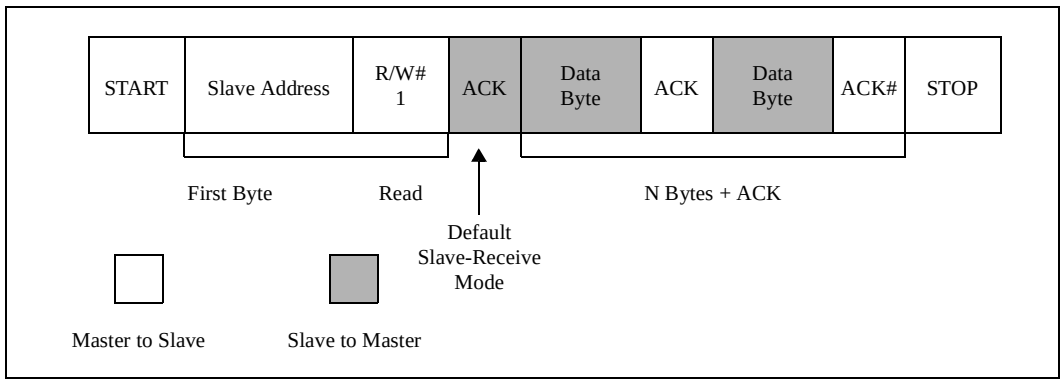
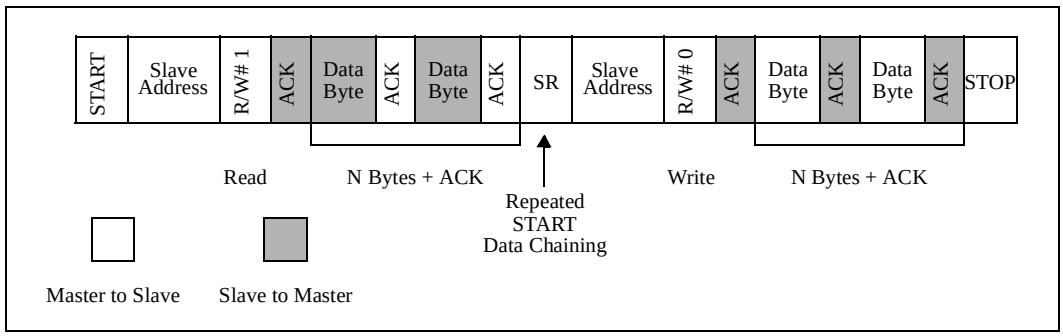


Figure 103. Master-Receiver Read to Slave-Transmitter / Repeated START / Master-Transmitter Write to Slave-Receiver



10.4.7 General Call Address

The I²C unit supports both sending and receiving general call address transfers on the I²C bus. When sending a general call message from the I²C unit, software must set the General Call Disable bit in the ICR to keep the I²C unit from responding as a slave. Failure to set this bit causes the I²C Bus to enter an indeterminate state.

A general call address is defined as a transaction with a slave address of 00H. When a device requires the data from a general call address, it acknowledges the transaction and stays in slave-receiver mode. Otherwise, the device can ignore the general call address. The second and following bytes of a general call transaction are acknowledged by every device using it on the bus. Any device not using these bytes must not Ack. The meaning of a general call address is defined in the second byte sent by the master-transmitter. Figure 104 shows a general call address transaction. The least significant bit (B) of the second byte defines the transaction. Table 287, "General Call Address Second Byte Definitions" on page 570 shows the valid values and definitions when B=0.

When the 80331 is acting as a slave, and the I²C unit receives a general call address and the ICR General Call Disable bit is clear the I²C unit:

- Sets the ISR general call address detected bit
- Sets the ISR slave address detected bit
- Interrupts (when enabled) the 80331

When the I²C unit receives a general call address and the ICR General Call Disable bit is set, the I²C unit ignores the general call address.

Figure 104. General Call Address

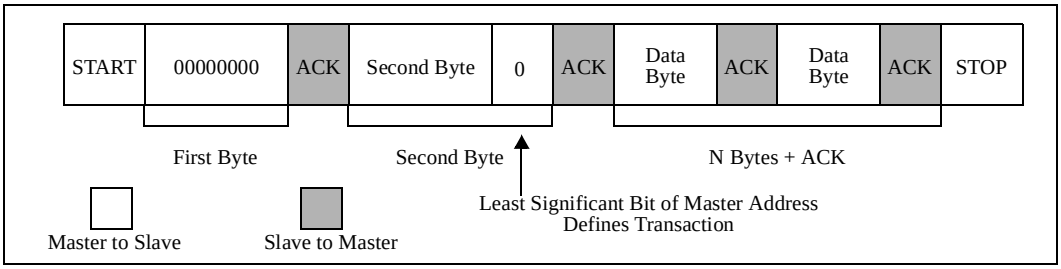


Table 287. General Call Address Second Byte Definitions

Least Significant Bit of Second Byte (B)	Second Byte Value	Definition
0	06H	2-byte transaction where the second byte tells the slave to reset and then store this value in the programmable part of their slave address.
0	04H	2-byte transaction where the second byte tells the slave to store this value in the programmable part of their slave address. No reset.
0	00H	Not allowed as a second byte

When directed to reset, the I²C Bus Interface Unit returns to its default reset condition with the exception of the ISAR. The 80331 is responsible for ensuring this occurs, not the I²C Bus Interface Unit hardware.

When B=1, the sequence is used as a hardware general call by hardware masters only they cannot transmit a slave address, only their own address. The I²C Bus Interface Unit does not support this mode of operation.

I²C 10-bit addressing and CBUS compatibility are not supported.

10.5 Slave Mode Programming Examples

10.5.1 Initialize Unit

1. Write ISAR: Set slave address
2. Write ICR: Enable all interrupts, set Unit Enable

10.5.2 Write 1 Byte as a Slave

1. Wait for Slave Address Detected interrupt.
   Read ISR: Slave Address Detected (set), Unit Busy (set), R/W# bit (1), Ack/Nack (Clear - Ack)
2. Write IDBR: Load data byte to transfer
3. Write ICR: Set Transfer Byte bit
4. Wait for IDBR Transmit Empty interrupt.
   Read ISR: IDBR Transmit Empty (set), Ack/Nack (set - indicates last byte write), R/W# bit (0)
5. Clear interrupt by clearing the IDBR Transmit Empty Interrupt bit.
6. Wait for interrupt.
   Read ISR: Unit Busy (clear), Slave STOP Detected (set)
7. Clear interrupt by clearing Slave STOP Detected Interrupt bit.

10.5.3 Read 2 Bytes as a Slave

1. Wait for Slave Address Detected interrupt.
   Read ISR: Slave Address Detected (set), Unit busy (set), R/W# bit (0)
2. Read byte 1 on I^2C bus
   Write ICR: Set Transfer Byte bit to initiate the transfer
3. Wait for interrupt.
   Read ISR: IDBR Receive Full (set), Ack/Nack (clear), R/W# bit (0)
   Clear interrupt by clearing IDBR Receive Full bit.
   Read IDBR: To get the data.
4. Read byte 2 on I^2C bus
   Write ICR: Set Transfer Byte bit to initiate the transfer
5. Wait for interrupt.
   Read ISR: IDBR Receive Full (set), Ack/Nack (clear), R/W# bit (0)
   Clear interrupt by clearing IDBR Receive Full bit.
   Read IDBR: To get the data.
   Write ICR: Set Transfer Byte bit (to release I^2C bus allowing next transfer)
6. Wait for interrupt.
   Read ISR: Unit busy (clear), Slave STOP Detected (set)
   Clear interrupt by clearing Slave STOP Detected bit.

10.6 Master Programming Examples

10.6.1 Initialize Unit

1. Write ISAR: Set slave address
2. Write ICR: Enable all interrupts (except Arb Loss), set **SCL** Enable, set Unit Enable

10.6.2 Write 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (0 for write)
2. Write ICR: Set START bit, Clear STOP bit, Set Transfer Byte bit to initiate the access
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
   Read status register: IDBR Transmit Empty (set), Unit Busy (set), R/W# bit (clear)
   Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.

Note: Arbitration Loss Detected bit may be set. When arbitration was lost, because Arb Loss interrupt was disabled, an address retry occurs when bus becomes free. Clear Arbitration Loss Detected bit when set.

4. Send byte with STOP
   Write IDBR: With data byte to send
   Write ICR: Clear START bit, Set STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit to initiate the access
5. Wait for Buffer empty interrupt. When interrupt arrives (Note: Unit is sending STOP):
   Read status register: IDBR Transmit Empty (set), Unit busy (set - maybe), R/W# bit (clear)
   Clear IDBR Transmit Empty Interrupt bit to clear the interrupt.
   Clear ICR STOP bit (optional)
   Wait until Unit busy is clear before clearing the ICR SCL Enable bit.

10.6.3 Read 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (1 for read)
2. Write ICR: Set START bit, Clear STOP bit, Disable Arb loss interrupt, Set Transfer Byte bit to initiate the access
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
   Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (set)
   Clear IDBR Transmit Empty bit to clear the interrupt.
4. Read byte with STOP
   Write ICR: Clear START bit, Set STOP bit, Enable arb loss interrupt, Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access
5. Wait for Buffer full interrupt. When interrupt arrives (Note: Unit is sending STOP):
   Read status register: IDBR Receive Full (set), Unit Busy (set - maybe), R/W# bit (Set), Ack/Nack bit (Set)
   Clear IDBR Receive Full bit to clear the interrupt.
   Read IDBR data.
   Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional)
   Wait until Unit busy is clear before clearing the ICR SCL Enable bit. (optional)

10.6.4 Write 2 Bytes and Repeated Start Read 1 Byte as a Master

1. Write IDBR: Target slave address and R/W# bit (0 for write)
2. Write ICR: Set START bit, Clear STOP bit, Set Transfer Byte bit to initiate the access
3. Wait for IDBR Transmit Empty interrupt. When interrupt arrives:
   Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (clear)
   Clear IDBR Transmit Empty bit to clear the interrupt.
4. Send byte 1
   Write IDBR: With data byte to send
   Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Set Transfer Byte bit to initiate the access
5. Wait for Buffer empty interrupt.
   Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (clear)
   Clear IDBR Transmit Empty bit to clear the interrupt.
6. Send byte 2
   Write IDBR: With data byte to send
   Write ICR: Clear START bit, Clear STOP bit, Set Transfer Byte bit to initiate the access
7. Wait for Buffer empty interrupt.
   Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (clear)
   Clear IDBR Transmit Empty bit to clear the interrupt.
8. Send repeated start as a master
   Write IDBR: Target slave address and R/W# bit (1 for read)
   Write ICR: Set START bit, Clear STOP bit, Disable Arb Loss interrupt, Set Transfer Byte bit the initiate the access
9. Wait for IDBR Transmit Empty interrupt. When interrupt comes.
   Read status register: IDBR Transmit Empty (set), Unit busy (set), R/W# bit (set)
   Clear IDBR Transmit Empty bit to clear the interrupt.
10. Read byte with STOP
    Write ICR: Clear START bit, Set STOP bit, Enable arb loss interrupt, Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access
11. Wait for Buffer full interrupt. When interrupt comes (Note: Unit is sending STOP).
    Read status register: IDBR Receive Full (set), Unit busy (set - maybe), R/W# bit (Set), Ack/Nack bit (Set)
    Clear IDBR Receive Full bit to clear the interrupt.
    Read IDBR data.
    Clear ICR STOP bit (optional), Clear ICR Ack/Nack Control bit (optional)
    Wait until Unit busy is clear before clearing the ICR SCL Enable bit. (optional)

10.6.5 Read 2 Bytes as a Master - Send STOP Using the Abort

1. Write IDBR: Target slave address and R/W# bit (1 for read)
2. Write ICR: Set START bit, Clear STOP bit, Disable Arb loss interrupt, Set Transfer Byte bit to initiate the access
3. Wait for IDBR Transmit Empty interrupt. When interrupt comes.
   Read status register: IDBR Transmit Empty (set), Unit Busy (set), R/W# bit (set)
   Clear IDBR Transmit Empty bit to clear the interrupt.
4. Read byte 1
   Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Clear Ack/Nack bit (Ack), Set Transfer Byte bit to initiate the access
5. Wait for Buffer full interrupt.
   Read status register: IDBR Receive Full (set), Unit busy (set), R/W# bit (Set), Ack/Nack bit (Clear)
   Clear IDBR Receive Full bit to clear the interrupt.
   Read IDBR data.
6. Read byte 2 with Nack (STOP is not set because STOP or Repeated START is decided on the byte read)
   Write ICR: Clear START bit, Clear STOP bit, Enable Arb Loss interrupt, Set Ack/Nack bit (Nack), Set Transfer Byte bit to initiate the access
7. Wait for Buffer full interrupt.
   Read status register: IDBR Receive Full (set), Unit Busy (set), R/W# bit (Set), Ack/Nack bit (Set)
   Clear IDBR Receive Full bit to clear the interrupt.
   Read IDBR data.

There are now two options based on the byte read:

- Send a repeated START
- Send a STOP only

Here, a STOP abort is sent.

Note: Had a NACK not been sent, the next transaction *must* involve another data byte read.

8. Send STOP abort condition. (STOP with no data transfer.)
   Write ICR: Set Master abort.

10.7 Glitch Suppression Logic

The I²C Bus Interface Unit has built-in glitch suppression logic. Glitches is suppressed according to: 2 * I²C clock period. For example, with the 33 MHz (30. ns period) I²C clock glitches of 60ns or less is suppressed. This is within the 50 ns glitch suppression specified.

10.8 Reset Conditions

The I²C unit is reset with **I_RST#**. Software is responsible for ensuring the I²C unit is not busy (ISR[3]) before asserting reset. Software is also responsible for ensuring the I²C bus is idle when the unit is enabled after reset. When directed to reset, the I²C unit returns to its default reset condition with the exception of the ISAR. ISAR is not affected by a reset.

When the Unit Reset bit in the ICRx is set, only the 80331 I²C unit resets, the associated I²C MMRs remain intact. When resetting the I²C unit with the ICRx units reset, use the following guidelines:

1. In the ICRx register, set the reset bit and clear the remainder of the register.
2. Clear the ISRx register.
3. Clear reset in the ICRx.

10.9 Register Definitions

The following registers are associated with the I²C Bus Interface Units. Each I²C Bus Interface Unit has five memory-mapped control registers for independent operation. In register titles, x is 0 or 1 for unit 0 or 1, respectively.

They are all located within the peripheral memory- mapped address space of the 80331. See Section 5.8, "Register Definitions" on page 341 for the register addresses

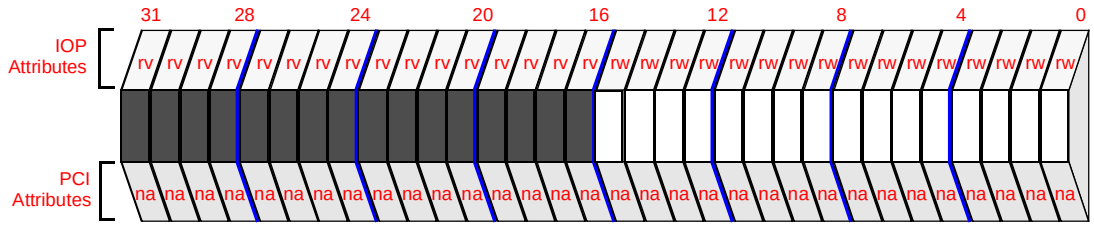
Table 288. I²C Register Summary

Section, Register Name, Acronym, Page
Section 10.9.1, "I²C Control Register x - ICRx" on page 576
Section 10.9.2, "I²C Status Register x - ISRx" on page 578
Section 10.9.3, "I²C Slave Address Register x - ISARx" on page 580
Section 10.9.4, "I²C Data Buffer Register x - IDBRx" on page 581
Section 10.9.5, "I²C Bus Monitor Register x - IBMRx" on page 582

10.9.1 I²C Control Register x - ICRx

The 80331 uses the bits in the I²C Control Register (ICRx) to control the I²C unit.

Table 289. I²C Control Register x - ICRx (Sheet 1 of 2)

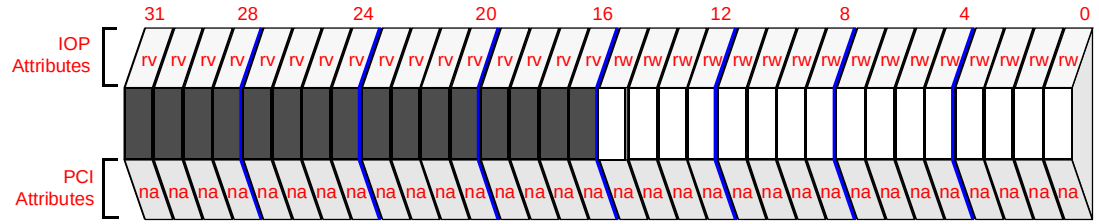


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F680H
1	FFFF F6A0H

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:16	0000H	Reserved
15	0	**Fast Mode:** 0 = 100 KBit/sec operation 1 = 400 KBit/sec operation
14	0_2	**Unit Reset**: 0 = No reset. 1 = Reset the I²C unit only.
13	0_2	**Slave Address Detected Interrupt Enable**: 0 = Disable interrupt. 1 = Enables the I²C unit to interrupt the 80331 upon detecting a slave address match or a general call address.
12	0_2	**Arbitration Loss Detected Interrupt Enable**: 0 = Disable interrupt. 1 = Enables the I²C unit to interrupt the 80331 upon losing arbitration while in master mode.
11	0_2	**Slave STOP Detected Interrupt Enable**: 0 = Disable interrupt. 1 = Enables the I²C unit to interrupt the 80331 when it detects a STOP condition while in slave mode.
10	0_2	**Bus Error Interrupt Enable**: 0 = Disable interrupt. 1 = Enables the I²C unit to interrupt the 80331 for the following I²C bus errors: • As a master transmitter, no Ack was detected after a byte was sent. • As a slave receiver, the I²C unit generated a Nack pulse. **NOTE:** Software is responsible for guaranteeing that misplaced START and STOP conditions do not occur. See Section 10.7, "Glitch Suppression Logic" on page 575.
09	0_2	**IDBR Receive Full Interrupt Enable**: 0 = Disable interrupt. 1 = Enables I²C unit to interrupt the 80331 when the IDBR has received a data byte from the I²C bus.
08	0_2	**IDBR Transmit Empty Interrupt Enable**: 0 = Disable interrupt. 1 = Enables the I²C unit to interrupt the 80331 after transmitting a byte onto the I²C bus.
07	0_2	**General Call Disable**: 0 = Enables the I²C unit to respond to general call messages. 1 = Disables I²C unit response to general call messages as a slave. This bit must be set when sending a master mode general call message from the I²C unit.

Table 289. I²C Control Register x - ICRx (Sheet 2 of 2)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F680H
1	FFFF F6A0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
06	0_2	I²C **Unit Enable**: 0 = Disables the unit and does not master any transactions or respond to any slave transactions. 1 = Enables the I²C unit (defaults to slave-receive mode). Software must guarantee the I²C bus is idle and that before setting this bit.
05	0_2	**SCL Enable**: 0 = Disables the I²C unit from driving the **SCL** line. 1 = Enables the I²C clock output for master mode operation.
04	0_2	**Master Abort**: used by the I²C unit when in master mode to generate a STOP without transmitting another data byte. 0 = The I²C unit transmits STOP using the STOP ICR bit only. 1 = The I²C unit sends STOP without data transmission. When in Master transmit mode, after transmitting a data byte, the ICR's Transfer Byte bit is clear and IDBR Transmit Empty bit is set. When no more data bytes need to be sent, setting master abort bit sends the STOP. The Transfer Byte bit (03) must remain clear. In master-receive mode, when a Nack is sent without a STOP (STOP ICR bit was not set) and the 80331 does not send a repeated START, setting this bit sends the STOP. Once again, the Transfer Byte bit (03) must remain clear.
03	0_2	**Transfer Byte**: used to send/receive a byte on the I²C bus. 0 = Cleared by I²C unit when the byte is sent/received. 1 = Send/receive a byte. The 80331 can monitor this bit to determine when the byte transfer has completed. In master or slave mode, after each byte transfer including Ack/Nack bit, the I²C unit holds the **SCL** line low (inserting wait states) until the Transfer Byte bit is set.
02	0_2	**Ack/Nack Control**: defines the type of Ack pulse sent by the I²C unit when in master receive mode. 0 = The I²C unit sends an Ack pulse after receiving a data byte. 1 = The I²C unit sends a negative Ack (Nack) after receiving a data byte. The I²C unit automatically sends an Ack pulse when responding to its slave address or when responding in slave-receive mode, independent of the Ack/Nack control bit setting.
01	0_2	**STOP**: used to initiate a STOP condition after transferring the next data byte on the I²C bus when in master mode. In master-receive mode, the Ack/Nack control bit must be set in conjunction with this bit. See Section 10.3.3.3, "STOP Condition" on page 557 for more details on the STOP state. 0 = Do not send a STOP. 1 = Send a STOP.
00	0_2	**START**: used to initiate a START condition to the I²C unit when in master mode. See Section 10.3.3.1, "START Condition" on page 557 for more details on the START state. 0 = Do not send a START. 1 = Send a START.

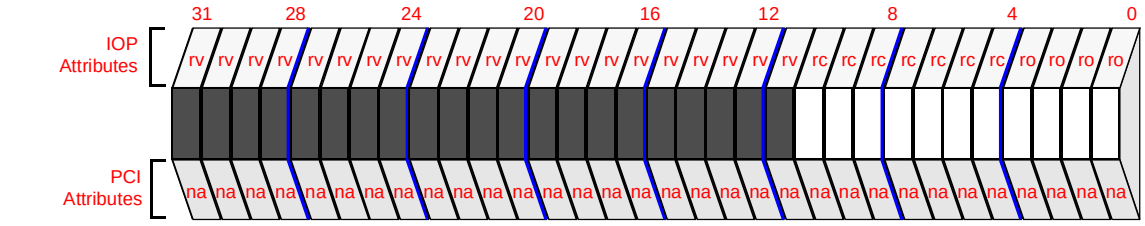
10.9.2 I²C Status Register x - ISRx

I²C interrupts are signalled to the 80331 interrupt controller by the I²C Interrupt Status Register (ISRx). Software uses the ISR bits to check the status of the I²C unit and bus. ISRx bits (bits 9-5) are updated after the Ack/Nack bit has completed on the I²C bus.

The ISRx is also used to clear interrupts signalled from the I²C Bus Interface Unit. These are:

- IDBRx Receive Full
- IDBRx Transmit Empty
- Slave Address Detected
- Bus Error Detected
- STOP Condition Detect
- Arbitration Lost

Table 290. I²C Status Register x - ISRx (Sheet 1 of 2)

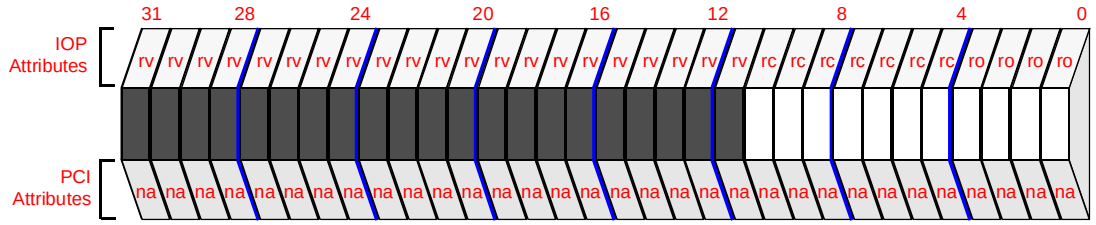


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F684H
1	FFFF F6A4H

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:11	000000H	Reserved
10	0_2	**Bus Error Detected**: 0 = No error detected. 1 = The I²C unit sets this bit when it detects one of the following error conditions: • As a master transmitter, no Ack was detected on the interface after a byte was sent. • As a slave receiver, the I²C unit generates a Nack pulse. **NOTE:** When an error occurs, I²C bus transactions continue. Software must guarantee that misplaced START and STOP conditions do not occur. See Section 10.4.4, "Arbitration" on page 562.
09	0_2	**Slave Address Detected**: 0 = No slave address detected. 1 = I²C unit detected a 7-bit address that matches the general call address or ISAR. An interrupt is signalled when enabled in the ICR.
08	0_2	**General Call Address Detected**: 0 = No general call address received. 1 = I²C unit received a general call address.
07	0_2	**IDBR Receive Full**: 0 = The IDBR has not received a new data byte or the I²C unit is idle. 1 = The IDBR register received a new data byte from the I²C bus. An interrupt is signalled when enabled in the ICR.

Table 290. I²C Status Register x - ISRx (Sheet 2 of 2)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F684H
1	FFFF F6A4H

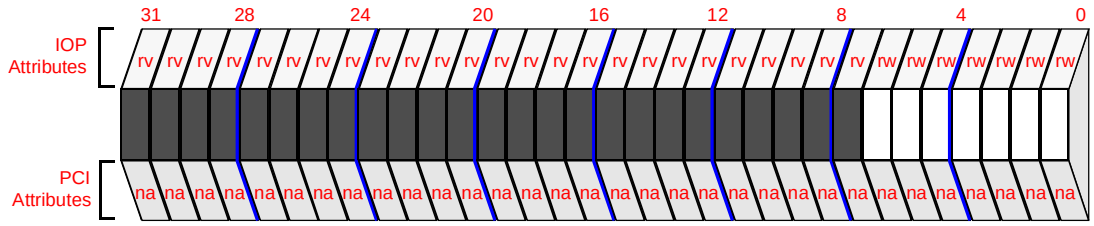
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
06	0_2	**IDBR Transmit Empty**: 0 = The data byte is still being transmitted. 1 = The I²C unit has finished transmitting a data byte on the I²C bus. An interrupt is signalled when enabled in the ICR.
05	0_2	**Arbitration Loss Detected**: used during multi-master operation. 0 = Cleared when arbitration is won or never took place. 1 = Set when the I²C unit loses arbitration.
04	0_2	**Slave STOP Detected**: 0 = No STOP detected. 1 = Set when the I²C unit detects a STOP while in slave-receive or slave-transmit mode.
03	0_2	I²C **Bus Busy**: 0 = I²C bus is idle or the I²C unit is using the bus (i.e., unit busy). 1 = Set when the I²C bus is busy but the 80331 I²C unit is not involved in the transaction.
02	0_2	**Unit Busy**: 0 = I²C unit not busy. 1 = Set when the 80331 I²C unit is busy. This is defined as the time between the first START and STOP.
01	0_2	**Ack/Nack Status**: 0 = The I²C unit received or sent an Ack on the bus. 1 = The I²C unit received or sent a Nack. This bit is used in slave transmit mode to determine when the byte transferred is the last one. This bit is updated after each byte and Ack/Nack information is received.
00	0_2	**Read/Write Mode**: 0 = The I²C unit is in master-transmit or slave-receive mode. 1 = The I²C unit is in master-receive or slave-transmit mode. This is the R/W# bit of the slave address. It is automatically cleared by hardware after a stop state.

10.9.3 I²C Slave Address Register x - ISARx

The I²C Slave Address Register (ISARx) (see Table 291) defines the I²C unit 7-bit slave address to which the 80331 responds when in slave-receive mode. This register is written by the 80331 before enabling I²C operations. The register is fully programmable (no address is assigned to the I²C unit) so it can be set to a value other than those of hard-wired I²C slave peripherals that might exist in the system. The ISAR is not affected by the 80331 being reset. The ISAR register default value is 0000000_2.

Table 291. I²C Slave Address Register x - ISARx



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F688H
1	FFFF F6A8H

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:07	000000H	Reserved
06:00	00H	I²C **Slave Address**: The 7-bit address to which the I²C unit responds when in slave-receive mode.

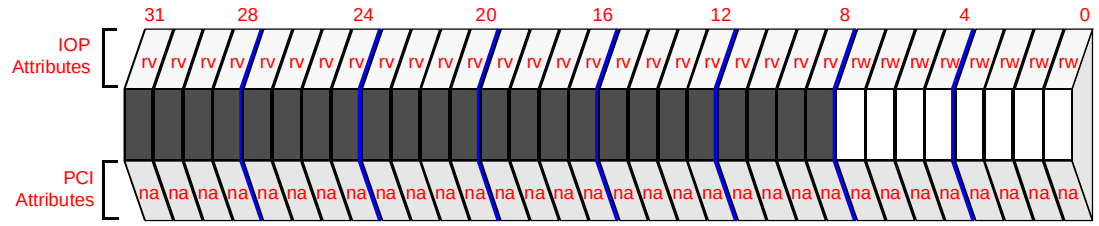
10.9.4 I²C Data Buffer Register x - IDBRx

The I²C Data Buffer Register (IDBRx) is used by the 80331 to transmit and receive data from the I²C bus. The accesses the IDBRx by the 80331 on one side and by the I²C shift register on the other. Data coming into the I²C Bus Interface Unit is received into the IDBRx after a full byte has been received and acknowledged. Data going out of the I²C Bus Interface Unit is written to the IDBRx by the Intel® XScale™ core and sent to the serial bus.

When the I²C Bus Interface Unit is in transmit mode (master or slave), the 80331 writes data to the IDBRx over the internal bus. This occurs when a master transaction is initiated or when the IDBRx Transmit Empty Interrupt is signalled. Data is moved from the IDBRx to the shift register when the Transfer Byte bit is set. The IDBR Transmit Empty Interrupt is signalled (when enabled) when a byte has been transferred on the I²C bus and the acknowledge cycle is complete. When the IDBRx is not written by the 80331 (and a STOP condition was not in place) before the I²C bus is ready to transfer the next byte packet, the I²C Bus Interface Unit inserts wait states until the Intel® XScale™ core writes the IDBRx and sets the Transfer Byte bit.

When the I²C Bus Interface Unit is in receive mode (master or slave), the processor reads IDBRx data over the internal bus. This occurs when the IDBRx Receive Full Interrupt is signalled. The data is moved from the shift register to the IDBRx when the Ack cycle is complete. The I²C Bus Interface Unit inserts wait states until the IDBR has been read. Refer to Section 10.4.3, "I²C Acknowledge" on page 561 for acknowledge pulse information in receiver mode. After the 80331 reads the IDBRx, the Ack/Nack Control bit is written and the Transfer Byte bit is written, allowing the next byte transfer to proceed on the I²C Bus. The IDBRx register is 00H after reset.

Table 292. I²C Data Buffer Register x - IDBRx



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F68CH
1	FFFF F6ACH

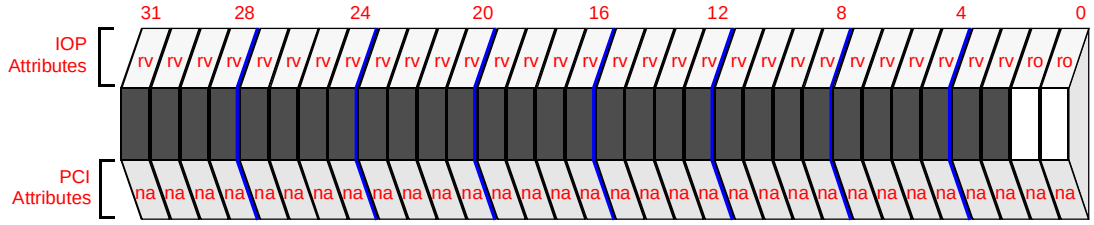
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:08	000000H	Reserved
07:00	00H	I²C **Data Buffer**: Buffer for I²C bus send/receive data.

10.9.5 I²C Bus Monitor Register x - IBMRx

The I²C Bus Monitor Register (IBMRx) tracks the status of the **SCL** and **SDA** pins. The values of these pins are recorded in this read-only register so that software may determine when the I²C bus is hung and the I²C unit must be reset.

Table 293. I²C Bus Monitor Register x - IBMRx



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F694H
1	FFFF F6B4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:02	0	Reserved
01	1	**SCL** Status: This bit continuously reflects the value of the **SCL** pin.
00	1	**SDA** Status: This bit continuously reflects the value of the **SDA** pin.

UARTs 11

This chapter describes the Universal Asynchronous Receiver/Transmitter (UART) serial ports. The Intel® 80331 I/O processor (80331) UARTs are controlled via programmed I/O through memory-mapped registers.

11.1 Overview

Each asynchronous serial port supports all the functions of a 16550 UART. Each 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 the processor. The processor can read the complete status of a UART at any time during the functional operation. Available status information includes the type and condition of the transfer operations being performed by a UART and any error conditions (parity, overrun, framing, or break interrupt).

Each serial port can operate in either FIFO or non-FIFO mode. In FIFO mode, a 64-byte transmit FIFO holds data from the processor to be transmitted on the serial link and a 64-byte Receive FIFO buffers data from the serial link until read by the processor.

Each UART includes a programmable baud rate generator which is capable of dividing the input clock by divisors of 1 to (2^{16}–1) and producing a 16X clock to drive the internal transmitter and receiver logic. Interrupts can be programmed to the user requirements, minimizing the computing required to handle the communications link. Each UART operates in a polled or an interrupt driven environment which is selected by software.

The UART hardware is responsible for executing serial protocol communication and for providing the programming interface. The UART features include:

- Registers are compatible with the 16550 and 16750
- 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
- Baud-rate generator allows division of clock by 1 to ($2^{16}-1$) and generates an internal 16X clock; baud-rate can be manually or automatically programmed via auto-baud-rate detection circuitry
- Modem control functions (CTS#, RTS#)
- Autoflow capability controls data I/O without generating Interrupts:
  - RTS# (output) controlled by UART Receiver FIFO
  - CTS# (input) from modem controls UART transmitter
- 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 115kbps)
- False start bit detection
- 64-byte Transmit FIFO
- 64-byte Receive FIFO with programmable threshold
- Complete status reporting capability
- 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

11.1.1 Compatibility with 16550 and 16750

The UARTs can be programmed to be functionally compatible with industry standard 16550 and 16750. Each UART supports most of the 16550 and 16750 functions and has additional features, as listed below.

- DMA requests for transmit and receive data services
- NRZ encoding/decoding function
- 64-byte Transmit/Receive FIFO buffers
- Programmable Receive FIFO threshold
- Auto baud-rate detection
- Auto flow

11.2 Signal Descriptions

The name and description of external signals connected to a UART module are shown in Table 294. These signals are multiplexed with the GPIO signals as specified in Chapter 16, "General Purpose I/O Unit". The selection between GPIO and UART function for these multiplexed pins is controlled by the UART Unit Enable bit (UUE bit 6) of the "UART x Interrupt Enable Register" on page 596.

Table 294. UART Signal Descriptions

Name*	Type	Description
Ux_RXD	Input	SERIAL INPUT: Serial data input from device pin to the receive shift register.
Ux_TXD	Output	SERIAL OUTPUT: Composite serial data output to the communications link-peripheral, modem, or data set. The TXD signal is set to the MARKING (logic 1) state upon a Reset operation.
Ux_CTS#	Input	CLEAR TO SEND: When low, this pin indicates that the receiving UART is ready to receive data. When the receiving UART deasserts **CTS#** high, the transmitting UART should stop transmission to prevent overflow of the receiving UARTs buffer. The **CTS#** signal is a modem-status input whose condition can be tested by the host processor or by the UART when in Autoflow mode as described below: Non-Autoflow Mode: When not in Autoflow mode, bit 4 (CTS) of the Modem Status register (MSR) indicates the state of **CTS#**. Bit 4 is the complement of the **CTS#** signal. Bit 0 (DCTS) of the Modem Status register indicates whether the **CTS#** input has changed state since the previous reading of the Modem Status register. **CTS#** has no effect on the transmitter. The user can program the UART to interrupt the processor when DCTS changes state. The programmer can then stall the outgoing data stream by starving the transmit FIFO or disabling the UART with the IER register. Note: When UART transmission is stalled by disabling the UART, the user does not receive an MSR interrupt when **CTS#** reasserts. This is because disabling the UART also disables interrupts. To get around this, the user can use Auto CTS in Autoflow Mode, or program the **CTS#** pin to interrupt. Autoflow Mode: In Autoflow mode, the UART Transmit circuity checks the state of **CTS#** before transmitting each byte. IF **CTS#** is high, no data is transmitted. See Section 11.4.7, UART x Modem Control Register for more information on Auto CTS mode.
Ux_RTS#	Output	REQUEST TO SEND: When low, this informs the remote device that the UART is ready to receive data. A reset operation sets this signal to its Inactive (high) state. LOOP mode operation holds this signal in its Inactive state. Non-Autoflow Mode: The **RTS#** output signal can be asserted by setting bit 1 (RTS) of the Modem Control register to a 1. The RTS bit is the complement of the **RTS#** signal. Autoflow Mode: **RTS#** is automatically asserted by the autoflow circuitry when the Receive buffer exceeds its programmed threshold. It is deasserted when enough bytes are removed from the buffer to lower the data level back to the threshold. See Section 11.4.7, UART x Modem Control Register for more information on Auto RTS mode.

Note: * "x" in signal name replaced with either "0" or "1" for UART-0 or UART-1 respectively.

11.3 Theory of Operation

The format of a UART data frame is shown in Figure 105.

Figure 105. Example UART Data Frame

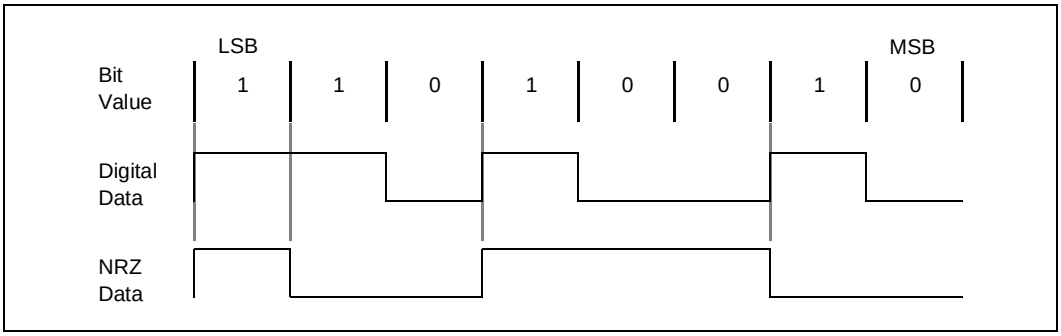
	Start Bit	Data< 0>	Data< 1>	Data< 2>	Data< 3>	Data< 4>	Data< 5>	Data< 6>	Data< 7>	Parity Bit	Stop Bit 1	Stop Bit 2	
TXD or RXD pin													
		LSB							MSB				

Shaded bits are optional and can be programmed by user. —>

Each data frame is between 7 bits and 12 bits long, depending on the size of data programmed and when parity and stop bits are enabled. The frame begins with a start bit that is represented by a high-to-low transition. Next, five to eight bits of data are transmitted, beginning with the least significant bit. An optional parity bit follows, which is set when even parity is enabled and an odd number of ones exist within the data byte; or when odd parity is enabled and the data byte contains an even number of ones. The data frame ends with one, one-and-one-half or two stop bits (as programmed by the user), which is represented by one or two successive bit periods of a logic one.

NRZ coding can be used by the UART to represent individual bit values. NRZ coding is enabled when Interrupt Enable Register (IER) bit-5 is set to high. A one is represented by a line transition and a zero is represented by no line transition. Figure 106 shows the NRZ coding of the data byte 8b 0100 1011. Note that the byte's LSB is transmitted first.

Figure 106. NRZ Bit Encoding Example – (0100 1011)



The unit is disabled upon reset, and users need to program GPIO registers first and then enable the unit by setting the UART Unit Enable bit (UUE, bit-6) of Interrupt Enable Register. When the unit is enabled, the receiver starts looking for the start bit of a frame; the transmitter sends data to the transmit data pin when there is data available in the transmit FIFO. Transmit data can be written to the FIFO before the unit is enabled. When the UART is disabled, the transmitter/receiver finishes the current byte being transmitted/received (when it is in the middle of transmitting/receiving a byte), and stops transmitting/receiving more data. Disabling the UART with the UUE bit does not clear transmission/reception with the original data.

Each UART has a Transmit FIFO and a Receive FIFO each holding 64 characters of data. There

are two methods for moving data into/out of the FIFOs: **Interrupts and Polling**.

11.3.1 FIFO Interrupt Mode Operation

11.3.1.1 Receiver Interrupt

When the Receive FIFO and receiver interrupts are enabled (FCR[0]=1 and IER[0]=1), receiver interrupts occur as follows:

- The Receive Data Available Interrupt is asserted when the FIFO has reached its programmed trigger level. The interrupt is cleared when the FIFO drops below the programmed trigger level.
- The IIR Receive Data Available indication also occurs when the FIFO trigger level is reached, and like the interrupt, the bits are cleared when the FIFO drops below the trigger level.
- The Data Ready bit (DR in LSR register) is set to 1 as soon as a character is transferred from the shift register to the Receive FIFO. This bit is reset to 0 when the FIFO is empty.

11.3.1.2 Transmit Interrupt

When the transmitter FIFO and transmitter interrupt are enabled (FCR[0]=1, IER[1]=1), transmit interrupts occur as follows:

- When the Flow Control Register Transmitter Interrupt Level (TIL) bit (FCR[3]) is clear (0), The Transmit Data Request interrupt occurs when the transmit FIFO is half empty or more than half empty. The interrupt is cleared when the data level exceeds the half-empty mark. The interrupt is cleared as soon as the Transmit Holding Register is written or the IIR is read. 1 to 32 characters may be written to the transmit FIFO while servicing the interrupt when TIL=0.
- When the Flow Control Register Transmitter Interrupt Level (TIL) bit is set (1), The Transmit Data Request Interrupt occurs when the Transmit FIFO is empty. The interrupt is cleared as soon as the Transmit Holding Register is written or the IIR is read. 1 to 64 characters may be written to the Transmit FIFO while servicing the interrupt when TIL = 1.

Users could cause the UART Transmit FIFO to overflow when too many characters are written. FIFO underflow does not cause an error as the UART waits for the Transmit FIFO to be serviced.

11.3.2 Removing Trailing Bytes In Interrupt Mode

When the number of entries in the Receive FIFO is less than its trigger level, and no additional data is received, the remaining bytes are called trailing bytes.When the receive FIFO is being serviced by processor interrupts, trailing bytes need to be removed via the processor using the 16550 compliant character timeout interrupt: Time Out Detected (TOD) bit of Interrupt Identification Register. To enter this mode, users need to insure that the character timeout interrupt is enabled via IER[4].

To remove trailing bytes in Interrupt mode, the user must wait for the character timeout interrupt and then read all remaining bytes as indicated in the FIFO Occupancy Register (FOR), or read one byte at a time until the FIFO is empty. This can be determined by polling the Line Status Register bit 0 through programmed I/O.

11.3.2.1 Character Timeout Interrupt

When the Receiver FIFO and Receiver Timeout Interrupt are enabled, a character timeout interrupt (TOD) occurs to signal the presence of trailing bytes. The Interrupt is cleared and the timer is reset when a character is read from the Receiver FIFO. When a timeout Interrupt has not occurred, the timeout timer is reset after a new character is received or after the processor reads the Receiver FIFO.

When enabled via IER[4], a character timeout occurs under the following conditions:

- At least one character is in the FIFO.
- A character has not been received for the amount of time it takes to receive four or more characters at the current baud rate.
- The FIFO has not been read for the amount of time it takes to receive four or more characters

11.3.3 FIFO Polled Mode Operation

With the FIFOs enabled (TRFIFOE bit of FCR set to 1), clearing IER[4:0] puts the serial port in the FIFO polled mode of operation. Since the receiver and the transmitter are controlled separately, either one or both can be in the Polled Operation mode. In this mode, software checks Receiver and Transmitter status via the LSR. The processor polls the following bits for Receive and Transmit Data Service.

11.3.3.1 Receive Data Service

- Processor should check *Data Ready bit of LSR* which is set when 1 or more bytes remains in the Receive FIFO or Receive Buffer register (RBR).

11.3.3.2 Transmit Data Service

- Processor should check *Transmit Data Request bit of LSR* which is set when transmitter needs data.
- • Processor can also check *Transmitter Empty bit of LSR*, which is set when the Transmit FIFO or Holding register is empty.

11.3.4 Autoflow Control

Autoflow Control uses the Clear-to-Send (**CTS#**) and Request-to-Send (**RTS#**) signals to automatically control the flow of data between the UART and external modem. When autoflow is enabled, the remote device is not allowed to send data unless the UART asserts nRTS low. When the UART deasserts **RTS#** while the remote device is sending data, the remote device is allowed to send one additional byte after **RTS#** is deasserted. An overflow could occur when the remote device violates this rule. Likewise, the UART is not allowed to transmit data unless the remote device asserts **CTS#** low. This feature increases system efficiency and eliminates the possibility of a Receive FIFO Overflow error due to long Interrupt latency.

Autoflow mode can be used in two ways: **Full autoflow,** automating both **CTS#** and **RTS#**, and **half autoflow,** automating only **CTS#**. Full Autoflow is enabled by writing a 1 to bits 1 and 5 of the Modem Control register (MCR). Auto-CTS-Only mode is enabled by writing a 1 to bit 5 and a 0 to bit 1 of the MCR register.

11.3.4.1 RTS Autoflow

When in full autoflow mode, **RTS#** is asserted when the UART FIFO is ready to receive data from the remote transmitter. This occurs when the amount of data in the Receive FIFO is below the programmable threshold value. When the amount of data in the Receive FIFO reaches the programmable threshold, **RTS#** is deasserted. It is asserted once again when enough bytes are removed from the FIFO to lower the data level below the threshold.

11.3.4.2 CTS Autoflow

When in Full or Half-Autoflow mode, **CTS#** is asserted by the remote receiver when the receiver is ready to receive data from the UART. The UART checks **CTS#** before sending the next byte of data and does not transmit the byte until **CTS#** is low. When **CTS#** goes high while the transfer of a byte is in progress, the transmitter completes this byte.

Note: Autoflow mode can be used only in conjunction with FIFO mode.

11.3.5 Auto-Baud-Rate Detection

Each UART supports auto-baud-rate detection. When enabled, the UART counts the number of 33.334 MHZ clock cycles within the *star*t-bit pulse. This number is then written into the Auto-Baud-Count register (ACR) and is used to calculate the baud rate. When the ACR is written, a Auto-Baud-Lock Interrupt is generated (when enabled), and the UART automatically programs the Divisor Latch registers with the appropriate baud rate. When preferred, the processor can read the Auto-Baud- Count register and use this information to program the Divisor-Latch registers with a baud rate calculated by the processor. After the baud rate has been programmed, it is the responsibility of the processor to verify that the predetermined characters (usually **AT** or **a**t) are being received correctly.

When the UART programs Divisor Latch registers, users can choose between two auto-baud calculation methods: **table-** and **formula-based**. The method is selected via bit *ABT* of the Auto-Baud Control register (ABR). When the formula method is used, any baudrate allowed in Equation 13 can be programmed by the UART. This method works well for higher baud rates, but could possibly fail below 28.8 kbps when the remote transmitter's actual baud rate differs by more than one percent of its target.

Equation 13. Baud-Rate Equation

$$BaudRate = \frac{33.334\text{Mhz}}{(16XDivisor)}$$

The table method is more immune to such errors as the table rejects uncommon baud rates and rounds to the common ones. The table method allows any baud rate in Equation 13 above 28.8 kbps. Below 28.8 kbps the only baud rates which can be programmed by the UART are 19200, 14400, 9600, 4800, 1200, and 300 baud. Some typical values for Divisor and corresponding baud rates are provided in Table 295. Baud rates above 3600 baud require only Divisor Latch Low Register to be programmed, as Divisor Latch High Register would be 0.

Table 295. Divisor Values for Typical Baud Rates

Baud Rate	Divisor	UART Rate	Error
115.2K	18	115.74K	0.47%
57.6K	36	57.87K	0.47%
38.4K	54	38.58K	0.47%
33.6K	62	33.60K	0.01%
28.8K	72	28.94K	0.47%
19.2K	109	19.29K	0.47%
14.4K	145	14.47K	0.47%
9600	217	9645	0.47%
4800	434	4800	0.01%
3600	579	3600	0.01%
2400	868	2400	0.01%
1200	1736	1200	0.01%
600	3472	600	0.01%
300	6944	300	0.01%

When the baud rate is detected, auto-baud circuitry disarms itself by clearing bit *ABE* of the Auto-Baud Control register (ABR). When users want to rearm the circuitry, the *ABE* bit must be rewritten.

Note: For the auto-baud-rate detection circuit to work correctly, the first data bit transmitted after the start bit must be a logic '1'. When a logic '0' is transmitted instead, the autobaud circuit counts the zero as part of the start bit, resulting in an incorrect baud rate being programmed into the DLL and DLH registers.

11.3.6 Manual Baud Rate Selection

Each UART contains a programmable Baud Rate Generator that is capable of taking the fixed input clock of 33.334 MHz and dividing it by any divisor from 1 to (2^{16}–1). The baud-rate generator output frequency is 16 times the baud rate. Two 8-bit registers store the divisor in a 16-bit binary format. These Divisor Registers must be loaded during initialization to ensure proper operation. When both Divisor Latches are loaded with 0, the 16X output clock is stopped. Access to the Divisor latch can be done with a word write. Equation 13 or Table 295 are used by the programmer to select the Divisor Latch value for the desired baud rate.

11.4 Register Descriptions

There are 15 registers in each UART. The registers are all 32 bit registers, but only lower 8 bits have valid data. The 12 UART registers share eight address locations in the MMR address space. Table 296 shows the registers and their addresses as offsets of a base address. The base address for each UART is 32 bits and is internal bus address FFFF F700H for UART 0, and FFFF F740H for UART 1. Note that the state of the Divisor Latch Bit (DLAB), which is the MOST significant bit of the Serial Line Control Register, affects the selection of certain of the UART registers. The DLAB bit must be set high by the system software to access the Baud Rate Generator Divisor Latches.

Table 296. UART Register Addresses as Offsets of a Base

UART Register Addresses	DLAB Bit Value	Name	Register Accessed
Base	0	UxRBR	UART x Receive BUFFER (read only)
Base	0	UxTHR	UART x Transmit BUFFER (write only)
Base + 04H	0	UxIER	UART x Interrupt Enable (R/W)
Base + 08H	X	UxIIR	UART x Interrupt I.D. (read only)
Base + 08H	X	UxFCR	UART x FIFO Control (write only)
Base + 0CH	X	UxLCR	UART x Line Control (R/W)
Base + 10H	X	UxMCR	UART x Modem Control (R/W)
Base + 14H	X	UxLSR	UART x Line Status (Read only)
Base + 18H	X	UxMSR	UART x Modem Status (Read only)
Base + 1CH	X	UxSPR	UART x Scratch Pad (R/W)
Base	1	UxDLL	UART x Divisor Latch (Low Byte, R/W)
Base + 04H	1	UxDLH	UART x Divisor Latch (High Byte, R/W)
Base + 24H	X	UxFOR	UART x FIFO Occupancy Register (R/W)
Base + 28H	X	UxABR	UART x Autobaud Control Register (R/W)
Base + 2CH	X	UxACR	UART x Autobaud Count Register (read only)

Table 297. UART Unit Registers

Section, Register Name, Acronym (page)
Section 11.4.1, "UART x Receive Buffer Register" on page 594
Section 11.4.2, "UART x Transmit Holding Register" on page 595
Section 11.4.3, "UART x Interrupt Enable Register" on page 596
Section 11.4.4, "UART x Interrupt Identification Register" on page 597
Section 11.4.5, "UART x FIFO Control Register" on page 599
Section 11.4.6, "UART x Line Control Register" on page 601
Section 11.4.7, "UART x Modem Control Register" on page 603
Section 11.4.8, "UART x Line Status Register" on page 605
Section 11.4.9, "UART x Modem Status Register" on page 608
Section 11.4.10, "UART x Scratchpad Register" on page 609
Section 11.4.11, "Divisor Latch Registers" on page 610
Section 11.4.13, "UART x Auto-Baud Control Register" on page 612
Section 11.4.14, "UART x Auto-Baud Count Register" on page 613

Table 298. UART Register MMR Addresses

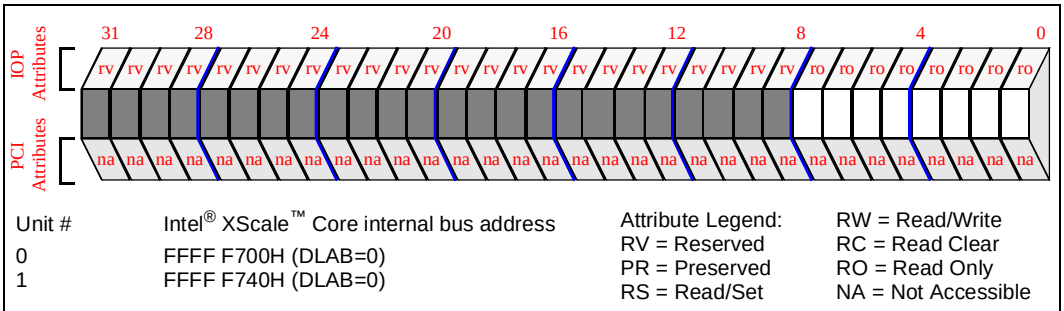
UART Register Addresses	DLAB Bit Value	Name	Register Accessed
FFFF F700H	0	U0RBR	UART 0 Receive BUFFER (read only)
	0	U0THR	UART 0 Transmit BUFFER (write only)
FFFF F704H	0	U0IER	UART 0 Interrupt Enable (R/W)
FFFF F708H	X	U0IIR	UART 0 Interrupt I.D. (read only)
	X	U0FCR	UART 0 FIFO Control (write only)
FFFF F70CH	X	U0LCR	UART 0 Line Control (R/W)
FFFF F710H	X	U0MCR	UART 0 Modem Control (R/W)
FFFF F714H	X	U0LSR	UART 0 Line Status (Read only)
FFFF F718H	X	U0MSR	UART 0 Modem Status (Read only)
FFFF F71CH	X	U0SPR	UART 0 Scratch Pad (R/W)
FFFF F700H	1	U0DLL	UART 0 Divisor Latch (Low Byte, R/W)
FFFF F704H	1	U0DLH	UART 0 Divisor Latch (High Byte, R/W)
FFFF F720H	X	n/a	Reserved
FFFF F724H	X	U0FOR	UART 0 FIFO Occupancy Register (R/W)
FFFF F728H	X	U0ABR	UART 0 Autobaud Control Register (R/W)
FFFF F72CH	X	U0ACR	UART 0 Autobaud Count Register (read only)
FFFF F730H through FFFF F73FH	X	n/a	Reserved
FFFF F740H	0	U1RBR	UART 1 Receive BUFFER (read only)
	0	U1THR	UART 1 Transmit BUFFER (write only)
FFFF F744H	0	U1IER	UART 1 Interrupt Enable (R/W)
FFFF F748H	X	U1IIR	UART 1 Interrupt I.D. (read only)
	X	U1FCR	UART 1 FIFO Control (write only)
FFFF F74CH	X	U1LCR	UART 1 Line Control (R/W)
FFFF F750H	X	U1MCR	UART 1 Modem Control (R/W)
FFFF F754H	X	U1LSR	UART 1 Line Status (Read only)
FFFF F758H	X	U1MSR	UART 1 Modem Status (Read only)
FFFF F75CH	X	U1SPR	UART 1 Scratch Pad (R/W)
FFFF F740H	1	U1DLL	UART 1 Divisor Latch (Low Byte, R/W)
FFFF F744H	1	U1DLH	UART 1 Divisor Latch (High Byte, R/W)
FFFF F760H	X	n/a	Reserved
FFFF F764H	X	U1FOR	UART 1 FIFO Occupancy Register (R/W)
FFFF F768H	X	U1ABR	UART 1 Autobaud Control Register (R/W)
FFFF F76CH	X	U1ACR	UART 1 Autobaud Count Register (read only)
FFFF F770H through FFFF F77FH	X	n/a	Reserved

11.4.1 UART x Receive Buffer Register

In non-FIFO mode, this register holds the character(s) received by the UART Receive Shift register. When it receives fewer than eight bits, the bits are right-justified and the leading bits are zeroed. Reading the register empties the register and resets the *data ready (DR)* bit in the Line Status register to 0. Other (error) bits in the Line Status register are not cleared. In FIFO mode, this register latches the value of the data byte(s) at the bottom of the FIFO.

When the UART is in eight-bit Peripheral Bus mode, the 24 most significant bits must be ignored and not used. Reading these bits returns unpredictable results.

Table 299. UART x Receive Buffer Register - (UxRBR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F700H (DLAB=0)
1	FFFF F740H (DLAB=0)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

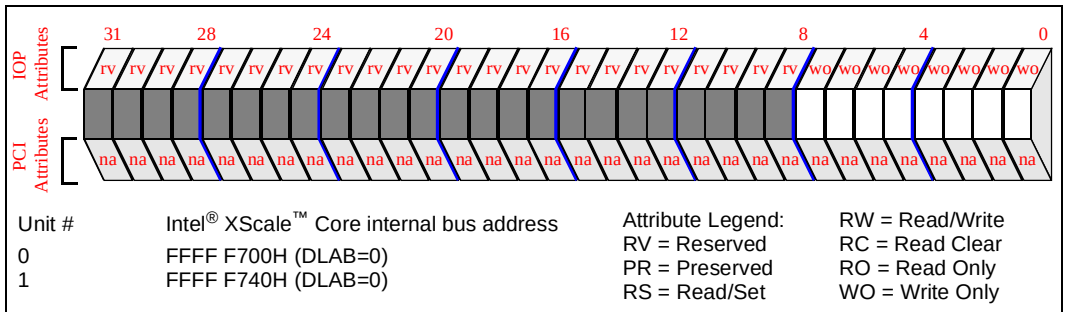
Bit	Default	Description
31:8	00h	Reserved
7:0	00h	Data Byte

11.4.2 UART x Transmit Holding Register

This register holds the next data byte(s) 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 one (see Table 303).

In FIFO mode, writing to THR puts data to the top of the FIFO. The data at the bottom of the FIFO is loaded to the Shift register when it is empty. In eight-bit Peripheral mode, the 24 most significant bits are ignored and is not transmitted.

Table 300. UART x Transmit Holding Register - (UxTHR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F700H (DLAB=0)
1	FFFF F740H (DLAB=0)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	WO = Write Only

Bit	Default	Description
31:8	00h	Reserved
7:0	00h	Data Byte

11.4.3 UART x Interrupt Enable Register

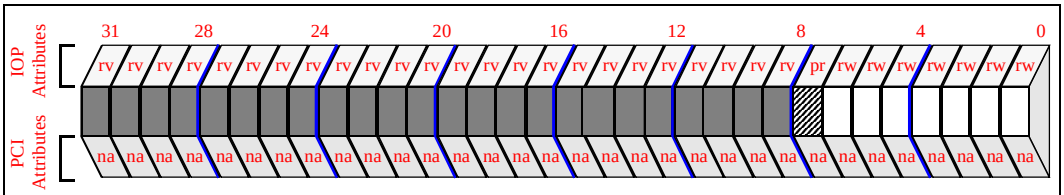
This register enables six types of interrupts which set a value in the Interrupt Identification register. Each of the six interrupt types can be disabled by clearing the appropriate bit of the IER register. Similarly, by setting the appropriate bits, selected interrupts can be enabled.

This register also has the control bits of the unit enable and NRZ coding enable. The use of bit 7 to bit 4 is different from the register definition of standard 16550.

Note: A global interrupt enable/disable exists in the Modem Control Register bit 3 (IE). After reset, this bit must be set or no interrupts occurs, regardless of the state of the IER bits. See Section 11.4.7, "UART x Modem Control Register" on page 603.

Note: Users need to program the GPIO registers before enabling the UART. See "GPIO Pin Multiplexing" on page 720.

Table 301. UART x Interrupt Enable Register - (UxIER)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F704H (DLAB=x)
1	FFFF F744H (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7	0_2	Preserved
6	0_2	UART Unit Enable (UUE): Controls UART operation and pin multiplexing with the GPIO. Refer to "GPIO Pin Multiplexing" on page 720 for details. 0 = the unit is disabled and the shared pins operate as GPIO signals. 1 = the unit is enabled and the shared pins operate as UART signals
5	0_2	**NRZ coding Enable (NRZE):** 0 = NRZ coding disabled 1 = NRZ coding enabled
4	0_2	Receiver Time Out Interrupt Enable: (RTOIE) 0 = Receiver data Time out Interrupt disabled 1 = Receiver data Time out Interrupt enabled
3	0_2	**Modem Interrupt Enable (MIE):** 0 = Modem Status interrupt disabled 1 = Modem Status interrupt enabled
2	0_2	**Receiver Line Status Interrupt Enable (RLSE):** 0 = Receiver Line Status interrupt disabled 1 = Receiver Line Status interrupt enabled
1	0_2	**Transmit Data request Interrupt Enable (TIE):** 0 = Transmit FIFO Data Request interrupt disabled 1 = Transmit FIFO Data Request interrupt enabled
0	0_2	**Receiver Data Available Interrupt Enable (RAVIE):** 0 = Receiver Data Available (Trigger level reached) interrupt disabled 1 = Receiver Data Available (Trigger level reached) interrupt enabled

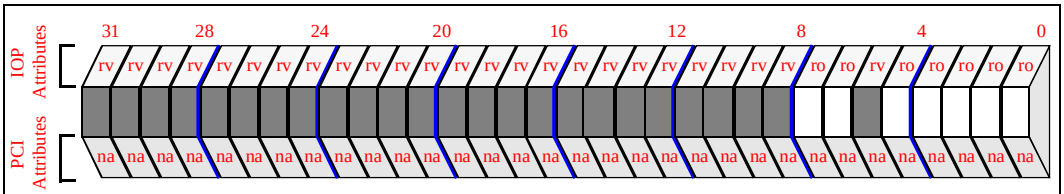
11.4.4 UART x Interrupt Identification Register

The IIR register is read to determine the type and source of UART interrupts. To be 16550 compatible, the lower 4 bits (0-3) of the IIR register are priority encoded as shown in Table 303. When two or more interrupts represented by bits (0-3) occur, only the interrupt with the highest priority is displayed. The upper 4 bits, (4-7) are not priority encoded. These bits asserts/deasserts independently of the lower 4 bits.

Bit 0 (nIP) is used to indicate the existence of an interrupt in the priority encoded bits (0-3) of the IIR register. A low signal on this bit indicates an encoded interrupt is pending. When this bit is high, no encoded interrupt is pending, regardless of the state of the other 3 bits. IP# has no effect or association with the upper bits four bits (4-7) which assert/deassert independently of IP#.

In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels (listed in Table 303) 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.

Table 302. UART x Interrupt Identification Register - (UxIIR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F708H (DLAB=x)
1	FFFF F748H (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7:6	00_2	**FIFO Mode Enable Status (FIFOES[1:0]):** 00 = Non-FIFO mode is selected 01 = Reserved 10 = Reserved 11 = FIFO mode is selected (TRFIFOE = 1)
5	0_2	Reserved
4	0_2	**Autobaud Lock (ABL)** 0 = Autobaud circuitry has not programmed Divisor Latch registers (DLL/DLH) 1 = Divisor Latch registers (DLL/DLH) programmed by autobaud circuitry
3	0_2	**Time Out Detected (TOD):** 0 = No time out interrupt is pending 1 = Time out interrupt is pending. (FIFO mode only)
2:1	0_2	**Interrupt Source Encoded (IID[1:0]):** indicates a Modem Status Interrupt when the IP# bit is low. When IP# bit is high, there is no Interrupt. 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	1_2	**Interrupt Pending (IP#):** 0 = Interrupt is pending. (Active low) 1 = No interrupt is pending

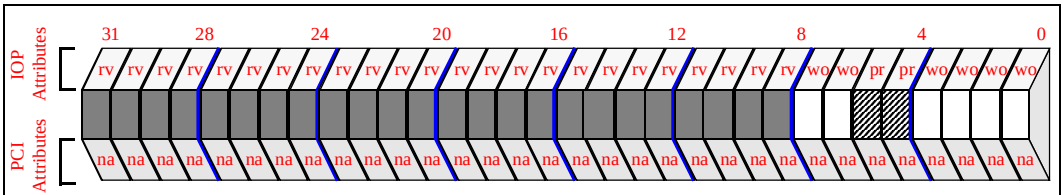
Table 303. Interrupt Identification Register Decode

	Interrupt ID bits				Interrupt SET/RESET Function			
	3	2	1	0	Priority	Type	Source	RESET Control
IP#	0	0	0	1	-	None	No Interrupt is pending.	-
IID[11]	0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt.	Reading the Line Status Register.
IID[10]	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.
TOD	1	1	0	0	Second Highest	Character Timeout 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.
IID[01]	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 (when the source of the interrupt) or writing into the Transmit Holding Register. Reading the IIR Register (when the source of the interrupt) or writing to the Transmitter FIFO.
IID[00]	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.
Non Prioritized Interrupts								
ABL	4				None	Autobaud Lock Indication	Autobaud circuitry has locked onto the baud rate.	Reading the IIR Register

11.4.5 UART x 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.

Table 304. UART x FIFO Control Register - (UxFCR) (Sheet 1 of 2)

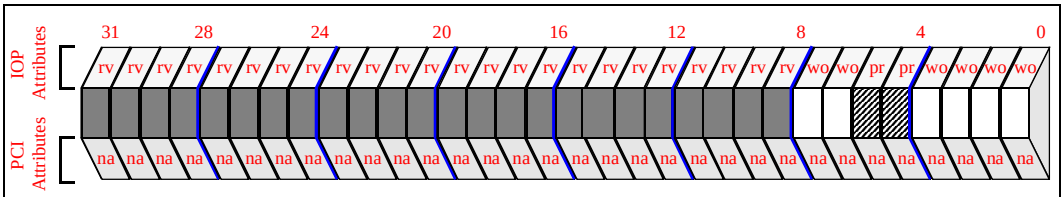


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F708H (DLAB=x)
1	FFFF F748H (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7:6	00_2	**Interrupt Trigger Level (ITL):** When the number of bytes in the receiver 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
5:4	0_2	Preserved
3	0_2	**Transmitter Interrupt Level (TIL):** Setting TIL causes Transmitter Interrupts to occur when the Transmit FIFO is empty. Clearing TIL causes Transmitter Interrupts to occur when the Transmit FIFO is half empty. 0 = Interrupt when FIFO is half empty 1 = Interrupt when FIFO is empty

Table 304. UART x FIFO Control Register - (UxFCR) (Sheet 2 of 2)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F708H (DLAB=x)
1	FFFF F748H (DLAB=x)

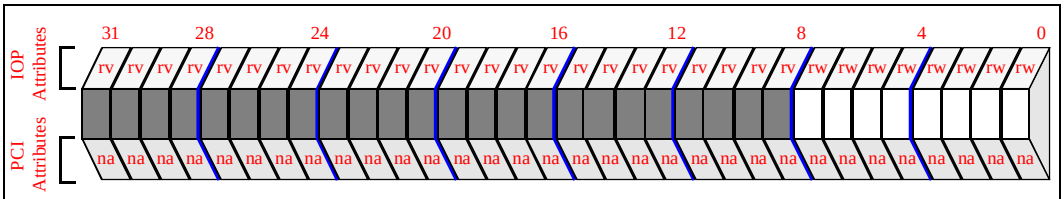
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
2	0_2	**Reset Transmitter FIFO (RESETTF):** When set, the Transmitter FIFO counter is reset to clear all the bytes in the FIFO. The *TDRQ* bit of LSR is set generating a Transmitter Requests Data Interrupt IID field of IIR when the *TIE* bit in the IER register is set. The Transmitter Shift register is not reset; it completes the current transmission. Any transmit FIFO Service-Request Interrupts are cleared. Note: After the FIFO is cleared, RESETTF is automatically reset to 0. 0 = no effect 1 = The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
1	0_2	**Reset Receiver FIFO (RESETRF):** When set, the receiver FIFO counter is reset to clear all the bytes in the FIFO. The DR bit in 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. Any Receive FIFO Service Request Interrupts are cleared. Note: After the FIFO is cleared, RESETRF is automatically reset to 0. 0 = no effect 1 = The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0
0	0_2	**Transmit and Receive FIFO Enable (TRFIFOE):** Enables/disables the transmitter and receiver FIFOs. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFOs are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFOs. When changing from FIFO mode to non-FIFO mode and vice versa, data is automatically cleared from the FIFOs. Any FIFO Service Request Interrupts are cleared when TRFIFOE is cleared. Note: This bit must be 1 when other bits in this register are written, or the other bits are not programmed. 0 = FIFOs are disabled 1 = FIFOs are enabled

11.4.6 UART x Line Control Register

In the Line Control Register, the system programmer specifies 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 LCR has bits for accessing the Divisor Latch registers and causing a Break condition. The programmer can also read the contents of the Line Control Register. The read capability simplifies system programming and eliminates the need for separate storage in system memory.

Table 305. UART x Line Control Register - (UxLCR) (Sheet 1 of 2)

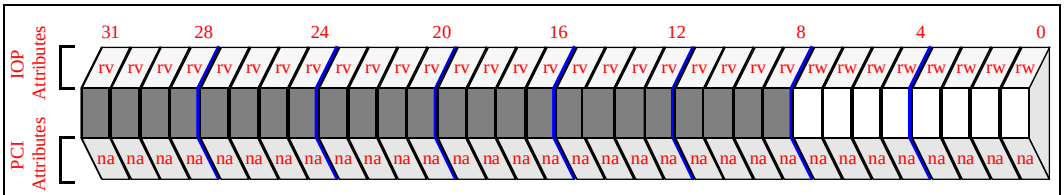


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F70CH (DLAB=x)
1	FFFF F74CH (DLAB=x)

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7	0_2	**Divisor Latch register Access Bit (DLAB):** This bit must be set (1) to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be clear (0) to access the Receiver Buffer, the Transmit-Holding Register, or the Interrupt-Enable Register. This bit does not have to be set when using autobaud. 0 = access Transmit Holding register (THR), Receive Buffer Register (RBR) and Interrupt Enable Register. 1 = access Divisor Latch Registers (DLL and DLH).
6	0_2	**Set break (SB):** Causes a Break condition to the receiving UART. When SB is set (1), the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is clear (0). This bit acts only on the TXD pin and has no effect on the transmitter logic. In FIFO mode, wait for the transmitter to be idle (TEMT=1) to set and clear the break bit. 0 = no effect on TXD output. 1 = forces TXD output to 0 (space).
5	0_2	**Sticky Parity (STKYP):** Can be used in multiprocessor communications. When PEN and STKYP are set (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. When EPS is 0, then the bit at the parity bit location is transmitted as a 1. In the receiver, when STKYP and PEN are 1, then the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. When the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt when line status interrupts were enabled. For example, when EPS is 0, the receiver expects the bit received at the parity bit location to be 1. When 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 can identify some transmitted characters as receiver addresses and the rest of the characters as data. When PEN = 0, STKYP is ignored. 0 = no effect on parity bit. 1 = Forces parity bit to be opposite of EPS bit value.

Table 305. UART x Line Control Register - (UxLCR) (Sheet 2 of 2)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F70CH (DLAB=x)
1	FFFF F74CH (DLAB=x)

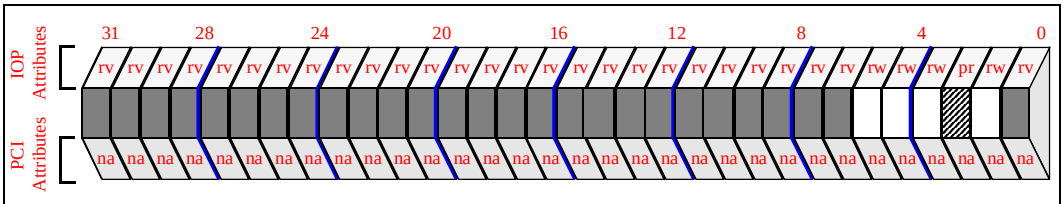
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
4	0_2	**Even Parity Select (EPS):** When PEN is set (1) and EPS is clear (0), an odd number of logic ones is transmitted or checked in the data word bits and the parity bit. When PEN is set (1) and EPS is also set (1), an even number of logic ones is transmitted or checked in the data word bits and parity bit. When PEN = 0, EPS is ignored. 0 = sends or checks for odd parity. 1 = sends or checks for even parity.
3	0_2	**Parity Enable (PEN):** When set (1), a parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The parity bit is used to produce an even or odd number of ones when the data word bits and the parity bit are summed.) 0 = no parity function. 1 = allows parity generation and checking.
2	0_2	**Stop bits (STB):** This bit specifies the number of stop bits transmitted and received in each serial character. When STB is clear (0), one stop bit is generated in the transmitted data. When STB is set (1) when a 5-bit word length is selected via WLS[1:0], then 1 and one half stop bits are generated. When STB is set (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 = 1 stop bit 1 = 2 stop bits, except for 5-bit character then 1-1/2 bits
1:0	00_2	**Word Length Select (WLS[1:0]):** Specifies 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

11.4.7 UART x Modem Control Register

This register controls the interface with the modem or data set (or a peripheral device emulating a modem). The contents of the Modem Control register are described below:

Table 306. UART x Modem Control Register - (UxMCR) (Sheet 1 of 2)

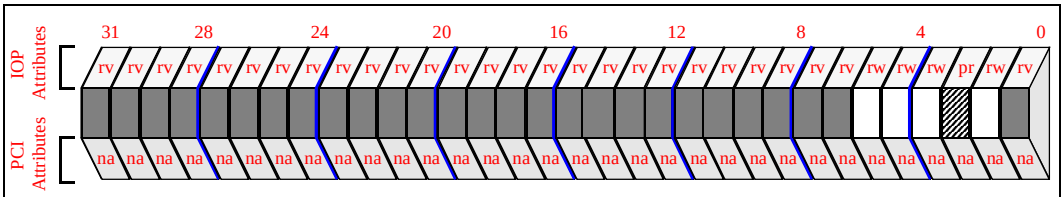


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F710H (DLAB=x)
1	FFFF F750H (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:6	000 0000h	Reserved
5	0_2	**Autoflow Control Enable (AFE):** When set, autoflow control is enabled. Only auto-CTS is enabled when *RTS* is cleared in MCR while AFE is set. Both auto-CTS and auto-RTS are enabled when *AFE* and *RTS in MCR* are set. Auto-RTS is not enabled when *AFE* is not set regardless of the state of *RTS*. Autoflow automates the flow of data between the UART and the remote device. See Section 11.3.4, "Autoflow Control" on page 589 for more details. 0 = Auto-RTS and Auto-CTS are disabled 1 = Auto-CTS is enabled. IF RTS in MCR is also set, both auto-CTS and auto-RTS is enabled
4	0_2	**Loop back test mode (LOOP):** This bit provides a local Loopback feature for diagnostic testing of the UART. In the Diagnostic mode, data that is transmitted is immediately received. This feature allows the processor to verify the UART Transmit and Receive data paths. The Transmit, Receive and Modem Control Interrupts are operational. The modem-control input **CTS#** is activated by MCR bit 1 instead of the modem-control input. A Break signal can also be transferred from the transmitter section to the receiver section in Loop-Back mode. When LOOP is set (1), the following occurs: • The TXD (transmitter output) pin is set to a logic-1 state. • The RXD (receiver input) pin is disconnected. • The output of the Transmitter Shift register is "looped back" into the Receiver-Shift register input. • The modem-control input **CTS#** is disconnected from the pins and the modem-control output pin **RTS#** is forced to the inactive state. The RTS bit of the Modem Control register is connected to bits of the Modem Status register bits. Flow control can be tested; when autoflow is enabled the RTS bit of the Modem Control Register has no effect on the **CTS#** input as **RTS#** is asserted by the autoflow logic.: • RTS = 1 forces CTS to a 1 ***Note:*** Note: Coming out of the Loop-Back Test mode may result in unpredictable activation of the delta bit (bit 0) in the Modem Status register (MSR). It is recommended that MSR is read once to clear the delta bits in the MSR. 0 = Normal UART operation 1 = Test mode UART operation

Table 306. UART x Modem Control Register - (UxMCR) (Sheet 2 of 2)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F710H (DLAB=x)
1	FFFF F750H (DLAB=x)

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
3	0_2	**Interrupt Enable (IE):** Global control all UART interrupts. 0 = interrupts disabled. 1 = interrupts enabled. **NOTE:** This bit is not valid when in Loopback mode.
2	0_2	Preserved.
1	0_2	**Request to Send (RTS):** **Non-Autoflow mode:** When not in Autoflow mode (AFE bit of MCR is clear), this bit controls the Request-to-Send (**RTS#**) output pin. 0 = RTS# pin is 1 1 = RTS# pin is 0 **Autoflow mode:** When in Autoflow mode (AFE bit of MCR is set), auto-RTS is enabled. **RTS#** behaves as follows: • Auto-RTS disabled. Autoflow works only with auto-CTS. • Auto-RTS enabled. Autoflow works with both auto-CTS and auto-RTS.
0	0_2	Reserved

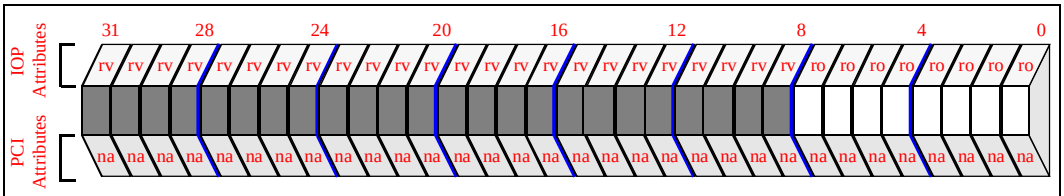
11.4.8 UART x Line Status Register

This register provides status information to the processor concerning the data transfers. Bits 5 and 6 show information about the transmitter section. The remainder of the bits contain information about the receiver.

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. In FIFO mode, these three status bits are stored with each received character in the FIFO. LSR shows the status bits of the character at the bottom of the FIFO. When the character at the bottom of the FIFO has errors, the LSR error bits are set and are not cleared until software reads LSR, even when the character in the FIFO is read and a new character is now at the bottom of the FIFO.

Bits 1 through 4 are the error conditions that produce a Receiver Line Status Interrupt when any of the corresponding conditions are detected and the interrupt is enabled. These bits are not cleared by reading the erroneous byte from the FIFO or receive buffer. They are cleared only by reading LSR. In FIFO mode, the Line Status Interrupt occurs only when the erroneous byte is at the bottom of the FIFO. When the erroneous byte being received is not at the bottom of the FIFO, an interrupt is generated only after the previous bytes are read and the erroneous byte is moved to the bottom of the FIFO.

Table 307. UART x Line Status Register - (UxLSR) (Sheet 1 of 3)

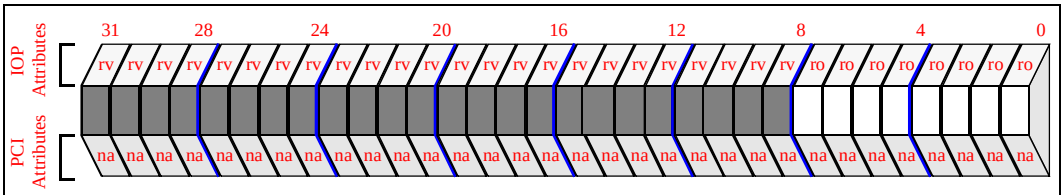


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F714H (DLAB=x)
1	FFFF F754H (DLAB=x)

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7	0_2	**FIFO Error Status (FIFOE):** In non-FIFO mode, this bit is clear (0). In FIFO Mode, this bit is set (1) when there is at least one parity error, framing error, or break indication for any of the characters in the FIFO. 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. FIFOE being set to 1 does NOT generate an interrupt. 0 = No errors exit in the receive FIFO 1 = At least one character in receiver FIFO has errors
6	1_2	**Transmitter Empty (TEMT):** Set (1) when the Transmit Holding register and the Transmitter Shift register are both empty. It is reset to zero (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. 0 = There is data in the Transmit Shift register, the Transmit Holding register, or the FIFO 1 = All the data in the transmitter has been shifted out

Table 307. UART x Line Status Register - (UxLSR) (Sheet 2 of 3)

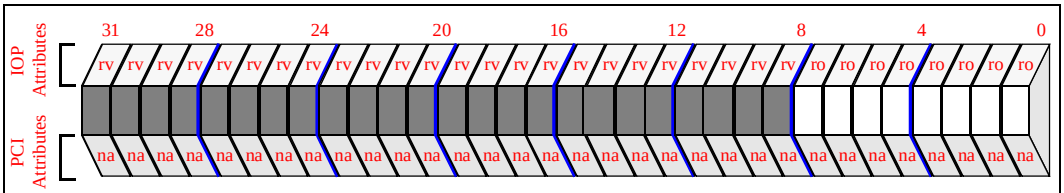


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F714H (DLAB=x)
1	FFFF F754H (DLAB=x)

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
5	1_2	**Transmit Data Request (TDRQ):** Indicates that the UART is ready to accept data for transmission. The assertion of this bit causes the UART to issue an interrupt when the Transmit Data Request Interrupt Enable is set. In non-FIFO mode, the TDRQ bit is set (1) when a character is transferred from the Transmit-Holding register. The bit is cleared (0) concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set (1) when the FIFO is less than half full. It is cleared when the FIFO is more than half full. When more than 64 characters are loaded into the FIFO, the excess characters are lost. 0 = The UART is NOT ready to receive data for transmission. 1 = The UART is ready to receive data for transmission.
4	0_2	**Break Indicator (BI):** Set (1) when the received data input is held in the Spacing (logic 0) state for longer than a full character transmission time (that is, the total time of *start* bit + *data* bits + *parity* bit + *stop* bits). The Break Indicator is reset (cleared to 0) when the processor reads the Line-Status register. In FIFO mode, only one Break character (equal to 0x00), 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 recent character received. 0 = No break signal has been received. 1 = Break signal has been received.
3	0_2	**Framing Error (FE):** Indicates that the received character did not have a valid stop bit. FE is set (1) when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). When the Line Control register had been set for two stop bits, the receiver does not check for a valid second stop bit. The FE indicator is reset when the processor reads the Line Status Register. The UART resynchronizes after a framing error by assuming that the framing error was due to the next start bit. Therefore 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. 0 = No Framing error. 1 = Invalid stop bit has been detected.

Table 307. UART x Line Status Register - (UxLSR) (Sheet 3 of 3)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F714H (DLAB=x)
1	FFFF F754H (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

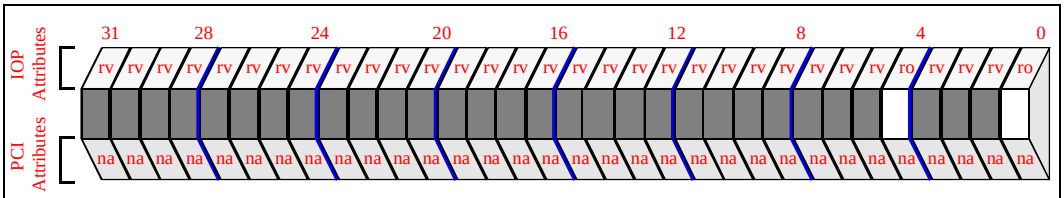
Bit	Default	Description
2	0_2	**Parity Error (PE):** Indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set (1) upon detection of a parity error and is cleared (0) when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the bottom of the FIFO, not the most recently received character. 0 = No Parity error. 1 = Parity error has occurred.
1	0_2	**Overflow Error (OE):** In non-FIFO mode, OE indicates that data in the Receiver Buffer register was not read 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 (1) upon detection of an overflow condition and reset when the processor reads the Line Status register. 0 = No overflow error. Data has not been lost. 1 = Overflow error. Receive data has been lost.
0	0_2	**Data Ready (DR):** Set to a 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 when the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in FCR. 0 = No data has been received. 1 = Data is available in RBR or the FIFO.

11.4.9 UART x 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). In addition to this current state information, the Modem Status register also provides change information. The change bit is set to a logic 1 when the control input from the Modem changes state. The change bit is reset to a logic 0 when the processor reads the Modem Status register.

Note: When the change bit (bit 0) is set to logic 1, a Modem Status interrupt is generated when bit 3 of the Interrupt Enable Register is set.

Table 308. UART x Modem Status Register - (UxMSR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F718H (DLAB=x)
1	FFFF F758H (DLAB=x)

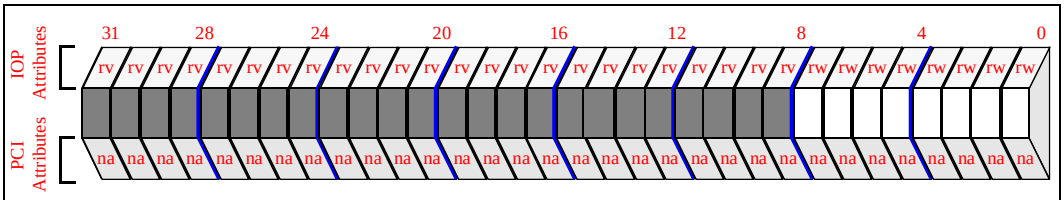
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:5	000 0000h	Reserved
4	0_2	**Clear to Send (CTS):** This bit is the complement of the Clear to Send (**CTS#**) input. This bit is equivalent to bit RTS of the Modem Control register when LOOP in the MCR is set to 1. 0 = **CTS#** pin is 1 1 = **CTS#** pin is 0
3:1	000_2	Reserved
0	0_2	**Delta Clear To Send (DCTS):** 0 = No change in **CTS#** pin since last read of MSR 1 = **CTS#** pin has changed state

11.4.10 UART x Scratchpad Register

This read/write register has no effect on the UART. It is intended as a scratchpad register for use by programmers.

Table 309. UART x Scratchpad Register - (UxSCR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F71CH (DLAB=x)
1	FFFF F75CH (DLAB=x)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7:0	00h	No effect on UART functionality

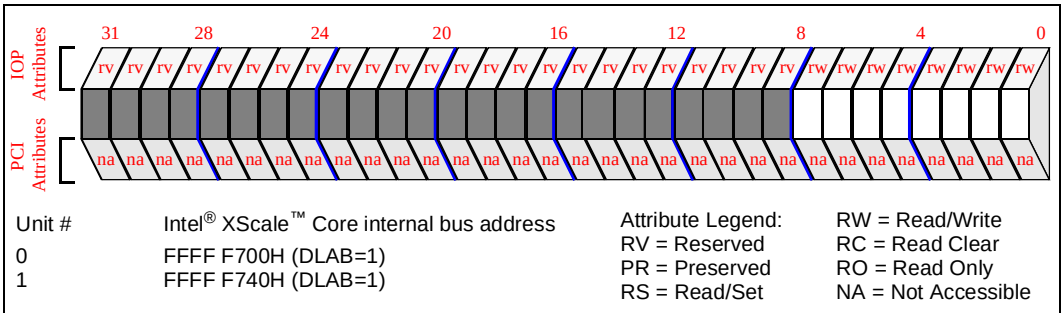
11.4.11 Divisor Latch Registers

The description of use for the Divisor Latch Registers are provided in Section 11.3.5, Auto-Baud-Rate Detection and Section 11.3.6, Manual Baud Rate Selection. Refer to those sections for details on how to program these registers.

Bit DLAB in the LCR register must be set high before the Divisor Latch registers can be accessed.

A Divisor value of 0 in the Divisor Latch Register is not allowed. A value of 0 has the affect of disabling the UART. The reset value of the divisor is 02.

Table 310. UART x Divisor Latch Low Register - (UxDLL)

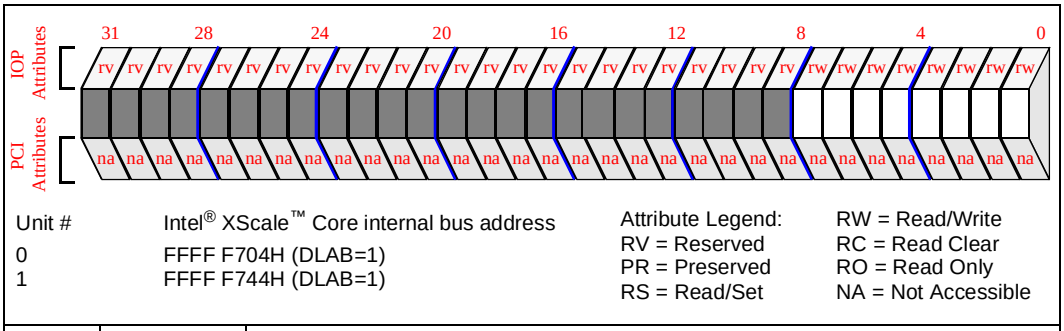


Unit #	Intel® XScale™ Core internal bus address
0	FFFF F700H (DLAB=1)
1	FFFF F740H (DLAB=1)

Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7:0	02h	Low byte compare value to generate baud rate

Table 311. UART x Divisor Latch High Register - (UxDLH)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F704H (DLAB=1)
1	FFFF F744H (DLAB=1)

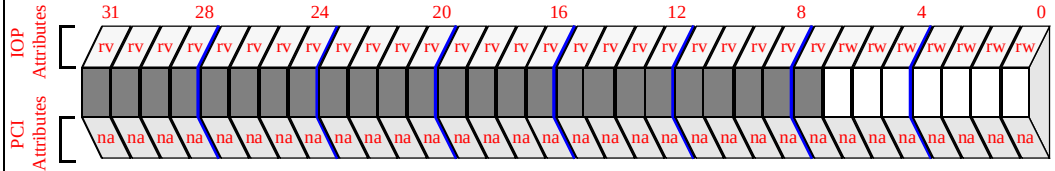
Attribute Legend:	
RV = Reserved	RW = Read/Write
PR = Preserved	RC = Read Clear
RS = Read/Set	RO = Read Only
	NA = Not Accessible

Bit	Default	Description
31:8	00 0000h	Reserved
7:0	00h	High byte compare value to generate baud rate

11.4.12 UART x FIFO Occupancy Register

This register shows the number of bytes currently remaining the Receive FIFO. It can be used by the processor to determine the number of trailing bytes to remove from the receive FIFO when the Character Time-out Interrupt is detected. Refer to Section 11.3.2, "Removing Trailing Bytes In Interrupt Mode" on page 588. The FOR register is incremented once for each byte of data written to the Receive FIFO and decremented once for each byte read.

Table 312. UART x FIFO Occupancy Register - (UxFOR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F724H (DLAB=x)
1	FFFF F764H (DLAB=x)

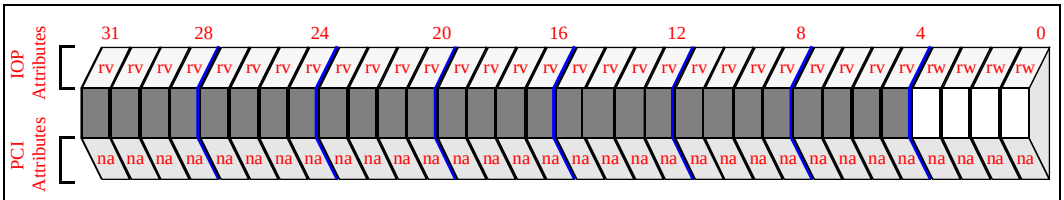
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:7	000 0000h	Reserved
6:0	000 0000_2	FOR[6:0]: Number of bytes (0-63) remaining in the Receiver FIFO

11.4.13 UART x Auto-Baud Control Register

This read/write register has no effect on the UART. It is intended as a scratchpad register for use by programmers.

Table 313. UART x Auto-Baud Control Register - (UxABR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F728H (DLAB=x)
1	FFFF F768H (DLAB=x)

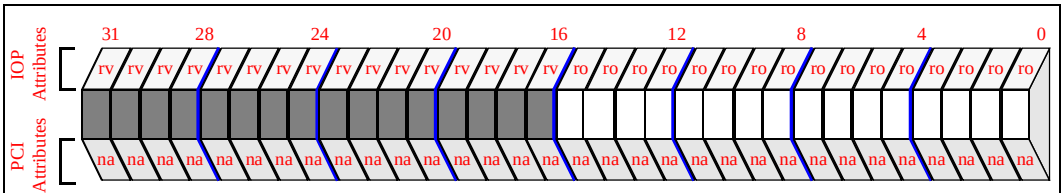
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:4	000 0000h	Reserved
3	0_2	**Auto-Baud Table (ABT):** Directs the auto-baud circuitry within the UART to use a table when selecting the final baud rate programmed into the Divisor Latch registers and written into the Auto-Baud Count register (ACR). When a table is not used, any value allowed in Section 13, "Baud-Rate Equation" on page 590 can be selected by the UART. See Section 11.3.5, "Auto-Baud-Rate Detection" on page 590 for more information on auto-baud. 0 = Formula used to calculate baud rates allowing all possible baud rates to be chosen by UART Equation 13. 1 = Table used to calculate baud rates which limits UART to choosing common baud rates.
2	0_2	**Auto-Baud UART Program (ABUP):** Allows the UART to automatically program the Divisor Latch registers (DLL and DLH) when the auto-baud circuit has detected the baud rate, regardless of the state of the *divisor-latch access* bit (DLAB). For this to occur, the *auto-baud enable (ABE)* bit must be set. Clearing this bit allows the processor to read the Auto-Baud Count register (ACR) and determine the baud rate using its own algorithm rather than using the UARTs. 0 = Software programs Divisor Latch Registers. 1 = UART Programs Divisor Latch Registers
1	0_2	**Auto-Baud Lock Interrupt Enable (ABLIE):** Enables the ABL Interrupt which occurs when the auto-baud circuit has detected the baud rate and written the value into the Auto-Baud Count register (ACR). The Divisor Latch registers can then be programmed by the processor or auto-baud circuitry as dictated by the state of the ABPP. 0 = Autobaud Lock Interrupt Disabled 1 = Autobaud Lock Interrupt Enabled
0	0_2	**Auto-Baud Enable (ABE):** Enables the auto-baud circuitry within the UART. The circuitry counts the number of clocks in the *start* bit and writes this count into the Autobaud Count register (ACR). It then interrupts the processor when the *auto-baud lock interrupt-enable (ABLIE)* bit is set. It also automatically programs the Divisor Latch registers (DLL and DLH) when the *auto-baud UART program (ABUP)* bit is set. 0 = Autobaud Disabled 1 = Autobaud Enabled

11.4.14 UART x Auto-Baud Count Register

The Auto-Baud Count register stores the number of 33.334 MHZ clock cycles within a *start* bit pulse. This value is then used by the processor or auto-baud circuitry within the UART to calculate the baud rate. When Auto-Baud mode and Auto-Baud Interrupts are enabled, the UART interrupt the processor with the Auto-Baud Lock Interrupt after it has written the count value into the ACR. The value is written regardless of the state of the *auto-baud UART program* bit.

Table 314. UART x Auto-Baud Count Register - (UxACR)



Unit #	Intel® XScale™ Core internal bus address
0	FFFF F72CH (DLAB=x)
1	FFFF F76CH (DLAB=x)

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	0000h	Reserved
15:0	0000h	ACR[15:0]: Number of 33.334 MHz clock cycles within a start bit pulse.

This Page Intentionally Left Blank

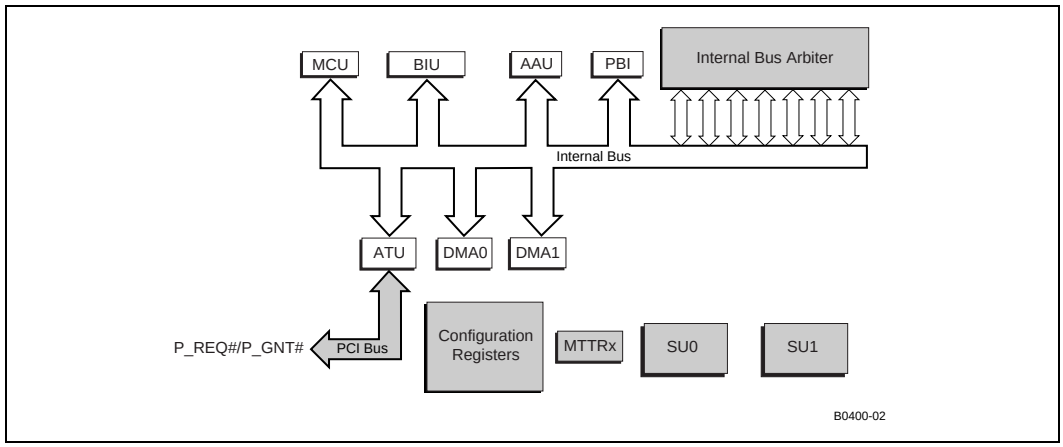
Intel® 80331 I/O Processor Arbitration Unit 12

This chapter describes the components comprising the Intel® 80331 I/O processor (80331) arbitration, including one Internal Bus Arbiter, one PCI Selector, and two Multi-Transaction Timers. The operation modes, setup, and implementation of these components are described in this chapter.

12.1 Arbitration Overview

The 80331 interfaces to one PCI bus and contains an internal PCI-like bus. Therefore, there are two buses which need an arbitration mechanism. Figure 107 illustrates all the potential bus initiators and which arbitration components are responsible for them.

Figure 107. Intel® 80331 I/O Processor Arbitration Block Diagram



The four components which comprise 80331 arbitration are:

- **Internal Bus Arbitrator (**Section 12.2, "Internal Bus Arbiter Overview" on page 616**)** - Arbitrates between multiple initiators. The arbitration scheme is a round-robin with priority/promotion capabilities. The 80331 requires one PCI arbiter: the Internal Bus Arbiter.
  - — Internal Bus Arbiter (IARB) arbitrates between six potential internal bus initiators (ATU, two DMA Channels, Application Accelerator, the PBI, and the Bus Interface Unit for the core).
- **Multi-Transaction Timer (** Section 12.4.2, "Multi-Transaction Timer Register 1 - MTTR1" on page 625Section 12.4.3, "Multi-Transaction Timer Register 2 - MTTR2" on page 626**)** - Two muti-transaction timers (MTTR[2:1]) control the arbitration control for the internal bus initiators. The MTTRs counts internal bus cycles, an initiator uses across back-to-back transactions. The arbiter continues to assert GNT# to current initiator until the respective MTTR expires, or the initiator deasserts its **REQ#**. The 80331 implements one MTTR for the internal Intel® XScale™ core Bus Interface Unit and one for all peripheral initiators on the internal bus.
- **Arbitration Configuration Registers (**Section 12.4, "Register Definitions" on page 623**)** - Priority and latency timer values for arbitration mechanism are programmable, as defined in Arbitration Configuration Registers.

12.2 Internal Bus Arbiter Overview

The *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0 requires a central arbitration resource for each PCI bus within a system environment. This section details the operation of the Internal Bus Arbiter block.

The Internal Bus Arbiter supports:

- Three priority levels for each bus initiator
- A "fairness" algorithm which ensures that each potential bus initiator is granted access to the PCI bus independent of other requests
- Hidden, access-based arbitration

PCI uses the concept of access-based arbitration rather than the traditional time slot approach. When a bus initiator requires the PCI bus for a transaction, the device requests the arbitration logic for the PCI bus. PCI arbitration consists of a simple **REQ#** and GNT# handshake protocol. When a device requires the Internal Bus, it asserts its **REQ#** output. The arbitration unit allows the requesting agent access to the bus by asserting that agent's GNT# input.

PCI arbitration is a hidden arbitration scheme where the arbitration sequence occurs in the background while another bus initiator may currently control the bus. Hidden arbitration has the advantage of not consuming any bus bandwidth for arbitration overhead.

The arbiter is required by the *PCI-X Addendum to the PCI Local Bus Specification,* Revision 1.0 to implement a "fair" arbitration algorithm. The Internal Bus Arbiter's algorithm guarantees there is only one GNT# active on the Internal bus at any one time.

12.2.1 Theory of Operation

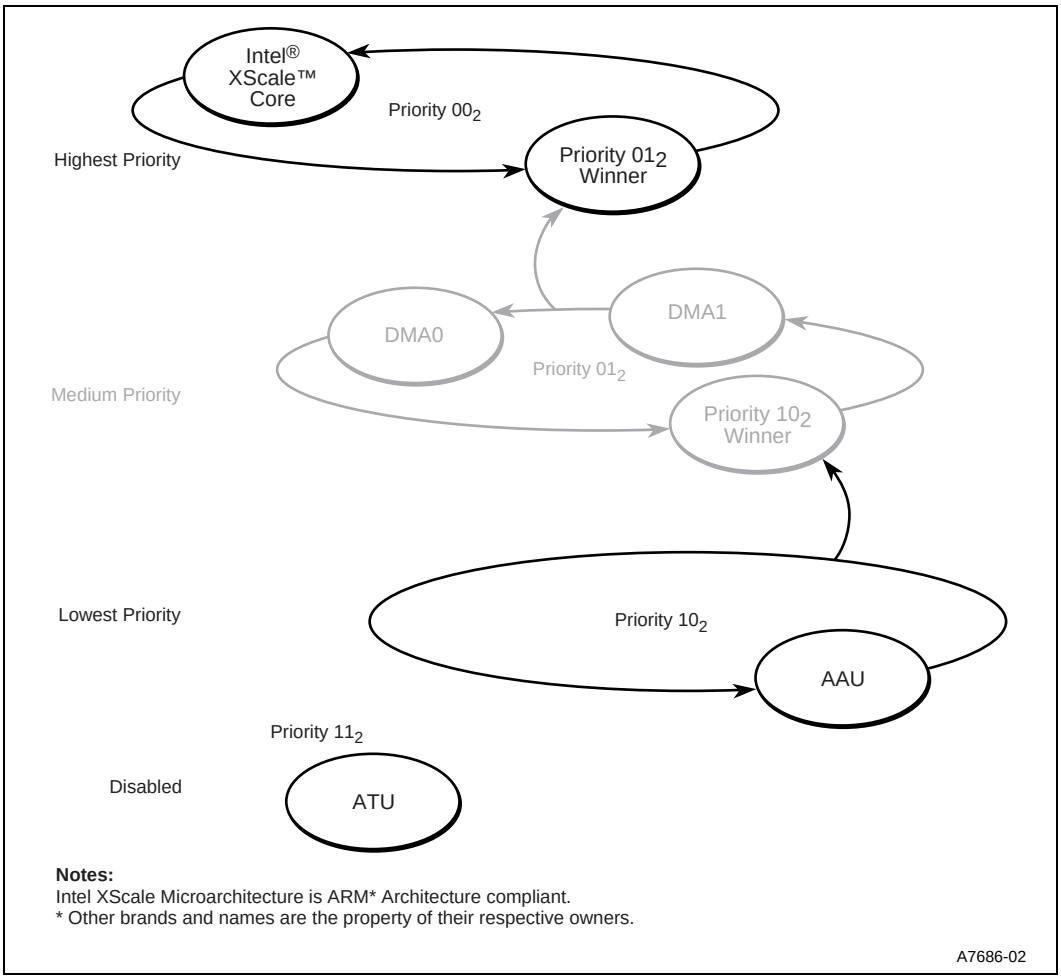
The arbiter's behavior is fully compliant with the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0.

12.2.1.1 Priority Mechanism

The Internal Bus Arbiter supports six bus initiators on the Internal Bus. Each request can be programmed to one of three priority levels or be disabled. Application software programs the Internal Arbiter Control Register (IACR) to set the initial priority for each bus initiator. The arbiter promotes the bus initiator priority levels using a round-robin scheme.

Figure 108 is an example showing the three priority levels and the reserved slots for the promoted requester.

Figure 108. Arbitration Example



In Figure 108, the bus initiators are initially programmed to the priorities shown in Table 315. The IACR defines the initial priority levels for the IARB.

Table 322 shows the 2-bit values that correspond to each priority level. A priority level of 11_2 effectively disables the associated device by removing it from the arbitration sequence. A device programmed with a 11_2 priority does not receive a grant to gain access to the bus.

Table 315. Priority Programming Example

Bus Initiator	Programmed Priority
Intel® XScale™ core (BIU)	High - 00_2
Memory Controller Unit	High - 00_2
DMA Channel 0	Medium - 01_2
ATU	Medium - 01_2
Application Accelerator	Low - 10_2
DMA Channel 1	Disabled - 11_2

The priority of the individual bus initiator determines the level to which the device is placed in the round-robin scheme. The programmed priority determines the starting priority or the lowest priority the device is. When the application programs the device for low priority, the device may be promoted up to medium and then high priority until it is granted the Internal Bus. Once the IARB grants the bus and the device asserts FRAME#, the device is reset to its initially programmed priority.

Note: When a low priority initiator requests the bus, with no other higher priority agent requesting the bus, that initiator is granted the bus the following clock. The promotion mechanism does not consume bus cycles.

The round-robin arbitration scheme supports three levels of round-robin arbitration: low, medium, and high priority. Using a round-robin mechanism ensures there is a winner for each priority level. To enforce the concept of fairness, a slot is reserved for the winner of each priority level (except the highest) in the next higher priority level. When the winner of a priority level is not granted the bus during that particular arbitration sequence, it is promoted to the next higher level of priority.

12.2.1.2 Priority Example with Three Bus Initiators

Table 316 illustrates an example of bus arbitration with three bus initiators:

Table 316. Bus Arbitration Example – Three Bus Initiators

Priority Level	Initial State	Winning Bus Initiator							
		A	**B**	**A**	**C**	**A**	**B**	**A**	**C**
High	A	B	A	C	A	B	A	C	A
Medium	B	C	C	B	B	–	C	B	B
Low	C	–	–	–	–	C	–	–	–

NOTE: In this example, all bus initiators are continually requesting the bus.

Each of the bus initiators (A, B, and C) are constantly requesting the bus and each is at a different priority level. The top row of Table 316 lists the current bus initiator/winner of the highest priority group. The three rows labelled as high, medium and low represent the actual priority levels that devices are currently at based on either their initial programmed priority or promotion through the levels. For example, device C starts out at low priority. Because it is the only device at this priority, it is the winner at low priority and is promoted to medium priority. Later, it wins at the medium priority level (against device B) and is promoted to high priority where it wins the level (against device A) and the bus. Device C is then put back at its programmed priority of low and starts the cycle over.

Continuing with Table 316, the winning bus initiator pattern would follow as:

ABACABACABACABAC

12.2.1.3 Priority Example with Six Bus Initiators

Table 317 illustrates an example of bus arbitration with six bus initiators:

Table 317. Bus Arbitration Example – Six Bus Initiators

Priority Level	Initial State	Winning Bus Initiator								
		A	B	C	A	B	D	A	B	E
High	AB	BC	AC	AB	BD	AD	AB	BE	AE	AB
Medium	CD	DE	DE	DE	CE	CE	CE	CDF	CDF	CDF
Low	EF	F	F	F	F	F	F	–	–	–

NOTE: In this example, all bus initiators are continually requesting the bus.

Each of the six bus initiators (A through F) are constantly requesting the bus. There are two initiators programmed at each priority level. The top row of Table 317 lists the current bus initiator/winner of the highest priority group. The three rows labelled as high, medium and low represent the actual priority levels that devices are currently at based on either their initial programmed priority or promotion through the levels.

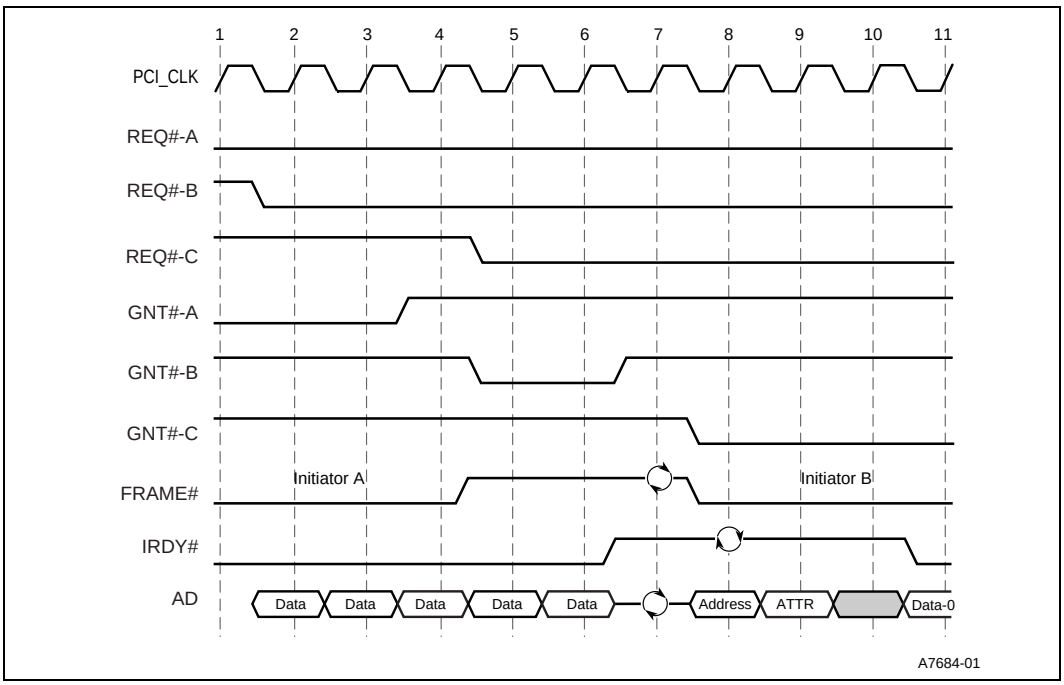
Continuing with Table 317, the winning bus initiator pattern would follow as:

ABCABDABFABCABDABEABCABDABF

12.2.1.4 Arbitration Signalling Protocol

The Internal Bus Arbiter interfaces to all requesting agents on the bus through the **REQ#**/GNT# handshaking protocol. A bus initiator asserts its **REQ#** to request ownership of the Internal Bus. When the arbiter determines an agent may use the bus, it asserts the agent GNT# input. Agents must only assert **REQ#** to signal a true need for the bus and not to *reserve* the bus. Figure 109 illustrates arbitration between initiators of equal priority. This Internal Bus Arbiter only operates in the PCI-X mode as defined in *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0.

Figure 109. Arbitration Between Three Initiators



An agent can be granted the bus while a previous bus owner still has control of the Internal Bus (hidden arbitration). The arbiter is responsible for deciding which device is granted the bus next while each initiator is responsible for determining when the bus actually becomes free and is allowed to initiate its transaction by asserting FRAME#.

PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0 states an initiator may deassert its **REQ#** pin before the arbiter grants the Internal bus to that initiator. When an initiator deasserts its **REQ#** pin, the Internal Bus Arbiter re-arbitrates, giving bus ownership to the next initiator based on priority algorithm defined in Section 12.2.1.1, "Priority Mechanism" on page 617.

Note: **The Internal Bus Arbiter arbitrates the Internal Bus by checking REQ[8:0]# on every cycle independent of any transactions on the bus.**

The Internal Bus Arbiter may deassert an agent's GNT# on any clock. An agent must ensure its GNT# is asserted two clocks prior to the clock edge where it initiates a transaction by asserting FRAME#. When GNT# is deasserted, the transaction may not proceed.

By monitoring **REQ[8:0]#**, the arbiter can control the arbitration algorithm described in Section 12.2.1.1, "Priority Mechanism" on page 617. The arbiter asserts GNT# two clocks after **REQ#** is asserted when the agent has won the bus. An example arbitration flow is:

Table 318. Arbitration Flow

Cycle	Event
0	The arbiter is currently driving Initiator_As GNT#. The arbitration flow is independent of whether or not Initiator_A is involved with a transaction. For example, the Internal Bus could be parked with Initiator_A.
1	Initiator_B asserts its **REQ#** for PCI bus ownership. The arbitration logic calculates that Initiator_B has a higher priority than Initiator_A.
3	The arbiter deasserts GNT# for Initiator_A since Initiator_B is higher priority.
4	The arbiter asserts GNT# for Initiator_B.
7	When Initiator_B drives FRAME#, any of the priority winners that were not granted the bus are promoted to a higher priority level when the reserved promotion slot is unoccupied (see Section 12.2.1.1, "Priority Mechanism" on page 617).

12.2.1.5 Internal Bus Arbitration Parking

Arbitration parking occurs when the arbiter asserts GNT# to a selected Internal Bus agent and no agent is currently using or requesting the bus.

Upon reset, the IARB parks the internal bus with the Intel® XScale™ core Bus Interface Unit (BIU). After an initiator requests, and is granted the bus, the arbiter parks the bus with that initiator. In other words, the last initiator that was granted the bus is responsible for parking.

When the Internal Bus is parked during an idle state, the parked agent loses the bus when the arbiter asserts another agent's GNT#. The parked agent relinquishes the bus and stops driving the address and command signals in one clock. When the arbiter deasserts an agent's GNT# on clock N-2, the agent can still initiate a normal bus transaction by driving FRAME# during clock N.

12.2.2 Intel® XScale™ Core Arbitration

The Intel® XScale™ core has an inherently small burst size. For this reason, a busy internal bus could inhibit data traffic for the core. To address this issue, the IARB implements a Multi-Transaction Timer (MTT1) which allocates a minimum timeslice where the Intel® XScale™ core BIU can request the IARB to keep the core processor's GNT# asserted potentially across multiple Internal Bus transactions. Refer to Section 12.2.2.1, "Multi-Transaction Timers" on page 622 for details.

12.2.2.1 Multi-Transaction Timers

The Internal Arbiter incorporates two Multi-Transaction Timers (MTT1 and MTT2) allowing the BIU or the other internal bus agents a guaranteed time-slice before losing arbitration to another internal bus agent. PCI is a transaction based protocol. In a system with long bursting agents, an agent such as the BIU with a small burst size could get starved.

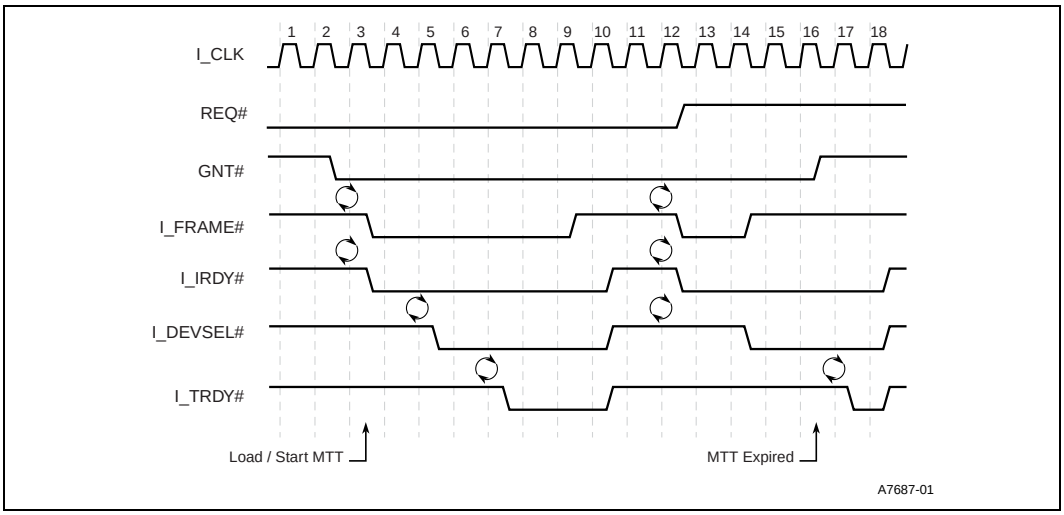
The MTT overcomes these potential bottlenecks by guaranteeing a programmed timeslice with which the BIU (MTTR1) or other internal bus agents (MTTR2) are granted the internal bus. For example, once the IARB grants the internal bus to the BIU and the BIU initially asserts **I_FRAME#**, MTT1 is loaded with the value programmed in the Multi-Transaction Timer Register 1 (MTTR1) and begins to decrement. The arbiter does not remove the BIU grant unless:

- BIU no longer requests bus by deasserting core processor **REQ#**.
- BIU continues to drive core processor **REQ#** and MTT expires.

Note: Even when a higher-priority initiator requests the internal bus, the arbiter does not deassert the BIU grant unless either of the above conditions occur.

Figure 110 illustrates an example of how the BIU uses MTT1 for efficient back-to-back transactions. For this example, the MTTR1 is programmed for 13 cycles.

Figure 110. Intel® XScale™ Core Back-to-Back Transactions with MTT1 Enabled



Note: Multi-Transaction Timers keep track of the maximum time that an internal bus agent may continue to initiate new transactions on the internal bus per arbitration cycle. The MTT governs the internal arbiter.

Warning: When the MTTRx are programmed with zero, the MTTx are effectively disabled. However, this is not recommended to avoid potential resource lock out conditions on the internal bus; this is due to the one-deep transaction queue of the MCU. The minimum (default) value for MTT1 is recommended to be 98H (152 clocks) and the minimum (default) value for MTT2 is recommended to be 38H (56 clocks).

12.3 Reset Conditions

Table 319 shows all the arbitration blocks and the signal responsible for resetting its logic:

Table 319. Arbitration Reset

Arbitration Block	Reset With
Internal Arbiter (IARB)	I_RST#
Multi-Transaction Timer 1	I_RST#
Multi-Transaction Timer 2	I_RST#

I_RST# moves all the internal agents to their programmed priority levels and starts the round robin arbitration sequence on the lowest number device at each priority level.

12.4 Register Definitions

Table 320 lists the Arbitration configuration registers which are detailed further in proceeding sections.

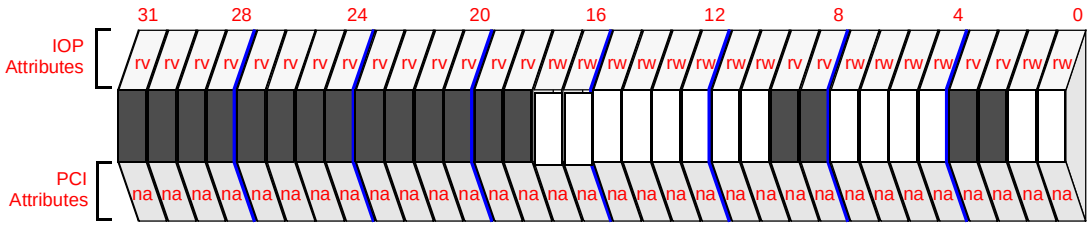
Table 320. Arbiter Register

Section, Register Name - Acronym (Page)
Section 12.4.1, "Internal Arbitration Control Register - IACR" on page 624
Section 12.4.2, "Multi-Transaction Timer Register 1 - MTTR1" on page 625
Section 12.4.3, "Multi-Transaction Timer Register 2 - MTTR2" on page 626

12.4.1 Internal Arbitration Control Register - IACR

The Internal Arbitration Control Register (IACR) sets the arbitration priority of each device that uses the internal bus. This register is part of the local arbitration configuration register space and is accessible from the 80331 core.

Table 321. Internal Arbitration Control Register - IACR



Intel® XScale™ Core Local Bus Address
FFFF E7F0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:18	0	Reserved
17:16	00_2	Memory Controller Priority
15:14	00_2	Peripheral Bus Interface Priority
13:12	00_2	Application Accelerator Priority
11:10	00_2	Intel® XScale™ core Bus Interface Unit Priority
9:8	00_2	Reserved
7:6	00_2	DMA Channel 1 Priority
5:4	00_2	DMA Channel 0 Priority
3:2	00_2	Reserved
1:0	00_2	ATU and Messaging Unit Priority

Each device is given a 2-bit priority shown in Table 322. The default values for the IACR give all the internal bus initiators the highest priority.

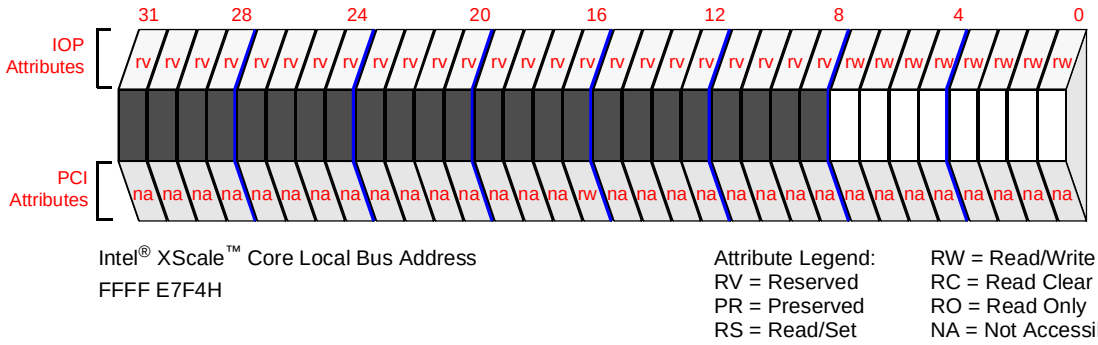
Table 322. Programmed Priority Control

2-Bit Programmed Value	Priority Level
00_2	High Priority
01_2	Medium Priority
10_2	Low Priority
11_2	Disabled

12.4.2 Multi-Transaction Timer Register 1 - MTTR1

The Multi-Transaction Timer Register 1 defines the duration with which the Intel® XScale™ core Bus Interface Unit retains the core processor GNT# across back-to-back transactions. This is an 8-bit value allowing up to 255 dedicated internal bus cycles for as long as the core processor GNT# is asserted. A value of zero effectively disables the MTTR1. This register is part of the local arbitration configuration register space and is accessible from the 80331 core.

Table 323. Multi-Transaction Timer Register - MTTR1

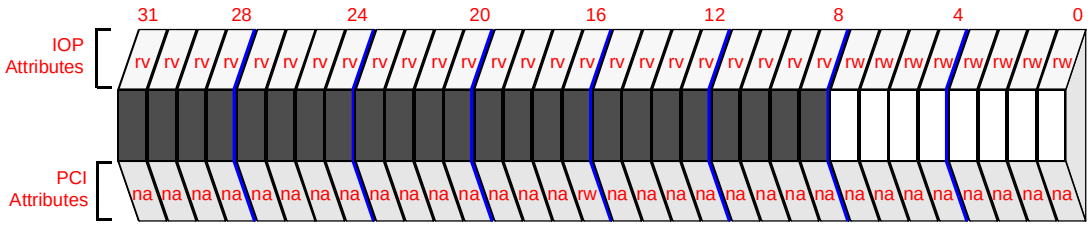


Bit	Default	Description
31:8	000000H	Reserved
7:0	98H	Multi-Transaction Timer 1 Preload Value - Indicates the minimum number of clocks the Intel® XScale™ core is allowed to retain ownership of the Internal Bus across back-to-back transactions. **NOTE:**

12.4.3 Multi-Transaction Timer Register 2 - MTTR2

The Multi-Transaction Timer Register 2 defines the duration with which agents other than the Intel® XScale™ core retains the core processor GNT# across back-to-back transactions. This is an 8-bit value allowing up to 255 dedicated internal bus cycles for as long as the core processor GNT# is asserted. A value of zero effectively disables the MTTR2. This register is part of the local arbitration configuration register space and is accessible from the 80331 core.

Table 324. Multi-Transaction Timer Register - MTTR2



Intel® XScale™ Core Local Bus Address
FFFF E7F8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:8	000000H	Reserved
7:0	38H	Multi-Transaction Timer 2 Preload Value - Indicates the minimum number of clocks internal bus agents other than the Intel® XScale™ core are allowed to retain ownership of the Internal Bus across back-to-back transactions. **NOTE:**

Intel® XScale™ Core and Core Performance Monitoring 13

This chapter describes the Intel® XScale™ core (ARM* architecture compliant) and its associated Performance Monitoring facilities within the Intel® 80331 I/O processor (80331). The events that are monitored can provide performance information for compiler writers, system application developers and software programmers.

13.1 High-Level Overview of Intel® XScale™ Core

The second generation Intel® XScale™ core is designed for high performance and low-power; leading the industry in mW/MIPs. Many of the architectural features added to the Intel® XScale™ core help hide memory latency which often is a serious impediment to high performance processors. This includes:

- the ability to continue instruction execution even while the data cache is retrieving data from external memory.
- a write buffer.
- write-back caching.
- various data cache allocation policies which can be configured different for each application.
- and cache locking.

All these features improve the efficiency of the external bus.

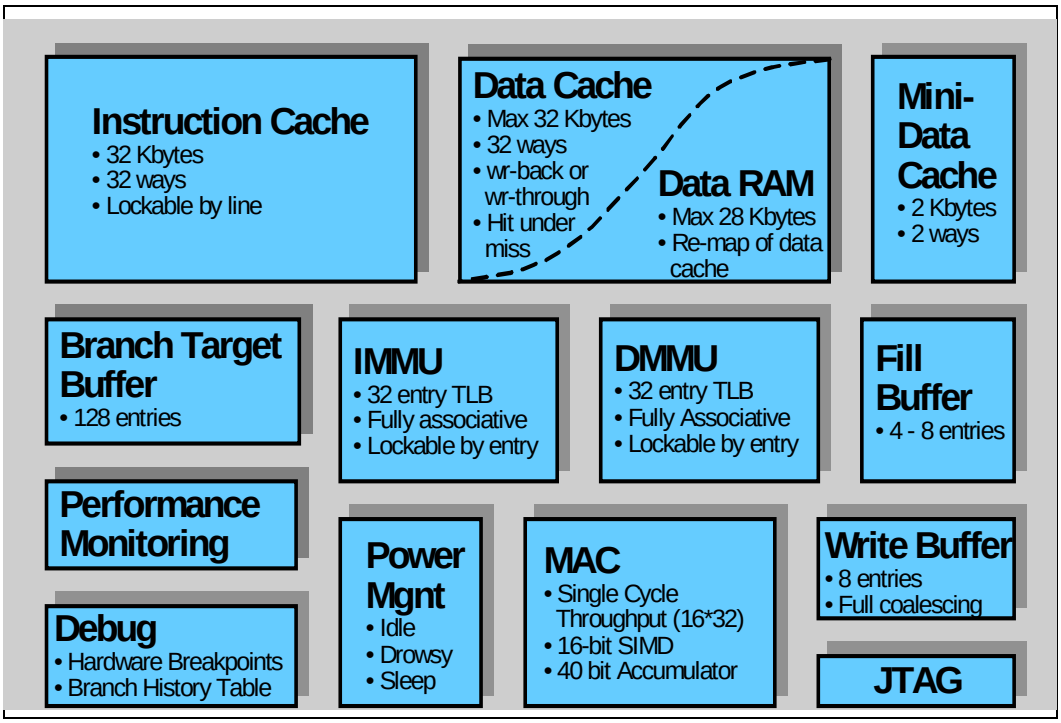
13.1.1 ARM Compatibility

ARM Version 5 (V5) Architecture added floating point instructions to ARM Version 4. The Intel® XScale™ core implements the integer instruction set architecture of ARM V5, but does not provide hardware support of the floating point instructions. The Intel® XScale™ core provides the Thumb instruction set (ARM V5T) and the ARM V5E DSP extensions. Backward compatibility with the first generation of Intel® StrongARM* products is maintained for user-mode applications. Operating systems may require modifications to match the specific hardware features of the and to take advantage of the performance enhancements added to the second generation Intel® XScale™ core.

13.1.2 Features

Figure 111 shows the major functional blocks of the Intel® XScale™ core. The following sections give a brief, high-level overview of these blocks.

Figure 111. The Intel® XScale™ Core Architecture Features



13.1.2.1 Multiply/ACcumulate (MAC)

The MAC unit supports early termination of multiplies/accumulates in two cycles and can sustain a throughput of a MAC operation every cycle. Several architectural enhancements were made to the MAC to support audio coding algorithms, which include a 40-bit accumulator and support for 16-bit packed data.

13.1.2.2 Memory Management

The Intel® XScale™ core implements the Memory Management Unit (MMU) Architecture specified in the *ARM Architecture Reference Manual*. The MMU provides access protection and virtual to physical address translation.

The MMU Architecture also specifies the caching policies for the instruction cache and data memory. These policies are specified as page attributes and include:

- identifying code as cacheable or non-cacheable
- selecting between the mini-data cache or data cache
- write-back or write-through data caching
- enabling data write allocation policy
- and enabling the write buffer to coalesce stores to external memory

13.1.2.3 Instruction Cache

The Intel® XScale™ core implements a 32-Kbyte, 32-way set associative instruction cache with a line size of 32 bytes. All requests that "miss" the instruction cache generate a 32-byte read request to external memory. A mechanism to lock critical code within the cache is also provided.

13.1.2.4 Branch Target Buffer

The Intel® XScale™ core provides a Branch Target Buffer (BTB) to predict the outcome of branch type instructions. It provides storage for the target address of branch type instructions and predicts the next address to present to the instruction cache when the current instruction address is that of a branch.

The BTB holds 128 entries.

13.1.2.5 Data Cache

The Intel® XScale™ core implements a 32-Kbyte, a 32-way set associative data cache and a 2-Kbyte, 2-way set associative mini-data cache. Each cache has a line size of 32 bytes, supports write-through or write-back caching.

The data/mini-data cache is controlled by page attributes defined in the MMU Architecture and by coprocessor 15.

The Intel® XScale™ core allows applications to re-configure a portion of the data cache as data RAM. Software may place special tables or frequently used variables in this RAM.

13.2 CP14 Registers

Table 325 lists the CP14 registers implemented in the I/O processor.

Table 325. CP14 Registers

Register (CRn)	Access	Description
0-3	Read / Write	Performance Monitoring Registers
4-5	Unpredictable	Reserved
6	Read Only	Core Frequency
7	Unpredictable	Reserved
8-15	Read / Write	Software Debug

13.2.1 Registers 0-3: Performance Monitoring

The performance monitoring unit contains the following:

- Control Register (PMNC)
- Clock Counter (CCNT)
- Interrupt Enable Register (INTEN)
- Overflow Flag Register (FLAG)
- Event Selection Register (EVTSEL)
- Four Event Counters (PMN0 - PMN3)

Opcode_2 should be zero on all accesses.

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

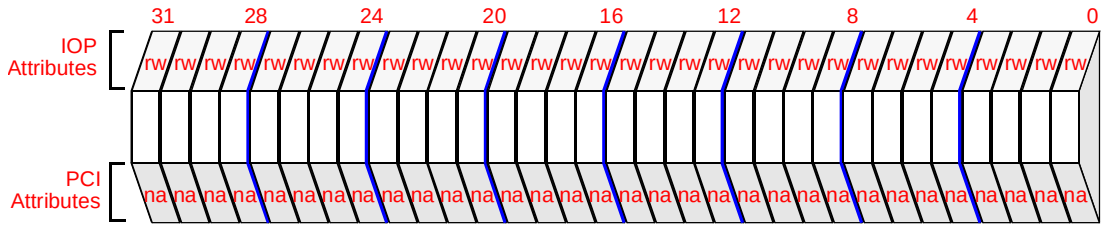
Table 326. Accessing the Performance Monitoring Registers

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) Performave Count Register 0	0b0000	0b0010	Read: MRC p14, 0, Rd, c0, c2, 0 Write: MCR p14, 0, Rd, c0, c2, 0
(PMN1) Performave Count Register 1	0b0001	0b0010	Read: MRC p14, 0, Rd, c1, c2, 0 Write: MCR p14, 0, Rd, c1, c2, 0
(PMN2) Performave Count Register 2	0b0010	0b0010	Read: MRC p14, 0, Rd, c2, c2, 0 Write: MCR p14, 0, Rd, c2, c2, 0
(PMN3) Performave Count Register 3	0b0011	0b0010	Read: MRC p14, 0, Rd, c3, c2, 0 Write: MCR p14, 0, Rd, c3, c2, 0

13.2.2 Register 1: Clock Count Register -- CCNT

The format of CCNT is shown in Table 351. The clock counter is reset to '0' by Performance Monitor Control Register (PMNC) or can be set to a predetermined value by directly writing to it. 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 6) in PMNC. An IRQ or FIQ is reported when it is enabled via bit 6 in the PMNC register.

Table 327. Clock Count Register -- CCNT



Intel® XScale™ Core Coprocessor address
CP14, Register 1

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

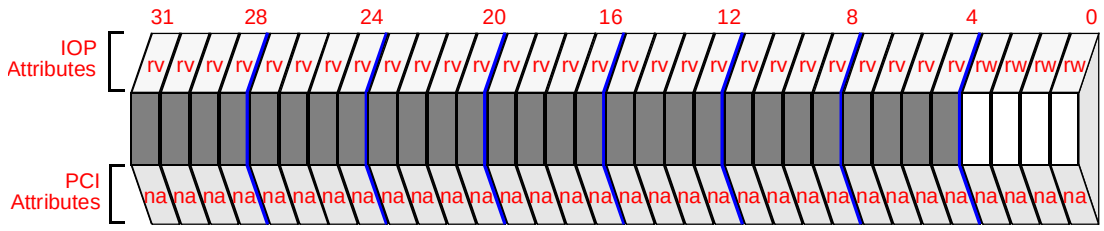
Bit	Default	Description
31:00	00000000H	**32-bit clock counter** - Reset to '0' by PMNC register. When the clock counter reaches its maximum value 0xFFFF,FFFF, the next cycle causes it to roll over to zero and generate an IRQ or FIQ when enabled.

13.2.3 Register 6: Core Clock Configuration Register -- CCLKCFG

Intel® XScale™ core frequency can be determined by reading this register.

Opcode_2 and CRm should be zero.

Table 328. Clock Count Register -- CCNT



Intel® XScale™ Core Coprocessor address
CP14, Register 1

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	000 0000H	**Reserved.**
03:00	Varies with product speed.	Core Clock Configuration (CCLKCFG): This field is used to determine the core frequency. 0000 - 500 0010 - 667 0100 - 800 all other values are reserved **NOTE:** Field is Read/Write however, writing value has no effect on core frequency and default value is overwritten.

13.2.4 Registers 8-15: Software Debug

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).

Opcode_2 and CRm are zero.

Table 329. Accessing the Debug Registers

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
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
Debug Saved Program Status Register	0b1111	MCR p14, 0, Rd, c15, c0, 0 MRC p14, 0, Rd, c15, c0, 0

13.3 CP15 Registers

Table 330 lists the CP15 registers implemented in the I/O processor.

Table 330. 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 / 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

13.3.1 Register 0: ID and Cache Type Registers

Register 0 houses two read-only registers 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 A0 stepping/revision. The low order four bits of the register are the chip revision number and is incremented for future steppings.F

Table 331. 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	0	1	0	0	0	0	0	0	1	x	0	1	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 = ARM* Version 5
15:4	Read / Write Ignored	Part Number (Implementation Specified) I/O processor: Bits[15:12] refer to the processor generation. = 0x4 Bits[11:8] refer to the implementation = 0x0 Bits[7:4] used for implementation derivatives = • 0x9 (BRG_EN = high) • 0xD (BRG_EN = low)
3:0	Read / Write Ignored	Revision number for the processor (Implementation Specified) A0 stepping = 0b0000

The Cache Type Register is selected when *opcode_2*=1 and describes the present I/O processor cache.

Table 332. Cache Type 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	0	0	0	1	0	1	1	0	0	0	1	1	0	1	0	1	0	1	0	0	0	0	1	1	0	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 = 0b110 = 32 kB
17:15	Read / Write Ignored	Data cache associativity = 0b101 = 32
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
8:6	Read / Write Ignored	Instruction cache size = 0b110 = 32 kB
5:3	Read / Write Ignored	Instruction cache associativity = 0b101 = 32 kB
2	Read-as-Zero / Write Ignored	Reserved
1:0	Read / Write Ignored	Instruction cache line length = 0b10 = 8 words/line

13.3.2 Register 1: Control and Auxiliary Control Registers

Register 1 is made up of two registers, one that is compliant with ARM Version 5 and is referenced by *opcode_2* = 0x0, and the other which is specific to Intel® XScale™ microarchitecture and is referenced by *opcode_2* = 0x1.

The Exception Vector Relocation bit (bit 13 of the ARM 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. When the MMU is enabled, the exception vectors are accessed via the usual translation method involving the PID register and the TLBs. To avoid automatic application of the PID to exception vector accesses, software may relocate the exceptions to high memory.

Table 333. ARM* 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
																		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* - ARM Limited. Order number: ARM DDI 0100E. 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* - ARM Limited. Order number: ARM DDI 0100E. 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
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 I/O processor 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 is setup before any data access is made that may be 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 can be changed.

Table 334. 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
																										MD			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 are coalesced in the write buffer as long as coalescing is globally enabled (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)** When set, page table accesses are protected by ECC. See Section 11, "Bus Controller" on page 11-1 of the *Intel® 80200 processor based on Intel® XScale Microarchitecture Developer's Manual* for more information.
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

13.3.3 Register 2: Translation Table Base Register

Table 335. 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

13.3.4 Register 3: Domain Access Control Register

Table 336. 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 can be found in the *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E..

13.3.5 Register 5: Fault Status Register

The Fault Status Register (FSR) indicates which fault has occurred, which is either a prefetch abort or a data abort. Bit 10 extends the encoding of the status field for prefetch aborts and data aborts. 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 337. 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.
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 was being accessed when a data abort occurred
3:0	Read / Write	**Status** - Type of data access being attempted

13.3.6 Register 6: Fault Address Register

Table 338. 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

13.3.7 Register 7: Cache Functions

All the functions defined in the first generation of Intel® XScale™ microarchitecture appear here. The I/O processor adds other functions as well. This register is 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 I/O processor does not check permissions on addresses supplied for cache or TLB functions. Because only privileged software may execute these functions, full accessibility is assumed. Cache functions do not generate any of the following:

- translation faults
- domain faults
- permission faults

The invalidate instruction cache line command does not invalidate the BTB. When software invalidates a line from the instruction cache and modifies the same location in external memory, it needs to 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 339 work regardless of whether the cache is enabled or disabled.

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

Table 339. 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. When a valid dirty line (with a different 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 when read.

This command may be used for cleaning the entire data cache on a context switch and also when re-configuring 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.

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 allocated into the data cache.
- When the targeted cache line is already resident, this command has no effect.
- This 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. However, when a valid store is made to that line it is marked as "dirty" and gets written back to external memory when another line is allocated to the same cache location. This eviction produces unpredictable results when the line-allocate command used a virtual address that mapped to non-existent memory.

  To avoid this situation, the line-allocate operation is only used when one of the following can be 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 must be invalidated before making another data access.

13.3.8 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 340 work regardless of whether the TLB is enabled or disabled.

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

Table 340. 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

13.3.9 Register 9: Cache Lock Down

Register 9 is used for locking down entries into the instruction cache and data cache.

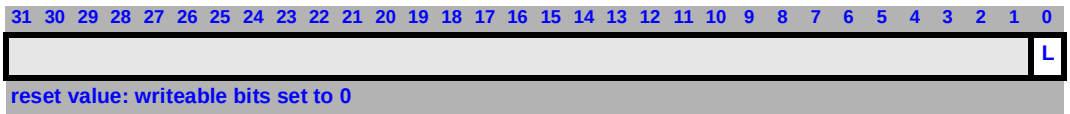
Table 341 shows the command for locking down entries in the instruction cache, instruction TLB, and data TLB. The entry to lock is specified by the virtual address in Rd. The data cache locking mechanism follows a different procedure than the others. The data cache is placed in lock down mode such that all subsequent fills to the data cache result in that line being locked in, as controlled by Table 342.

Lock/unlock operations on a disabled cache have an undefined effect. This register is accessed as write-only. Reads from this register, as with an MRC, have an undefined effect.

Table 341. Cache Lockdown 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 342. Data Cache Lock Register



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

13.3.10 Register 10: TLB Lock Down

Register 10 is used for locking down entries into the instruction TLB, and data TLB. Lock/unlock operations on a TLB when the MMU is disabled have an undefined effect.

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

Table 343 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 343. 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

13.3.11 Register 13: Process ID

The I/O 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 effectively remaps the address to one of 128 "slots" in the 4 Gbytes of address space. When 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 I/O processor do not require invalidating on a process switch.

Table 344. 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 345. 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

13.3.11.1 The PID Register Affect On Addresses

All addresses generated and used by User Mode code are eligible for being "PIDed" as described in the previous section. Privileged code, however, must be aware of certain special cases in which 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 invalidate the BTB, which prevents any virtual addresses from being double mapped between two processes.

A breakpoint address must be expressed as an MVA when written to the breakpoint register. This means the value of the PID must be combined appropriately with the address before it is written to the breakpoint register. All virtual addresses in translation descriptors are MVAs.

13.3.12 Register 14: Breakpoint Registers

The I/O 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 I/O processor also supports a 256 entry, trace buffer that records program execution information. The registers to control the trace buffer are located in CP14.

Table 346. 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

13.3.13 Register 15: Coprocessor Access Register

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

- The operating system, during a context switch, saves the state of the coprocessor when 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 needs to 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 7. 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 347. Coprocessor Access 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	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** - Program to zero for future compatibility
15:14	Read-as-Zero/Write-as-Zero	**Reserved** - Program to zero for future compatibility
13:0	Read / Write	**Coprocessor Access Rights**- Each bit in this field corresponds to the access rights for each coprocessor. For each bit: 0 = Access denied. Attempts to access corresponding coprocessor generates an undefined exception. 1 = Access allowed. Includes read and write accesses. Setting any of bits 12:1 has an undefined effect.

13.4 Core Performance Monitoring Unit (CPMON)

13.4.1 CPMON Overview

The Intel® XScale™ core hardware provides two 32-bit performance counters that allow two unique events to be monitored simultaneously. In addition, Intel® XScale™ core implements a 32-bit clock counter that can be used in conjunction with the performance counters; its sole purpose is to count the number of core clock cycles which is useful in measuring total execution time.

Intel® XScale™ core can monitor either 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 while a specified condition is true. When any of the 3 counters overflow, an IRQ or FIQ is generated when it is enabled. (IRQ or FIQ selection is programmed in the interrupt controller.) Each counter has its own interrupt enable. The counters continue to monitor events even after an overflow occurs, until disabled by software.

Each of these counters can be programmed to monitor any one of various events.

To further augment performance monitoring, the Intel® XScale™ core clock counter can be 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.

Each of the three counters and the performance monitoring control register are accessible through Coprocessor 14 (CP14), registers 0-3. Refer to Section 7.3.1, "Registers 0-3: Performance Monitoring", in the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual* (Order Number: 273411), for more details on accessing these registers with **MRC**, **MCR**, **LDC**, and **STC** coprocessor instructions. Access is allowed in privileged mode only.

13.4.2 Performance Monitoring Events

Table 348 lists events that may be monitored by the CPMON. Each of the Performance Monitor Count Registers (PMN0 and PMN1) can count any listed event. Software selects which event is counted by each PMNx register by programming the evtCountx fields of the PMNC register.

Table 348. Performance Monitoring Events

Event Number (evtCount0 or evtCount1)	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 in which the condition is present.
0x2	Stall due to a data dependency. This event occurs every cycle in which 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.
0x6	Branch mispredicted.
0x7	Instruction executed.
0x8	Stall because the data cache buffers are full. This event occurs every cycle in which 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 Section 7.2.8 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*).
0xB	Data cache miss, not including Cache Operations (defined in Section 7.2.8 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*).
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. This event occurs any time the PC is changed by software and there is not a mode change. For example, a **mov** instruction with PC as the destination triggers this event. Executing a **swi** from User mode does not trigger this event, because it incurs a mode change.
0x10	BCU detected a 1-bit error while reading data from the bus. This event may be counted even when reporting of 1-bit errors is disabled. See Section 11.3, "Error Handling" on page 11-2 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*, for a description of 1-bit errors.
0x11	The BCUs request queue is full. This event takes place each clock cycle in which the condition is met. A high incidence of this event indicates the BCU is often waiting for transactions to complete on the external bus.
0x12	The BCUs request queue is not empty and the bus is in hold (see Section 10.2.5, "Multimaster Support" on page 10-9 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*, for a description of hold mode on the external bus). This event takes place each clock cycle in which the condition is met. A high incidence of this event indicates the BCU is starved for access to the external bus.
0x13	RMW cycle occurred due to narrow write on ECC-protected memory (see Section 11.2, "ECC" on page 11-1 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*, for a description of ECC and RMW cycles).
0x14	The BCU detected an ECC error, but no ELOG register was available in which to log the error. (See Section 11.4.2, "ECC Error Registers" on page 11-9 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*, for a description of the ELOG registers).
all others	Reserved, unpredictable results

Some typical combination of counted events are listed in this section and summarized in Table 349. In this section, we call such an event combination a *mode*.

Table 349. Some Common Uses of the CPMON

Mode	PMNC.evtCount0	PMNC.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)

13.4.2.1 Instruction Cache Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions fetched from the instruction cache that were never executed. This can happen when 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 can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

13.4.2.2 Data Cache Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note that **STM** and **LDM** each count 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 Section 7.2.8 of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*, 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.

13.4.2.3 Instruction Fetch Latency Mode

PMN0 accumulates the number of cycles when the instruction-cache is not able to deliver an instruction to Intel® XScale™ core 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 the instruction cache efficiency mode and is included in this mode for convenience so that only one performance monitoring run is need.

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. When the average is high then Intel® XScale™ core may be starved of the bus external to Intel® XScale™ core.
- *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, which was used to measure total execution time.

13.4.2.4 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. When 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 Intel® XScale™ core and what the memory access latency is for Data Cache miss requests to external memory. When the Intel® XScale™ core memory access latency is high, possibly due to starvation, these Data Cache buffers iscome full. This performance monitoring mode is provided to see when Intel® XScale™ core is being starved of the bus external to Intel® XScale™ core, which effects the performance of the application running on Intel® XScale™ core.

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 when the duration event cycles are due to many requests or are attributed to just a few requests. When the average is high then Intel® XScale™ core may be starved of the bus external to Intel® XScale™ core.
- *Percent of total execution cycles processor stalled because Data Cache request buffer was not available*. Calculate by dividing PMN0 by CCNT, used to measure total execution time.

13.4.2.5 Stall/Writeback Statistics

When an instruction requires the result of a previous instruction and that result is not yet available, Intel® XScale™ core stalls in order 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: try to use load result before load completes. To avoid penalty, software should delay using load result until available. Penalty shows data-cache access latency effect.
- Multiply/Accumulate-use penalty: try to use multiply or multiply/accumulate operation result before operation completes. To avoid penalty, software should delay using result until 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 Chapter 14, "Performance Considerations" of the *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual*.

PMN1 counts the number of writeback operations emitted by the data cache. These writebacks 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:

- *Percent of total execution cycles processor stalled because of data dependency.* Calculate by dividing PMN0 by CCNT, used to measure total execution time. Often a compiler can reschedule code to avoid these penalties, given the right optimization switches.
- Total number of data writeback requests to external memory can be derived solely with PMN1.

13.4.2.6 Instruction TLB Efficiency Mode

PMN0 totals the number of instructions that were executed, which does not include instructions that were translated by the instruction TLB and never executed. This can happen when 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, which occurs when there is a TLB miss. When 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).* CPI can be derived by dividing CCNT by PMN0, where CCNT was used to measure total execution time.

13.4.2.7 Data TLB Efficiency Mode

PMN0 totals the number of data cache accesses, which includes cacheable and non-cacheable accesses, mini-data cache access and accesses made to locations configured as data RAM.

Note that **STM** and **LDM** each count 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, which occurs when there is a TLB miss. When 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.

13.4.3 Statistics from Multiple Performance Monitoring Runs

Even though only two events can be monitored at any given time, multiple performance monitoring runs can be done, capturing different events from different modes. For example, the first run could monitor the number of writeback operations (PMN1 of mode, Stall/Writeback) and the second run could monitor the total number of data cache accesses (PMN0 of mode, Data Cache Efficiency). From the results, a percentage of writeback operations to the total number of data accesses can be derived.

13.4.4 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 which generates an IRQ interrupt.

Example 8. Configuring the Performance Monitor

```
; Configure PMNC with the following values:
;   evtCount0 = 7, evtCount1 = 0instruction cache efficiency
;   inten = 0x7set all counters to trigger an interrupt on
        overflow
;   C = 1reset CCNT register
;   P = 1reset PMN0 and PMN1 registers
;   E = 1enable counting
MOV R0,#0x7777
MCR P14,0,R0,C0,c0,0;write R0 to PMNC
; Counting begins
```

Counter overflow can be dealt with in the IRQ interrupt service routine as shown below:

Example 9. Interrupt Handling

```
IRQ_INTERRUPT_SERVICE_ROUTINE:
; Assume that performance counting interrupts are the only IRQ in the system
MRC P14,0,R1,C0,c0,0; read the PMNC register
BIC R2,R1,#1; clear the enable bit
MCR P14,0,R2,C0,c0,0; clear interrupt flag and disable counting
MRC P14,0,R3,C1,c0,0; read CCNT register
MRC P14,0,R4,C2,c0,0; read PMN0 register
MRC P14,0,R5,C3,c0,0; read PMN1 register

<process the results>
SUBSPC,R14,#4; return from interrupt
```

As an example, assume the following values in CCNT, PMN0, PMN1 and PMNC:

Example 10. 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 was 2.4.

13.4.5 CPMON Registers

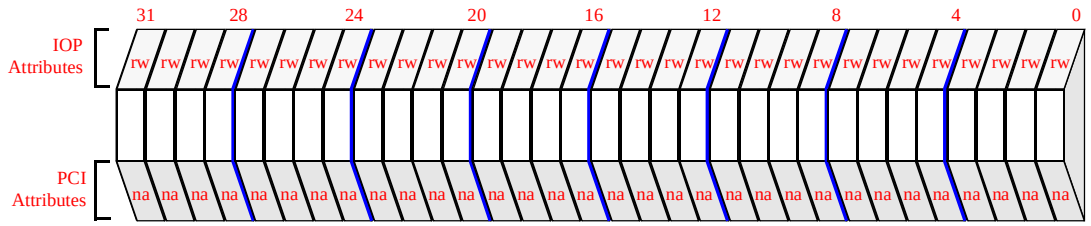
Table 350. CPMON Registers

Register	Description
CCNT	Section 13.4.5.1, "Clock Count Register -- CCNT" on page 653
PMNx	Section 13.4.5.2, "Performance Count Registers -- PMNx" on page 654
PMNC	Section 13.4.5.3, "Performance Monitor Control Register -- PMNC" on page 655

13.4.5.1 Clock Count Register -- CCNT

The format of CCNT is shown in Table 351. The clock counter is reset to '0' by Performance Monitor Control Register (PMNC) or can be set to a predetermined value by directly writing to it. 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 6) in PMNC. An IRQ or FIQ is reported when it is enabled via bit 6 in the PMNC register.

Table 351. Clock Count Register -- CCNT



Intel® XScale™ Core Coprocessor address
CP14, Register 1

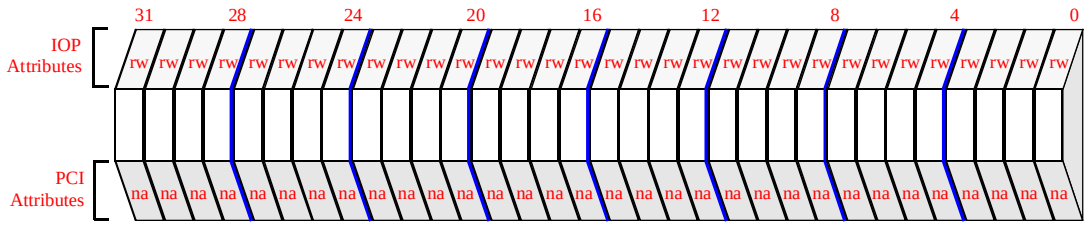
Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	**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 IRQ or FIQ when enabled.

13.4.5.2 Performance Count Registers -- PMNx

There are two 32-bit event counters; their format is shown in Table 352. The event counters are reset to '0' by the PMNC register or can be set to a predetermined value by directly writing to them. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count causes it to roll over to zero and set the overflow flag (bit 8 or 9) in PMNC. An IRQ interrupt is reported when it is enabled via bit 4 or 5 in the PMNC register.

Table 352. Performance Count Register -- PMNx



Intel® XScale™ Core Coprocessor Address

PMN0 CP14 Register 2
PMN1 CP14 Register 3

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	00000000H	**32-bit event counter** - Reset to '0' by PMNC register. When an event counter reaches its maximum value 0xFFFF,FFFF, the next event it needs to count causes it to roll over to zero and generate an IRQ interrupt when enabled.

13.4.5.2.1 Extending Count Duration Beyond 32 Bits

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 can be 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}).

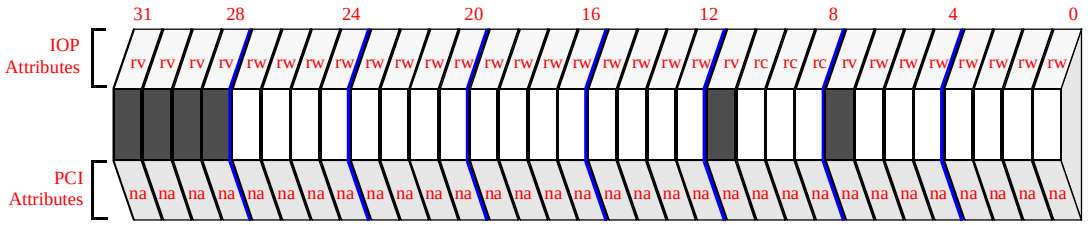
13.4.5.3 Performance Monitor Control Register -- PMNC

The performance monitor control register (PMNC) is a coprocessor register that:

- controls which events PMN0 and PMN1 monitors.
- detects which counter overflowed.
- enables/disables interrupt reporting.
- extends CCNT counting by six more bits (cycles between counter rollover = 2^{38}).
- resets all counters to zero.
- and enables the entire mechanism.

Table 353 shows the format of the PMNC register.

Table 353. Performance Monitor Control Register -- PMNC (Sheet 1 of 2)

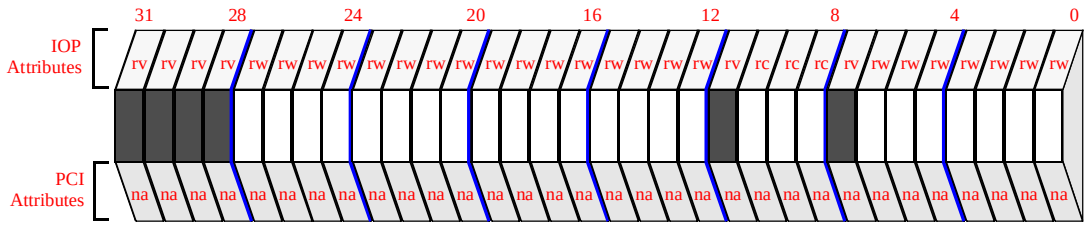


Intel® XScale™ Core Coprocessor address
CP14, Register 0

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0H	Reserved
27:20	00H	**Event Count1** - identifies the source of events that PMN1 counts. See Table 348 for a description of the values this field may contain.
19:12	00H	**Event Count0** - identifies the source of events that PMN0 counts. See Table 348 for a description of the values this field may contain.
11	0_2	Reserved
10:08	000_2	**Overflow/Interrupt Flag** - identifies which counter overflowed Bit 10 = clock counter overflow flag Bit 9 = performance counter 1 overflow flag Bit 8 = performance counter 0 overflow flag Read Values: 0 = no overflow 1 = overflow has occurred Write Values: 0 = no change 1 = clear this bit
7	0_2	Reserved

Table 353. Performance Monitor Control Register -- PMNC (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP14, Register 0

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
06:04	000_2	**Interrupt Enable** - used to enable/disable interrupt reporting for each counter Bit 6 = clock counter interrupt enable 0 = disable interrupt 1 = enable interrupt Bit 5 = performance counter 1 interrupt enable 0 = disable interrupt 1 = enable interrupt Bit 4 = performance counter 0 interrupt enable 0 = disable interrupt 1 = enable interrupt
03	0_2	**Clock Counter Divider (D)** - 0 = CCNT counts every processor clock cycle 1 = CCNT counts every 64th processor clock cycle
02	0_2	**Clock Counter Reset (C)** - 0 = no action 1 = reset the clock counter to '0x0'
01	0_2	**Performance Counter Reset (P)** - 0 = no action 1 = reset both performance counters to '0x0'
00	0_2	**Enable (E)** - 0 = all 3 counters are disabled 1 = all 3 counters are enabled

13.4.5.4 Managing PMNC

The following are a few notes about controlling the performance monitoring mechanism:

- An interrupt is reported when a counter's overflow flag is set and its associated interrupt enable bit is set in the PMNC register. The interrupt remains asserted until software clears the overflow flag by writing a one to the flag that is set. Note that the interrupt unit (Chapter 15, "Interrupt Controller Unit") and the CPSR must have enabled the interrupt in order for software to receive it.
- The counters continue to record events even after they overflow.

This Page Intentionally Left Blank

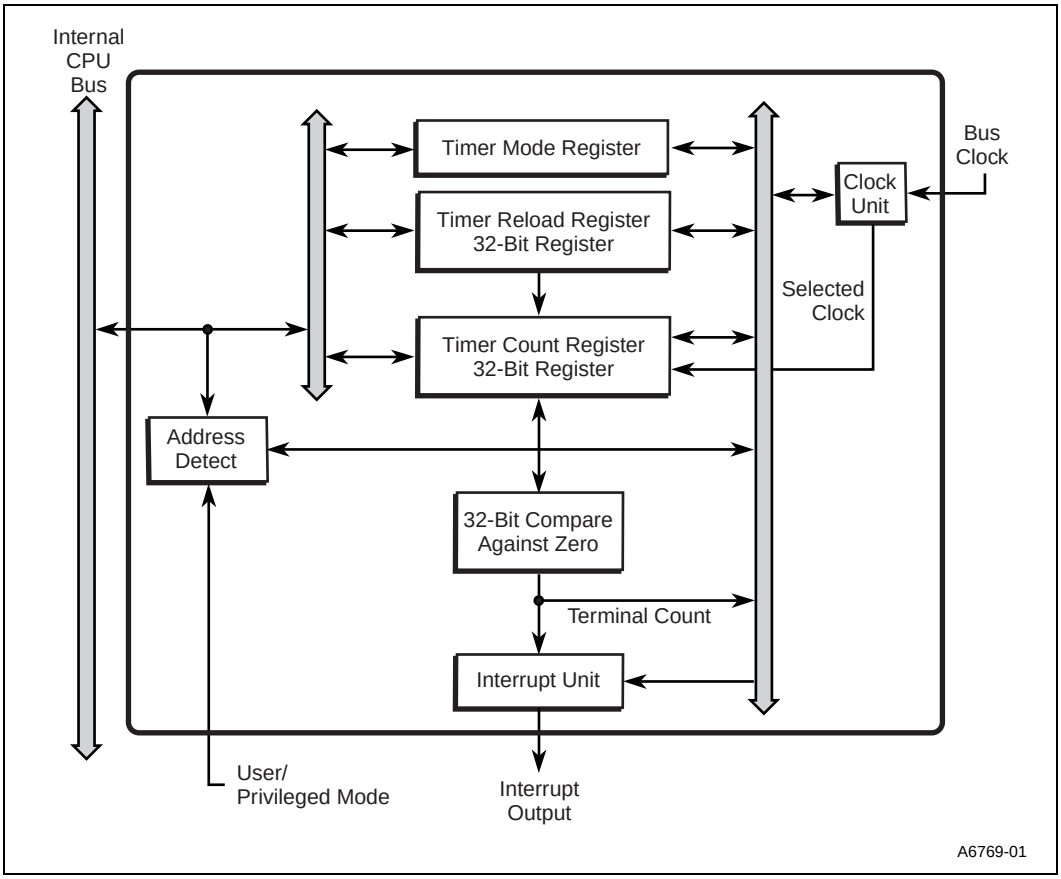
Timers 14

This chapter describes the Intel® 80331 I/O processor (80331) dual-programmable 32-bit timers and Watch Dog Timer. Topics include timer registers (TMRx, TCRx and TRRx), timer operation, timer interrupts, and timer register values at initialization.

Each timer is programmed by the timer registers. These registers are mapped into Intel® XScale™ microarchitecture Coprocessor 6, registers 0 to 5. They may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The *CRn* field of the instruction denotes the register number to be accessed. The *opcode_1* and *opcode_2* fields of the instruction should be zero. The *CRm* field of the instruction should be one. Most systems restricts access to CP6 to privileged processes. To control access to CP6, use the Coprocessor Access Register.

Figure 112 shows a diagram of the timer functions. See also Figure 113 for the Programmable Timer state diagram.

Figure 112. Programmable Timer Functional Diagram



When enabled, a timer decrements the user-defined count value with each Timer Clock (TCLOCK) cycle. The countdown rate is also user-configurable to be equal to the internal bus frequency, or the internal bus clock rate divided by 4, 8 or 16. The timers can be programmed to either stop when the count value reaches zero (single-shot mode) or run continuously (auto-reload mode). When a timer's count reaches zero, the timer's interrupt unit signals the processor's interrupt controller.

Table 354. Timer Performance Ranges

Internal Bus Frequency (MHz)	Max Resolution (ns)	Max Range (mins)
266	3.75	4.30

14.1 Timer Operation

This section summarizes the programmable timer and Watch Dog Timer operation and describes load/store access latency for the timer registers.

14.1.1 Basic Programmable Timer Operation

Each timer has a programmable enable bit in its control register (TMRx.enable) to start and stop counting. This allows the programmer to prevent user mode tasks from enabling or disabling the timer. Once the timer is enabled, the value stored in the Timer Count Register (TCRx) decrements every Timer Clock (TCLOCK) cycle. TCLOCK is determined by the Timer Input Clock Select (TMRx.csel) bit setting. The countdown rate can be set to equal the internal bus clock frequency, or the internal bus clock rate divided by 4, 8 or 16. Setting TCLOCK to a slower rate lets the user specify a longer count period with the same 32-bit TCRx value.

Software can read or write the TCRx value whether the timer is running or stopped. This lets the user monitor the count without using hardware interrupts.

When the TCRx value decrements to zero, the unit's interrupt request signals the processor's interrupt controller. See Section 14.2, "Timer Interrupts" on page 664 for more information. The timer checks the value of the timer reload bit (TMRx.reload) setting. When TMRx.reload. = 1, the processor:

- Automatically reloads TCRx with the value in the Timer Reload Register (TRRx).
- Decrements TCRx until it equals 0 again.

This process repeats until software clears TMRx.reload or TMR.enable.

When TMRx.reload = 0, the timer stops running and sets the terminal count bit (TMRx.tc). This bit remains set until user software reads or writes the TMRx register. Either access type clears the bit. The timer ignores any value specified for TMRx.tc in a write request.

Table 355. Timer Mode Register Control Bit Summary

TRRx	TCRx	Bit 2 (TMRx.reload)	Bit 1 (TMRx.enable)	Action
X	X	X	0	Timer disabled.

NOTE: X = don't care
N = a number between 1H and FFFF FFFFH

14.1.2 Watch Dog Timer Operation

The Watch Dog Timer (WDT) is a 32-bit down counter that can be used to reset the Internal Bus and the Intel® XScale™ core when software gets stuck in an infinite loop. A reset of the Internal Bus also results in the **M_RST#** output to be asserted which can be used to reset the system or as an external indicator of the WDT expiration.

Following **P_RST#** assertion, the WDT is disabled.

The software can enable the WDT by using coprocessor instructions (i.e, MCR or LDC) to write the value 1E1E 1E1EH followed by the value E1E1 E1E1H to the WDT Control register. When enabled, the WDT is initialized with FFFF FFFFH and begin to decrement towards 0000 0000H.

The software is required periodically to write the WDT initialization sequence (the value 1E1E 1E1EH followed by the value E1E1 E1E1H) to the WDT Control register in order to reset the timer value to FFFF FFFFH. For a 266 MHz Internal Bus frequency, this means that the sequence must be written approximately every fifteen seconds.

Note: The WDT always runs at Internal Bus frequency speed of 266MHz.

When the software fails to reinitialize the WDT prior to the timer value transitioning to zero, an Internal Bus Reset is generated. The Internal Bus Reset reinitializes all Internal Bus peripherals and the Intel® XScale™ core.

Warning: Once enabled, the WDT can **not** be disabled without resetting the Intel® XScale™ core.

14.1.3 Load/Store Access Latency for Timer Registers

As with all other load accesses from internal memory-mapped registers, a load instruction that accesses a timer register has a latency of one internal processor cycle. With one exception, a store access to a timer register completes and all state changes take effect before the next instruction begins execution. The exception to this is when disabling a timer. Latency associated with the disabling action is such that a timer interrupt may be posted immediately after the disabling instruction completes. This can occur when the timer is near zero as the store to TMRx occurs. In this case, the timer interrupt is posted immediately after the store to TMRx completes and before the next instruction can execute. Table 356 summarizes the timer access and response timings. Refer also to the individual register descriptions for details.

Note that the processor may delay the actual issuing of the load or store operation due to previous instruction activity and resource availability of processor functional units.

The processor ensures that the TMRx.tc bit is cleared within one internal bus clock after a load or store instruction accesses TMRx.

Table 356. Timer Responses to Register Bit Settings

Name	Status	Action
(TMRx.tc) Terminal Count Bit 0	READ	Timer clears this bit when user software accesses TMRx. This bit can be set 1 internal bus clock later. The timer sets this bit within 1 internal bus clock of TCRx reaching zero when TMRx.reload=0.
	WRITE	Timer clears this bit within 1 internal bus clock after the software accesses TMRx. The timer ignores any value specified for TMRx.tc in a write request.
(TMRx.enable) Timer Enable Bit 1	READ	Bit is available 1 internal bus clock after executing a read instruction from TMRx.
	WRITE	Writing a '1' enables the internal bus clock to decrement TCRx within 1 internal bus clock after executing a store instruction to TMRx.
(TMRx.reload) Timer Auto Reload Enable Bit 2	READ	Bit is available 1 internal bus clock after executing a read instruction from TMRx.
	WRITE	Writing a '1' enables the reload capability within 1 internal bus clock after the store instruction to TMRx has executed. The timer loads TRRx data into TCRx and decrements this value during the next internal bus clock cycle.
(TMRx.csel1:0) Timer Input Clock Select Bits 4-5	READ	Bits are available 1 internal bus clock after executing a read instruction from TMRx.csel1:0 bit(s).
	WRITE	The timer re-synchronizes the clock cycle used to decrement TCRx within one internal bus clock cycle after executing a store instruction to TMRx.csel1:0 bit(s).
(TCRx.d31:0) Timer Count Register	READ	The current TCRx count value is available within 1 internal bus clock cycle after executing a read instruction from TCRx. When the timer is running, the pre-decremented value is returned as the current value.
	WRITE	The value written to TCRx becomes the active value within 1 internal bus clock cycle. When the timer is running, the value written is decremented in the current clock cycle.
(TRRx.d31:0) Timer Reload Register	READ	The current TRRx count value is available within 1 internal bus clock after executing a read instruction from TRRx. When the timer is transferring the TRRx count into TCRx in the current count cycle, the timer returns the new TCRx count value to the executing read instruction.
	WRITE	The value written to TRRx becomes the active value stored in TRRx within 1 internal bus clock cycle. When the timer is transferring the TRRx value into the TCRx, data written to TRRx is also transferred into TCRx.

14.2 Timer Interrupts

Each timer is the source for one interrupt. When a timer detects a zero count in its TCRx, the timer generates an internal level-detected Timer Interrupt signal (TINTx) to the interrupt controller, and the interrupt source (INTSRC[1:0]) bit is set in the interrupt controller. Each timer interrupt can be selectively masked in the Interrupt Control (INTCTL[1:0]) registers. Refer to Section 15.5, "Intel® 80331 I/O Processor Interrupt Controller Unit" for a description of interrupt controller operation.

After servicing the timer interrupt, the interrupt service routine clears the pending request by writing a '1' to the appropriate bit of the Timer Interrupt Status Register (TISR).

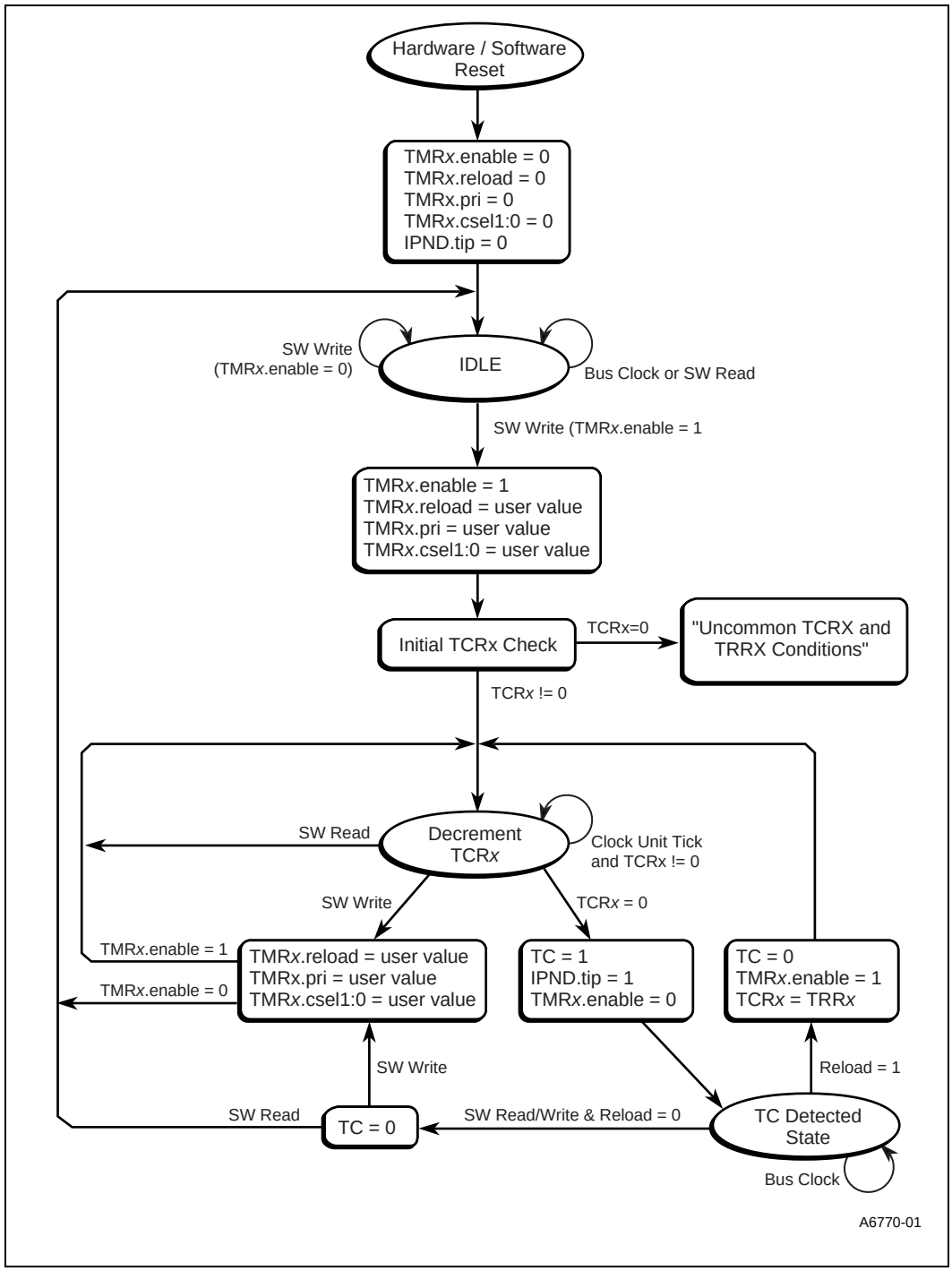
When a timer generates a second interrupt request before the CPU services the first interrupt request, the second request may be lost.

When auto-reload is enabled for a timer, the timer continues to decrement the value in TCRx even after entry into the timer interrupt handler.

14.3 Timer State Diagram

Figure 113 shows the common states of the Timer Unit. For uncommon conditions see Section 14.5, "Uncommon TCRX and TRRX Conditions" on page 674.

Figure 113. Timer Unit State Diagram



14.4 Timer Registers

As shown in Table 357, each timer has three memory-mapped registers:

- Timer Mode Register - programs the specific mode of operation or indicates the current programmed status of the timer. This register is described in Section 14.4.2, "Timer Mode Registers – TMR0:1" on page 667.
- Timer Count Register - contains the timer's current count. See Section 14.4.3, "Timer Count Register – TCR0:1" on page 670.
- Timer Reload Register - contains the timer's reload count. See Section 14.4.4, "Timer Reload Register – TRR0:1" on page 671.

Table 357. Timer Registers

Timer Unit	Register Acronym	Register Name
Timer 0	TMR0	Timer Mode Register 0
	TCR0	Timer Count Register 0
	TRR0	Timer Reload Register 0
Timer 1	TMR1	Timer Mode Register 1
	TCR1	Timer Count Register 1
	TRR1	Timer Reload Register 1

14.4.1 Power Up/Reset Initialization

Upon **P_RST#** assertion, the timer registers are initialized to the values shown in Table 358.

Table 358. Timer Powerup Mode Settings

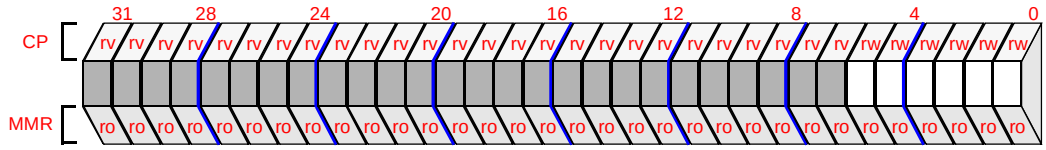
Mode/Control Bit	Notes
TMRx.tc = 0	No terminal count
TMRx.enable = 0	Prevents counting and assertion of TINTx
TMRx.reload = 0	Single terminal count mode
TMRx.pri = 0	Privileged Mode and User Mode Writes Allowed
TMRx.csel1:0 = 0	Timer Clock = internal bus clock
TCRx.d31:0 = 0	Undefined
TRRx.d31:0 = 0	Undefined
TINTx output	Deasserted

14.4.2 Timer Mode Registers – TMR0:1

The Timer Mode Register (TMRx) lets the user program the mode of operation and determine the current status of the timer. TMRx bits are described in the subsections following Table 359 and are summarized in Table 355.

Note: TMR0:1 registers are read-only from Memory-Mapped Register Address space.

Table 359. Timer Mode Register – TMRx



Intel® XScale™ Core Local Bus address
TMR0: FFFF E7D0H
TMR1: FFFFE7D4H

Intel® XScale™ Core Coprocessor address
TMR0: CP6, Register 0
TMR1: CP6, Register 1

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

31:06	0000 000H	Reserved. Initialize to 0.
05:04	00_2	Timer Input Clock Selects - TMRx.csel1:0 (00) 1:1 Timer Clock = internal bus clock (01) 4:1 Timer Clock = internal bus clock / 4 (10) 8:1 Timer Clock = internal bus clock / 8 (11) 16:1 Timer Clock = internal bus clock / 16
03	0_2	Timer Register Privileged Write Control - TMRx.pri (0) Privileged and User Mode Write Enabled (1) Privileged Mode Only Write Enabled
02	0_2	Timer Auto Reload Enable - TMRx.reload (0) Auto Reload Disabled (1) Auto Reload Enabled
01	0_2	Timer Enable - TMRx.enable (0) Disabled (1) Enabled
00	0_2	Terminal Count Status - TMRx.tc (0) No Terminal Count (1) Terminal Count

14.4.2.1 Bit 0 - Terminal Count Status Bit (TMRx.tc)

The TMRx.tc bit is set when the Timer Count Register (TCRx) decrements to 0 and bit 2 (TMRx.reload) is not set for a timer. The TMRx.tc bit allows applications to monitor timer status through software instead of interrupts. TMRx.tc remains set until software accesses (reads or writes) TMRx. The access clears TMRx.tc. The timer ignores any value specified for TMRx.tc in a write request.

When auto-reload is selected for a timer and the timer is enabled, the TMRx.tc bit status is unpredictable. Software should not rely on the value of the TMRx.tc bit when auto-reload is enabled.

The processor also clears the TMRx.tc bit upon hardware or software reset. Refer to Section 18.2, "Reset Overview" on page 755.

14.4.2.2 Bit 1 - Timer Enable (TMRx.enable)

The TMRx.enable bit allows user software to control the timer's RUN/STOP status. When:

TMRx.enable = 1	The Timer Count Register (TCRx) value decrements every Timer Clock (TCLOCK) cycle. TCLOCK is determined by the Timer Input Clock Select (TMRx.csel bits 0-1). See Section 14.4.2.5. When TMRx.reload=0, the timer automatically clears TMRx.enable when the count reaches zero. When TMRx.reload=1, the bit remains set. See Section 14.4.2.3.
TMRx.enable = 0	The timer is disabled and ignores all input transitions.

User software sets this bit. Once started, the timer continues to run, regardless of other processor activity. Three events can stop the timer:

- User software explicitly clearing this bit (i.e., TMRx.enable = 0).
- TCRx value decrements to 0, and the Timer Auto Reload Enable (TMRx.reload) bit = 0.
- Hardware or software reset. Refer to Section 18.2, "Reset Overview" on page 755.

14.4.2.3 Bit 2 - Timer Auto Reload Enable (TMRx.reload)

The TMRx.reload bit determines whether the timer runs continuously or in single-shot mode. When TCRx = 0 and TMRx.enable = 1 and:

TMRx.reload = 1 The timer runs continuously. The processor:

1. Automatically loads TCRx with the value in the Timer Reload Register (TRRx), when TCRx value decrements to 0.
2. Decrements TCRx until it equals 0 again.

Steps 1 and 2 repeat until software clears TMRx bits 1 or 2.

TMRx.reload = 0	The timer runs until the Timer Count Register = 0. TRRx has no effect on the timer.

User software sets this bit. When TMRx.enable and TMRx.reload are set and TRRx does not equal 0, the timer continues to run in auto-reload mode, regardless of other processor activity. Two events can stop the timer:

- User software explicitly clearing either TMRx.enable or TMRx.reload.
- Hardware or software reset.

The processor clears this bit upon hardware or software reset.

14.4.2.4 Bit 3 - Timer Register Privileged Read/Write Control (TMRx.pri)

The TMRx.pri bit enables or disables user mode writes to the timer registers (TMRx, TCRx, TRRx). Privileged mode writes are allowed regardless of this bit's condition. Software can read these registers from either mode.

When:

TMRx.pri = 1	The timer ignores the user mode write to the timer registers; however, writes from the privileged modes are allowed.
TMRx.pri = 0	The timer registers can be written from either the user mode or the privileged modes.

The processor clears TMRx.pri upon hardware or software reset.

14.4.2.5 Bits 4, 5 - Timer Input Clock Select (TMRx.csel1:0)

User software programs the TMRx.csel bits to select the Timer Clock (TCLOCK) frequency. See Table 360. As shown in Figure 112, the internal bus clock is an input to the timer clock unit. These bits allow the application to specify whether TCLOCK runs at or slower than the internal bus clock frequency.

Table 360. Timer Input Clock (TCLOCK) Frequency Selection

Bit 5 TMRx.csel1	Bit 4 TMRx.csel0	Timer Clock (TCLOCK)
0	0	Timer Clock = internal bus clock
0	1	Timer Clock = internal bus clock / 4
1	0	Timer Clock = internal bus clock / 8
1	1	Timer Clock = internal bus clock / 16

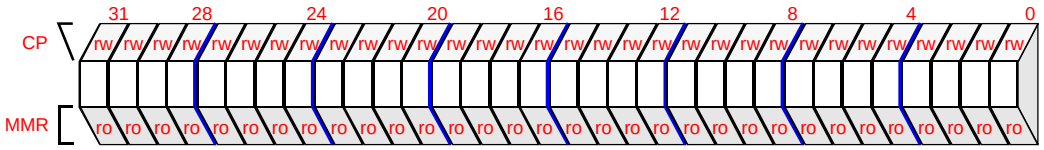
The processor clears these bits upon hardware or software reset (TCLOCK = Core Clock).

14.4.3 Timer Count Register – TCR0:1

The Timer Count Register (TCRx) is a 32-bit register that contains the timer's current count. The register value decrements with each timer clock tick. When this register value decrements to zero (terminal count), a timer interrupt is generated. When TMRx.reload is not set for the timer, the status bit in the timer mode register (TMRx.tc) is set and remains set until the TMRx register is accessed. Table 361 shows the timer count register.

Note: TCR0:1 registers are read-only from Memory-Mapped Register Address space.

Table 361. Timer Count Register – TCRx



Intel® XScale™ Core Local Bus address
TCR0: FFFF E7D8H
TCR1: FFFF E7DCH

Intel® XScale™ Core Coprocessor address
TCR0: CP6, Register 2
TCR1: CP6, Register 3

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

31:00	0000 0000H	Timer Count Value - TCRx.d31:0

The valid programmable range is from 1H to FFFF FFFFH. Avoid programming TCRx to 0 as it has varying results as described in Section 14.5, "Uncommon TCRX and TRRX Conditions" on page 674.

User software can read or write TCRx whether the timer is running or stopped. Bit 3 of TMRx determines user read/write control (Section 14.4.2.5). The TCRx value is undefined after hardware or software reset.

14.4.4 Timer Reload Register – TRR0:1

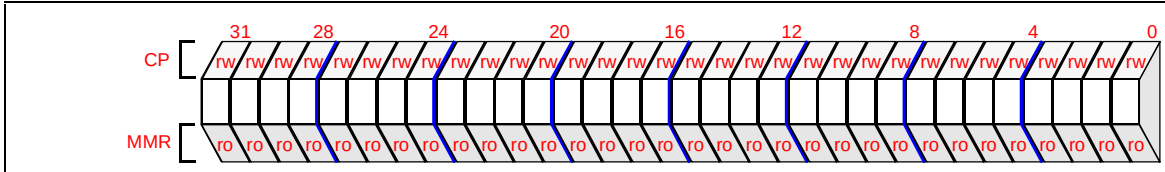
The Timer Reload Register (TRRx; Table 362) is a 32-bit register that contains the timer's reload count. The timer loads the reload count value into TCRx when TMRx.reload is set (1), TMRx.enable is set (1) and TCRx equals zero.

As with TCRx, the valid programmable range is from 1H to FFFF FFFFH. Avoid programming a value of 0, as it may prevent TINTx from asserting continuously. (See Section 14.5, "Uncommon TCRX and TRRX Conditions" on page 674 for more information.)

User software can access TRRx whether the timer is running or stopped. Bit 3 of TMRx determines read/write control (Section 14.4.2.5, "Bits 4, 5 - Timer Input Clock Select (TMRx.csel1:0)" on page 669). TRRx value is undefined after hardware or software reset.

Note: TRR0:1 registers are read-only from Memory-Mapped Register Address space.

Table 362. Timer Reload Register – TRRx



Intel® XScale™ Core Local Bus address
TRR0: FFFF E7E0H
TRR1: FFFF E7E4H

Intel® XScale™ Core Coprocessor address
TRR0: CP6, Register 4
TRR1: CP6, Register 5

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

31:00	0000 0000H	Timer Auto-Reload Value - TRRx.d31:0

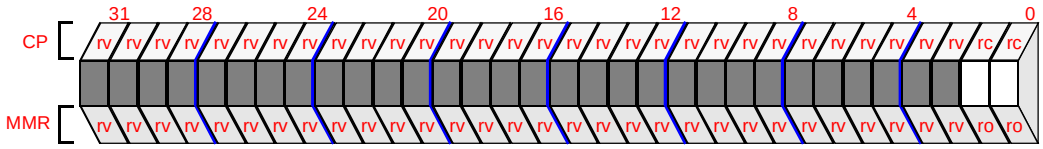
14.4.5 Timer Interrupt Status Register – TISR

The Timer Interrupt Status Register (TISR; Table 363) is a two-bit register that contains the timer's pending interrupt status. The setting of these status bits represents the assertion of a "level-sensitive" interrupt request to the "Intel® 80331 I/O Processor Interrupt Controller Unit". After the interrupt service routine completes processing of the interrupt request, it needs to write a '1' to the appropriate bit in the TISR to clear the pending request.

TISR interrupt requests are cleared after hardware or software reset.

Note: TISR register is read-only from Memory-Mapped Register Address space.

Table 363. Timer Interrupt Status Register – TISR



Intel® XScale™ Core Local Bus address
TISR: FFFF E7E8H

Intel® XScale™ Core Coprocessor address
TISR: CP6, Register 6

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

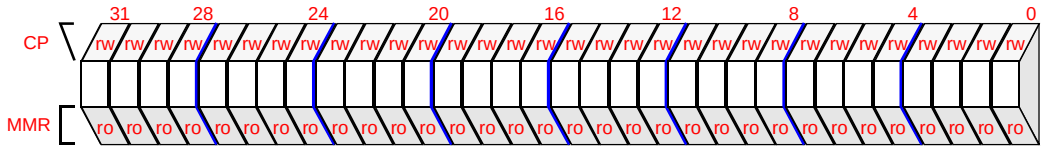
31:02	0000 0000H	Reserved.
01	0_2	Timer 1 Interrupt Pending - When set, there is an interrupt pending from Timer 1. This occurs when Timer 1 detects a zero count in TCR1. After servicing the interrupt, SW needs to write a '1' to this bit to clear the pending request.
00	0_2	Timer 0 Interrupt Pending - When set, there is an interrupt pending from Timer 0. This occurs when Timer 0 detects a zero count in TCR0. After servicing the interrupt, SW needs to write a '1' to this bit to clear the pending request.

14.4.6 Watch Dog Timer Control Register – WDTCR

The Watch Dog Timer Control Register (WDTCR) is a 32-bit register that software can use to enable the WDT or read the current WDT count value. The register value decrements with each internal bus clock tick. When this register value decrements to zero (terminal count), an Internal Bus Reset is generated. The timer can be enabled and/or reinitialized by writing 1E1E 1E1EH immediately followed by E1E1 E1E1H to the WDTCR.

Note: WDTCR register is read-only from Memory-Mapped Register Address space.

Table 364. Watch Dog Timer Control Register -- WDTCR



Intel® XScale™ Core Local Bus address
WDTCR: FFFF E7ECH

Intel® XScale™ Core Coprocessor address
WDTCR: CP6, Register 7

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

31:00	0000 0000H	Watch Dog Timer Count Value - By writing 1E1E 1E1EH followed by E1E1 E1E1H to this register, software can enable the WDT and reset the count value to FFFF FFFFH. When read, this register returns the current value contained in the WDT.

14.5 Uncommon TCRx and TRRx Conditions

Table 355 summarizes the most common settings for programming the timer registers. Under certain conditions, however, it may be useful to set the Timer Count Register or the Timer Reload Register to zero before enabling the timer. Table 365 details the conditions and results when these conditions are set.

Table 365. Uncommon TMRx Control Bit Settings

TRRx	TCRx	Bit 2 (TMRx.reload)	Bit 1 (TMRx.enable)	Action
X	0	0	1	TMRx.tc and TINTx set, TMR.enable cleared
0	0	1	1	Timer and auto reload enabled, TINTx not generated and timer enable remains set.
0	N	1	1	Timer and auto reload enabled. TINTx set when TCRx=0. The timer remains enabled but further TINTx's are not generated.

NOTE: X = don't care
N = a number between 1H and FFFF FFFFH

Interrupt Controller Unit 15

This chapter describes the Intel® 80331 I/O processor (80331) Interrupt Controller Unit. The operation modes, setup, external memory interface, and implementation of the interrupts are described in this chapter.

15.1 Overview

The interrupt control unit manages the interrupt routing and interrupt sources to the Intel® XScale™ core.

An interrupt is an event that causes a temporary break in program execution so the processor can handle another task. Interrupts commonly request I/O services or synchronize the processor with some external hardware activity. For interrupt handler portability across the Intel® XScale™ microarchitecture family (ARM* architecture compliant), the architecture defines a consistent exception handling mechanism. To manage exceptions which include interrupt requests in parallel with processor execution, the 80331 provides an on-chip programmable interrupt controller.

Requests for interrupt service come from many sources and are prioritized such that instruction execution is redirected only when an exception interrupt request is of higher priority than that of the executing task. On the 80331, interrupt requests may originate from external hardware sources, internal peripherals or software. The 80331 contains a number of integrated peripherals which may generate interrupts, including:

- DMA Channel 0
- DMA Channel 1
- ATU
- UART 0 and 1
- Timers 0 and 1
- Performance Monitoring Unit
- I^2C Bus Interface Units 0 and 1
- Application Accelerator Unit
- Messaging Unit
- Memory Controller Unit
- Peripheral Bus Interface Unit
-

The interrupt controller can also intercept external processor interrupts and forward them to the PCI interrupt pins.

Interrupts are detected with the chip 4-bit interrupt port, an 12-bit GPIO port, and with a dedicated High-Priority Interrupt (**HPI#**) input in the Intel® XScale™ core interrupt controller. Interrupt requests originate from software by the **SWI** instruction.

Ultimately, all interrupt sources that are steered to the Intel® XScale™ core processor are combined into one of two internal interrupt exceptions: IRQ and FIQ.

15.2 Theory of Operation

15.2.1 Interrupt Controller Unit

The 80331 Interrupt Controller Unit (ICU) provides the ability to generate interrupts to both the Intel® XScale™ core and the PCI interrupt pins.

In addition to the internal peripherals, external devices may also generate interrupts to the Intel® XScale™ core. External devices can generate interrupts via the **S_INT[D:A]#** pins, the **GPIO[7:0]**/**XINT[15:8]#** pins, and the **HPI#** pin. The Peripheral Interrupt Controller provides the ability to direct PCI interrupts. The routing logic enables, under software control, the ability to intercept external PCI interrupts and forward to the PCI interrupt output pinsor to the Intel® XScale™ core.

The Interrupt Controller has two functions:

- Internal Peripheral Interrupt Control
- External Interrupt Routing

The internal peripheral interrupt control mechanism consolidates a number of interrupt sources for a given peripheral into a single interrupt driven to the Intel® XScale™ core. High performance data movement associated interrupts are fully demultiplexed into the ICU, however. In order to provide the executing software with the knowledge of interrupt source, coprocessor mapped status registers describe the source of the active interrupts and the vectors to interrupt handlers for the highest priority active sources. All of the peripheral interrupts are individually enabled from the respective peripheral control registers.

The Peripheral interrupt routing mechanism allows the host software (or Intel® XScale™ microarchitecture software) to route external interrupts to either the Intel® XScale™ core or the **P_INTA#**, **P_INTB#**, **P_INTC#**, and **P_INTD#** output pins. This routing mechanism is controlled through a memory-mapped register accessible from the 80331.

15.3 The Intel® XScale™ Core Exceptions Architecture

The Intel® XScale™ core supports five types of exceptions[1], and a privileged processing mode for each type.

- IRQ and FIQ internal interrupt exceptions otherwise known as the normal and fast interrupts, respectively
- memory aborts (used to implement memory protection or virtual memory)
- attempted execution of an undefined instruction
- software interrupts (SWIs) (used to make a call to an Operating System)

When an exception occurs, some of the standard registers are replaced with registers specific to the exception mode. All exceptions have replacement (or banked) registers for R14 and R13, and one interrupt mode has more registers for fast interrupt processing.

After an exception, R14 holds the return address for exception processing, which is used both to return after the exception is processed and to address the instruction that caused the exception.

R13 is banked across exception modes to provide each exception handler with a private stack pointer (SP). The fast interrupt mode also banks R8 to R12, so that interrupt processing can begin without the need to save or restore these registers. There is a seventh processing mode, System Mode, that does not have any banked registers (it uses the User mode registers), which is used to run normal (non-exception) tasks that require a privileged processor mode.

15.3.1 CPSR and SPSR

All other processor state is held in status registers. The current operating processor status is in the Current Program Status Register or CPSR. The CPSR holds:

- Four condition code flags (Negative, Zero, Carry and Overflow)
- Two interrupt disable bits (one for each type of interrupt)
- Five bits which encode the current processor mode

All five exception modes also have a Saved Program Status Register (SPSR) which holds the CPSR of the task immediately before the exception occurred. Both the CPSR and SPSR are accessed with special instructions.

15.3.2 The Exception Process

When an exception occurs, the Intel® XScale™ core halts execution after the current instruction and begins execution at a fixed address in low memory, known as the exception vectors. There is a separate vector location for each exception (and two for memory aborts to distinguish between data and instruction accesses).

An operating system installs a handler on every exception at initialization. Privileged operating system tasks normally run in System mode to allow exceptions to occur within the operating system without state loss (exceptions overwrite their R14 when an exception occurs, and System mode is the only privileged mode that cannot be entered by an exception).

1. Exception Description from the ARM Architecture Reference Manual, p. 1-3, Copyright Advanced RISC Machines Ltd. (ARM) 1996

15.3.3 Exception Priorities and Vectors

It is important to note that fast interrupt (FIQ) is higher priority than the normal interrupt (IRQ). In addition, while an FIQ exception is executing, the IRQ exception is masked out.

When an exception is taken by the processor, the Program Counter (PC) is loaded with the vector associated with that exception as specified by Table 366.

Generally, the instruction at this location is required to be a branch instruction to the associated exception handler. However, in the case of an FIQ, this is not necessary since the vector location is at the very bottom of all the defined exception vectors, thus the entire FIQ exception handler can be placed at that vector location.

Table 366. Exception Priorities And Vectors

Exception	Priority	Vector[a]
Reset	1 (Highest)	0000 0000H
Data Abort	2	0000 0010H
FIQ	3	0000 001CH
IRQ	4	0000 0018H
Prefetch Abort	5	0000 000CH
Undefined Instructions	6 (Lowest)	0000 0004H
Software Interrupt (SWI)[b]	6 (Lowest)	0000 0008H

a. By enabling the Exception Vector Relocation mode (bit 13, CP15, Register 1), the Vectors (except Reset Vector) can be relocated to be based at FFFF 0000H rather than 0000 0000H. (i.e., FIQ Vector located at FFFF 001CH)
b. Undefined Instruction and SWI can not occur at the same time since SWI is a particular instruction decoding.

15.3.4 Software Requirements For Exception Handling

To use the processor's exception handling facilities, user software must provide the following items in memory:

- Exception Handler Routines
- Software handler to nest certain exceptions (i.e., FIQ and IRQ)

These items are established in memory as part of the initialization procedure.

15.3.4.1 Nesting FIQ and IRQ Exceptions

Hardware does not provide support for nesting of any particular exception, including the FIQ and IRQ exceptions.

In order to provide support for nested interrupts, a software handler must be provided to save the Link Register (R14) and the SPSR (Saved Program Status Register) before reenabling the FIQ or IRQ exception.

15.4 Intel® 80331 I/O Processor External Interrupt Interface

The interrupt controller attached to the Intel® XScale™ core, has the facilities necessary to handle all core processor and peripheral internal interrupts, as well as four external interrupts (**S_INT[D:A]#XINT[3:0]#**), twelve GPIO inputs (**GPIO[7:0]**), and one High Priority Interrupt (**HPI#**).

80331 Primary PCI local bus interface includes four interrupt output signals. Interrupts from the Secondary PCI bus, the Messaging Unit and Address Translation Unit are routed to these interrupt output signals via the interrupt controller steering logic.

15.4.1 Interrupt Inputs

The 17 external interrupt input pins of the 80331 have the following definitions:

S_INT[D:A]#XINT[3:0]#External Interrupt (Input) - These pins cause interrupts to be requested. Each pin is a level-detect input only.These pins are internally synchronized. These pins only act as interrupt inputs when they are unmasked in the INTCTL[1:0] registers.

GPIO[7:0] **External Interrupt (Input)** - These pins cause interrupts (**XINT[15:8]#**) to be requested. Each pin is a level-detect input only.These pins are internally synchronized. These pins only act as interrupt inputs when they are configured as general purpose inputs and are unmasked in the INTCTL[1:0] registers.

HPI# **High-Priority Interrupt (Input)** - Causes a high priority interrupt event to occur. The external **HPI#** input requires a level input and is maskable by the INTCTL[1:0] registers. This pin is internally synchronized.

The external interrupt input interface for the 80331 consists of the pins shown in Table 367.

Table 367. Interrupt Input Pin Descriptions

Signal	Description
S_INTA#/XINT0#	Directed to the **P_INTA#** (**XINT4#**) output or the 80331 Interrupt Controller input **XINT0#**. When directed to the 80331 Interrupt Controller input **XINT0#**, the **XINT0#** input can be steered to either the FIQ or the IRQ internal interrupt input of the Intel® XScale™ core.
S_INTB#/XINT1#	Directed to the **P_INTB#** (**XINT5#**) output or the 80331 Interrupt Controller input **XINT1#**. When directed to the 80331 Interrupt Controller input **XINT1#**, the **XINT1#** input can be steered to either the FIQ or the IRQ internal interrupt input of the Intel® XScale™ core.
S_INTC#/XINT2#	Directed to the **P_INTC#** (**XINT6#**) output or the 80331 Interrupt Controller input **XINT2#**. When directed to the 80331 Interrupt Controller input **XINT2#**, the **XINT2#** input can be steered to either the FIQ or the IRQ internal interrupt input of the Intel® XScale™ core.
S_INTD#/XINT3#	Directed to the **P_INTD#** (**XINT7#**) output or the 80331 Interrupt Controller input **XINT3#**. When directed to the 80331 Interrupt Controller input **XINT3#**, the **XINT3#** input can be steered to either the FIQ or the IRQ internal interrupt input of the Intel® XScale™ core.
XINT[15:8]#	Interrupt Inputs dedicated to the Intel® XScale™ core. Functionary is muxed with **GPIO[7:0]** functionality. **XINT8#** corresponds to **GPIO[0]**, **XINT9#** corresponds to **GPIO[1]**, and so on. Interrupt functionality is available on these pins only when configured as General Purpose Inputs. To enable a given pin as an interrupt, it needs to be unmasked in the INTCTL1 register.
HPI#	**HPI#** is a dedicated interrupt input to the Intel® XScale™ core. This interrupt can be disabled by the INTCTL0 register, and can be steered to either the FIQ or the IRQ internal interrupt (not the interrupt outputs).

15.4.2 Outbound Interrupts

The Messaging Unit (MU) has the capability of generating interrupts on the PCI interrupt output pin . The MU has four distinct messaging mechanisms. Each allows a host processor or external PCI agent and the 80331 to communicate through message passing and interrupt generation. The four mechanisms are:

- **Message Registers** — allow the 80331 and external PCI agents to communicate by passing messages in one of four 32-bit Message Registers. In this context, a message is any 32-bit data value. Message registers combine aspects of mailbox registers and doorbell registers. Writes to the message registers may optionally cause interrupts.
- **Doorbell Registers** — allow the 80331 to assert the PCI interrupt signals and allow external PCI agents to generate an interrupt to the Intel® XScale™ core.
- **Circular Queues** — support a message passing scheme that uses four circular queues.
- **Index Registers** — support a message passing scheme that uses a portion of the 80331 local memory to implement a large set of message registers.

All four mechanisms can result in Outbound Interrupts to a host processor.

The Messaging Unit (MU) is a part of the Address Translation Unit (ATU) and uses the ATU interrupt pin.

The external interrupt output interface for 80331 consists of the pins shown in Table 368.

Table 368. Interrupt Output Pin Descriptions

Signal	Description
P_INTA#	Primary PCI Interrupt output of 80331 source from **S_INTA#** based on interrupt steering control. When the 80331 bridge is disabled (**BRG_EN** is low), and arbiter/central resource is disabled (**ARB_EN** is low), the 80331 ATU/MU interrupt is also connected to this Primary PCI Output.
P_INTB#	Primary PCI Interrupt output of 80331 source from **S_INTB#** based on interrupt steering control.
P_INTC#	Primary PCI Interrupt output of 80331 source from **S_INTC#** based on interrupt steering control. The 80331 ATU/MU interrupt is also connected to this Primary PCI Output based on internally wiring the ATU IDSEL input to S_AD30. This routing applies to 80331 modes with bridge enabled (**BRG_EN** is high), and bridge disabled (**BRG_EN** is low) and arbiter/central resource is enabled (**ARB_EN** is high).
P_INTD#	Primary PCI Interrupt output of 80331 source from **S_INTD#** based on interrupt steering control.

15.4.3 Interrupt Routing

The four PCI interrupt inputs of the 80331, **S_INT[D:A]#/XINT[3:0]#**, can be routed to either Intel® XScale™ core interrupt inputs or to the primary PCI interrupt output pins **P_INT[D:A]#/XINT[7:4]#**.

Routing of interrupt inputs is controlled by the PCI Interrupt Routing Select Register (PIRSR). Table 369 summarizes the usage of the bits in the PIRSR.

Table 369. Interrupt Routing Summary

PIRSR Select Bit	Bit Value	Description
bit 0	1	**S_INTA#** Input Pin routed to ICU **XINT0#** Input Pin
	0	**S_INTA#** Input Pin routed to **P_INTA#** Output Pin
bit 1	1	**S_INTB#** Input Pin routed to ICU **XINT1#** Input Pin
	0	**S_INTB#** Input Pin routed to **P_INTB#** Output Pin
bit 2	1	**S_INTC#** Input Pin routed to ICU **XINT2#** Input Pin
	0	**S_INTC#** Input Pin routed to **P_INTC#** Output Pin
bit 3	1	**S_INTD#** Input Pin routed to ICU **XINT3#** Input Pin
	0	**S_INTD#** Input Pin routed to **P_INTD#** Output Pin

Note: **XINT0#** through **XINT3#** of the 80331 Interrupt Controller only handles level sensitive inputs such as PCI interrupts. When any Select bit is cleared, the logic external to the ICU input must drive an inactive level ('1') to the corresponding interrupt input of the 80331 Interrupt Controller.

15.5 Intel® 80331 I/O Processor Interrupt Controller Unit

The 80331 Interrupt Controller Unit (ICU) provides a flexible, low-latency means for requesting interrupts and minimizing the core's interrupt handling burden.

All interrupt sources are combined into one of the two internal interrupt exceptions: IRQ and FIQ.

The interrupt controller provides the following features for managing hardware-requested interrupts:

- Flexibility to direct interrupt sources to either the FIQ or IRQ internal interrupt exception
- 13 external interrupt pins.
  - — One high-priority maskable interrupt pin, **HPI#**.
  - — Four maskable Inputs, **S_INT[D:A]#/XINT[3:0]#**.
  - — Twelve GPIO Pins (when configured as Inputs), **GPIO[7:0]/XINT[15:8]#**
- Two internal programmable timer sources.
- Peripheral interrupt sources.

All interrupts are *level sensitive*: interrupt sources must keep asserting the interrupt signal until software causes the source to deassert it.

All interrupt sources are individually maskable with the ICUs Interrupt Control registers (

). Additionally, all interrupts may be quickly disabled by altering the F and I bits in the CPSR as specified in the *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E..

When software running on the 80331 is vectored to an Interrupt Service Routine (ISR), it reads the ICUs IRQ Interrupt Vector Register (IINTVEC) or FIQ Interrupt Vector register (FINTVEC) to quickly retrieve the address for the interrupt handler of the highest priority active interrupt source.

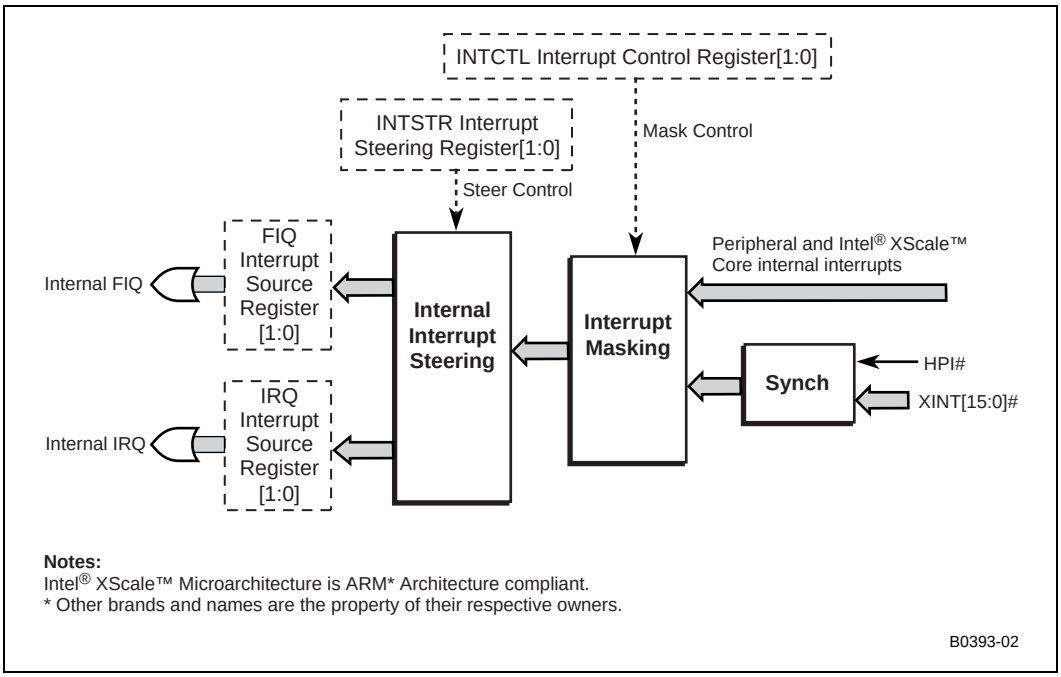
15.5.1 Programmer Model

Software has access to 15 registers in the ICU. These registers control, masking, prioritization, and vector generation for all interrupt sources.

15.5.1.1 Active Interrupt Source Control and Status

The INTCTL[1:0] register is used to enable or disable (mask) individual interrupts. As mentioned, masking of all interrupts may still be accomplished via the CPSR register in the core. INTSTR[1:0] are used to direct internal interrupts to either FIQ or IRQ. IINTSRC[1:0] and FINTSRC[1:0] are read-only registers that record all currently active and unmasked interrupt sources; the architecture for the interrupt source registers and FIQ/IRQ generation is illustrated in Figure 114.

Figure 114. Interrupt Controller Block Diagram (Active Interrupt Source Registers)



15.5.1.2 Prioritization and Vector Generation for Active Interrupt Sources

The INTPRI[3:0] registers reserve two bits for each source to assign one of four priority levels.

00_2 - High Priority
01_2 - Med/High Priority
10_2 - Med/Low Priority
11_2 - Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[1:0] or the IINTSRC[1:0] registers, the prioritizer selects a highest priority active source for each source register.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the highest priority active source is selected according to a fixed priority based on bit location. Highest order bit is first.

The INTBASE and INTSIZE registers are used to establish a contiguous Interrupt Service Routine (ISR) memory range for all of 64 possible sources. The architecture provides for an ISR ranging from 256 bytes to 64 Kbytes per source.

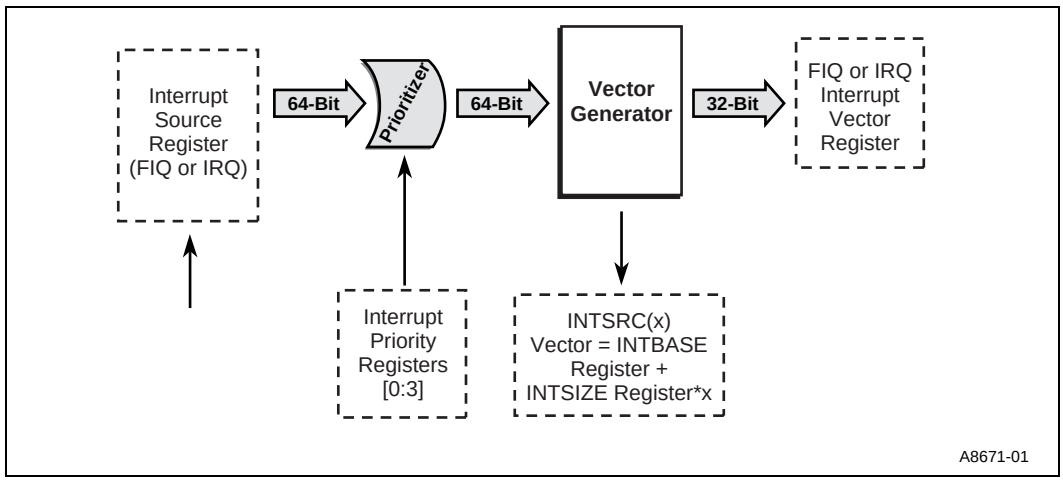
The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 64 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., IINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source)*63 with source 63 (i.e, IINTSRC1 bit 31).

Example 11. Determining the Location of the Interrupt Handler for Source 25

```
INTBASE = 0x81200000 ; 2 Mbyte Aligned Base Address
INTSIZE = 0xE ; 32 Kbytes per source (ISR Memory Range of 2 Mbytes)
ISR Address(25) = 0x81200000 + conv_hex(2^15*25) = 0x812C8000
```

Based on IINTSRC[1:0], FINTSRC[1:0], INTPRI[3:0], INTBASE, and INTSIZE, the interrupt controller generates the values provided by the IINTVEC and FINTVEC registers as illustrated in Figure 115. The IINTVEC and FINTVEC registers presents the vector for the active interrupt source with the highest priority to the IRQ and FIQ exception handlers, respectively.

Figure 115. Interrupt Controller Block Diagram (FIQ/IRQ Interrupt Vector Generation)



Note: The 80331 does not use all 64 possible sources.

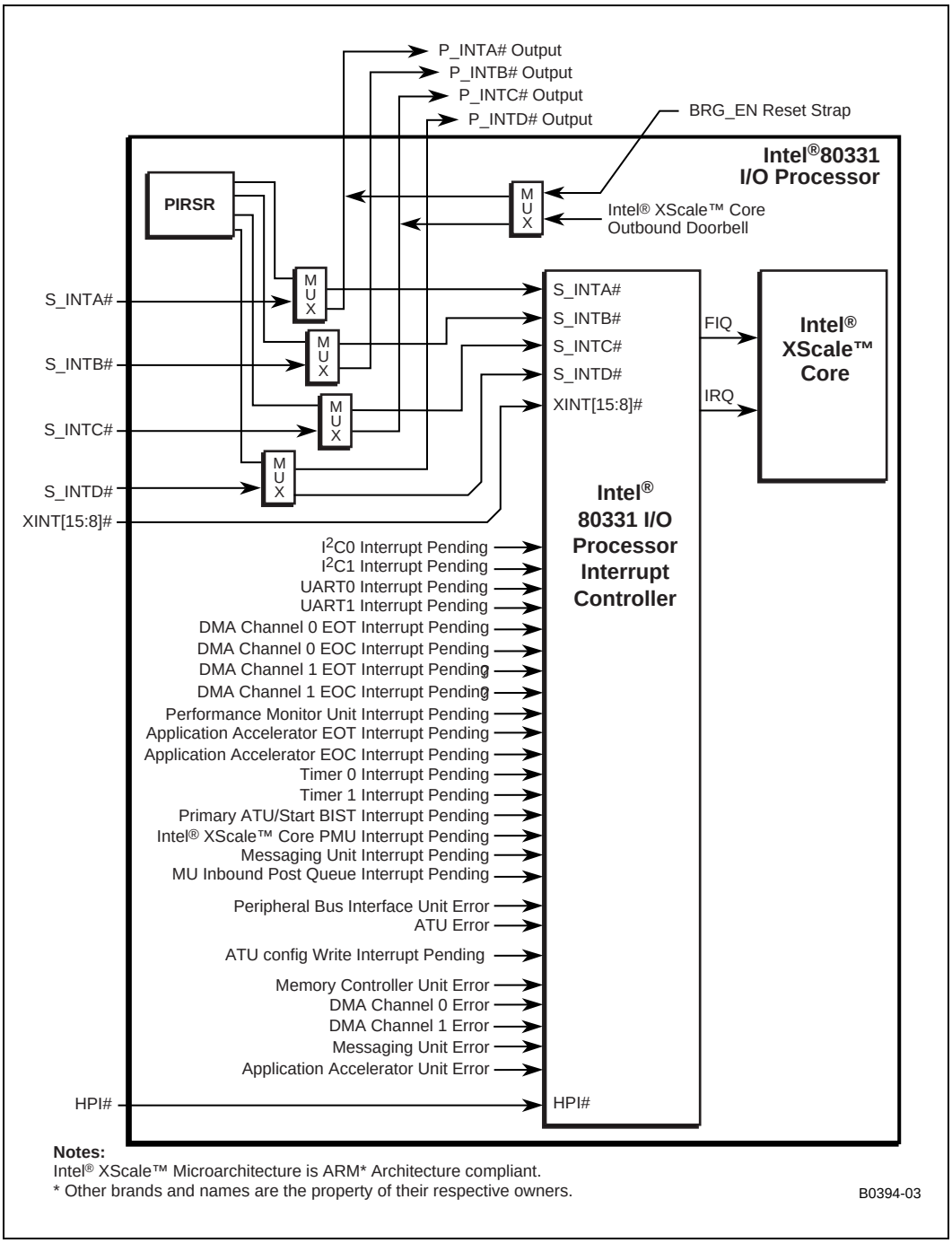
ICU registers reside in Coprocessor 6 (CP6). They may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The instruction *CRn* field denotes the accessed register number. The instruction *opcode_1*, *opcode_2*, and *CRm* fields should be zero. Most systems restricts access to CP6 to privileged processes. To control access to CP6, use Coprocessor Access Register.

An instruction that modifies an ICU register is guaranteed to take effect before the next instruction executes. For example, when an instruction masks an interrupt source, subsequent instructions execute in an environment in which the masked interrupt does not occur.

15.5.2 Operational Blocks

The ICU provides the connections to the Intel® XScale™ core. These connections are shown in Figure 116.

Figure 116. Intel® 80331 I/O Processor Interrupt Controller Connections



15.5.3 Intel® 80331 I/O Processor: Internal Peripheral Interrupt

The 80331 Interrupt Controller receives inputs from multiple internal interrupt sources. All pending interrupts required during normal operation of the various peripheral units are available in either the IINTSRC0, IINTSRC1, FINTSRC0 or FINTSRC1 registers depending on the value in INTSTR0 or INTSTR1. To provide the best latency for high performance event driven activities, the DMA channel and AAU interrupts are fully demultiplexed into the interrupt source registers for FIQ and IRQ so that software does not need to access these peripheral units to diagnose the exact source and cause of the interrupt. The IINTSRC[1:0] and the FINTSRC[1:0] registers also include pending interrupts that indicate that an error has occurred in one of the peripheral units. For the interrupts that indicate errors, more detail about the exact cause of the interrupt can be determined by reading the status register of the respective peripheral unit.

15.5.3.1 Normal Interrupt Sources

The 80331 Interrupt Controller receives normal interrupts from the two DMA channels, Performance Monitoring Unit, the I^2C Bus Interface Unit, the ATU, the Programmable Timers, the Messaging Unit, the Application Accelerator Unit (AAU) and the UARTs. The DMA/AAU channel interrupts for End of Transfer interrupt or End of Chain interrupt are demultiplexed into the interrupt controller. A Performance Monitoring Unit interrupt implies that at least one of the fourteen programmable event counters and/or the Global Time Stamp Counter has a pending interrupt condition.

A valid interrupt from any of these sources outputs a *level-sensitive* interrupt to the 80331 Interrupt Controller input. The corresponding IRQ or FIQ interrupt source register bit in the interrupt controller should remain active as long as the interrupt is pending in the peripheral unit. The appropriate interrupt source bit is cleared by clearing the source of the interrupt at the internal peripheral.

The normal interrupt sources which drive the inputs to the 80331 Interrupt Controller are detailed in Table 370.

Note: The UART and I^2C Bus Interface Unit interrupt sources are combined as a single interrupt, and include both normal and error conditions within the respective units.

Table 370. Normal Interrupt Sources (Sheet 1 of 2)

Unit	Interrupt Condition	Register
DMA Channel 0	End of Chain	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
DMA Channel 0	End of Transfer	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
DMA Channel 1	End of Chain	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
DMA Channel 1	End of Transfer	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
Application Accelerator	End of Chain	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
Application Accelerator	End of Transfer	Section 6.14.2, "Channel Status Register x - CSRx" on page 372
Core Performance Monitor	Counter Overflow	Section 13.4.5.3, "Performance Monitor Control Register -- PMNC" on page 655

Table 370. Normal Interrupt Sources (Sheet 2 of 2)

Unit	Interrupt Condition	Register
I^2C Bus Interface Unit 0	I^2C Status Register 0	Receive Buffer Full
		Transmit Buffer Empty
		Slave Address Detect (General Call Address Detect)
		STOP Detected
		Bus Error Detected
		Arbitration Lost Detected
I^2C Bus Interface Unit 1	I^2C Status Register 1	Receive Buffer Full
		Transmit Buffer Empty
		Slave Address Detect (General Call Address Detect)
		STOP Detected
		Bus Error Detected
		Arbitration Lost Detected
Messaging Unit	Inbound Interrupt Status Register	Index Register Interrupt
		Inbound Post Queue Interrupt
		Inbound Doorbell Interrupt
		Inbound Message 1 Interrupt
		Inbound Message 0 Interrupt
ATU	ATU Interrupt Status Register	ATU BIST Start
	Configure Register Write	Any of the ATU Configuration registers written by an inbound Configuration Write cycle. This includes dedicated interrupt status bits for configuration writes to the VPDAR, IABAR1 and IAUBAR1 registers.
Timer 0	Timer Mode Register 0	Timer 0 has decremented to 0 interrupt.
Timer 1	Timer Mode Register 1	Timer 1 has decremented to 0 interrupt.
UART Unit 0	UART 0 Interrupt ID Register	Received Line Status
		Received Data is Available
		Character Timeout Indications
		Transmit FIFO Data Request
		Autobaud Lock Indication
UART Unit 1	UART 1 Interrupt ID Register	Received Line Status
		Received Data is Available
		Character Timeout Indications
		Transmit FIFO Data Request
		Autobaud Lock Indication

15.5.3.2 Error Interrupt Sources

The 80331 Interrupt Controller receives error interrupts from the ATU, the Messaging Unit, the two DMA channels and the AAU. Each of these interrupts represent an error condition in the peripheral unit. Refer to the appropriate units for more details.

A valid interrupt from any of these sources outputs a *level-sensitive* interrupt to the 80331 Interrupt Controller input. The corresponding FIQ or IRQ interrupt source register bit in the interrupt controller should remain active as long as the interrupt is pending in the peripheral unit. The appropriate FIQ or IRQ interrupt source bit is cleared by clearing the source of the interrupt at the internal peripheral.

Table 371. Error Interrupt Sources

Unit	Register	Error Condition
ATU	ATU Interrupt Status Register	Initiated Split Completion Error Message
		Received Split Completion Error Message
		Power State Transition
		P_SERR# Asserted
		PCI Detected Parity Error
		IB Master Abort
		P_SERR# Detected
		PCI Master Abort
		PCI Target Abort (master)
		PCI Target Abort (target)
		PCI Master Parity Error
Messaging Unit	Inbound Interrupt Status Register	Outbound Free Queue Full Interrupt
		Error Doorbell Interrupt
DMA Channel 0	Channel Status Register 0	IB Master Abort
		PCI Master Abort
		PCI Target Abort (master)
		Unexpected Split Completion
DMA Channel 1	Channel Status Register 1	IB Master Abort
		PCI Master Abort
		PCI Target Abort (master)
		Unexpected Split Completion
Application Accelerator (AAU Error)	Accelerator Status Register	IB Master Abort
Memory Controller	Memory Controller Interrupt Status Register	ECC Error 0
		ECC Error 1
		ECC Error N
		Address Region Error
		IB Discard Timer Expired

The PCI Interrupt Routing Select Register and the Interrupt Source Register are described in Section 15.7.

15.5.4 High-Priority Interrupt (HPI#)

The **HPI#** pin generates an interrupt for implementation of critical interrupt routines.

15.5.5 Timer Interrupts

Each of the two timer units has an associated interrupt. Timer interrupts are connected directly to the 80331 interrupt controller and are posted in either the IINTSRC0 or FINTSRC0 registers. These interrupts are set up through the timer control registers described in Chapter 14, "Timers."

15.5.6 Software Interrupts

The application program may use the **SWI** instruction to request interrupt service.

15.6 Default Status

The interrupt logic is reset by the PCI reset signal or through software. Table 372 shows the power-up and reset values.

Table 372. Default Interrupt Routing and Status Values

Register	Default Value	Description
INTCTL0	0000 0000H	All interrupts 31:0 masked.
INTCTL1	0000 0000H	All interrupts 63:32 masked.
INTSTR0	0000 0000H	All interrupts 31:0 steered to IRQ.
INTSTR1	0000 0000H	All interrupts 63:32 steered to IRQ.
IINTSRC0	0000 0000H	All IRQ interrupts 31:0 inactive.
IINTSRC1	0000 0000H	All IRQ interrupts 63:32 inactive.
FINTSRC0	0000 0000H	All FIQ interrupts 31:0 inactive.
FINTSRC1	0000 0000H	All FIQ interrupts 63:32 inactive.
PIRSR	0000 0000H	Interrupts **S_INT[D:A]#**/**XINT[3:0]#** routed to the **P_INT[D:A]#** Outputs. Interrupts **XINT[7:4]#** disabled.

15.7 Interrupt Control Unit Registers

All Interrupt Controller registers are visible as 80331 memory mapped registers and can be accessed through the internal memory bus. Each is a 32-bit register and is memory-mapped in the Intel® XScale™ core memory space.

The PCI Interrupt Routing Select Register is accessible from the internal memory bus and also during PCI configuration cycles through the PCI configuration register space. See Chapter 3, "Address Translation Unit" for additional information regarding the PCI configuration cycles that access the PCI Interrupt Routing Select Register. The programmer's interface to the interrupt controller is through both coprocessor registers and memory-mapped control register. Table 373 describes these registers.

The coprocessor registers may be accessed/manipulated with the MCR, MRC, STC, and LDC instructions. The *CRn* field of the instruction denotes the register number to be accessed. The *opcode_1*, *opcode_2*, and *CRm* fields of the instruction should be zero. Most systems restricts access to CP6 to privileged processes. To control access to CP6, use the Coprocessor Access Register.

Table 373. Interrupt Controller Register Addresses

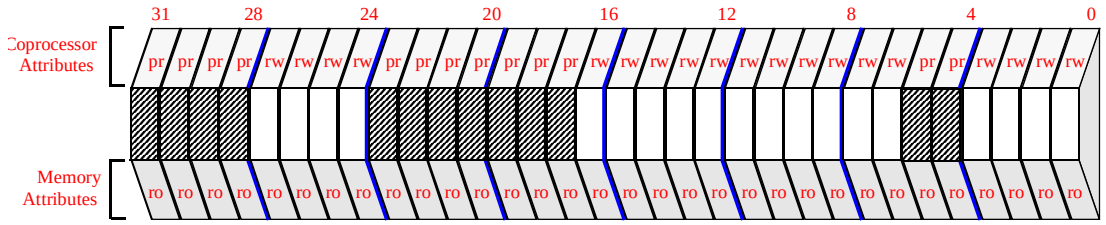
Register Name	Description	Coprocessor Register (CR_m= 0) or MMR Address
INTCTL0	Interrupt Control Register 0	CP6, Register 0 (Read/Write) FFFF E790H (Read-Only)
INTCTL1	Interrupt Control Register 1	CP6, Register 1 (Read/Write) FFFF E794H (Read-Only)
INTSTR0	Interrupt Steer Register 0	CP6, Register 2 (Read/Write) FFFF E798H (Read-Only)
INTSTR1	Interrupt Steer Register 1	CP6, Register 3 (Read/Write) FFFF E79CH (Read-Only)
IINTSRC0	IRQ Interrupt Source Register 0	CP6, Register 4 (Read-Only) FFFF E7A0H (Read-Only)
IINTSRC1	IRQ Interrupt Source Register 1	CP6, Register 5 (Read-Only) FFFF E7A4H (Read-Only)
FINTSRC0	FIQ Interrupt Source Register 0	CP6, Register 6 (Read-Only) FFFF E7A8H (Read-Only)
FINTSRC1	FIQ Interrupt Source Register 1	CP6, Register 7 (Read-Only) FFFF E7ACH (Read-Only)
IPR0	Interrupt Priority Register 0	CP6, Register 8 (Read/Write) FFFF E7B0H (Read-Only)
IPR1	Interrupt Priority Register 1	CP6, Register 9 (Read/Write) FFFF E7B4H (Read-Only)
IPR2	Interrupt Priority Register 2	CP6, Register 10 (Read/Write) FFFF E7B8H (Read-Only)
IPR3	Interrupt Priority Register 3	CP6, Register 11 (Read/Write) FFFF E7BCH (Read-Only)
INTBASE	Interrupt Base Register	CP6, Register 12 (Read/Write) FFFF E7C0H (Read-Only)
INTSIZE	Interrupt Size Register	CP6, Register 13 (Read/Write) FFFF E7C4H (Read-Only)
IINTVEC	IRQ Interrupt Vector Register	CP6, Register 14 (Read/Write) FFFF E7C8H (Read-Only)
FINTVEC	FIQ Interrupt Vector Register	CP6, Register 15 (Read/Write) FFFF E7CCH (Read-Only)
PIRSR	PCI Interrupt Routing Select Register	FFFF E1ECH

15.7.1 Interrupt Control Register 0 - INTCTL0

The Interrupt Control register 0 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Note: INTCTL0 register is read-only from Memory-Mapped Register Address space.

Table 374. Interrupt Control Register 0 - INTCTL0 (Sheet 1 of 2)



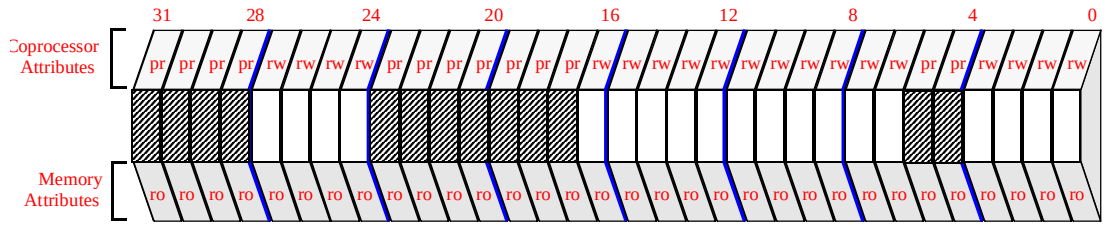
Intel® XScale™ Core Coprocessor address
CP6, Register 0
Intel® XScale™ Core Local Bus Address
FFFF E790H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0000_2	Preserved
27	0_2	**XINT3#** Interrupt Mask 0 = Masked 1 = Not Masked
26	0_2	**XINT2#** Interrupt Mask 0 = Masked 1 = Not Masked
25	0_2	**XINT1#** Interrupt Mask 0 = Masked 1 = Not Masked
24	0_2	**XINT0#** Interrupt Mask 0 = Masked 1 = Not Masked
23:17	$000\ 0000_2$	Preserved
16	0_2	Intel® XScale™ Core PMU Interrupt Mask 0 = Masked 1 = Not Masked
15	0_2	Peripheral Performance Monitor Interrupt Mask 0 = Masked 1 = Not Masked
14	0_2	ATU/Start BIST Interrupt Mask 0 = Masked 1 = Not Masked
13	0_2	Messaging Unit Inbound Post Queue Interrupt Mask 0 = Masked 1 = Not Masked

Table 374. Interrupt Control Register 0 - INTCTL0 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 0
Intel® XScale™ Core Local Bus Address
FFFF E790H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

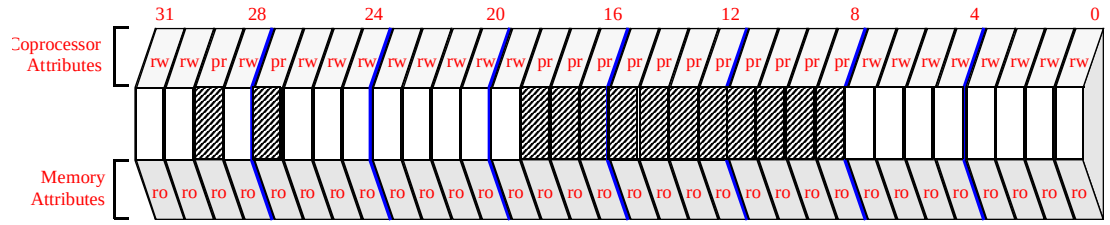
Bit	Default	Description
12	0_2	Messaging Unit Interrupt Mask 0 = Masked 1 = Not Masked
11	0_2	I^2C Bus Interface 1 Interrupt Mask 0 = Masked 1 = Not Masked
10	0_2	I^2C Bus Interface 0 Interrupt Mask 0 = Masked 1 = Not Masked
9	0_2	Timer 1 Interrupt Mask 0 = Masked 1 = Not Masked
8	0_2	Timer 0 Interrupt Mask 0 = Masked 1 = Not Masked
7	0_2	Application Accelerator End-Of-Chain Interrupt Mask 0 = Masked 1 = Not Masked
6	0_2	Application Accelerator End-Of-Transfer Interrupt Mask 0 = Masked 1 = Not Masked
5:4	00_2	Preserved
3	0_2	DMA Channel 1 End-Of-Chain Interrupt Mask 0 = Masked 1 = Not Masked
2	0_2	DMA Channel 1 End-Of-Transfer Interrupt Mask 0 = Masked 1 = Not Masked
1	0_2	DMA Channel 0 End-Of-Chain Interrupt Mask 0 = Masked 1 = Not Masked
0	0_2	DMA Channel 0 End-Of-Transfer Interrupt Mask 0 = Masked 1 = Not Masked

15.7.2 Interrupt Control Register 1 - INTCTL1

The Interrupt Control register 1 is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts are masked.

Note: INTCTL1 register is read-only from Memory-Mapped Register Address space.

Table 375. Interrupt Control Register 1 - INTCTL1 (Sheet 1 of 2)



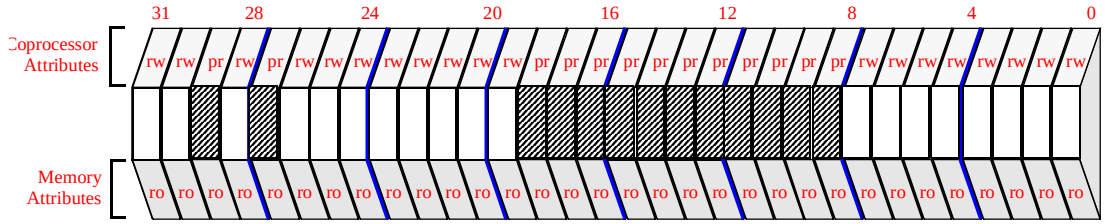
Intel® XScale™ Core Coprocessor address
CP6, Register 1
Intel® XScale™ Core Local Bus Address
FFFF E794H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	HPI# Interrupt Mask 0 = Masked 1 = Not Masked
30	0_2	Messaging Unit Error Interrupt Mask 0 = Masked 1 = Not Masked
29	0_2	Preserved
28	0_2	Application Accelerator Unit Error Interrupt Mask 0 = Masked 1 = Not Masked
27	0_2	Preserved
26	0_2	DMA Channel 1 Error Interrupt Mask 0 = Masked 1 = Not Masked
25	0_2	DMA Channel 0 Error Interrupt Mask 0 = Masked 1 = Not Masked
24	0_2	Memory Controller Unit Error Interrupt Mask 0 = Masked 1 = Not Masked
23	0_2	ATU Error Interrupt Mask 0 = Masked 1 = Not Masked
22	0_2	ATU Configuration Register Write Interrupt 0 = Masked 1 = Not Masked

Table 375. Interrupt Control Register 1 - INTCTL1 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 1
Intel® XScale™ Core Local Bus Address
FFFF E794H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
21	0_2	Peripheral Bus Unit Error Interrupt Mask 0 = Masked 1 = Not Masked
20	0_2	UART 1 Interrupt Mask 0 = Masked 1 = Not Masked
19	0_2	UART 0 Interrupt Mask 0 = Masked 1 = Not Masked
18:08	000H	Preserved
7	0_2	XINT15# Interrupt Mask 0 = Masked 1 = Not Masked
6	0_2	XINT14# Interrupt Mask 0 = Masked 1 = Not Masked
5	0_2	XINT13# Interrupt Mask 0 = Masked 1 = Not Masked
4	0_2	XINT12# Interrupt Mask 0 = Masked 1 = Not Masked
3	0_2	XINT11# Interrupt Mask 0 = Masked 1 = Not Masked
2	0_2	XINT10# Interrupt Mask 0 = Masked 1 = Not Masked
1	0_2	XINT9# Interrupt Mask 0 = Masked 1 = Not Masked
0	0_2	XINT8# Interrupt Mask 0 = Masked 1 = Not Masked

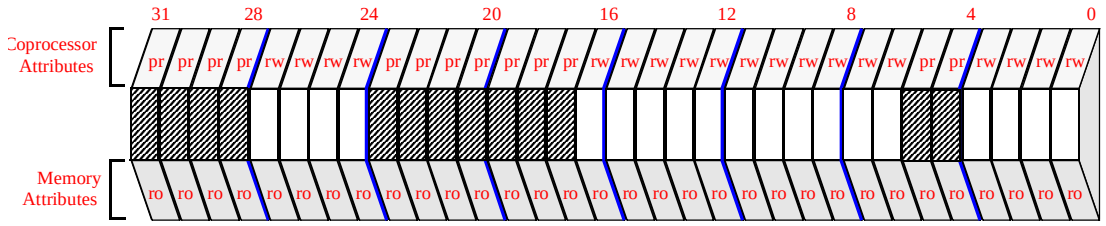
15.7.3 Interrupt Steering Register 0 - INTSTR0

The Interrupt Steering Register 0 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL0 register, this register steers the interrupt to an internal interrupt exception.

Note: INTSTR0 register is read-only from Memory-Mapped Register Address space.

Table 376. Interrupt Steering Register 0 - INTSTR0 (Sheet 1 of 2)



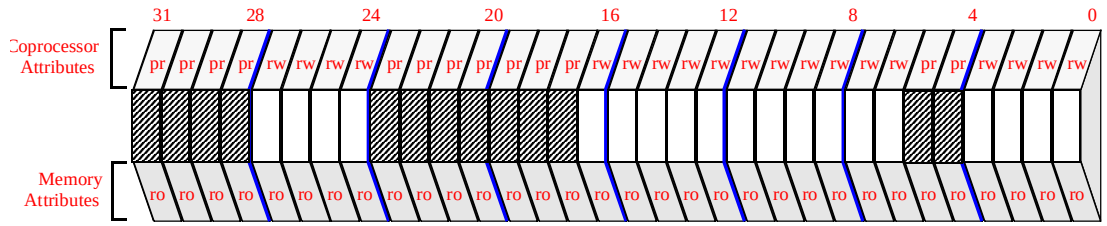
Intel® XScale™ Core Coprocessor address
CP6, Register 2
Intel® XScale™ Core Local Bus Address
FFFF E798H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0000_2	Preserved
27	0_2	**XINT3#** Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
26	0_2	**XINT2#** Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
25	0_2	**XINT1#** Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
24	0_2	**XINT0#** Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
23:17	$000\ 0000_2$	Preserved
16	0_2	Intel® XScale™ Core PMU Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
15	0_2	Peripheral Performance Monitor Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
14	0_2	ATU/Start BIST Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
13	0_2	Messaging Unit Inbound Post Queue Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ

Table 376. Interrupt Steering Register 0 - INTSTR0 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 2
Intel® XScale™ Core Local Bus Address
FFFF E798H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
12	0_2	Messaging Unit Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
11	0_2	I^2C Bus Interface 1 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
10	0_2	I^2C Bus Interface 0 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
9	0_2	Timer 1 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
8	0_2	Timer 0 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
7	0_2	Application Accelerator End-Of-Chain Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
6	0_2	Application Accelerator End-Of-Transfer Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
5:4	00_2	Preserved
3	0_2	DMA Channel 1 End-Of-Chain Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
2	0_2	DMA Channel 1 End-Of-Transfer Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
1	0_2	DMA Channel 0 End-Of-Chain Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
0	0_2	DMA Channel 0 End-Of-Transfer Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ

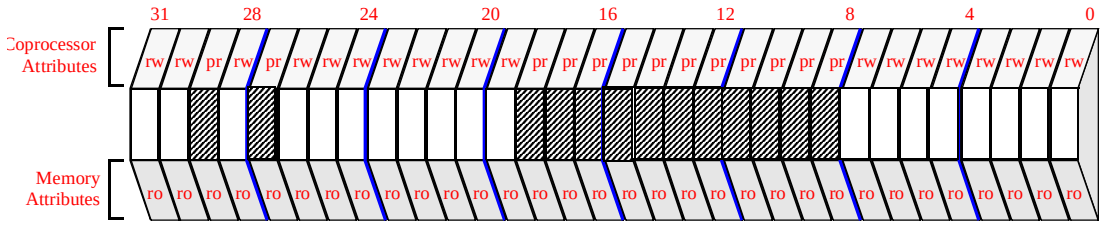
15.7.4 Interrupt Steering Register 1 - INTSTR1

The Interrupt Steering Register 1 allows system designers to direct any of 32 internal or external interrupt sources to either one of the two internal interrupt exceptions, FIQ and IRQ.

When an interrupt is enabled with the INTCTL1 register, this register steers the interrupt to an internal interrupt exception.

Note: INTSTR1 register is read-only from Memory-Mapped Register Address space.

Table 377. Interrupt Steering Register 1 - INTSTR1 (Sheet 1 of 2)

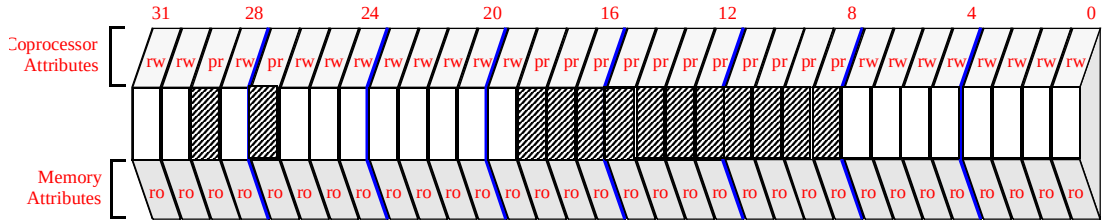


Intel® XScale™ Core Coprocessor address
CP6, Register 3
Intel® XScale™ Core Local Bus Address
FFFF E79CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	HPI# Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
30	0_2	Messaging Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
29	0_2	Preserved
28	0_2	Application Accelerator Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
27	0_2	Preserved
26	0_2	DMA Channel 1 Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
25	0_2	DMA Channel 0 Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
24	0_2	Memory Controller Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
23	0_2	ATU Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
22	0_2	ATU Configuration Register Write Interrupt 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ

Table 377. Interrupt Steering Register 1 - INTSTR1 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 3
Intel® XScale™ Core Local Bus Address
FFFF E79CH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
21	0_2	Peripheral Bus Interface Unit Error Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
20	0_2	UART 1 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
19	0_2	UART 0 Interrupt Steering 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
18:08	000H	Preserved
7	0_2	XINT15# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
6	0_2	XINT14# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
5	0_2	XINT13# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
4	0_2	XINT12# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
3	0_2	XINT11# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
2	0_2	XINT10# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
1	0_2	XINT9# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ
0	0_2	XINT8# Interrupt Mask 0 = Interrupt Directed to Internal IRQ 1 = Interrupt Directed to Internal FIQ

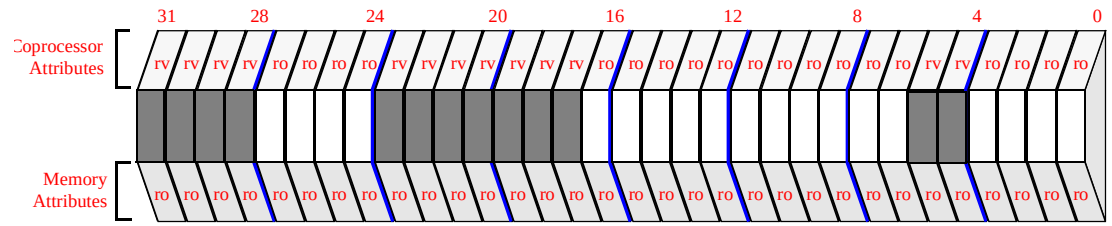
15.7.5 IRQ Interrupt Source Register 0 - IINTSRC0

The IRQ Interrupt Source register is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts that are steered to the internal IRQ exception are unmasked by the INTCTL0 register and active. The INTSTR0 control register is used to steer individual interrupts to the IRQ exception.

The IINTSRC0 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an IRQ interrupt.

Note: IINTSRC0 register is read-only from Memory-Mapped Register Address space.

Table 378. IRQ Interrupt Source Register 0 - IINTSRC0 (Sheet 1 of 2)



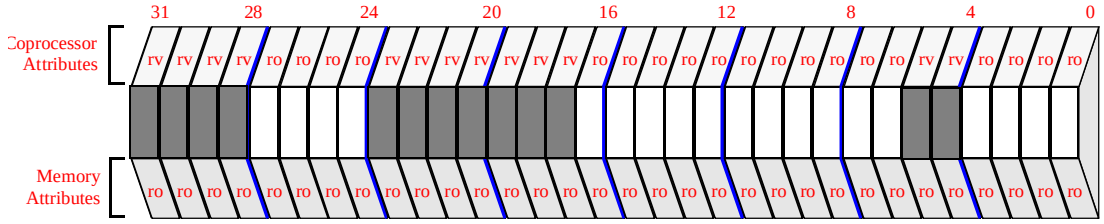
Intel® XScale™ Core Coprocessor address
CP6, Register 4
Intel® XScale™ Core Local Bus Address
FFFF E7A0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0000_2	Reserved
27	0_2	**XINT3#** Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
26	0_2	**XINT2#** Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
25	0_2	**XINT1#** Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
24	0_2	**XINT0#** Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
23:17	$00\ 0000_2$	Reserved
16	0_2	Intel® XScale™ Core PMU Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
15	0_2	Peripheral Performance Monitor Interrupt - when set, at least one of the programmable event counters and/or the Global Time Stamp Counter contains an overflow condition. Application software identifies the counter by reading the Event Monitoring Interrupt Status register (EMISR). 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
14	0_2	ATU/Start BIST Interrupt - when set, the host processor has set the start BIST request in the ATUBISTR register. 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0

Table 378. IRQ Interrupt Source Register 0 - IINTSRC0 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 4
Intel® XScale™ Core Local Bus Address
FFFF E7A0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
13	0_2	Messaging Unit Inbound Post Queue Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
12	0_2	Messaging Unit Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
11	0_2	I^2C Bus Interface 1 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
10	0_2	I^2C Bus Interface 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
9	0_2	Timer 1 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
8	0_2	Timer 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
7	0_2	Application Accelerator End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
6	0_2	Application Accelerator End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
5:4	00_2	Reserved
3	0_2	DMA Channel 1 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
2	0_2	DMA Channel 1 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
1	0_2	DMA Channel 0 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0
0	0_2	DMA Channel 0 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL0 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL0

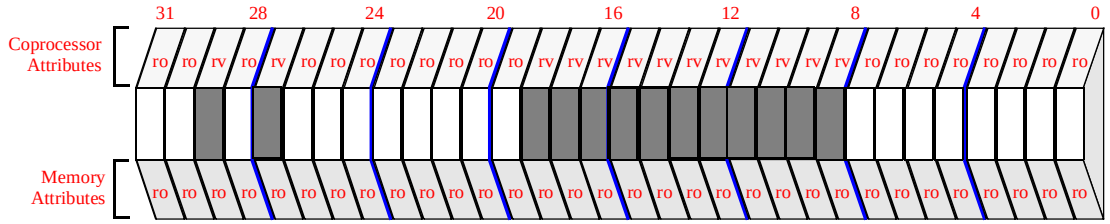
15.7.6 IRQ Interrupt Source Register 1 - IINTSRC1

The IRQ Interrupt Source register is a 32-bit Coprocessor 6 control register used to specify which of 32 interrupts that are steered to the internal IRQ exception are unmasked by the INTCTL1 register and active. The INTSTR1 control register is used to steer individual interrupts to the IRQ exception.

The IINTSRC1 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an IRQ interrupt.

Note: IINTSRC1 register is read-only from Memory-Mapped Register Address space.

Table 379. IRQ Interrupt Source Register 1 - IINTSRC1 (Sheet 1 of 2)



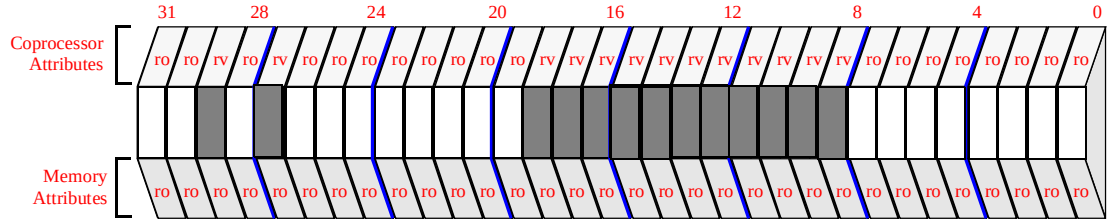
Intel® XScale™ Core Coprocessor address
CP6, Register 5
Intel® XScale™ Core Local Bus Address
FFFF E7A4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	HPI# Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
30	0_2	Messaging Unit Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
29	0_2	Reserved
28	0_2	Application Accelerator Unit Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
27	0_2	Reserved
26	0_2	DMA Channel 1 Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
25	0_2	DMA Channel 0 Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
24	0_2	Memory Controller Unit Error Interrupt - when set, an error condition exists within the MCU. The bit indicates one of the following conditions: • A single-bit correctable or uncorrectable ECC error. • A multi-bit correctable or uncorrectable ECC error. 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
23	0_2	ATU Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1

Table 379. IRQ Interrupt Source Register 1 - IINTSRC1 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 5
Intel® XScale™ Core Local Bus Address
FFFF E7A4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
22	0_2	ATU Configuration Register Write Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
21	0_2	Peripheral Bus Interface Unit Error Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
20	0_2	UART 1 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
19	0_2	UART 0 Interrupt 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
18:08	0_2	Reserved
7	0_2	XINT15# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
6	0_2	XINT14# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
5	0_2	XINT13# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
4	0_2	XINT12# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
3	0_2	XINT11# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
2	0_2	XINT10# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
1	0_2	XINT9# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1
0	0_2	XINT8# Interrupt Mask 0 = Not Interrupting or Not steered to internal IRQ exception or masked by INTCTL1 1 = Interrupting and steered to internal IRQ exception and unmasked by INTCTL1

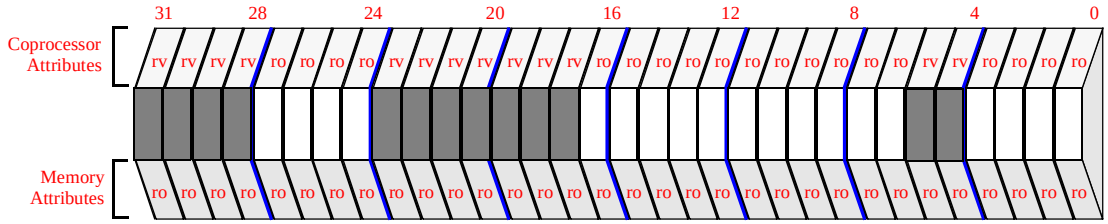
15.7.7 FIQ Interrupt Source Register 0 - FINTSRC0

The FIQ Interrupt Source register 0 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL0 register and active. The INTSTR0 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC0 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Note: FINTSRC0 register is read-only from Memory-Mapped Register Address space.

Table 380. FIQ Interrupt Source Register 0 - FINTSRC0 (Sheet 1 of 2)



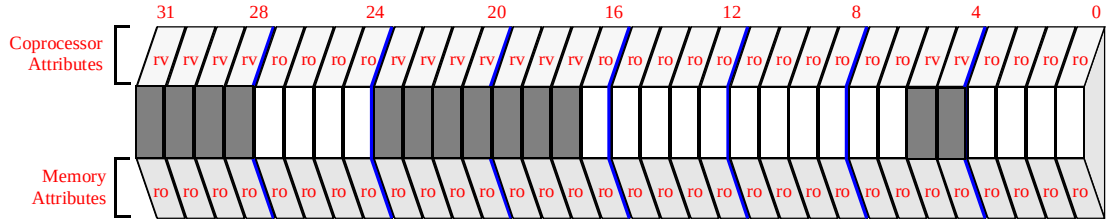
Intel® XScale™ Core Coprocessor address
CP6, Register 6
Intel® XScale™ Core Local Bus Address
FFFF E7A8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:28	0000_2	Reserved
27	0_2	**XINT3#** Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
26	0_2	**XINT2#** Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
25	0_2	**XINT1#** Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
24	0_2	**XINT0#** Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
23:17	$000\ 0000_2$	Reserved
16	0_2	Intel® XScale™ Core PMU Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
15	0_2	Peripheral Performance Monitor Interrupt - when set, at least one of the programmable event counters and/or the Global Time Stamp Counter contains an overflow condition. Application software identifies the counter by reading the Event Monitoring Interrupt Status register (EMISR). 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
14	0_2	ATU/Start BIST Interrupt - when set, the host processor has set the start BIST request in the ATUBISTR register. 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0

Table 380. FIQ Interrupt Source Register 0 - FINTSRC0 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 6
Intel® XScale™ Core Local Bus Address
FFFF E7A8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
13	0_2	Messaging Unit Inbound Post Queue Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
12	0_2	Messaging Unit Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
11	0_2	I^2C Bus Interface 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
10	0_2	I^2C Bus Interface 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
9	0_2	Timer 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
8	0_2	Timer 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
7	0_2	Application Accelerator End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
6	0_2	Application Accelerator End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
5:4	00_2	Reserved
3	0_2	DMA Channel 1 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
2	0_2	DMA Channel 1 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
1	0_2	DMA Channel 0 End-Of-Chain Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0
0	0_2	DMA Channel 0 End-Of-Transfer Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL0 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL0

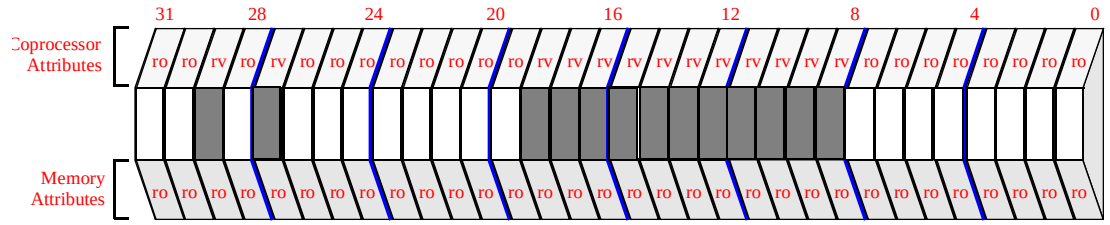
15.7.8 FIQ Interrupt Source Register 1 - FINTSRC1

The FIQ Interrupt Source register 1 is a 32-bit Coprocessor 6 control register used to specify which interrupts that are steered to the internal FIQ exception are unmasked by the INTCTL1 register and active. The INTSTR1 control register is used to steer individual interrupts to the FIQ exception.

The FINTSRC1 register may be used by an Interrupt Service Routine (ISR) to determine quickly the source of an FIQ interrupt.

Note: FINTSRC1 register is read-only from Memory-Mapped Register Address space.

Table 381. FIQ Interrupt Source Register 1 - FINTSRC1 (Sheet 1 of 2)



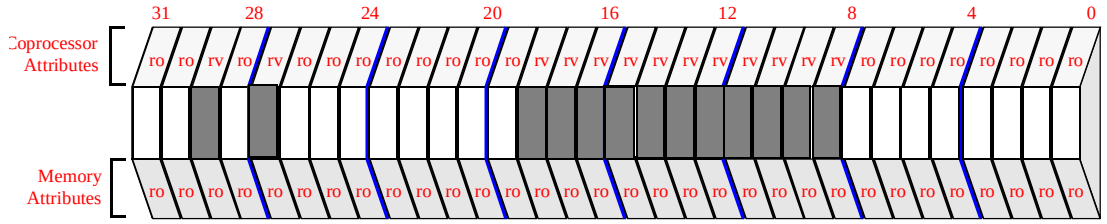
Intel® XScale™ Core Coprocessor address
CP6, Register 7
Intel® XScale™ Core Local Bus Address
FFFF E7ACH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31	0_2	HPI# Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
30	0_2	Messaging Unit Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
29	0_2	Reserved
28	0_2	Application Accelerator Unit Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
27	0_2	Reserved
26	0_2	DMA Channel 1 Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
25	0_2	DMA Channel 0 Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
24	0_2	Memory Controller Unit Error Interrupt - when set, an error condition exists within the MCU. The bit indicates one of the following conditions: • A single-bit correctable or uncorrectable ECC error. • A multi-bit correctable or uncorrectable ECC error. 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
23	0_2	ATU Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1

Table 381. FIQ Interrupt Source Register 1 - FINTSRC1 (Sheet 2 of 2)



Intel® XScale™ Core Coprocessor address
CP6, Register 7
Intel® XScale™ Core Local Bus Address
FFFF E7ACH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
22	0_2	ATU Configuration Register Write Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
21	0_2	Peripheral Bus Interface Unit Error Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
20	0_2	UART 1 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
19	0_2	UART 0 Interrupt 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
18:08	000H	Reserved
7	0_2	XINT15# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
6	0_2	XINT14# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
5	0_2	XINT13# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
4	0_2	XINT12# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
3	0_2	XINT11# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
2	0_2	XINT10# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
1	0_2	XINT9# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1
0	0_2	XINT8# Interrupt Mask 0 = Not Interrupting or Not steered to internal FIQ exception or masked by INTCTL1 1 = Interrupting and steered to internal FIQ exception and unmasked by INTCTL1

15.7.9 Interrupt Priority Register 0 - IPR0

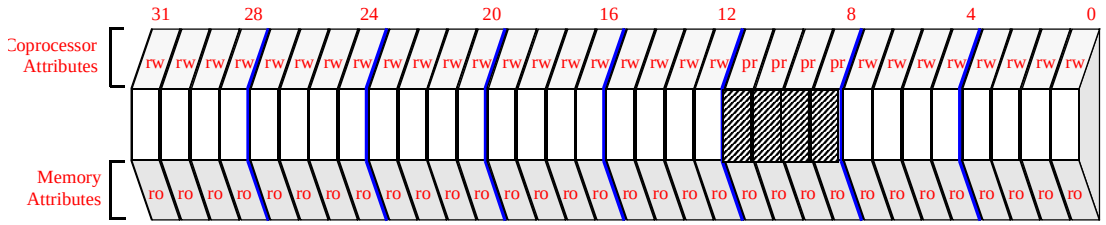
The Interrupt Priority Register 0 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 15 down to 0. The IPR0 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[1:0] registers:

00_2 - High Priority
01_2 - Med/High Priority
10_2 - Med/Low Priority
11_2 - Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[1:0] or the IINTSRC[1:0] registers, the highest priority vectors pending for either FIQ or IRQ is presented in the FINTVEC or IINTVEC, respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 382. Interrupt Priority Register 0 - IPR0



Intel® XScale™ Core Coprocessor address
CP6, Register 8
Intel® XScale™ Core Local Bus Address
FFFF E7B0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:30	00_2	Peripheral Performance Monitor Interrupt Priority
29:28	00_2	ATU/Start BIST Interrupt Priority
27:26	00_2	Messaging Unit Inbound Post Queue Interrupt Priority
25:24	00_2	Messaging Unit Interrupt Priority
23:22	00_2	I^2C Bus Interface 1 Interrupt Priority
21:20	00_2	I^2C Bus Interface 0 Interrupt Priority
19:18	00_2	Timer 1 Interrupt Priority
17:16	00_2	Timer 0 Interrupt Priority
15:14	00_2	Application Accelerator End-Of-Chain Interrupt Priority
13:12	00_2	Application Accelerator End-Of-Transfer Interrupt Priority
11:08	00_2	Preserved
07:06	00_2	DMA Channel 1 End-Of-Chain Interrupt Priority
05:04	00_2	DMA Channel 1 End-Of-Transfer Interrupt Priority
03:02	00_2	DMA Channel 0 End-Of-Chain Interrupt Priority
01:00	00_2	DMA Channel 0 End-Of-Transfer Interrupt Priority

15.7.10 Interrupt Priority Register 1 - IPR1

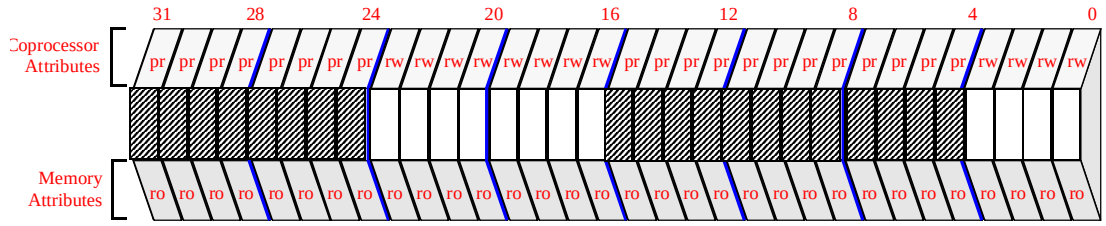
The Interrupt Priority Register 1 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 31 down to 15. The IPR1 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[1:0] registers:

00_2 - High Priority
01_2 - Med/High Priority
10_2 - Med/Low Priority
11_2 - Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[1:0] or the IINTSRC[1:0] registers, the highest priority vectors pending for either FIQ or IRQ is presented in the FINTVEC or IINTVEC, respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 383. Interrupt Priority Register 180331 - IPR1



Intel® XScale™ Core Coprocessor address
CP6, Register 9
Intel® XScale™ Core Local Bus Address
FFFF E7B4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:24	00H	Preserved
23:22	00_2	XINT3# Interrupt Priority
21:20	00_2	XINT2# Interrupt Priority
19:18	00_2	XINT1# Interrupt Priority
17:16	00_2	XINT0# Interrupt Priority
15:02	0000H	Preserved
01:00	00_2	Intel® XScale™ Core PMU Interrupt Priority

15.7.11 Interrupt Priority Register 2 - IPR2

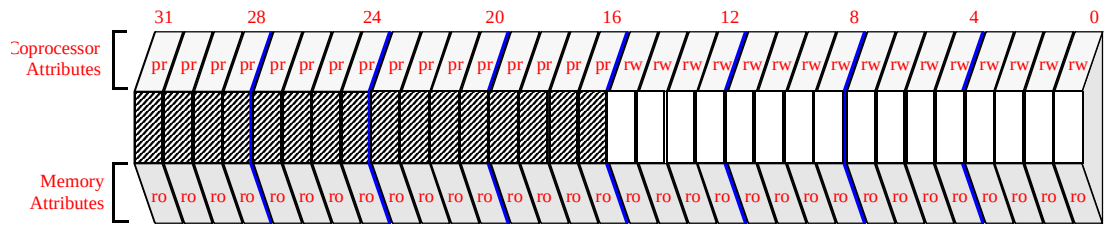
The Interrupt Priority Register 2 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 47 down to 32. The IPR2 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[1:0] registers:

00_2 - High Priority
01_2 - Med/High Priority
10_2 - Med/Low Priority
11_2 - Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[1:0] or the IINTSRC[1:0] registers, the highest priority vectors pending for either FIQ or IRQ is presented in the FINTVEC or IINTVEC, respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 384. Interrupt Priority Register 2 - IPR2



Intel® XScale™ Core Coprocessor address
CP6, Register 10
Intel® XScale™ Core Local Bus Address
FFFF E7B8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:16	0000H	Preserved
15:14	00_2	XINT15# Interrupt Priority
13:12	00_2	XINT14# Interrupt Priority
11:10	00_2	XINT13# Interrupt Priority
09:08	00_2	XINT12# Interrupt Priority
07:06	00_2	XINT11# Interrupt Priority
05:04	00_2	XINT10# Interrupt Priority
03:02	00_2	XINT9# Interrupt Priority
01:00	00_2	XINT8# Interrupt Priority

15.7.12 Interrupt Priority Register 3 - IPR3

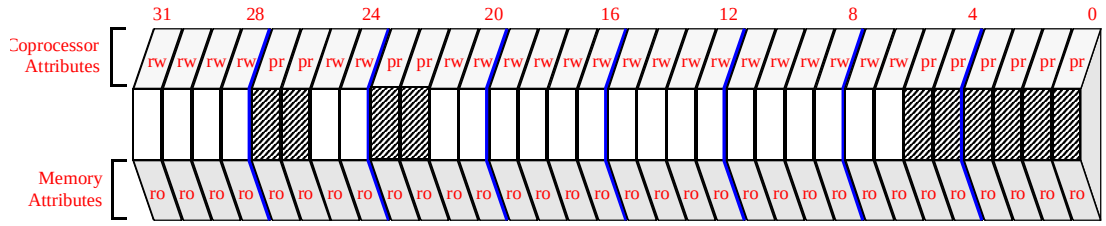
The Interrupt Priority Register 3 is a 32-bit Coprocessor 6 control register used to assign a priority level to interrupt sources 63 down to 48. The IPR3 control register is used to assign one of 4 priority levels to each interrupt source independent of the INTSTR[1:0] registers:

00_2 - High Priority
01_2 - Med/High Priority
10_2 - Med/Low Priority
11_2 - Low Priority

When interrupt vector generation is enabled and there are multiple requests pending either in the FINTSRC[1:0] or the IINTSRC[1:0] registers, the highest priority vectors pending for either FIQ or IRQ is presented in the FINTVEC or IINTVEC, respectively.

Note: When multiple interrupts at the same priority level are pending for either FIQ or IRQ, the vector is selected according to a fixed priority based on bit location. Highest order bit is first.

Table 385. Interrupt Priority Register 3 - IPR3



Intel® XScale™ Core Coprocessor address
CP6, Register 11
Intel® XScale™ Core Local Bus Address
FFFF E7BCH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:30	00_2	HPI# Interrupt Priority
29:28	00_2	Messaging Unit Error Interrupt Priority
27:26	00_2	Preserved
25:24	00_2	Application Accelerator Unit Error Interrupt Priority
23:22	00_2	Preserved
21:20	00_2	DMA Channel 1 Error Interrupt Priority
19:18	00_2	DMA Channel 0 Error Interrupt Priority
17:16	00_2	Memory Controller Unit Error Interrupt Priority
15:14	00_2	ATU Error Interrupt Priority
13:12	00_2	ATU Configuration Register Write Interrupt Priority
11:10	00_2	Peripheral Bus Interface Unit Error Interrupt Priority
9:8	00_2	UART 1 Interrupt Priority
7:6	00_2	UART 0 Interrupt Priority
5:0	$00\ 0000_2$	Preserved

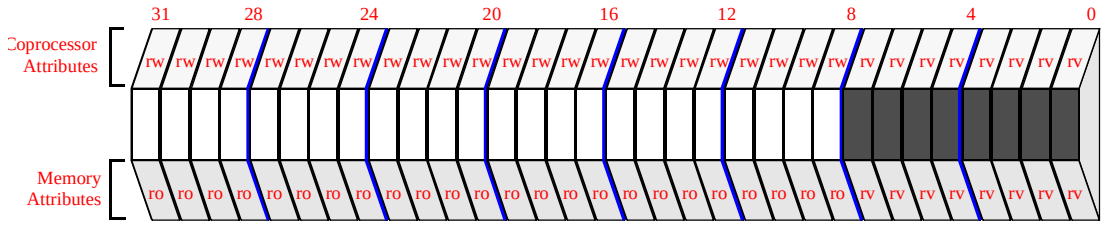
15.7.13 Interrupt Base Register - INTBASE

The Interrupt Base Register indicates the beginning of the Interrupt Service Routine (ISR) memory range that contains the interrupt service routines for up to 64 sources. The starting address must be on a boundary equal to the granularity of the ISR memory range as specified by the INTSIZE register.

For instance, the upper 24 bits are used for a 256 byte range, upper 16 bits for a 64 Kbyte range.

Note: INTBASE register is read-only from Memory-Mapped Register Address space.

Table 386. Interrupt Base Register - INTBASE



Intel® XScale™ Core Coprocessor address
CP6, Register 12
Intel® XScale™ Core Local Bus Address
FFFF E7C0H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:08	0	Interrupt Base These bits define the upper 24 bits of the base address for the ISR memory range.
07:00	0	Reserved

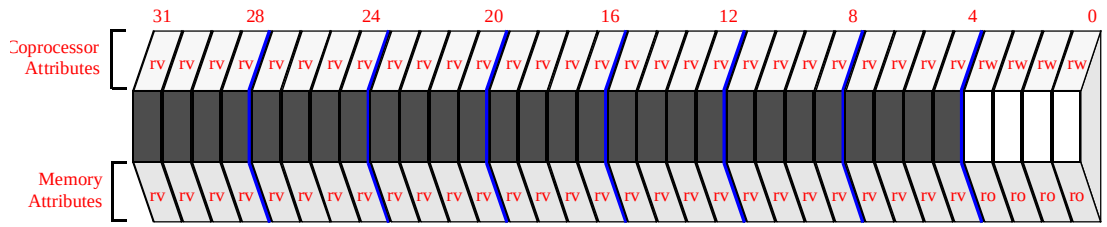
15.7.14 Interrupt Size Register - INTSIZE

The Interrupt Size Register indicates the size of the Interrupt Service Routine (ISR) memory range that contains the interrupt service routines for up to 64 sources. The INTSIZE register can allocate from 4 bytes to 64 Kbytes of memory address space for the ISR per source. This means that the INTSIZE register can allocate a total ISR memory space that ranges in size from 256 bytes to 4 Mbytes.

Along with the starting address defined in the INTBASE register, the INTSIZE register fully specifies the ISR memory range.

Note: INTSIZE register is read-only from Memory-Mapped Register Address space.

Table 387. Interrupt Size Register - INTSIZE



Intel® XScale™ Core Coprocessor address
CP6, Register 13
Intel® XScale™ Core Local Bus Address
FFFF E7C4H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:04	0	Reserved
03:00	0	ISR Memory Range Size These bits define the size of the ISR memory range: **INTSIZE** / **ISR Range Size** / **ISR Size (per Source)** 0 (Disabled) 1 256 bytes 4 bytes 2 512 bytes 8 bytes 3 1 Kbytes 16 bytes 4 2 Kbytes 32 bytes 5 4 Kbytes 64 bytes 6 8 Kbytes 128 bytes 7 16 Kbytes 256 bytes 8 32 Kbytes 512 bytes 9 64 Kbytes 1 Kbytes 10 128 Kbytes 2 Kbytes 11 256 Kbytes 4 Kbytes 12 512 Kbytes 8 Kbytes 13 1 Mbytes 16 Kbytes 14 2 Mbytes 32 Kbytes 15 4 Mbytes 64 Kbytes

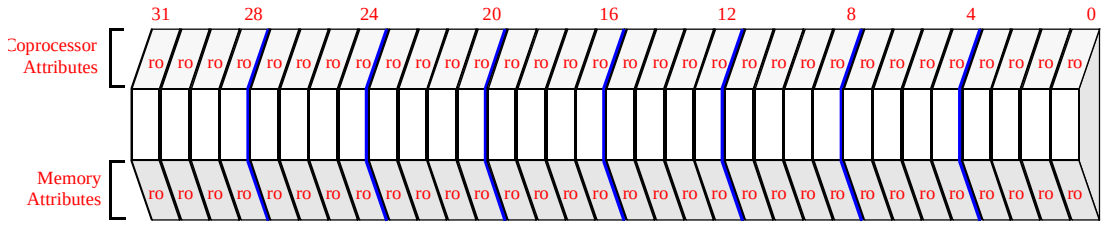
15.7.15 IRQ Interrupt Vector Register - IINTVEC

The IRQ Interrupt Vector Register is a 32-bit Coprocessor 6 control register. Following an IRQ exception, the IRQ interrupt service routine reads the 32-bit vector to the ISR for the active IRQ source with the highest priority.

The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 64 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., IINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source)*63 with source 63 (i.e, IINTSRC1 bit 31).

Before returning to User Mode from Interrupt Mode, the software reads the IINTVEC register and process any lower priority IRQ sources that are active. When there are no longer any active IRQ sources, a read from the IINTVEC register returns FFFF FFFFH.

Table 388. IRQ Interrupt Vector Register- IINTVEC



Intel® XScale™ Core Coprocessor address
CP6, Register 14
Intel® XScale™ Core Local Bus Address
FFFF E7C8H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0_2	IRQ Interrupt Vector -- Vector to the highest priority active IRQ source. This register reads FFFF FFFFH when there are no active IRQ sources.

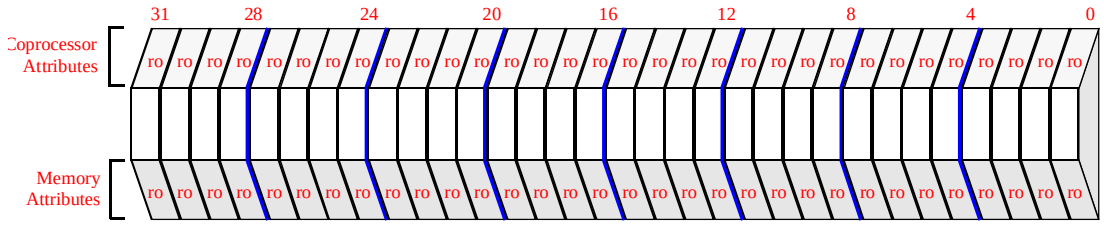
15.7.16 FIQ Interrupt Vector Register - FINTVEC

The FIQ Interrupt Vector Register is a 32-bit Coprocessor 6 control register. Following an FIQ exception, the FIQ interrupt service routine reads the 32-bit vector to the ISR for the active FIQ source with the highest priority.

The actual vector value is a function of the INTBASE and the INTSIZE registers and is based on a fixed order of all 64 possible interrupt sources. The vectors begin at INTBASE with source 0 (i.e., FINTSRC0 bit 0), and end at INTBASE + INTSIZE (per source)*63 with source 63 (i.e, FINTSRC1 bit 31).

Before returning to User Mode from Interrupt Mode, the software reads the FINTVEC register and process any lower priority FIQ sources that are active. When there are no longer any active FIQ sources, a read from the FINTVEC register returns FFFF FFFFH.

Table 389. FIQ Interrupt Vector Register- FINTVEC



Intel® XScale™ Core Coprocessor address
CP6, Register 15
Intel® XScale™ Core Local Bus Address
FFFF E7CCH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:00	0_2	FIQ Interrupt Vector -- Vector to the highest priority active FIQ source. This register reads FFFF FFFFH when there are no active FIQ sources.

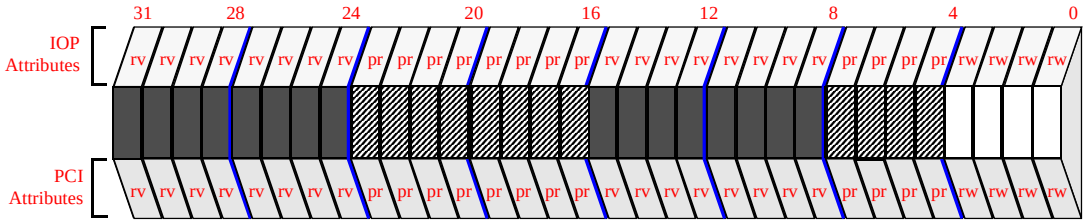
15.7.17 PCI Interrupt Routing Select Register - PIRSR

The PCI Interrupt Routing Select Register (PIRSR) determines the routing of the external input pins. The input pins consist of four external interrupt inputs which are routed to either PCI interrupts or Intel® XScale™ core interrupts. The external interrupt pins are defined as "level sensitive," asserted low. The assertion and deassertion of the interrupt pins are synchronous to the PCI or processor clock.

All external interrupt inputs, **S_INT[D:A]#XINT[3:0]#**, are required to be "level sensitive," asserted low. The interrupt assertion is cleared at the source by the interrupt handler.

Table 390 shows the bit definitions for programming the PCI Interrupt Routing Select Register.

Table 390. PCI Interrupt Routing Select Register- PIRSR



PCI Configuration Offset
ECH - EFH

Intel® XScale™ Core Local Bus Address
FFFF E1ECH

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA= Not Accessible

Bit	Default	Description
31:24	000 0000_2	Reserved
23:16	00H	Preserved
15:8	00H	Reserved
7:4	0H	Preserved
3	0_2	**XINT3#** Select Bit 0 = Interrupt routed to **P_INTD#** pin 1 = Interrupt routed to Intel® XScale™ core interrupt controller input (**XINT3#**)
2	0_2	**XINT2#** Select Bit 0 = Interrupt routed to **P_INTC#** pin 1 = Interrupt routed to Intel® XScale™ core interrupt controller input (**XINT2#**)
1	0_2	**XINT1#** Select Bit 0 = Interrupt routed to **P_INTB#** pin 1 = Interrupt routed to Intel® XScale™ core interrupt controller input (**XINT1#**)
0	0_2	**XINT0#** Select Bit 0 = Interrupt routed to **P_INTA#** pin 1 = Interrupt routed to Intel® XScale™ core interrupt controller input (**XINT0#**)

This Page Intentionally Left Blank

General Purpose I/O Unit 16

This chapter describes the Intel® 80331 I/O processor (80331) General Purpose I/O Unit. The operation modes, setup, external memory interface, and implementation of the General Purpose I/Os (GPIOs) are described in this chapter.

16.1 General Purpose Input Output Support

Eight pins are provided as General Purpose Input Output (GPIO) pins. The twelve pins are **GPIO[7:0]**. These pins can be used by the Intel® XScale™ core to control or monitor external devices in the I/O subsystem.

The interface for the two serial UART ports are multiplexed on to **XINT[15:8]#/GPIO[7:0]**.

Warning: To avoid serial port integrity problems, the user needs to ensure that the GPIO Output Data Register - GPOD bits associated with a UART port is cleared prior to setting the enable bit for that serial UART port. The user prepares to enable the serial UART ports by clearing GPOD bits associated with the respective shared GPIO pins. Refer to Section 16.1.4, "GPIO Pin Multiplexing" for the details on shared GPIO pins.

When a UART port is enabled through the UxIER Register bit 6 ("UART x Interrupt Enable Register" on page 596), the GPIO functionality described in the following sections is not available on the associated pins. UART ports are disabled following system reset.

16.1.1 General Purpose Inputs

The current state of the twelve GPIO pins can be read in Section 16.2.2, "GPIO Input Data Register - GPID" on page 723).

Note: When configured as general purpose inputs, the twelve GPIO pins can be used as up to 12 additional external interrupt inputs dedicated to the Intel® XScale™ core. This feature is available on a per pin basis simply by programming the INTCTL[1:0] registers.

16.1.2 General Purpose Outputs

The output function of the GPIO pins is controlled by two registers, as stated in Section 16.2.3, "GPIO Output Data Register - GPOD" on page 724) and Section 16.2.1, "GPIO Output Enable Register - GPOE" on page 722).

The output enables are mapped on a per bit basis to each of the data bits in the GPIO Output Data Register. When a bit of the GPIO Output Enable Register is cleared, the corresponding data bit value in the GPIO Output Data Register is actively driven on the appropriate GPIO pin.

16.1.3 Reset Initialization of General Purpose Input Output Function

The GPIO Input Data Register is initialized to the state of **GPIO[7:0]** upon assertion of **P_RST#**.

The GPIO Output Data Register is initialized to all zeros upon assertion of **P_RST#**.

The GPIO Output Enable Register is initialized to FFH upon assertion of **P_RST#**. This means that **GPIO[7:0]** initializes as inputs.

The **GPIO[7:0]** pins is tristated during **P_RST#** assertion.

16.1.4 GPIO Pin Multiplexing

The GPIO pins are multiplexed with the UART ports as specified in. The selection between GPIO and UART function for these multiplexed pins is controlled by the UART Unit Enable bit (UUE bit 6) of the Section 11.4.3, "UART x Interrupt Enable Register" on page 596.

Table 391. GPIO Pin Multiplexing

GPIO Pin	UART 0 Shared Port	UART 0 Port Signal	Default
GPIO[0]	UART0	U0_RXD	GPIO[0] - Input
GPIO[1]	UART0	U0_TXD	GPIO[1] - Input
GPIO[2]	UART0	U0_CTS#	GPIO[2] - Input
GPIO[3]	UART0	U0_RTS#	GPIO[3] - Input
GPIO Pin	**UART 1 Shared Port**	**UART 1 Port Signal**	**Default**
GPIO[4]	UART1	U1_RXD	GPIO[4] - Input
GPIO[5]	UART1	U1_TXD	GPIO[5] - Input
GPIO[6]	UART1	U1_CTS#	GPIO[6] - Input
GPIO[7]	UART1	U1_RTS#	GPIO[7] - Input

16.2 Register Definitions

All GPIO registers are visible as 80331 memory mapped registers and can be accessed through the internal memory bus. Each is a 32-bit register and is memory-mapped in the Intel® XScale™ core memory space.

The programmer interface to the General Purpose I/O interface is through memory-mapped control registers. Table 392 describes these registers.

Table 392. General Purpose I/O Registers Addresses

Register Name	Description	MMR Address
GPOE	GPIO Output Enable Register	FFFF F780H
GPID	GPIO Input Data Register	FFFF F784H
GPOD	GPIO Output Data Register	FFFF F788H

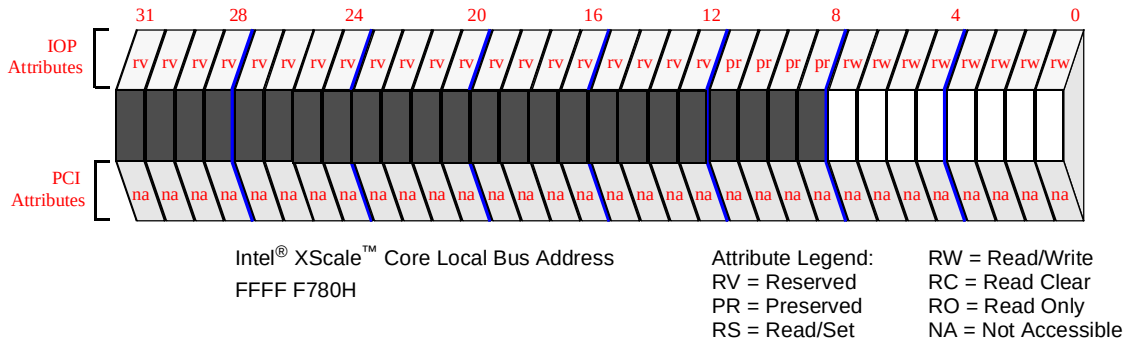
16.2.1 GPIO Output Enable Register - GPOE

The GPIO Output Enable Register enables on a per pin basis the output value contained in the GPIO Output Data Register onto the appropriate pin.

The GPIO Output Enable Register is initialized to FFH such that all of **GPIO[7:0]** are inputs.

In order to enable a particular GPIO pin to operate as an output following the deassertion of **P_RST#**, the user needs to write a '0' into the appropriate GPOE bit.

Table 393. GPIO Output Enable Register - GPOE

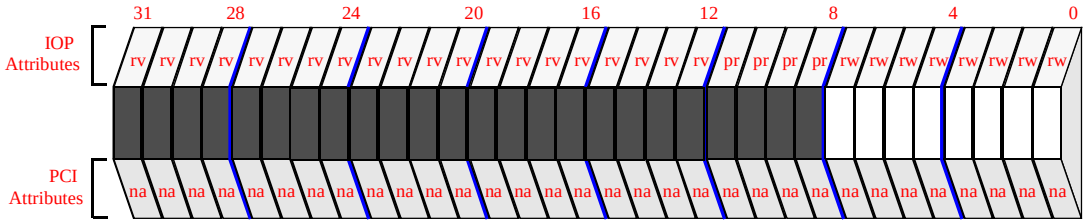


Bit	Default	Description
31:12	0	Reserved.
11:08	0	Preserved.
07	1_2	GPIO7 Output Enable -- When clear, bit 7 of the GPIO Output Data Register is enabled onto the **GPIO[7]** pin.
06	1_2	GPIO6 Output Enable -- When clear, bit 6 of the GPIO Output Data Register is enabled onto the **GPIO[6]** pin.
05	1_2	GPIO5 Output Enable -- When clear, bit 5 of the GPIO Output Data Register is enabled onto the **GPIO[5]** pin.
04	1_2	GPIO4 Output Enable -- When clear, bit 4 of the GPIO Output Data Register is enabled onto the **GPIO[4]** pin.
03	1_2	GPIO3 Output Enable -- When clear, bit 3 of the GPIO Output Data Register is enabled onto the **GPIO[3]** pin.
02	1_2	GPIO2 Output Enable -- When clear, bit 2 of the GPIO Output Data Register is enabled onto the **GPIO[2]** pin.
01	1_2	GPIO1 Output Enable -- When clear, bit 1 of the GPIO Output Data Register is enabled onto the **GPIO[1]** pin.
00	1_2	GPIO0 Output Enable -- When clear, bit 0 of the GPIO Output Data Register is enabled onto the **GPIO[0]** pin.

16.2.2 GPIO Input Data Register - GPID

The GPIO Input Data Register reflects the state of the appropriate **IRQ** bus pin following the deassertion of **P_RST#**.

Table 394. GPIO Input Data Register - GPID



Intel® XScale™ Core Local Bus Address
FFFF F784H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

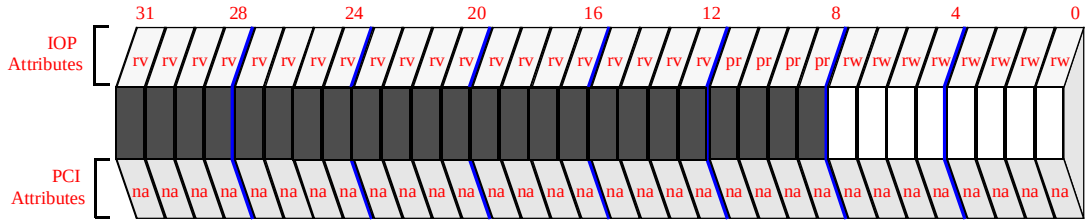
RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	0	Reserved.
11:08	0	Preserved.
07	**GPIO[7]** during **P_RST#** rising edge	GPIO7 Input Data -- This bit reflects the state of the **GPIO[7]** pin.
06	**GPIO[6]** during **P_RST#** rising edge	GPIO6 Input Data -- This bit reflects the state of the **GPIO[6]** pin.
05	**GPIO[5]** during **P_RST#** rising edge	GPIO5 Input Data -- This bit reflects the state of the **GPIO[5]** pin.
04	**GPIO[4]** during **P_RST#** rising edge	GPIO4 Input Data -- This bit reflects the state of the **GPIO[4]** pin.
03	**GPIO[3]** during **P_RST#** rising edge	GPIO3 Input Data -- This bit reflects the state of the **GPIO[3]** pin.
02	**GPIO[2]** during **P_RST#** rising edge	GPIO2 Input Data -- This bit reflects the state of the **GPIO[2]** pin.
01	**GPIO[1]** during **P_RST#** rising edge	GPIO1 Input Data -- This bit reflects the state of the **GPIO[1]** pin.
00	**GPIO[0]** during **P_RST#** rising edge	GPIO0 Input Data -- This bit reflects the state of the **GPIO[0]** pin.

16.2.3 GPIO Output Data Register - GPOD

The GPIO Output Data Register is driven on a per bit basis on the appropriate **IRQ** bus pin following the deassertion of **P_RST#** when the corresponding bit in the GPOE register is cleared.

Table 395. Output Data Register - GPOD



Intel® XScale™ Core Local Bus Address
FFFF F788H

Attribute Legend:
RV = Reserved
PR = Preserved
RS = Read/Set

RW = Read/Write
RC = Read Clear
RO = Read Only
NA = Not Accessible

Bit	Default	Description
31:12	0	Reserved.
11:08	0	Preserved.
07	0_2	GPIO7 Output Data -- This bit value is driven on the **GPIO[7]** pin when bit 7 of the GPOE register is cleared.
06	0_2	GPIO6 Output Data -- This bit value is driven on the **GPIO[6]** pin when bit 6 of the GPOE register is cleared.
05	0_2	GPIO5 Output Data -- This bit value is driven on the **GPIO[5]** pin when bit 5 of the GPOE register is cleared.
04	0_2	GPIO4 Output Data -- This bit value is driven on the **GPIO[4]** pin when bit 4 of the GPOE register is cleared.
03	0_2	GPIO3 Output Data -- This bit value is driven on the **GPIO[3]** pin when bit 3 of the GPOE register is cleared.
02	0_2	GPIO2 Output Data -- This bit value is driven on the **GPIO[2]** pin when bit 2 of the GPOE register is cleared.
01	0_2	GPIO1 Output Data -- This bit value is driven on the **GPIO[1]** pin when bit 1 of the GPOE register is cleared.
00	0_2	GPIO0 Output Data -- This bit value is driven on the **GPIO[0]** pin when bit 0 of the GPOE register is cleared.

Peripheral Registers 17

This chapter summarizes the registers for the integrated peripherals (MMR). Each register is defined in detail in the corresponding unit chapter.

17.1 Overview

The Peripheral Registers of the Intel® 80331 I/O processor (80331) can be accessed via three methods: the PCI Configuration Register interface, Peripheral Memory-Mapped Register interface and the high performance Intel® XScale™ core (ARM* architecture compliant) Coprocessor Register interface.

The PCI Configuration Register interface is supported by the PCI interface and PCI Configuration Cycle transaction type. The 80331 does not support access to the PCI Configuration Register Interface by the core processor.

The Peripheral Memory-Mapped Register (PMMR) interface gives software the ability to read and modify internal control registers. 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 Intel® XScale™ core.

The Intel® XScale™ core Coprocessor Register (CCR) interface gives software the ability to read and modify internal control registers at a very low latency as compared to the PMMR interface.

These registers are specific to the 80331 only. They support the:

- DMA Controller Unit
- Memory Controller
- Intel® XScale™ Core Bus Interface Unit
- Interrupt Controller Unit
- Messaging Unit
- Intel® 80331 I/O Processor Arbitration Unit
- UARTs
- Address Translation Unit
- I^2C Bus Interface Units
- Application Accelerator Unit
- General Purpose I/O Unit
- Peripheral Bus Interface Unit
- I/O Level Control

Each of these peripherals fully describe the independent functionality of the registers, control and usage.

Control and status registers for the Intel® XScale™ core use the CCR interface. Accesses to coprocessor registers do not generate external bus cycles. See the Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual (Order Number: 273411) for a full description of the usage of the coprocessor register space in the Intel® XScale™ core. For completeness, these registers are included in the tables of registers for the CCR interface. These registers can only be accessed using the Intel® XScale™ core coprocessor instructions.

The PMMR interface provides full accessibility from the ATU, and the Intel® XScale™ core. Addresses FFFF E000H through FFFF FFFFH are allocated to the PMMR interface.

17.2 Accessing the PCI Configuration Registers

The 80331 PCI configuration space consists of the PCI-to-PCI bridge and the ATU. Table 396 summarizes the PCI programming model of the I/O processor. The configuration registers of the bridge are not memory mapped, and are only accessible via PCI congruent space from the PCI interface.

The registers of the ATU are both memory mapped and mapped to PCI configuration space. The

Table 396. Intel® 80331 I/O Processor PCI Programming Model

Bus	Device	Function	Unit
Primary PCI Bus #	0	0	PCI-to-PCI Bridge
Secondary PCI Bus #	0	0	Address Translation Unit

ATU registers are listed in the Peripheral Memory Mapped Register section. The configuration registers of the bridge are summarized in the PCI Configuration Register section, including address and access type.

For PCI Configuration Read transactions, the PMMR shall return a value of zero for registers declared as "reserved". For PCI Configuration Write transactions, the PMMR shall discard the data. For all other types of access, reading or writing a register declared as "reserved" is undefined.

17.3 Accessing Peripheral Memory-Mapped Registers

The PMMR interface is a slave device connected to the 80331 internal bus. This interface accepts data transactions which appear on the internal bus from the ATU and the Intel® XScale™ core.

The PMMR interface allows these devices to perform read, write, or read-modify-write transactions. The specific actions taken when modifying any value in the PMMR space is independently defined within each chapter which describes the functionality of the register.

Note: The PMMR interface does not support multi-DWORD burst accesses from any internal bus master.

All PMMR transactions are allowed from the Intel® XScale™ core operating in either user or supervisor mode. In addition, the PMMR does not provide any access exception to the Intel® XScale™ core.

17.4 Accessing Peripheral Registers Using the Core Coprocessor Register Interface

Registers may be accessed/manipulated through the CCR interface with the MCR, MRC, STC, and LDC instructions. The *CRn* field of the instruction denotes the register number to be accessed. The *opcode_1*,and *opcode_2* fields of the instruction should be zero. The *CRm* field must be set to 0 for the Interrupt Controller Unit and to 1 for the Programmable Timers. Most systems restricts access to coprocessor registers to privileged processes. To control access to a coprocessor register, use the Coprocessor Access Register as described in the *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E..

17.5 Architecturally Reserved Memory Space

The 80331 provides 4 Gbytes of address space. Portions of this address space is architecturally reserved and users are restricted as to their function. Figure 117 shows the reserved address space.

Addresses 0000 0000H through 0000 001FH are reserved for the exception vectors of the Intel® XScale™ Core.

Addresses FFFF 0000H through FFFF 001FH are reserved for the relocated exception vectors of the Intel® XScale™ Core.[1]

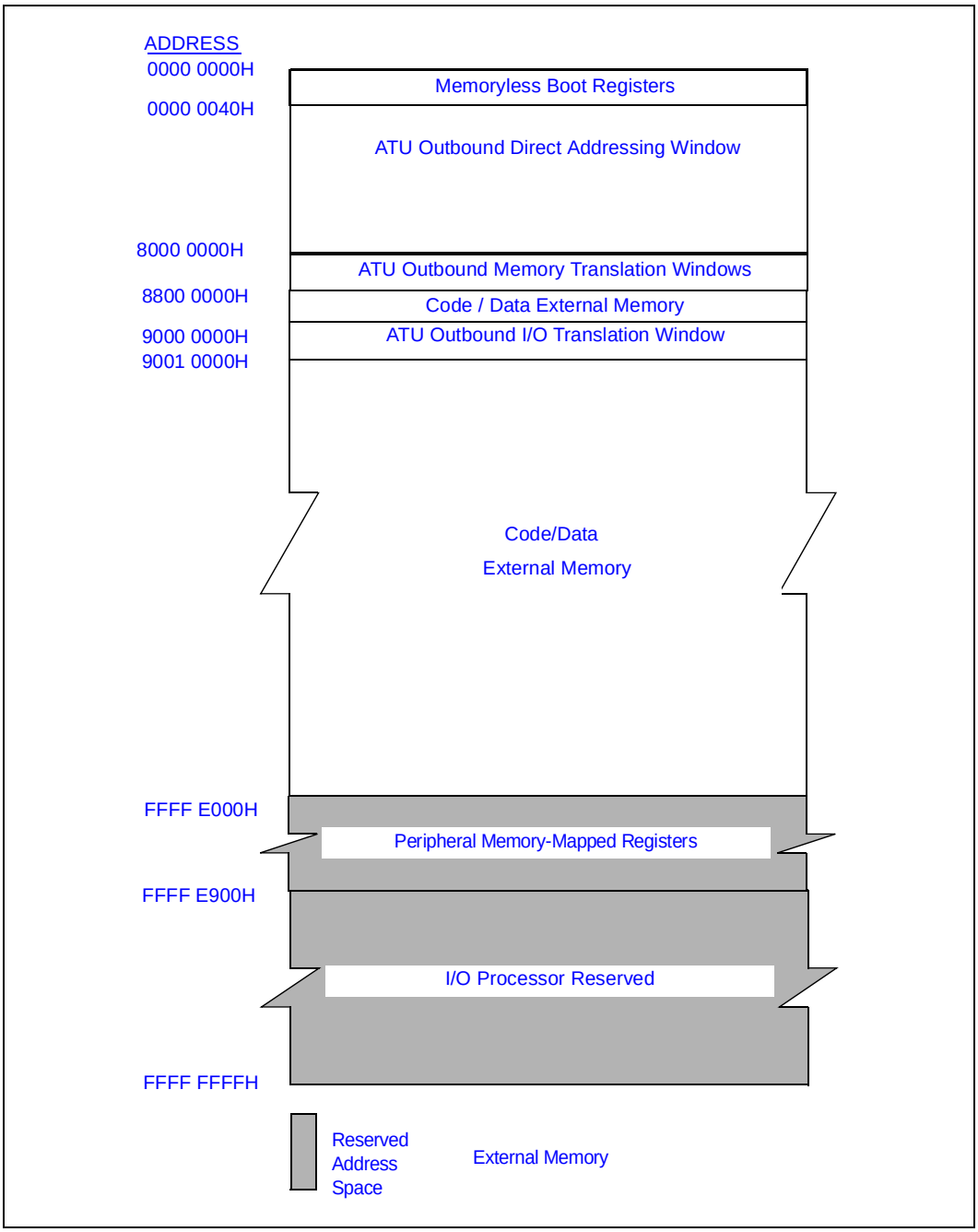
Addresses FFFF E000H through FFFF E8FFH are allocated to the PMMR interface. These registers are reserved for 80331 use and should not be written by the system designer.

Addresses FFFF E900H through FFFF FFFFH are allocated for future expansion of the PMMR interface. These registers are reserved for 80331 use and should not be written by the system designer.

1. By enabling the Exception Vector Relocation mode (bit 13, CP15, Register 1), the Exception Vectors (except Reset Vector at 0000 0000H) can be relocated to be based at FFFF 0000H rather than 0000 0000H. (i.e., FIQ Vector located at FFFF 001CH)

Figure 117 shows the Intel® XScale™ Core address space and addresses available to the applications.

Figure 117. Memory Address Space



For a full description of the address space for the Intel® XScale™ core Reset and Exception Vectors, refer to the *ARM Architecture Reference Manual*.

17.6 PCI Configuration Register Address Space

The PCI Configuration space is accessible via configuration transactions from the Primary PCI bus interface. The Intel® XScale core of 80331 does not have access to the bridge configuration space.

Table 397. PCI Configuration Register Locations (Sheet 1 of 2)

PCI Device	Register Description (Name)	Register Size in Bits	Register Access	PCI Configuration Space Register Number
PCI-to-PCI Bridge	Identifiers Register	32	PCI Standard Configuration Header Device 0 Function 0	00H
	Primary Command Register	16		01H
	Primary Status Register	16		01H
	Revision ID Register	8		02H
	Class Code Register	24		02H
	Cacheline Size Register	8		03H
	Latency Timer Register	8		03H
	Header Type Register	8		03H
	Reserved	8		03H
	Reserved	32		04H
	Reserved	32		05H
	Bus Number Register	24		06H
	Secondary Latency Timer Register	8		06H
	I/O Base and Limit Register	16		07H
	Secondary Status Register	16		07H
	Memory Base and Limit Register	32		08H
	Prefetchable Memory Base and Limit Register	32		09H
	Prefetchable Memory Base Upper 32-bits Register	32		0AH
	Prefetchable Memory Limit Upper 32-bits Register	32		0BH
	I/O Base and Limit Upper 16-bits Register	32		0CH
	Capabilities Pointer Register	8		0DH
	Reserved	24		0DH
	Reserved	32		0EH
	Interrupt Information Register	16		0FH
	Bridge Control Register	16		0FH

Table 397. PCI Configuration Register Locations (Sheet 2 of 2)

PCI Device	Register Description (Name)	Register Size in Bits	Register Access	PCI Configuration Space Register Number
PCI-to-PCI Bridge	Diagnostic Control Register	8	PCI Extended Configuration Header Device 0 Function 0	10H
	Secondary Arbiter Control/Status Register	16		10H
	Bridge Control Register 0	8		10H
	Bridge Control Register 1	16		11H
	Bridge Control Register 2	16		11H
	Bridge Status Register	16		12H
	Reserved	32		13H
	Bridge Multi-Transaction Timer Register	16		14H
	Read Prefetch Policy Register	16		14H
	P_SERR# Assertion Control Register	16		15H
	Pre-boot Status Register	8		15H
	Reserved	8		15H
	Reserved	32		16H
	Secondary IDSEL Select Register	16		17H
	Secondary Decode Enable Register	16		17H
	Primary Bridge Interrupt Status Register	32		18H
	Secondary Bridge Interrupt Status Register	32		19H
	Reserved	x		20H through 36H
	Power Management Capabilities Identifier Register	8		37H
	Power Management Next Item Pointer Register	8		37H
	Power Management Capabilities Register	16		37H
	Power Management Control/Status Register	16		38H
	Power Management Extended Bridge Control/Status Register	8		38H
	Reserved	8		38H
	Reserved	32		39H
	Reserved	32		3AH
	Reserved	32		3BH
	PCI-X Capabilities Identifier Register	8		3CH
	PCI-X Capabilities Next Item Pointer Register	8		3CH
	PCI-X Secondary Status Register	16		3CH
	PCI-X Bridge Status Register	32		3DH
	PCI-X Upstream Split Transaction Control Register	32		3EH
	PCI-X Downstream Split Transaction Control Register	32		3FH

17.7 Peripheral Memory-Mapped Register Address Space

The PMMR address space is divided to support the integrated peripherals on the 80331. Table 400 shows all of the 80331 integrated peripheral memory-mapped registers and their internal bus addresses.

Table 398. Intel® XScale™ Core Local Addresses Assigned to Integrated Peripherals

Integrated Peripheral	Internal Address Block
Address Translation Unit	FFFF E100H through FFFF E1FFH
Reserved	FFFF E200H through FFFF E2FFH
Messaging Unit	FFFF E300H through FFFF E3FFH
DMA Controller	FFFF E400H through FFFF E4FFH
Memory Controller	FFFF E500H through FFFF E5FFH
Intel® XScale™ Core Bus Interface Unit	FFFF E600H through FFFF E67FH
Peripheral Bus Interface Unit	FFFF E680H through FFFF E6FFH
Peripheral Performance Monitoring Unit	FFFF E700H through FFFF E77FH
Interrupt Controller Unit	FFFF E780H through FFFF E7EFH
Internal Arbitration Unit	FFFF E7F0H through FFFF E7FFH
Application Accelerator Unit	FFFF E800H through FFFF E8FFH
Reserved	FFFF E900H through FFFF F4FFH
DDR I/O Control	FFFF F500H through FFFF F5FFH
Reserved	FFFF F600H through FFFF F67FH
I^2C Bus Interface Units	FFFF F680H through FFFF F6FFH
UART Units	FFFF F700H through FFFF F77FH
GPIO Unit	FFFF F780H through FFFF F7BFH
Reserved	FFFF F7C0H through FFFF F7FFH
I/O Pad Control	FFFF F800H through FFFF F8FFH
Reserved	FFFF F900H through FFFF FFFFH

The memory-mapped registers that are also accessible via PCI configuration transactions are:

- Address Translation Unit

The registers which must have the address translation logic configured to translate PCI addresses into the Intel® XScale™ Core address space, to access the memory-mapped registers from the PCI interface are:

- DMA Controllers
- Memory Controller
- I^2C Bus Interface Unit
- Synchronous Serial Port UART Unit
- Messaging Unit
- Application Accelerator Unit
- Internal Arbitration Unit
- Peripheral Bus Interface Unit
- Performance Monitoring
- Interrupt Controller and General Purpose I/O Unit
- Interface Pad Control Registers

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 1 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
	Reserved	x	FFFF E000H through FFFF E0FFH	
Address Translation Unit	ATU Vendor ID Register	16	FFFF E100H	00H
	ATU Device ID Register	16	FFFF E102H	00H
	ATU Command Register	16	FFFF E104H	01H
	ATU Status Register	16	FFFF E106H	01H
	ATU Revision ID Register	8	FFFF E108H	02H
	ATU Class Code Register	24	FFFF E109H	02H
	ATU Cacheline Size Register	8	FFFF E10CH	03H
	ATU Latency Timer Register	8	FFFF E10DH	03H
	ATU Header Type Register	8	FFFF E10EH	03H
	BIST Register	8	FFFF E10FH	03H
	Inbound ATU Base Address Register 0	32	FFFF E110H	04H
	Inbound ATU Upper Base Address Register 0	32	FFFF E114H	05H
	Inbound ATU Base Address Register 1	32	FFFF E118H	06H
	Inbound ATU Upper Base Address Register 1	32	FFFF E11CH	07H
	Inbound ATU Base Address Register 2	32	FFFF E120H	08H
	Inbound ATU Upper Base Address Register 2	32	FFFF E124H	09H
	Reserved	32	FFFF E128H	0AH
	ATU Subsystem Vendor ID Register	16	FFFF E12CH	0BH
	ATU Subsystem ID Register	16	FFFF E12EH	0BH
	Expansion ROM Base Address Register	32	FFFF E130H	0CH
	ATU Capabilities Pointer Register	8	FFFF E134H	0DH
	Reserved	24	FFFF E135H	0DH
	Reserved	32	FFFF E138H	0EH
	ATU Interrupt Line Register	8	FFFF E13CH	0FH
	ATU Interrupt Pin Register	8	FFFF E13DH	0FH
	ATU Minimum Grant Register	8	FFFF E13EH	0FH
	ATU Maximum Latency Register	8	FFFF E13FH	0FH

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 2 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Address Translation Unit Extended Configuration Registers	Inbound ATU Limit Register 0	32	FFFF E140H	10H
	Inbound ATU Translate Value Register 0	32	FFFF E144H	11H
	Expansion ROM Limit Register	32	FFFF E148H	12H
	Expansion ROM Translate Value Register	32	FFFF E14CH	13H
	Inbound ATU Limit Register 1	32	FFFF E150H	14H
	Inbound ATU Limit Register 2	32	FFFF E154H	15H
	Inbound ATU Translate Value Register 2	32	FFFF E158H	16H
	Outbound I/O Window Translate Value Register	32	FFFF E15CH	17H
	Outbound Memory Window Value Register 0	32	FFFF E160H	18H
	Outbound Upper 32-bit Memory Window Value Register 0	32	FFFF E164H	19H
	Outbound Memory Window Value Register 1	32	FFFF E168H	1AH
	Outbound Upper 32-bit Memory Window Value Register 1	32	FFFF E16CH	1BH
	Reserved	32	FFFF E170H	1CH
	Reserved	32	FFFF E174H	1DH
	Outbound Upper 32-bit Direct Window Value Register	32	FFFF E178H	1EH
	Reserved	8	FFFF E17CH	1FH
	Reserved	8	FFFF E17DH	1FH
	Reserved	8	FFFF E17EH	1FH
	Reserved	8	FFFF E17FH	1FH
	ATU Configuration Register	32	FFFF E180H	20H
	PCI Configuration and Status Register	32	FFFF E184H	21H
	ATU Interrupt Status Register	32	FFFF E188H	22H
	ATU Interrupt Mask Register	32	FFFF E18CH	23H
	Inbound ATU Base Address Register 3	32	FFFF E190H	24H
	Inbound ATU Upper Base Address Register 3	32	FFFF E194H	25H
	Inbound ATU Limit Register 3	32	FFFF E198H	26H
	Inbound ATU Translate Value Register 3	32	FFFF E19CH	27H
	Reserved	32	FFFF E1A0H	28H
	Outbound Configuration Cycle Address Register	32	FFFF E1A4H	29H
	Reserved	32	FFFF E1A8H	2AH
	Outbound Configuration Cycle Data Register	32	FFFF E1ACH	2BH
	Reserved	32	FFFF E1B0H	2CH
	Reserved	32	FFFF E1B4H	2DH
	VPD Capabilities Identifier Register	8	FFFF E1B8H	2EH
	VPD Next Item Pointer Register	8	FFFF F1B9H	2EH
	VPD Address Register	16	FFFF F1BAH	2EH
	VPD Data Register	32	FFFF F1BCH	2FH

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 3 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Address Translation Unit Extended Configu-ration Registers	Power Management Capability Identifier Register	8	FFFF E1C0H	30H
	Power Management Next Item Pointer Register	8	FFFF E1C1H	30H
	Power Management Capabilities Register	16	FFFF E1C2H	30H
	Power Management Control/Status Register	16	FFFF E1C4H	31H
	Reserved	16	FFFF E1C6H	31H
	Reserved	32	FFFF E1C8H	32H
	Reserved	32	FFFF E1CCH	33H
	MSI Capability Identifier Register	8	FFFF E1D0H	34H
	MSI Next Item Pointer Register	8	FFFF E1D1H	34H
	MSI Message Control Register	16	FFFF E1D2H	34H
	MSI Message Address Register	32	FFFF E1D4H	35H
	MSI Message Upper Address Register	32	FFFF E1D8H	36H
	MSI Message Data Register	16	FFFF E1DCH	37H
	Reserved	16	FFFF E1DEH	37H
	PCI-X Capability Identifier Register	8	FFFF E1E0H	38H
	PCI-X Next Item Pointer Register	8	FFFF E1E1H	38H
	PCI-X Command Register	16	FFFF E1E2H	38H
	PCI-X Status Register	32	FFFF E1E4H	39H
	Reserved	32	FFFF E1E8H	3AH
	PCI Interrupt Routing Select Register	32	FFFF E1ECH	3BH
	Reserved		FFFF E1F0H through FFFF E1FFH	
	Reserved	x	FFFF E200H through FFFF E2FFH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 4 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Messaging Unit	Reserved	x	FFFF E300H through FFFF E30CH	Available through ATU Inbound Translation Window or must translate PCI address to the Intel® XScale™ Core Memory-Mapped Address
	Inbound Message Register 0	32	FFFF E310H	
	Inbound Message Register 1	32	FFFF E314H	
	Outbound Message Register 0	32	FFFF E318H	
	Outbound Message Register 1	32	FFFF E31CH	
	Inbound Doorbell Register	32	FFFF E320H	
	Inbound Interrupt Status Register	32	FFFF E324H	
	Inbound Interrupt Mask Register	32	FFFF E328H	
	Outbound Doorbell Register	32	FFFF E32CH	
	Outbound Interrupt Status Register	32	FFFF E330H	
	Outbound Interrupt Mask Register	32	FFFF E334H	
	Reserved	x	FFFF E338H through FFFF E34FH	Must Translate PCI address to the Intel® XScale™ Core Memory-Mapped Address
	MU Configuration Register	32	FFFF E350H	
	Queue Base Address Register	32	FFFF E354H	
	Reserved	32	FFFF E358H	
	Reserved	32	FFFF E35CH	
	Inbound Free Head Pointer Register	32	FFFF E360H	
	Inbound Free Tail Pointer Register	32	FFFF E364H	
	Inbound Post Head Pointer Register	32	FFFF E368H	
	Inbound Post Tail Pointer Register	32	FFFF E36CH	
	Outbound Free Head Pointer Register	32	FFFF E370H	
	Outbound Free Tail Pointer Register	32	FFFF E374H	
	Outbound Post Head Pointer Register	32	FFFF E378H	
	Outbound Post Tail Pointer Register	32	FFFF E37CH	
	Index Address Register	32	FFFF E380H	
	Reserved	x	FFFF E384H through FFFF E3FFH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 5 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
DMA Controller	Channel 0 Channel Control Register	32	FFFF E400H	Must Translate PCI address to the Intel® XScale™ Core Memory-Mapped Address
	Channel 0 Channel Status Register	32	FFFF E404H	
	Reserved	32	FFFF E408H	
	Channel 0 Descriptor Address Register	32	FFFF E40CH	
	Channel 0 Next Descriptor Address Register	32	FFFF E410H	
	Channel 0 PCI Address Register	32	FFFF E414H	
	Channel 0 PCI Upper Address Register	32	FFFF E418H	
	Channel 0 Internal Bus Address Register	32	FFFF E41CH	
	Channel 0 Byte Count Register	32	FFFF E420H	
	Channel 0 Descriptor Control Register	32	FFFF E424H	
	Reserved	x	FFFF E428H through FFFF E43FH	
	Channel 1 Channel Control Register	32	FFFF E440H	
	Channel 1 Channel Status Register	32	FFFF E444H	
	Reserved	32	FFFF E448H	
	Channel 1 Descriptor Address Register	32	FFFF E44CH	
	Channel 1 Next Descriptor Address Register	32	FFFF E450H	
	Channel 1 PCI Address Register	32	FFFF E454H	
	Channel 1 PCI Upper Address Register	32	FFFF E458H	
	Channel 1 Internal Bus Address Register	32	FFFF E45CH	
	Channel 1 Byte Count Register	32	FFFF E460H	
	Channel 1 Descriptor Control Register	32	FFFF E464H	
	Reserved	x	FFFF E468H through FFFF E4FFH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 6 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Memory Controller	SDRAM Initialization Register	32	FFFF E500H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	SDRAM Control Register 0	32	FFFF E504H	
	SDRAM Control Register 1	32	FFFF E508H	
	SDRAM Base Register	32	FFFF E50CH	
	SDRAM Bank 0 Size Register	32	FFFF E510H	
	SDRAM Bank 1 Size Register	32	FFFF E514H	
	SDRAM 32-bit Region Size Register	32	FFFF E518H	
	ECC Control Register	32	FFFF E51CH	
	ECC Log 0 Register	32	FFFF E520H	
	ECC Log 1 Register	32	FFFF E524H	
	ECC Address 0 Register	32	FFFF E528H	
	ECC Address 1 Register	32	FFFF E52CH	
	ECC Test Register	32	FFFF E530H	
	Memory Controller Interrupt Status Register	32	FFFF E534H	
	Reserved	32	FFFF E538H	
	MCU Port Transaction Count Register	32	FFFF E53CH	
	MCU Preemption Control Register	32	FFFF E540H	
	Refresh Frequency Register	32	FFFF E548H	
	Reserved	32	FFFF E550H	
	Reserved	32	FFFF E554H	
	Reserved	32	FFFF E558H	
	Reserved	32	FFFF E55CH	
	Reserved	32	FFFF E560H	
	Reserved	32	FFFF E564H	
	Reserved	32	FFFF E568H	
	Reserved	32	FFFF E56CH	
	Reserved	x	FFFF E58CH through FFFF E5FFH	
Intel® XScale core Bus Interface Unit	BIU Status Register	32	FFFF E600H	
	BIU Error Address Register	32	FFFF E604H	
	BIU Control Register	32	FFFF E608H	
	Reserved	x	FFFF E60CH through FFFF E67FH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 7 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Peripheral Bus Interface Unit	PBI Control Register	32	FFFF E680H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Reserved	32	FFFF E684H	
	PBI Base Address Register 0	32	FFFF E688H	
	PBI Limit Register 0	32	FFFF E68CH	
	PBI Base Address Register 1	32	FFFF E690H	
	PBI Limit Register 1	32	FFFF E694H	
	Reserved	x	FFFF E698H through FFFF E6BCH	
	PBI Memory-less Boot Register 0	32	FFFF E6C0H	
	Reserved	x	FFFF E6C4H through FFFF E6DFH	
	PBI Memory-less Boot Register 1	32	FFFF E6E0H	
	PBI Memory-less Boot Register 2	32	FFFF E6E4H	
	Reserved	x	FFFF E6E8H through FFFF E6FFH	
Peripheral Performance Monitoring Unit	Global Timer Mode Register	32	FFFF E700H	Must Translate PCI address to the Intel® XScale™ Core Memory-Mapped Address
	Event Select Register	32	FFFF E704H	
	Event Monitoring Interrupt Status Register	32	FFFF E708H	
	Reserved	32	FFFF E70CH	
	Global Time Stamp Register	32	FFFF E710H	
	Programmable Event Counter Register 1	32	FFFF E714H	
	Programmable Event Counter Register 2	32	FFFF E718H	
	Programmable Event Counter Register 3	32	FFFF E71CH	
	Programmable Event Counter Register 4	32	FFFF E720H	
	Programmable Event Counter Register 5	32	FFFF E724H	
	Programmable Event Counter Register 6	32	FFFF E728H	
	Programmable Event Counter Register 7	32	FFFF E72CH	
	Programmable Event Counter Register 8	32	FFFF E730H	
	Programmable Event Counter Register 9	32	FFFF E734H	
	Programmable Event Counter Register 10	32	FFFF E738H	
	Programmable Event Counter Register 11	32	FFFF E73CH	
	Programmable Event Counter Register 12	32	FFFF E740H	
	Programmable Event Counter Register 13	32	FFFF E744H	
	Programmable Event Counter Register 14	32	FFFF E748H	
	Reserved		FFFF E74CH through FFFF E77FH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 8 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Interrupt Controller, and Timers	Processor Device ID Register	32	FFFF E780H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Reserved		FFFF E784H	
	Reserved		FFFF E788H	
	Reserved		FFFF E78CH	
	Interrupt Control Register 0	32	FFFF E790H	
	Interrupt Control Register 1	32	FFFF E794H	
	Interrupt Steer Register 0	32	FFFF E798H	
	Interrupt Steer Register 1	32	FFFF E79CH	
	IRQ Interrupt Source Register 0	32	FFFF E7A0H	
	IRQ Interrupt Source Register 1	32	FFFF E7A4H	
	FIQ Interrupt Source Register 0	32	FFFF E7A8H	
	IRQ Interrupt Source Register 1	32	FFFF E7ACH	
	Interrupt Priority Register 0	32	FFFF E7B0H	
	Interrupt Priority Register 1	32	FFFF E7B4H	
	Interrupt Priority Register 2	32	FFFF E7B8H	
	Interrupt Priority Register 3	32	FFFF E7BCH	
	Interrupt Base Register	32	FFFF E7C0H	
	Interrupt Size Register	32	FFFF E7C4H	
	IRQ Interrupt Vector Register	32	FFFF E7C8H	
	FIQ Interrupt Vector Register	32	FFFF E7CCH	
Timer Unit	Timer Mode Register 0	32	FFFF E7D0H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Timer Mode Register 1	32	FFFF E7D4H	
	Timer Count Register 0	32	FFFF E7D8H	
	Timer Count Register 1	32	FFFF E7DCH	
	Timer Reload Register 0	32	FFFF E7E0H	
	Timer Reload Register 1	32	FFFF E7E4H	
	Timer Interrupt Status Register	32	FFFF E7E8H	
	Watch Dog Timer Control Register	32	FFFF E7ECH	
Internal Arbitration Unit	Internal Arbitration Control Register	32	FFFF E7F0H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Multi-Transaction Timer Register 1	32	FFFF E7F4H	
	Multi-Transaction Timer Register 2	32	FFFF E7F8H	
	Reserved	x	FFFF E7FCH through FFFF E7FFH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 9 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Application Accelerator Unit	Accelerator Control Register	32	FFFF E800H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Accelerator Status Register	32	FFFF E804H	
	Accelerator Descriptor Address Register	32	FFFF E808H	
	Accelerator Next Descriptor Address Register	32	FFFF E80CH	
	Accelerator Source Address 1 Register	32	FFFF E810H	
	Accelerator Source Address 2 Register	32	FFFF E814H	
	Accelerator Source Address 3 Register	32	FFFF E818H	
	Accelerator Source Address 4 Register	32	FFFF E81CH	
	Destination Address Register	32	FFFF E820H	
	Accelerator Byte Count Register	32	FFFF E824H	
	Accelerator Descriptor Control Register	32	FFFF E828H	
	Accelerator Source Address 5 Register	32	FFFF E82CH	
	Accelerator Source Address 6 Register	32	FFFF E830H	
	Accelerator Source Address 7 Register	32	FFFF E834H	
	Accelerator Source Address 8 Register	32	FFFF E838H	
	Extended Descriptor Control Register 0	32	FFFF E83CH	
	Accelerator Source Address 9 Register	32	FFFF E840H	
	Accelerator Source Address 10 Register	32	FFFF E844H	
	Accelerator Source Address 11 Register	32	FFFF E848H	
	Accelerator Source Address 12 Register	32	FFFF E84CH	
	Accelerator Source Address 13 Register	32	FFFF E850H	
	Accelerator Source Address 14 Register	32	FFFF E854H	
	Accelerator Source Address 15 Register	32	FFFF E858H	
	Accelerator Source Address 16 Register	32	FFFF E85CH	
	Extended Descriptor Control Register 1	32	FFFF E860H	
	Accelerator Source Address 17 Register	32	FFFF E864H	
	Accelerator Source Address 18 Register	32	FFFF E868H	
	Accelerator Source Address 19 Register	32	FFFF E86CH	
	Accelerator Source Address 20 Register	32	FFFF E870H	
	Accelerator Source Address 21 Register	32	FFFF E874H	
	Accelerator Source Address 22 Register	32	FFFF E878H	
	Accelerator Source Address 23 Register	32	FFFF E87CH	
	Accelerator Source Address 24 Register	32	FFFF E880H	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 10 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
Application Accelerator Unit	Extended Descriptor Control Register 2	32	FFFF E884H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	Accelerator Source Address 25 Register	32	FFFF E888H	
	Accelerator Source Address 26 Register	32	FFFF E88CH	
	Accelerator Source Address 27 Register	32	FFFF E890H	
	Accelerator Source Address 28 Register	32	FFFF E894H	
	Accelerator Source Address 29 Register	32	FFFF E898H	
	Accelerator Source Address 30 Register	32	FFFF E89CH	
	Accelerator Source Address 31 Register	32	FFFF E8A0H	
	Accelerator Source Address 32 Register	32	FFFF E8A4H	
	Reserved	x	FFFF E8A8H through FFFF E8FFH	
	Reserved	x	FFFF E900H through FFFF EFFFH	
	Reserved	x	FFFF F000H through FFFF F4FFH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 11 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
DDR I/O Control	DCAL Control and Status Register	32	FFFF F500H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	DCAL Address Register	32	FFFF F504H	
	DCAL Data Register 0	32	FFFF F508H	
	DCAL Data Register 1	32	FFFF F50CH	
	DCAL Data Register 2	32	FFFF F510H	
	DCAL Data Register 3	32	FFFF F514H	
	DCAL Data Register 4	32	FFFF F518H	
	DCAL Data Register 5	32	FFFF F51CH	
	DCAL Data Register 6	32	FFFF F520H	
	DCAL Data Register 7	32	FFFF F524H	
	DCAL Data Register 8	32	FFFF F528H	
	DCAL Data Register 9	32	FFFF F52CH	
	DCAL Data Register 10	32	FFFF F530H	
	DCAL Data Register 11	32	FFFF F534H	
	DCAL Data Register 12	32	FFFF F538H	
	DCAL Data Register 13	32	FFFF F53CH	
	DCAL Data Register 14	32	FFFF F540H	
	DCAL Data Register 15	32	FFFF F544H	
	DCAL Data Register 16	32	FFFF F548H	
	DCAL Data Register 17	32	FFFF F54CH	
	Receive Enable Delay Register	32	FFFF F550H	
	Slave Low Mix 0	32	FFFF F554H	
	Slave Low Mix 1	32	FFFF F558H	
	Slave High Mix 0	32	FFFF F55CH	
	Slave High Mix 1	32	FFFF F560H	
	Slave Length	32	FFFF F564H	
	Master Mix	32	FFFF F568H	
	Master Length	32	FFFF F56CH	
	DDR Drive Strength Status Register	32	FFFF F570H	
	DDR Drive Strength Control Register	32	FFFF F574H	
	DDR Miscellaneous Pad Control Register	32	FFFF F578H	
	Reserved	32	FFFF F57CH	
	PBI Drive Strength Control Register	32	FFFF F580H	
	Reserved	x	FFFF F584H through FFFF F5BFH	
PCI I/O Control	Secondary PCI Drive Strength Control Register	32	FFFF F5C0H	
	Reserved	32	FFFF F5C4H	
	Primary PCI Drive Strength Control Register	32	FFFF F5C8H	
	Reserved	32	FFFF F5CCH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 12 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
	Reserved	x	FFFF F5D0H through FFFF F67FH	
I^2C Bus Interface Units	I^2C Control Register 0	32	FFFF F680H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	I^2C Status Register 0	32	FFFF F684H	
	I^2C Data Buffer Register 0	32	FFFF F68CH	
	Reserved	32	FFFF F690H	
	I^2C Bus Monitor Register 0	32	FFFF F694H	
	Reserved	32	FFFF F698H	
	Reserved	32	FFFF F69CH	
	I^2C Control Register 1	32	FFFF F6A0H	
	I^2C Status Register 1	32	FFFF F6A4H	
	I^2C Slave Address Register 1	32	FFFF F6A8H	
	I^2C Data Buffer Register 1	32	FFFF F6ACH	
	Reserved	32	FFFF F6B0H	
	I^2C Bus Monitor Register 1	32	FFFF F6B4H	
	Reserved	x	FFFF F6B8H through FFFF F6FFH	
UART Channel 0	UART 0 Receive Buffer Register (Read Only) (DLAB=0)	32	FFFF F700H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	UART 0 Transmit Holding Register (Write Only) (DLAB=0)	32		
	UART 0 baud Divisor Latch Low byte (DLAB=1)	8		
	UART 0 Interrupt Enable Register (DLAB=0)	8	FFFF F704H	
	UART 0 baud Divisor Latch High byte (DLAB=1)	8		
	UART 0 Interrupt ID Register (Read Only)	8	FFFF F708H	
	UART 0 FIFO Control Register (Write Only)	8		
	UART 0 Line Control Register	8	FFFF F70CH	
	UART 0 Modem Control Register	8	FFFF F710H	
	UART 0 Line Status Register	8	FFFF F714H	
	UART 0 Modem Status Register	8	FFFF F718H	
	UART 0 Scratch Pad Register	8	FFFF F71CH	
	Reserved	32	FFFF F720H	
	UART 0 FIFO Occupancy Register	8	FFFF F724H	
	UART 0 Autobaud Control Register	8	FFFF F728H	
	UART 0 Autobaud Count Register	16	FFFF F72CH	
	Reserved		FFFF F730H through FFFF F73FH	

Table 399. Peripheral Memory-Mapped Register Locations (Sheet 13 of 13)

Peripheral	Register Description (Name)	Register Size in Bits	Internal Bus Address	PCI Configuration Space Register Number
UART Channel 1	UART 1 Receive Buffer Register (Read Only) (DLAB=0)	32	FFFF F740H	Must Translate PCI address to the Intel® XScale™ Core Memory-mapped Address
	UART 1 Transmit Holding Register (Write Only) (DLAB=0)	32		
	UART 1 baud Divisor Latch Low byte (DLAB=1)	8		
	UART 1 Interrupt Enable Register (DLAB=0)	8	FFFF F744H	
	UART 1 baud Divisor Latch High byte (DLAB=1)	8		
	UART 1 Interrupt ID Register (Read Only)	8	FFFF F748H	
	UART 1 FIFO Control Register (Write Only)	8		
	UART 1 Line Control Register	8	FFFF F74CH	
	UART 1 Modem Control Register	8	FFFF F750H	
	UART 1 Line Status Register	8	FFFF F754H	
	UART 1 Modem Status Register	8	FFFF F758H	
	UART 1 Scratch Pad Register	8	FFFF F75CH	
	Reserved	32	FFFF F760H	
	UART 1 FIFO Occupancy Register	8	FFFF F764H	
	UART 1 Autobaud Control Register	8	FFFF F768H	
	UART 1 Autobaud Count Register	16	FFFF F76CH	
	Reserved	x	FFFF F770H through FFFF F77FH	
GPIO	GPIO Output Enable Register	32	FFFF F780H	
	GPIO Input Data Register	32	FFFF F784H	
	GPIO Output Data Register	32	FFFF F788H	
	Reserved	32	FFFF F78CH	
	Reserved	x	FFFF F790H through FFFF F7FFH	
	Reserved	x	FFFF F800H through FFFF F9FFH	

17.8 Coprocessor Register Space

The CCR address space is assigned to support the integrated peripherals on the 80331 that require low latency register access. Table 400 shows all of the 80331 integrated coprocessor registers and assigned coprocessor space. The *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E. provides for a total of 16 coprocessors each of which can contain up to 256 32 bit registers. For completeness, the coprocessor space reserved by the *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E. is shown.

All accesses to coprocessor registers that are not implemented have 'unpredictable' behavior and are designated as 'undefined' in the register table. Accesses to these registers results in an undefined instruction exception.

Table 400. Intel® XScale™ Core Coprocessor Registers Assigned to Integrated Peripherals

Integrated Peripheral	Coprocessor
Interrupt Control Unit	CP6
Programmable Timers	CP6
Core Performance Monitoring Unit	CP14
System Control[a]	CP15

a. Reserved by the *ARM Architecture Reference Manual* - ARM Limited. Order number: ARM DDI 0100E..

Table 401. Coprocessor Register Locations (Sheet 1 of 2)

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
Interrupt Control Unit	Interrupt Control Register 0	CP6	0	Register 0
	Interrupt Control Register 1			Register 1
	Interrupt Steering Register 0			Register 2
	Interrupt Steering Register 1			Register 3
	IRQ Interrupt Source Register 0			Register 4
	IRQ Interrupt Source Register 1			Register 5
	FIQ Interrupt Source Register 0			Register 6
	FIQ Interrupt Source Register 1			Register 7
	Interrupt Priority Register 0			Register 8
	Interrupt Priority Register 1			Register 9
	Interrupt Priority Register 2			Register 10
	Interrupt Priority Register 3			Register 11
	Interrupt Base Register			Register 12
	Interrupt Size Register			Register 13
	IRQ Interrupt Vector Register			Register 14
	FIQ Interrupt Vector Register			Register 15
Programmable Timers Unit	Timer Mode Register 0		1	Register 0
	Timer Mode Register 1			Register 1
	Timer Count Register 0			Register 2
	Timer Count Register 1			Register 3
	Timer Reload Register 0			Register 4
	Timer Reload Register 1			Register 5
	Timer Interrupt Status Register			Register 6
	Watch Dog Timer Control Register			Register 7
	Undefined			Register 8 through Register 15

Table 401. Coprocessor Register Locations (Sheet 2 of 2)

Peripheral	Register Description (Name)	Coprocessor	Field CR_m	Coprocessor Register (Field CR_n)
Core Performance Monitoring Unit	Performance Monitor Control Register	CP14	CP14 Function	Register 0
	Clock Count Register			Register 1
	Performance Count Register 0			Register 2
	Performance Count Register 1			Register 3
	Reserved			Register 4 and Register 5
	Core Clock Configuration Register (CCLKCFG)			Register 6
	Reserved			Register 7
	Access Transmit Debug Register (TX)			Register 8
	Access Receive Debug Register (RX)			Register 9
	Access Debug Control and Status Register (DBGCSR)			Register 10
	Access Trace Buffer Register (TBREG)			Register 11
	Access Checkpoint 0 Register (CHKPT0)			Register 12
	Access Checkpoint 1 Register (CHKPT1)			Register 13
	Access Transmit and Receive Debug Control Register			Register 14
	Debug Saved Program Status Register			Register 15
System Controls	ID and Cache Type Registers	CP15	CP 15 Function[a]	Register 0
	Control and Auxiliary Control Registers			Register 1
	Translation Table Base Register			Register 2
	Domain Access Control Register			Register 3
	Undefined			Register 4
	Fault Status Register			Register 5
	Fault Address Register			Register 6
	Cache Operations Register			Register 7
	TLB Operations Register			Register 8
	Cache Lock Down			Register 9
	TLB Lock Down			Register 10
	Reserved			Register 11 and Register 12
	Process ID Register			Register 13
	Breakpoint Registers			Register 14
	CP Access (PID) Coprocessor Access Control Register			Register 15

a. Some of the CP15 registers are differentiated by the Opcode_2 field. For CP15, the CR_m is used in some cases to denote different control functions for a given coprocessor register rather than a distinct register decode. Please refer to the Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Developer's Manual (Order Number: 273411), for more details on the operation of CP15.

This Page Intentionally Left Blank

Clocking and Reset 18

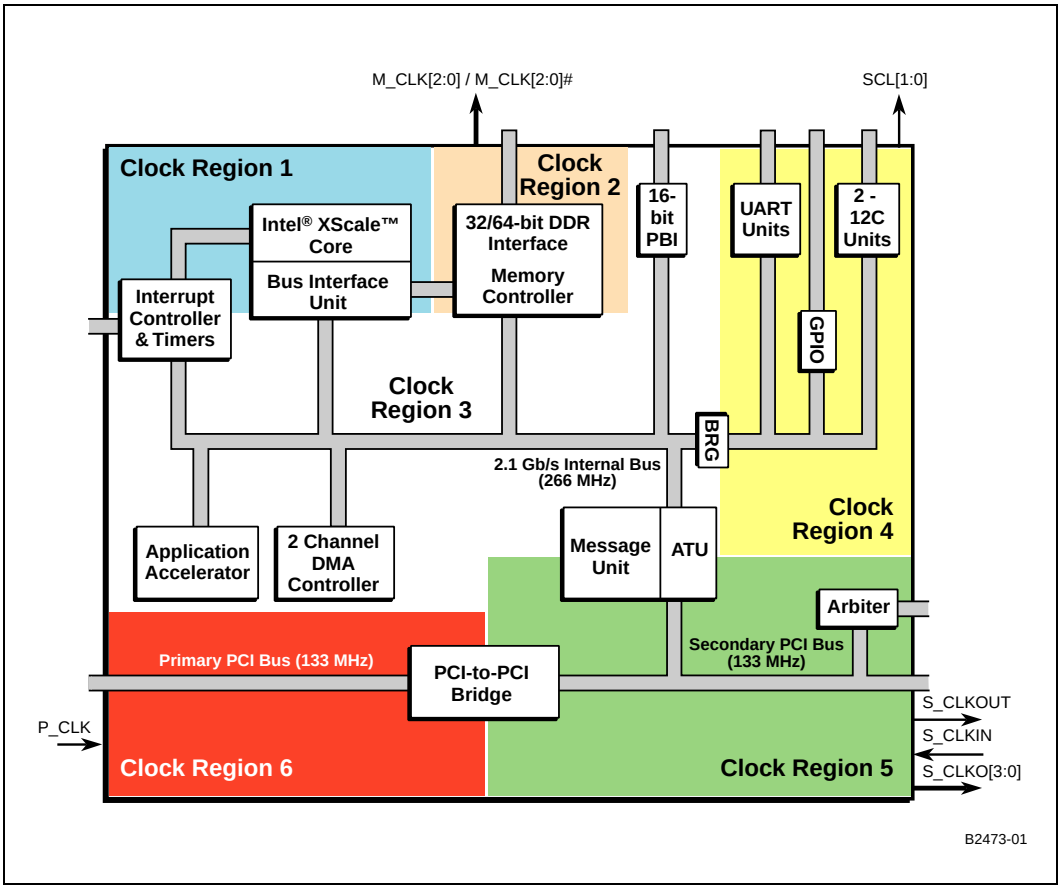
This chapter describes the clocking and reset function. The intent of this chapter is to elaborate and clarify descriptions of the clocking and reset mechanisms.

18.1 Clocking Overview

The Intel® 80331 I/O processor (80331) contains various clocking boundaries internally. The clocks for all of the units within the 80331 are generated from a single input clock. Clock regions are buffered or PLL versions of this clock input for the processor design. This input feeds the Phase Lock Loop (PLL) circuitry which generates the internal clocks for all clock regions.

The block diagram of the 80331 shown in Figure 118, highlights the six clocking regions for this processor.

Figure 118. Intel® 80331 I/O Processor Clocking Regions Diagram



Within each of the clocking regions identified exist various clock requirements for the 80331 units and for the output clocks pins provided for the external subsystem.

18.1.1 Clocking Theory of Operation

Each region within the 80331 contains different clocking requirements. These requirements are summarized in the following sections.

18.1.2 Clocking Region 1

Region 1 is the Intel® XScale™ core (ARM* architecture compliant). It supports clock frequencies up to a maximum of 800 MHz operation. Region 1 also incudes the core interface of the Intel® XScale™ core Bus Interface Unit. The region 1 clock is an integer multiple of one-half the frequency of the DDR SDRAM clocks of Region 2. The creates a synchronous architecture between the Bus Interface Unit and Memory Controller Unit for minimal latency between the Core processor and DDR SDRAM.

18.1.3 Clocking Region 2

Region 2 provides six DDR SDRAM output clocks, and the clocking for the Memory Controller Unit. The clocking unit contains three output clocks, called **M_CLK[2:0]**, and three complement clock outputs called **M_CLK[2:0]#**. The **M_CLK[2:0]** and **M_CLK[2:0]#** outputs are used by the DDR SDRAM memory subsystem. It supports clock frequencies up to a maximum of 333 MHz operation for DDR333 SDRAM.

18.1.4 Clocking Region 3

Region 3 covers the 80331 internal bus. It supports clock frequencies up to 266 MHz. The IOP units interface to or reside in this region. Units which interface to this clock region include the Intel® XScale™ core Bus Interface Unit, the Memory Controller Unit, Interrupt Controller, Timers, ATU, MU, Performance Monitor Unit and the bridge interface to the AHB bus. Units which reside in this clock region, and are based entirely on clock region 3 clocking include DMA units, AAU unit, and the Peripheral Bus interface. The peripheral bus interface operates at one-quarter of the Internal Bus frequency or 66MHz.

18.1.5 Clocking Region 4

Region 4 obtains its input clock from the clocking unit specified in clocking region 3. This region is for use by low-speed peripheral units. Currently, these include the I^2C bus interface, the General Purpose I/O unit and the UART serial bus interface.

Region 4 contains an output clock (**SCL**) used for the I^2C bus interface (see Chapter 10, "I^2C Bus Interface Units"). The **SCL** clock frequency for I^2C operation is 100 KHz or 400 KHz. **SCL** is generated from the internal bus clock. In order to use the I^2C interface, a clock divider value must be written into the I^2C Clock Count Register. The UART input clock is driven at 33.334 MHz,and divided within the unit for the serial interface baud rate.

18.1.6 Clocking Region 5

Region 5 covers the Secondary PCI bus , and obtains its input clock **S_CLKIN** as the board level feedback of from the **S_CLKOUT** output clock. Region 5 also generates four secondary PCI bus segment output clocks the 80331 based on a dedicated PLL. Eight of these output clocks, **S_CLKO[3:0]** are used to drive the Primary PCI input clocks of secondary PCI devices attached to the 80331 secondary PCI interfaces and the, **S_CLKOUT** used as the reference clock to this region's PCI interface. This region is the secondary PCI interface for 80331 and meets clocking requirements as specified in *PCI-X Addendum to the PCI Local Bus Specification,* Revision 1.0a. This region supports clock frequencies up to 133 MHz.

This region also includes the PCI interface of the ATU.

18.1.7 Clocking Region 6

Region 6 covers the Primary PCI bus , and obtains its input clock **P_CLK** as the Primary PCI interface reference clock. This region is the primary PCI interface for 80331 and meets clocking requirements as specified in *PCI-X Addendum to the PCI Local Bus Specification,* Revision 1.0a. This region supports clock frequencies up to 133 MHz.

Region 6 is the main input clock for the Intel® 80331 I/O processor. The clocks for all regions one through 6 are generated by the **P_CLK** input clock of this region.

18.1.8 Output Clocks

Table 402 shows the loading requirements for each of the output clocks. Refer to the respective JEDEC specifications for the DDR and DDR-II clock skew and loading requirements for the DDR and DDR-II SDRAM devices. Refer to the *PCI Local Bus Specification*, Revision 2.3 (e.g., 33 MHz, 66 MHz) and the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0 (e.g., 66 MHz, 100 MHz, 133 MHz) for details on PCI bus clock loading requirements.

Table 402. Output Clocks Loading Summary

Pin	Minimum	Maximum
M_CLK[2:0]	tbd pF + Trace	tbd pF + Trace
M_CLK[2:0]#	tbd pF + Trace	tbd pF + Trace
P_CLK	tbd pF + Trace	tbd pF + Trace
S_CLKOUT	tbd pF + Trace	tbd pF + Trace
S_CLKO[3:0]	tbd pF + Trace	tbd pF + Trace
SCL	tbd pF + Trace	tbd pF + Trace

18.1.9 Clocking Region Summary

Table 403 summarizes all of the input clock pins, output clock pins, and clock strapping option pins used in the .

Table 403. Clock Pin Summary

Pin	Input/Output	Description
P_CLK	Input	Primary PCI Input Clock
S_CLKO[3:0]	Output	Secondary PCI Output Clocks
S_CLKOUT	Output	Secondary PCI Output Reference Clock
S_CLKIN	Input	Secondary PCI Reference Clock Input
P_M66EN, S_M66EN	Input	PCI 66 MHz Enable
S_PCIX133EN	Input	Reset Strap to Limit Secondary Clock Frequency to 100MHz or 133MHz in PCI-X Mode
M_CLK[2:0]	Output	SDRAM Output Clocks
M_CLK[2:0]#	Output	Complementary SDRAM Output Clocks
SCL[1:0]	Output	I^2C Output Clocks

Table 404 summarizes all of the clocks generated to the six regions within the 80331. The clock units initializes appropriately based on the PCI-X initialization pattern driven during the rising edge of **P_RST#**.

Table 404. Intel® 80331 I/O Processor Clock Region Summary

Region	Interface	Unit	Frequencies		
1		Intel® XScale™ core	500 MHz	667 MHz	800 MHz
2	DDR SDRAM	MCU	333 MHz	333 MHz	N/A
	DDR-II SDRAM		400 MHz	N/A	400 MHz
3	Internal Bus	DMA[1:0], AAU, ATU	266 MHz		
	Peripheral Bus Interface	PBI	66 MHz		
4	AHB	I^2C	33 MHz		
		UART	33.334 MHz		
		GPIO	33 MHz		
5	Secondary PCI	Secondary PCI side of Bridge	33, 66, 100, 133MHz based on PCI reset strapping and initiation sequence.		
6	Primary PCI	Primary PCI side of Bridge	33, 66, 100, 133MHz based on PCI reset strapping and initiation sequence.		

18.2 Reset Overview

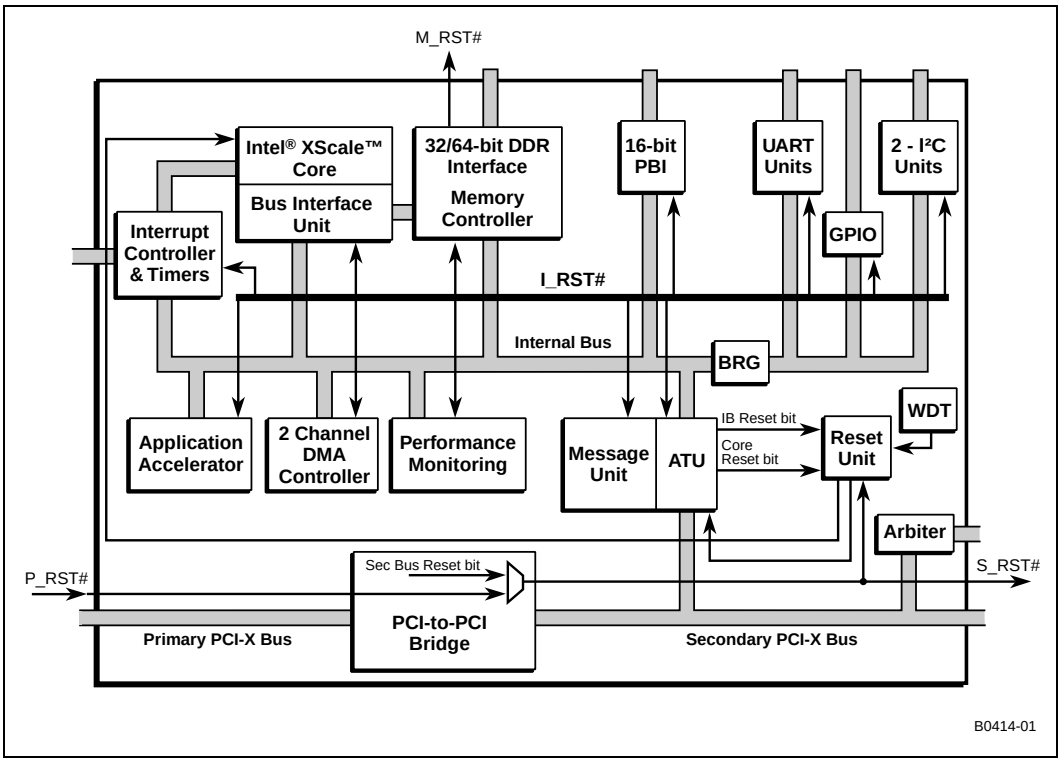
There are four levels of reset for the 80331. The main reset is controlled through the Primary PCI bus reset signal (**P_RST#**). When this signal is asserted, the entire 80331 is placed in a reset state. In addition to the reset pin, the 80331 provides software control of internal units such as the PCI-to-PCI Bridge, Internal Bus, and Intel® XScale™ core.

The 80331 four levels of reset are:

- **P_RST#** - this is an asynchronous reset to 80331 resets the entire chip. All reset straps are sampled at the falling edge of **P_RST#**.
- Software PCI Reset - this reset is initiated by writing to bridge control register of the PCI configuration space. This is also commonly referred to as the SBR (Secondary Bus Reset). As the I/O processor is integrated to the secondary PCI bus of 80331, all I/O processor units and registers are also reset by the Secondary Bus Reset.
- Internal Bus Reset - this reset is initiated by writing to ATU control register in PCI configuration space. This reset is specific to the integrated I/O processor and the associated peripheral units. This reset may also be generated by the expiration of the Watch Dog Timer as described in Section 14.1.2, "Watch Dog Timer Operation" on page 662.
- Intel® XScale™ core reset - this reset is initiated by two mechanisms. First is the **CORE_RST#** reset strap, and the other is via software through ATU control register in PCI configuration space.

Figure 119 shows the logical block diagram of the reset conditions.

Figure 119. Intel® 80331 I/O Processor Reset Block Diagram



18.2.1 Primary PCI Bus Reset Mechanism

The **P_RST#** reset clears all internal state machines and logic, and initialize all registers to their default states. The assertion and deassertion of the PCI reset signal **P_RST#** is asynchronous with respect to P_CLK. The rising edge of the **P_RST#** signal must be monotonic through the input switching range and must meet the minimum slew rate. The PCI local bus specifications define the assertion of **P_RST#** for a period of 1 ms after power is stable.

Upon the assertion of **P_RST#**, all units within the 80331 are reset including the bridge and I/O processor units. The affects on the units beyond the bridge are described in Section 18.2.3, "I/O Processor Reset" on page 757.

Upon the deassertion of **P_RST#**, the 80331 samples a series of strapping pins to set configuration modes (refer to Section 18.3, "Reset Strapping Options" on page 762).

18.2.2 Software PCI Reset Mechanism

This reset is initiated by a write to the bridge control register and resets the secondary PCI segment and is often referred to as the Secondary Bus Reset (SBR). This reset can be used for various reasons in including recovering from error conditions on the secondary bus, to redo enumeration, to change the operating frequency of the bus (33/66/100/133 MHz), to change the operating mode of the bus (PCI or PCI-X) etc. This reset is synchronous to the PCI clock domain. Writes to the bridge control register with a new frequency etc, has no effect until the SBR happens. The power up frequency of the PCI bus is shown in Table 2.2.4, "Bus Mode and Frequency Initialization" on page 46. The frequency depends on the M66EN and the PCIXCAP pins.

The I/O processor units of 80331, as a device on the secondary PCI bus segment, is completely reset by a SBR. Refer to Section 18.2.3, "I/O Processor Reset" on page 757 for details of the I/O processor reset.

18.2.3 I/O Processor Reset

Those reset mechanisms which assert secondary bus reset, results in a reset of the I/O processor in addition to the additional reset actions described in the respective reset mechanism descriptions above.

When the I/O processor is reset by the Secondary Bus Reset signal S_RST#, the I/O processor:

- resets the internal bus, refer to Section 18.2.4, "Internal Bus Reset" on page 758.
- resets the Intel® XScale™ core, refer to Section 18.2.5, "Intel® XScale™ core Reset Mechanism" on page 761
- resets all internal units
- resets all Memory Mapped Registers
- all configuration straps are not affected by S_RST#, refer to Section 18.3.

The assertion and deassertion of the PCI reset signal (S_RST#) is asynchronous with respect to **P_CLK**. The rising edge of the S_RST# signal must be monotonic through the input switching range and must meet the minimum slew rate. The *PCI Local Bus Specification*, Revision 2.3 defines the assertion of S_RST# for a period of 1 ms after power is stable.

Upon the assertion of S_RST#, all units within the 80331, excluding the bridge are reset. This reset resets all internal memory mapped registers (MMRs) to their default configuration state. The reset value for each register is defined within each register description.

Upon the deassertion of S_RST#, the 80331 samples a series of strapping pins to set configuration modes (refer to Section 18.3, "Reset Strapping Options" on page 762). One strap which alters the behavior of the 80331 on the deassertion of S_RST# is the **CORE_RST#** strap (refer to Section 18.2.5, "Intel® XScale™ core Reset Mechanism" on page 761).

18.2.4 Internal Bus Reset

The Reset Internal Bus bit in the PCI Configuration and Status Register (see Table 125, "PCI Configuration and Status Register - PCSR" on page 253) resets the Intel® XScale™ core and all units on the internal bus. Expiration of the Watch Dog Timer as described in Section 14.1.2, "Watch Dog Timer Operation" on page 662 also results in the same internal bus reset.

Before resetting, the ATU shall gracefully halt all PCI bus transactions when operating in PCI mode. Following an Internal Bus Reset when the PCI Bus is operating in PCI-X mode, the ATU may allow Split Completion transactions due to Outbound Split Requests to Master Abort. Also, the ATU may not initiate all of the Split Completion transactions required by any outstanding Inbound Split Requests at the time of the Internal Bus Reset.

While the *PCI-X Addendum to the PCI Local Bus Specification*, Revision 1.0 requires host software to gracefully handle both these cases without reinitializing the PCI bus, the user may want to avoid this scenario. This can be done by taking the following steps:

1. Disable the ATU from either claiming or initiating new transactions by clearing the *Bus Master Enable* and the *Memory Enable* in the ATU Command Register (see Section 3.10.3, "ATU Command Register - ATUCMD" on page 214).
2. Monitor the *Inbound Read Transaction Queue Status* and the *Outbound Read Transaction Queue Status* in the PCSR.
3. When both the Inbound Read Transaction queue and the Outbound Read Transaction queue are empty, software writes to the PCSR to initiate the Internal Bus Reset.

It is the responsibility of the software to ensure that the I^2C bus and UART are idle before the reset occurs. The Intel® XScale™ core may or may not be held in reset when the Reset Internal Bus bit is cleared by software. This depends on the **CORE_RST#** strap as described in section Section 18.2.5.

Expiration of the Watch Dog Timer does not result in a graceful Internal Bus Reset, as the queues, I^2C bus and UART may be active when the reset occurs.

When the Internal Bus Reset is initiated there are sideband signals notifying the ATU that a reset is coming. The following describes the operation of each unit:

Table 405. Internal Bus Reset Summary (Sheet 1 of 2)

Unit	Preparation for Reset	Reset Status
ATU/MU	The ATU and MU detect via a sideband signal that a internal bus reset is coming. Upon detecting this signal, the ATU and MU complete the following: PCI Interface Outbound Transaction: • When the ATU has already asserted its PCI request signal, and not yet started a transaction, the ATU deasserts its request and not start its transaction. • When the ATU has not yet requested the PCI bus, the ATU never asserts its request for the PCI bus. • When the ATU is in the middle of a transaction, the ATU performs the existing transaction. This means that an outbound write, the ATU transfers as much data as available in the queue. When an outbound read, the ATU reads the data until the transaction stops naturally (meaning that the target has ended the transaction or the ATU has read all of the data it has been requested to read). Once terminated by the ATU or the target, the ATU no longer requests the PCI bus. • In PCI-X mode, the ATU allows any outstanding split completions due to prior outstanding Split Requests to Master-Abort on the PCI Bus. Since, the IOP is only accessing Prefetchable Memory on the host, no error condition is created in the system. PCI Interface Inbound Transaction: • Once the ATU and MU have detected that an internal bus reset is coming, they no longer claim any new transactions on the PCI bus. This results in a master abort to the initiating master. • In PCI-X mode, this may mean that data from an outstanding split request may not be returned to the host. It is the responsibility of the host software to handle this condition as a consequence of writing to the Internal Bus Reset bit. • The other requirement for the inbound ATU is to complete the configuration cycle which set the IB reset bit in the ATUs configuration space. In PCI-X Mode this means returning a Split Completion Message to the host indicating that the write has completed. Internal Bus Interface: Inbound Transaction: • The ATU and MU go ahead and assert their IB request signals, and try to continue any pending transactions as normal. There are no special actions taken on the internal bus for inbound transactions. Internal Bus Interface: Outbound Transaction: • For all ATU and MU outbound transactions, there are no special requirements since the BIU/Core/DMA is reset. Upon meeting both the outbound and inbound transaction requirements, the ATU and MU assert the sideband signal to the reset unit notifying ready-for-reset.	Clear all ATU and MU interrupts and HPIs. All ATU and MU MMRs reset to the default value.
DMA0/1/	No special requirements. DMA can be reset at any time.	Clear all interrupts and HPIs. All DMA MMRs reset to the default value.

Table 405. Internal Bus Reset Summary (Sheet 2 of 2)

Unit	Preparation for Reset	Reset Status
BIU	No special requirements. BIU can be reset at any time.	Clear all interrupts and HPIs. The BIU behaves as the Core after reset. Meaning, the CORE_RST# reset strap determines when the core is to be held in reset. All BIU mmrs reset to the default value.
AA	No special requirements. AA can be reset at any time.	Clear all interrupts and HPIs. All AA MMRs reset to the default value.
IB-ARB	No special requirements. IB-ARB can be reset at any time.	Clear all interrupts and HPIs. All IB-ARB MMRs reset to the default value.
I^2C	No special requirements. I^2C can be reset at any time.When the I^2C is in the middle of a transmission, the protocol may be violated when the I^2C unit resets.	Clear all interrupts and HPIs. All I^2C MMRs reset to the default value.
UART	No special requirements. UART can be reset at any time.When the UART is in the middle of a transmission, the protocol may be violated when the UART unit resets.	Clear all interrupts and HPIs. All UART MMRs reset to the default value.
MCU	No special requirements. MCU can be reset at any time.	Clear all interrupts and HPIs. All MCU MMRs reset to the default value. **M_RST#** is asserted.
PBI	No special requirements. PBI can be reset at any time.	Clear all interrupts and HPIs. All PBI MMRs reset to the default value.

18.2.5 Intel® XScale™ core Reset Mechanism

This reset is initiated the **CORE_RST#** reset strap, and exited via software through ATU PCSR register in PCI configuration space. Upon the de-assertion of the reset signal, the **CORE_RST#** is sampled as described in Section 18.3, "Reset Strapping Options" on page 762. When asserted, the Intel® XScale™ core processor is held in the reset state. Software is required to clear this bit to deassert Intel® XScale™ core reset. Software cannot set this bit.

18.2.6 Reset Summary

Table 406. Reset Summary

Reset Source	Affected Resets	Affected Units
P_RST#	All	Full Chip including Reset Straps
Secondary Bus Reset	S_RST#, **I_RST#**	Secondary bridge logic (not configuration register) and all IOP units.
Internal Bus Reset bit in ATU	**I_RST#**	All IOP units
Watch Dog Timer Expiration	**P_RST#**	Full chip. Applies only in no bridge mode with arbiter and central resources enabled (**BRG_EN** is low and **ARB_EN** is high)
	I_RST#	All IOP units. Applies to all other modes for 80331
CORE_RST#	Intel® XScale™ core reset	Intel® XScale™ core

18.3 Reset Strapping Options

There are many initialization modes that can be selected when the processor is reset. Table 407 shows the configuration modes. All of the configuration modes defined are determined on the rising edge of **P_RST#**.

Upon the deassertion of **P_RST#**, the 80331 samples a series of strapping pins to set configuration modes. One strap which alters the behavior of the 80331 on the deassertion of **P_RST#** is the **CORE_RST#** strap. When the **CORE_RST#** pin is asserted on the rising edge of **P_RST#**, the 80331 continues to assert the individual reset to the Intel® XScale™ core. This mode, holds the Intel® XScale™ core in reset until the Core Processor Reset Bit in the PCI Configuration and Status Register (ATU) is cleared, thus allowing the Intel® XScale™ core to enter its initialization procedure.

The Primary PCI interface of the 80331 samples the **P_REQ64#** signal to determine when 80331 is connected to a 64-bit data path. The central resource is required to drive the **P_REQ64#** signal low during the time that **P_RST#** is asserted. The state of **P_REQ64#** on the rising edge of the **P_RST#** signal notifies 80331 PCI-to-PCI Bridge primary bus interface it is connected to a 64-bit or 32-bit PCI bus. Similarly, the PCI interface of the ATU is integrated to the Secondary PCI bus of 80331 and therefore samples the **S_REQ64#** signal to determine when the secondary bus is connected to a 64-bit external data path.

The remaining reset straps are sampled similarly and are described in Table 407 below.

Table 407. Reset Strap Signals (Sheet 1 of 2)

NAME	DESCRIPTION
AD[4] / **P_BOOT16#**	Bus Width: **P_BOOT16#** is latched at the rising edge of **P_RST#** and it indicates the default bus width for the PBI Memory Boot window. 0 = 16 bits wide (Default mode). 1 = 8 bits wide (Requires pull-up resistor).
AD[6] / **RETRY**	Configuration Retry Mode: **RETRY** is latched at the rising edge of **P_RST#** and it determines when the PCI interface of the ATU disables PCI configuration cycles by signaling a Retry until the Configuration Cycle Retry bit is cleared in the PCI Configuration and Status Register. 0 = Configuration Cycles enabled (Requires pull-down resistor). 1 = Configuration Retry enabled (Default mode).
AD[5] / **CORE_RST#**	RESET MODE is latched at the rising edge of **P_RST#** and it determines when the Intel® XScale™ core is held in reset until the Intel® XScale™ Processor Reset bit is cleared in the PCI Configuration and Status Register. 0 = Hold in reset (Requires pull-down resistor). 1 = Don't hold in reset (Default mode).
AD[2] / **MEM_TYPE**	Memory Frequency: **MEM_TYPE** is latched at the rising edge of **P_RST#** and it indicates the speed of the DDR SDRAM interface. 0 = DDR-II SDRAM @ 400MHz (Requires pull-down resistor). 1 = DDR SDRAM @ 333MHz (Default mode).
AD[3] / **S_PCIX133EN**	Secondary PCI Bus Segment 133MHz Enable: **S_PCIX133EN** is latched at the rising edge of **P_RST#** and it determines the maximum PCI-X mode operating frequency. 0 = 100MHz enabled (Requires pull-down resistor). 1 = 133MHz enabled (Default mode).
AD[1]/**ARB_EN**	Internal Arbiter Enable: **ARB_EN** is latched at the rising edge of P_RST# and it determines when the PCI interface enables the integrated arbiter and central resource support, or use an external arbiter and external clock input. This arbiter and central resource support includes the four S_REQ#/S_GNT# arbitration pairs, the four **S_CLKO[3:0]** output clocks, the **S_CLKOUT**/**S_CLKIN** reference clock output/input signals and driving the appropriate PCI interface reset pattern. Applicable only when the **BRG_EN** is pulled low. 0 = Internal Arbiter/Central Resource disabled (Requires pull-down resistor). 1 = Internal Arbiter/Central Resource enabled (Default mode).
AD[20] / **PCIODT_EN**	**PCI Bus ODT ENABLE: PCIODT_EN** is latched on the rising edge of **P_RST#** and it determines when the PCI-X interface has On Die Termination enabled. PCI ODT enable is valid for 80331 secondary PCI bus only. 0 = ODT disabled (Requires pull-down resistor). 1 = ODT enabled (Default mode).
A[2]/ **P_32BITPCI#**	32 BIT PCI: **P_32BITPCI#** is latched at the rising edge of **P_RST#** and it indicates the width of the Primary PCI-X bus to the PCI-X Status Register. 0 = 32 Bit PCI-X Bus (Requires pull-down resistor). 1 = 64 Bit PCI-X Bus (Default mode).

Table 407. Reset Strap Signals (Sheet 2 of 2)

NAME	DESCRIPTION
A[1] / **PRIVMEM**	Private Memory Enable: **PRIVMEM** is latched at the rising edge of**P_RST#** and it determines when the 80331 operates with the Private Memory Space on the secondary PCI bus of the PCI-to-PCI Bridge. 0 = Normal addressing mode. (Requires pull-down resistor) 1 = Private Addressing enable in PCI-to-PCI Bridge. (Default mode).
A[0] / **PRIVDEV**	Private Device Enable: **PRIVDEV** is latched at the rising edge of**P_RST#** and it determines when the 80331 operates with Private Device enabled on the secondary PCI bus of the PCI-to-PCI Bridge. 0 = All Secondary PCI devices are accessible to Primary PCI configure cycles. (Requires pull-down resistor) 1 = Private Devices enabled in PCI-to-PCI Bridge. (Default mode).
AD[0] / **BRG_EN**	No Bridge Mode: **BRG_EN** is latched at the rising edge of**P_RST#** and it determines when the 80331 operates with the PCI-to-PCI Bridge. 0 = Disable bridge, enable external arbiter on **S_REQ0#/S_GNT[0]#** pins, enable **P_CLK** input on **S_CLKIN** input. (Requires pull-down resistor). 1 = Normal 80331 mode (Default mode).

Test Logic Unit and Testability 19

19.1 Overview

This chapter summarizes the testability features (DFT) that are incorporated in theIntel® 80331 I/O processor. The Test Logic Unit (TLU) is based on the IEEE 1149.1a *Standard Test Access Port and Boundary-Scan Architecture* and supports boundary scan.

19.2 Test Control/Observe Pins

The following table lists the complete set of test pins required.

Table 408. Test Control/Observe Pins

Pin Name	I/O	Test Purpose
TCK	I	Test Clock. Input clock for the test logic defined by IEEE1149.1Std. Maximum frequency: 200 MHz
TDI	I	Test Data Input. TAP controller defined by IEEE1149.1 Std receives serial test instructions and data via this pin.
TDO	O	Test Data Output. TAP controller defined by IEEE1149.1 Std sends the serial output from the instruction and data registers via this pin.
TMS	I	Test Mode Select. TAP controller defined by IEEE1149.1 decodes the signal received at this pin to **TMS** to control test operations.
TRST#	I	Test Reset. The optional **TRST#** input provides for asynchronous initialization of the TAP controller.

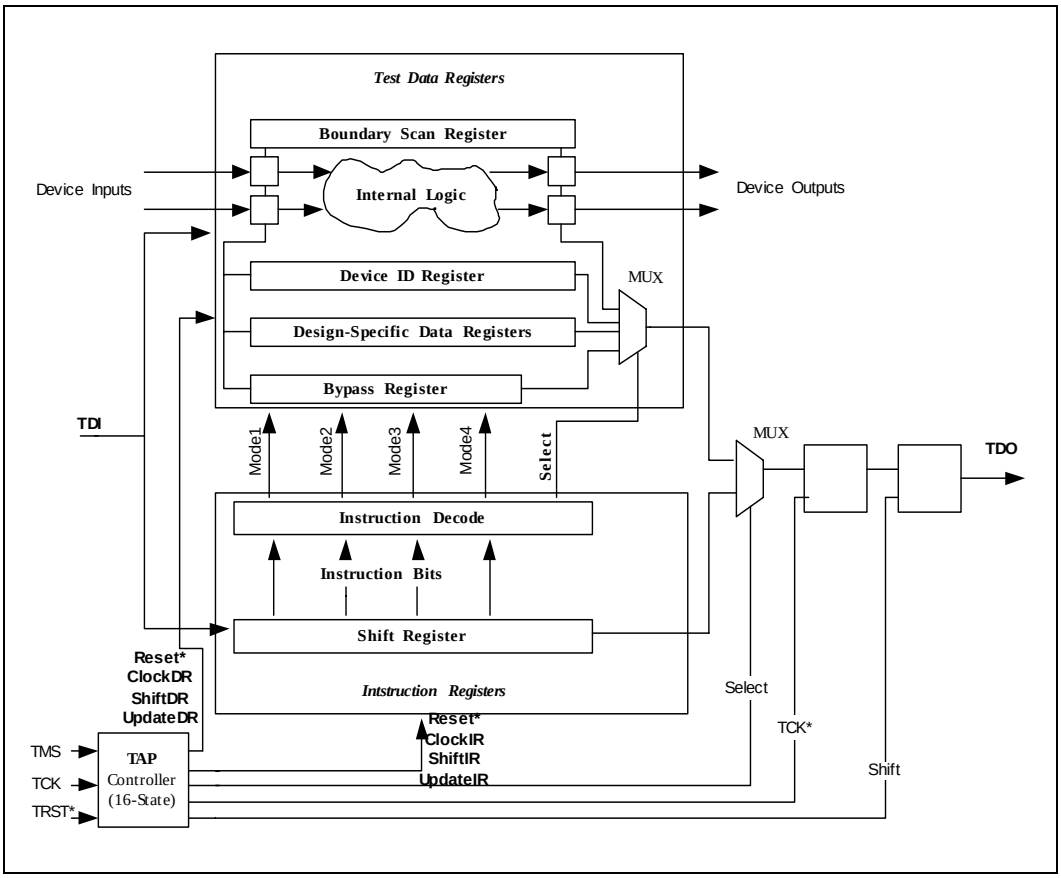
19.3 IEEE 1149.1 Standard Test Access Port (TAP)

The IOP contains test logic that is compatible with the IEEE Standard 1149.1-1990 Test Access Port (TAP) and Boundary Scan Architecture. Logic that conforms to this standard contains the following:

1. A Test Access Port (TAP) - four inputs (**TDI**, **TMS**, **TRST#** and TCLK) and a single output (**TDO**).
2. A TAP controller.
3. An instruction register.
4. A group of test data registers.

Each of these is described in more detail below. Figure 120 shows a generic diagram for logic conforming to the IEEE 1149.1 test standard.

Figure 120. IEEE 1149.1 Standard. Block Diagram



19.3.1 TAP Pin Description

The internal test logic is accessed through the TAP pins. The following sections describe some of the rules and permissions of the IEEE 1149.1a Standard for the TAP pins.

19.3.1.1 Test Clock (TCK)

This is the clock input for the test logic defined by this standard, i.e. the TAP controller and associated registers. The TLU is a fully static design, thus all registers retain their states indefinitely when **TCK** is stopped at "0" or "1".

19.3.1.2 Test Mode Select (TMS)

This pin is used to control the operation of the TAP controller. The signal received at **TMS** is decoded by the TAP controller to control test operations. The state of **TMS** is sampled on the rising edge of **TCK**. Internally, there is a weak pull-up on this pin to provide a logic high when not driven, per standard definition.

19.3.1.3 Test Data Input (TDI)

This pin is used to provide serial input data to the instruction and test data registers. Data at **TDI** is sampled on the rising edge of **TCK**. Data shifted from **TDI** through a register to **TDO** appears non-inverted at **TDO** after a number of rising and falling edges of **TCK** determined by the length of the instruction or test data register selected. Internally, there is a weak pull-up on this pin to provide a logic high when not driven, per standard definition.

19.3.1.4 Test Data Output (TDO)

This is the serial data output pin. Changes in the state of **TDO** occur only following the falling edge of **TCK** while performing a shift operation. This pin is only driven while scanning (in SHDR or SHIR states) otherwise it is in inactive (high Z) state. The non-shift inactive state is provided to support parallel connection of **TDO** outputs at the board or module level.

19.3.1.5 Asynchronous Reset (TRST#)

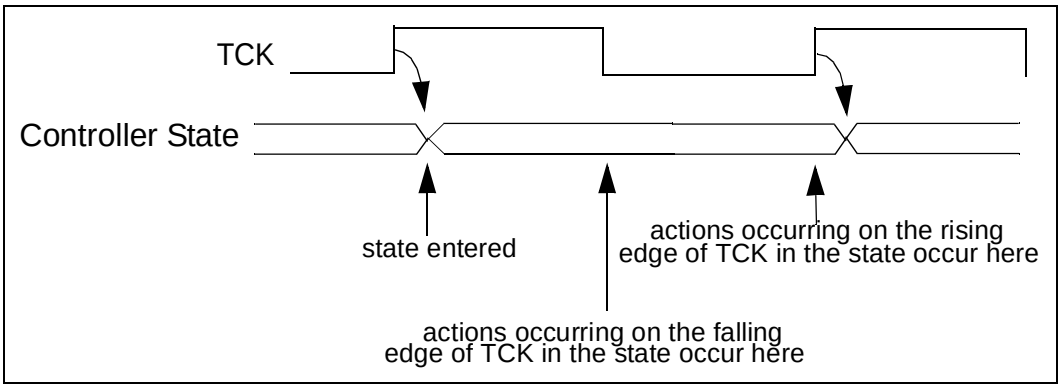
The **TRST#** signal is used to asynchronously reset the TAP controller and boundary-scan registers. The TAP controller is not initialized by any other system input, including system reset. The TAP controller initializes asynchronously on the falling edge of **TRST#** to the Test-Logic-Reset (initial) state. The TAP controller is initialized at power-up by cirrostrati built into the test logic. Upon reset, the TAP instruction register initializes to the IDCODE instruction. Internally, there is a weak pull-up on this pin to provide a logic high when not driven.

19.3.2 TAP Controller

The TAP controller, shown in Figure 122, is a sixteen-state synchronous finite state machine that changes state on the rising edge of **TCK**. The controller's next state is controlled by the state present at the **TMS** input. The TAP controller generates control signals, which together with **TCK** and control signals decoded from the instruction active in the instruction register, determine the operation of the test circuitry as defined by the IEEE Standard.

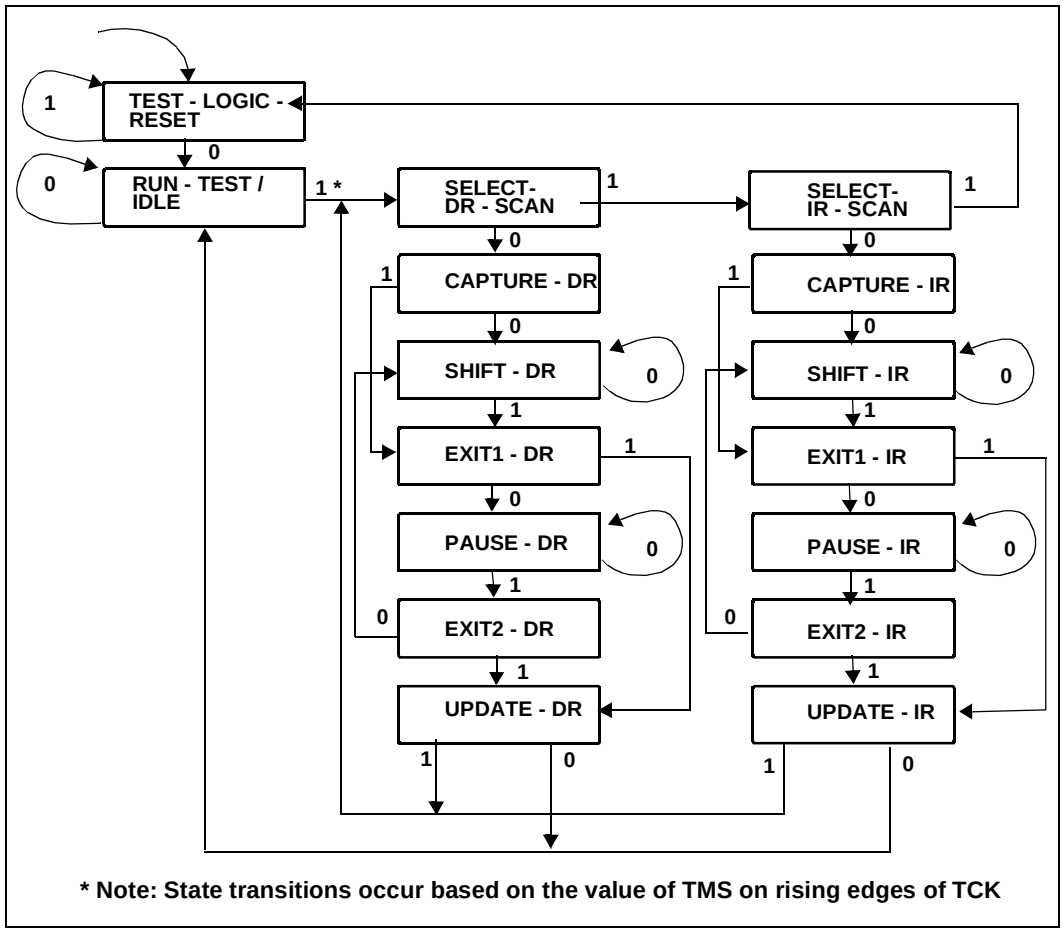
All state transitions occur based on the values of TMS on the rising edge of **TCK** Actions of the test logic (instruction register, data registers, etc.) occur on either the rising or falling edge of **TCK**, as show in Figure 121.. See the description of each state to learn which.

Figure 121. Timing of Actions in a TAP Controller State



For greater detail on the state machine and the public instructions, refer to IEEE 1149.1a Standard Test Access Port and Boundary-Scan Architecture Specification.

Figure 122. TAP Controller State Diagram



19.3.2.1 Test-Logic-Reset State

In this state, test logic is disabled to allow normal operation of the IOP. This is achieved by loading the instruction register with the IDCODE instruction. No matter what he state of the controller, it enters Test-Logic-Reset state when the **TMS** input is held high for at least five rising edges of **TCK**. The controller remains in this state while **TMS** is high. The TAP controller is also forced to enter this state by enabling **TRST#**.

When the controller exits the Test-Logic-Reset controller state as a result of an erroneous low signal on the **TMS** line at the time of a rising edge on **TCK** (for example, a glitch due to external interference), it returns to the Test-Logic-Reset state following three rising edges of **TCK** with the **TMS** line at the intended high logic level. Test logic operation is such that no disturbance is caused to on-chip system logic operation as the result of such an error.

Transition to next state: On the rising edge of **TCK**, when **TMS** is low move to Run-Test/Idle, else **TMS** remains high so stay in Reset.

19.3.2.2 Run-Test/Idle State

This state is a controller state between scan operations. The controller remains in this state as long as TMS is held low. In the Run-Test/Idle state, activity in selected test logic occurs only when certain instructions are present. Instructions that do not cause functions to execute generate no activity in the test logic while the controller is in this state.

The instruction register and all test data registers retain their current value in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Select-DR-Scan, else remain in Idle.

19.3.2.3 Select-DR-Scan State

This is a temporary controller state. Here the decision is made to enter the Capture-DR column and initiate a scan sequence for the selected test data register.

All test data registers selected by the current instruction retain their previous value in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Capture-DR, else move to Select-IR.

19.3.2.4 Capture-DR State

When the controller is in this state data is parallel-loaded into test data registers selected by the current instruction on the rising edge of **TCK**. Test data registers that do not have parallel inputs are not changed. Also when capturing is not required for the selected instruction, the register retains its previous state.

The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Shift-DR, else move to Exit1-DR.

19.3.2.5 Shift-DR State

In this controller state, the test data register, which is connected between **TDI** and **TDO** as a result of the current instruction, shifts data one bit position nearer to its serial output on each rising edge of **TCK**. Test data registers that the current instruction select but do not place in the serial path, retain their previous value during this state.

The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit1-DR, else remain at Shift-DR.

19.3.2.6 Exit1-DR State

This is a temporary controller state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is held high on the rising edge of **TCK**, the controller enters the Update-DR state and the scanning process terminates. When **TMS** is held low on the rising edge of **TCK**, the controller enters the Pause-DR state.

19.3.2.7 Pause-DR State

The Pause-DR state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between **TDI** and **TDO**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to the Exit2-DR, else remain at Pause-DR.

19.3.2.8 Exit2-DR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, the controller enters the Update-DR state and the scanning process terminates. When TMS is held low on the rising edge of **TCK**, the controller re-enters the Shift-DR state

19.3.2.9 Update-DR State

Data is latched into the parallel output of shift registers from the shift register path, on the falling edge of **TCK**.

All of the test data register's shift-register bit positions selected by the current instruction retain their previous values. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** remains high on the rising edge of **TCK**, then the controller moves to the Select-DR state, else the controller moves to the Run-Test/Idle state.

19.3.2.10 Select-IR-Scan State

This is a temporary controller state. Here the decision is made to enter the Capture-IR column and initiate a scan sequence for the instruction register or to return to Test-Logic-Reset.

All test data registers selected by the current instruction retain their previous value during this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Capture-IR, else move to Test-Logic-Reset.

19.3.2.11 Capture-IR State

In this state, the shift register contained in the instruction register loads the fixed value 0000001_2 on the rising edge of **TCK**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is low on the rising edge of **TCK**, move to Shift-IR, else move to Exit1-IR.

19.3.2.12 Shift-IR State

In this state, the shift register contained in the instruction register is connected between **TDI** and **TDO** and shifts data one bit position nearer to its serial output on each rising edge of **TCK**.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit1-IR, else remain at Shift-IR.

19.3.2.13 Exit1-IR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high during the next rising edge of **TCK**, the controller enters the Update-IR state and the scanning process terminates. When **TMS** is held low during the next rising edge of **TCK**, the controller enters the Pause-IR state.

19.3.2.14 Pause-IR State

This state allows the TAP controller to temporarily halt the shifting of data through the instruction register.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is high on the rising edge of **TCK**, move to Exit2-IR, else remain in the Pause-IR state.

19.3.2.15 Exit2-IR State

This is a temporary state.

All test data registers selected by the current instruction retain their previous value during this state. The instruction does not change while the TAP controller is in this state.

Transition to next state: When **TMS** is held high during the next rising edge of **TCK**, the controller enters the Update-IR state and the scanning process terminates. When **TMS** is held low during the next rising edge of **TCK**, the controller re-enters the Shift-IR state

19.3.2.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of **TCK**. Once latched, the new instruction becomes the current instruction.

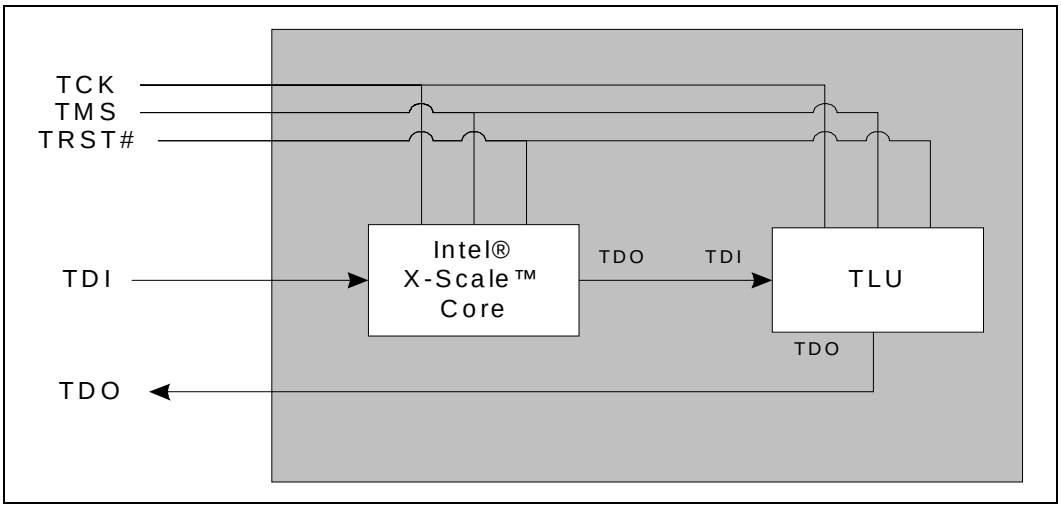
All test data registers selected by the current instruction retain their previous values in this state.

Transition to next state: When **TMS** remains high on the rising edge of **TCK**, then the controller moves to the Select-DR state, else the controller moves to the Run-Test/Idle state.

19.3.3 TAP Controller Configuration

The Test Logic Unit (TLU) and the Intel® XScale™ core each contain a separate TAP controller. They are connected in series through the TDI-TDO stream as shown in Figure 123.

Figure 123. TAP Controller Configuration



19.3.4 TAP Controller Registers

The IEEE 1149.1 architecture specifies an instruction register and a minimum of two test data registers: bypass and boundary scan. The IOP TAP controller meets this requirement.

19.3.4.1 Instruction Register

Each of the TAP controllers in the design contain an instruction register (IR). Each IR is a 7-bit, master/slave-configured, parallel-loadable, serial-shift register with latched outputs. They are used in each unit to select the test data register to be accessed.

When the TAP controller is in the Shift-IR state, instructions are loaded serially via **TDI** clocked by the rising edge of **TCK**. The shifted-in instruction becomes active upon latching from the serial-stages to the parallel-stages in the Update-IR state. At that time the IR outputs, along with the TAP finite state machine outputs, are decoded to select and control the test data register selected by that instruction. Upon latching, all actions caused by any previous instructions must terminate.

On activation of **TRST#**, the latched instruction asynchronously changes to the IDCODE instruction. Additionally, upon entering the Test-Logic-Reset state, the IDCODE instruction is latched and becomes active on the falling edge of **TCK.**

Because the TAP controllers are connected in series, each unit must receive its own 7-bit instruction during the Shift-IR state. The BYPASS instruction should be loaded into the unit that is not to be accessed.

For example, to load the IDCODE instruction into the TLUs IR, the data shifted into TDI during the Shift-IR state is:

```
 1 1 1 1 1 1 1 1 1 1 1 1 0 ------> TDI
| BYPASS into |    IDCODE   |
| X-Scale core  |   into TLU   |
```

As another example, to load the IDCODE instruction into the Intel® XScale™'s instruction register, the data shifted into TDI during the Shift-IR state is:

```
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 ------> TDI
| IDCODE into |    BYPASS   |
| X-Scale core  |    into TLU  |
```

19.3.4.2 Instructions

Since the instruction set for each TAP controllers is independent of one another, each is listed separately below.

Table 409. TLU TAP Controller Instruction Set

Instruction	Opcode	Description
EXTEST	0000000_2	Intended for supporting the boundary-scan feature for testing device interconnects at the board/system level, the EXTEST instruction connects only the boundary-scan register between TDI and TDO in the Shift-DR state. When EXTEST is selected, all output signal pin values are driven by values shifted into the boundary-scan register and may change only on the falling-edge of **TCK** in the Update_DR state. Also, when extest is selected, all system input pin states must be loaded into the boundary-scan register on the rising-edge of **TCK** in the Capture_DR state. Note: Data would typically be loaded onto the latched parallel outputs of boundary-scan shift registers using the SAMPLE/PRELOAD instruction prior to selection of the EXTEST instruction.
SAMPLE/ PRELOAD	0000001_2	SAMPLE/PRELOAD performs two functions: • When the TAP controller is in the Capture-DR state, the sample instruction occurs on the rising edge of **TCK** and provides a snapshot of the component's normal operation without interfering with that normal operation. The instruction causes Boundary-Scan register cells associated with outputs to sample the value being driven by or to the processor. • When the TAP controller is in the Update-DR state, the preload instruction occurs on the falling edge of **TCK**. This instruction causes the transfer of data held in the boundary-scan cells to the slave register cells. Typically the slave latched data is then applied to the system outputs by means of the extest instruction.
HIGHZ	0110001_2	HIGHZ puts all output pins into a tri-state mode. When this instruction is active, the bypass register is connected between **TDI** and **TDO**.
IDCODE	1111110_2	IDCODE is used in conjunction with the device identification register. When selected, IDCODE parallel-loads the hard-wired identification code (32 bits) into the identification register on the rising edge of **TCK** following entry into the Capture-DR state. The instruction selects only the identification register for connection between **TDI** and **TDO** in the Shift-DR state for serial access. NOTE: The device identification register is not altered by data being shifted in on **TDI**.
BYPASS	1111111_2	BYPASS selects the bypass register between TDI and TDO while in Shift-DR state, effectively bypassing the processor's test logic. '0' is captured in the Capture-DR state. While this instruction is in effect, all other test data registers have no effect on the operation of the system.

19.3.4.2.1 High-Z

High-Z aids in board-level testing. A mounted device effectively removes itself electrically from a circuit board. This allows for system-level testing where a remote tester exercises the processor system.

"HIGH-Z" is an instruction defined by the IEEE 1149.1 Standard. The requirement for this instruction are 1) all system logic outputs are high impedance; 2) the TAP controller continues to operate with the bypass register connected between TDI and TDO. The part may have other settings, as long as they do not interfere with these two requirements.

Note: Following the use of High-Z the part may be in an indeterminate state. Therefore, the IOP requires a reset in order to continue normal operation.

Table 410. Intel® XScale™ Core TAP Controller Instruction Set

Instruction	Opcode	Description
DBGRXx	0000010_2	Used by the Intel® XScale™ core debugger during software debug. See core EAS Ch 9 for more information.
LDIC	0000111_2	Used by the Intel® XScale™ core to load the instruction cache. Used during core burn-in testing. See core EAS Ch 9 for more information.
DCSR	0001001_2	Used by the Intel® XScale™ core debugger during software debug. See core EAS Ch 9 for more information.
DBGTXx	0010000_2	Used by the Intel® XScale™ core debugger during software debug. See core EAS Ch 9 for more information.
TRACE	0010001_2	Selects ETM registers.
IDCODE	1111110_2	IDCODE is used in conjunction with the device identification register. When selected, IDCODE parallel-loads the hard-wired identification code (32 bits) into the identification register on the rising edge of **TCK** following entry into the Capture-DR state. The instruction selects only the identification register for connection between **TDI** and **TDO** in the Shift-DR state for serial access. NOTE: The device identification register is not altered by data being shifted in on **TDI**.
BYPASS	1111111_2	BYPASS selects the bypass register between TDI and TDO while in Shift-DR state, effectively bypassing the processor's test logic.'0' is captured in the Capture-DR state. While this instruction is in effect, all other test data registers have no effect on the operation of the system.

19.3.4.3 Boundary-Scan Register

The boundary-scan register is a set of serial-shiftable register cells, connected between each of the system pins and the on-chip system logic (power, ground and TAP pins excluded). This forms a single shift-register between TDI and TDO of all the system pins.

This is the most extensive, complex register in the test circuitry. This register allows testing of circuitry external to the component (e.g. board interconnect) in addition to device system logic. It permits the system signals (into and out of the system logic) to be sampled and examined without causing interference with the normal operation of the system logic. Further definition, rules, and specifics of the boundary-scan register can be found in the IEEE Std, 1149.1-1990, Chapter 10.

19.3.4.4 Bypass Register

The bypass register is a single-bit, serial-shift register that connects **TDI** and **TDO** when the BYPASS instruction is in effect. This allows rapid movement of test data to and from other components on the board, since this register provides the shortest path between **TDI** and **TDO**. This path can be selected when no test operation is being performed. While the bypass register is selected, data is transferred from **TDI** to **TDO** without inversion.

19.3.4.5 Device Identification Register

The device identification (ID) register is a 32-bit register used for storing the manufacturer identification, part number, and the version of the processor. It is a dedicated part of the test logic and is not usable in system functionality.

The identification register is selected only by the IDCODE instruction. When the Test-Logic-Reset state of the TAP controller is entered, the IDCODE instruction is automatically loaded into the instruction register. The generic format of the register is discussed in chapter 11 of the IEEE 1149.1 Standard.

Figure 124. Device ID Register

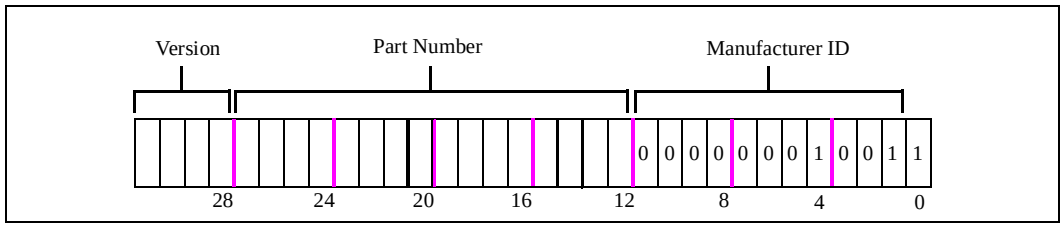


Table 411. Device ID Register Field Definitions

Field	Definition
Version	Indicates stepping changes.
Part Number	part number
Manufacturer ID	Manufacturer ID assigned by IEEE. (00000001001h indicates Intel)
1	IEEE 1149.1 requirement

Table 412. Intel® XScale™ Core Device ID Register Settings

Intel® 80331 I/O processor	0x09279013	0000 1001 0010 0111 1001 0000 0001 0011

Table 413. TLU Device ID Register Settings

Intel® 80331 I/O processor	0x09279013	0000 1001 0010 0111 1001 0000 0001 0011

The TLU and the Intel® XScale™ core both have the same device ID number.

This Page Intentionally Left Blank