# TMS320DM646x DMSoC Peripheral Component Interconnect (PCI)

# User's Guide

Literature Number: SPRUER2B
November 2009

## List of Figures

## List of Tables

# Read This First

## About This Manual

This document describes the peripheral component interconnect (PCI) module in the TMS320DM646x Digital Media System-on-Chip (DMSoC). The DM646x DMSoC PCI is compliant to the *PCI Local Bus Specification* (revision 2.3). See that document for details on the protocol, electrical, and mechanical specifications of the PCI.

## Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

## Related Documentation From Texas Instruments

The following documents describe the TMS320DM646x Digital Media System-on-Chip (DMSoC). Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the DM646x DMSoC, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

**SPRUEP8 — *TMS320DM646x DMSoC DSP Subsystem Reference Guide.*** Describes the digital signal processor (DSP) subsystem in the TMS320DM646x Digital Media System-on-Chip (DMSoC).

**SPRUEP9 — *TMS320DM646x DMSoC ARM Subsystem Reference Guide.*** Describes the ARM subsystem in the TMS320DM646x Digital Media System-on-Chip (DMSoC). The ARM subsystem is designed to give the ARM926EJ-S (ARM9) master control of the device. In general, the ARM is responsible for configuration and control of the device; including the DSP subsystem and a majority of the peripherals and external memories.

**SPRUEQ0 — *TMS320DM646x DMSoC Peripherals Overview Reference Guide.*** Provides an overview and briefly describes the peripherals available on the TMS320DM646x Digital Media System-on-Chip (DMSoC).

**SPRAA84 — *TMS320C64x to TMS320C64x+ CPU Migration Guide.*** Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) to the TMS320C64x+ DSP. The objective of this document is to indicate differences between the two cores. Functionality in the devices that is identical is not included.

**SPRU732 — *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.*** Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

**SPRU871 — *TMS320C64x+ DSP Megamodule Reference Guide.*** Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

User's Guide
SPRUER2B–November 2009

# Peripheral Component Interconnect (PCI)

## 1 Introduction

This document describes the peripheral component interconnect (PCI) module in the TMS320DM646x Digital Media System-on-Chip (DMSoC). The DM646x PCI is compliant to the *PCI Local Bus Specification* (revision 2.3). See that document for details on the protocol, electrical, and mechanical specifications of the PCI.

### *1.1 Purpose of the Peripheral*

The DM646x PCI module allows communication with devices compliant to the *PCI Local Bus Specification* (revision 2.3) via a 32-bit address/data bus operating at speeds up to 33 MHz and up to 66 MHz (for DM6467T devices only).

### *1.2 Features*

The PCI module supports the following features:
- PCI Local Bus Specification (revision 2.3) compliant
- Single function PCI interface provided
- 32-bit address/data bus width
- Operation up to 33 MHz and up to 66 MHz (for DM6467T devices only)
- Optimized burst behavior supported for system cache line sizes of 16, 32, 64 and 128 bytes
- PCI is only accessible from the ARM.

The PCI operates as a PCI slave device for configuration cycles and memory cycles. It also acts as a PCI master device for configuration cycles, IO cycles, and memory accesses to other devices.

As a slave, the PCI includes the following features:
- Response to accesses as a 32-bit agent with medium DEVSEL timing (single wait state)
- Direct support of the Memory Read, Memory Read Multiple, Memory Read Line, Memory Write, Configuration Read and Configuration Write transactions
- Aliases Memory Write and Invalidate to the Memory Write command
- Support of variable length burst transfers up to a cache line for Memory Read Line transactions
- Support of unlimited length burst transfers for Memory Read Multiple and Memory Write transactions
- Support of single data phase transfers with disconnect for Memory Read, Configuration Read and Configuration Write transactions
- Support of both immediate or timeout forced delayed transactions for Memory Read, Memory Read Line, and Memory Read Multiple transactions
- Support of posting of Memory Write transactions
- Support of up to six base address registers (PCIBAR0-PCIBAR5)
- Support of programmable cache line size of 4, 8, 16, 32, 64, or 128 bytes
- Ports provided to set configuration space registers to specific values after reset

As a master, the PCI includes the following features:

- Transaction initiation as a 32-bit agent
- Support of the Configuration Read, Configuration Write, IO Read, IO Write, Memory Read, Memory Read Line, Memory Read Multiple, Memory Write, and Memory Write and Invalidate PCI Bus commands
- Support of bursts transfers of up to 256 data phases for Memory Read Line, Memory Read Multiple, and Memory Write transactions
- Support of single data phase transfers for Memory Read transactions
- Automatic selection between Memory Read, Memory Read Line, and Memory Read Multiple based on the requested transaction length and the cache line size
- Assertion of the $\overline{\text{PCI\_IRDY}}$ signal one clock cycle after the $\overline{\text{PCI\_FRAME}}$ signal is asserted. For optimal performance, it does not insert wait states during a burst.

## 1.3 Features Not Supported

The PCI Module does not support:

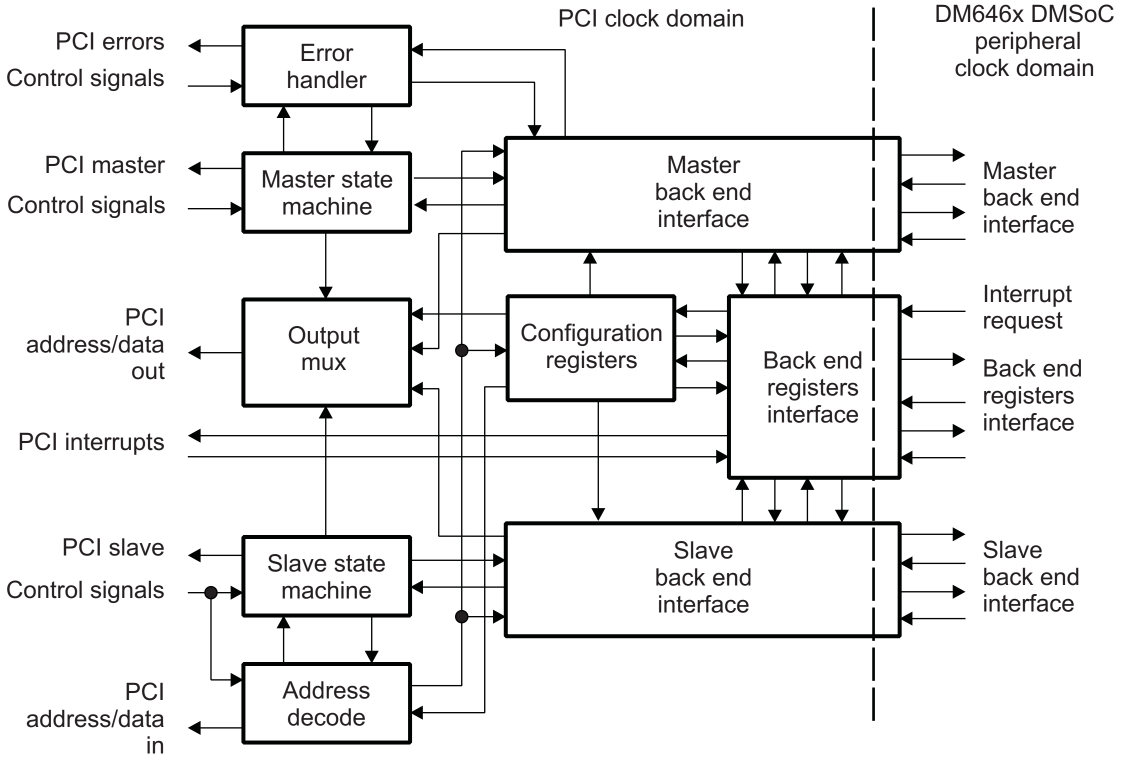
- PCI special cycles
- PCI interrupt acknowledge cycles
- PCI lock
- 64-bit bus operation
- Operation at frequencies greater than 33 MHz for DM646x devices (operation up to 66 MHz is supported for DM6467T devices only)
- Address/data stepping
- Combining (for write posting)
- Collapsing
- Merging
- Cache line wrap accesses
- Reserved accesses
- Message signaled interrupts
- Vital product data
- Slave IO Read and IO Write Transactions
- Accessing the PCI peripheral from the DSP on the DM646x DMSoC

## 1.4 Functional Block Diagram

The PCI (Figure 1) consists of the following blocks:

- Address decoder
- Slave state machine
- Slave back end interface
- Master back end interface
- Master state machine
- Output multiplexer
- Configuration registers
- Error handler
- Back end registers interface

**Figure 1. PCI Block Diagram**



### 1.4.1 Address Decoder

This block latches transaction control information from the PCI bus and decodes that information to determine if the transaction was targeted to its PCI slave. This block instructs the Slave State machine to either accept or ignore slave transactions as they are presented on the PCI bus.

### 1.4.2 Slave State Machine

This block generates and monitors all of the PCI signals necessary for accepting transactions on the bus. All of the slave PCI protocols handling functions are split between the address decoder and slave state machine blocks.

### 1.4.3 Slave Back End PCI Interface

This block accepts transactions from the slave state machine and passes those transactions to the back end interface. This block performs the asynchronous decoupling between the PCI clock domain and the peripheral clock domain for slave transactions. It also implements the slave address translation control registers and performs address translation for slave transactions.

### 1.4.4 Master Back End PCI Interface

This block accepts bus transactions from the back end interface and passes those transactions on to the master state machine. It performs the asynchronous decoupling between the peripheral clock domain and the PCI clock domain for master transactions. This block implements the master address translation control registers and also performs the address translation for master transactions.

### 1.4.5 Master State Machine

This block generates and monitors all of the PCI signals necessary for initiating transactions on the bus. The majority of the master PCI protocols handling functions are implemented in this block. This block responds to transfer requests that are presented to it from the master back end interface.

### 1.4.6 Output Multiplexer

This block multiplexes the master address, master write data, slave configuration read data, and slave memory read data on to the PCI_AD[31:0] pins at the appropriate times. This block is controlled by several of the other blocks in the PCI.

### 1.4.7 Configuration Registers

This block implements the required PCI configuration registers and some of the back end registers. These registers control the modes and options in the PCI and provide vital information to the PCI host.

### 1.4.8 Error Handler

This block monitors for error conditions that may occur on the PCI bus.

### 1.4.9 Back End Registers Interface

This block implements the asynchronous bridging function that allows DM646x DMSoC masters to access select PCI configuration registers, the master and slave address translation registers, and other miscellaneous PCI control/status registers. This block also implements some PCI control/status registers that reside in the back end peripheral clock domain.

## *1.5 Terminology Used in This Document*

The following is a brief explanation of some terms used in this document:

| Term | Meaning |
|---|---|
| Back End | The internal infrastructure of the DM646x DMSoC. |
| Host | A PCI-capable device other than the DM646x DMSoC on the system that initially enumerates the PCI devices on the PCI bus. It is also capable of booting the DM646x DMSoC via the option of a PCI boot. |

## *1.6 Industry Standard(s) Compliance Statement*

The PCI module is compliant with the *PCI Local Bus Specification* (revision 2.3).

# 2 Architecture

## *2.1 Clocks*

The PCI module uses the following clocks:

- Main PCI clock from the PCI_CLK pin
  - 33 MHz from PCI_CLK pin when in 33 MHz mode
  - 66 MHz from PCI_CLK pin when in 66 MHz mode (for DM6467T devices only)
- Internal DM646x DMSoC peripheral clock
  - Sourced by the DM646x DMSoC PLL controller, see the device-specific data manual for more information.
  - Peripheral clock frequency must not be less than the main PCI clock frequency.

## 2.2 Signal Descriptions

Figure 2 lists the PCI signals that are used by the PCI. Table 1 shows the PCI pin name with the signal direction and description.

**Figure 2. PCI Signals**

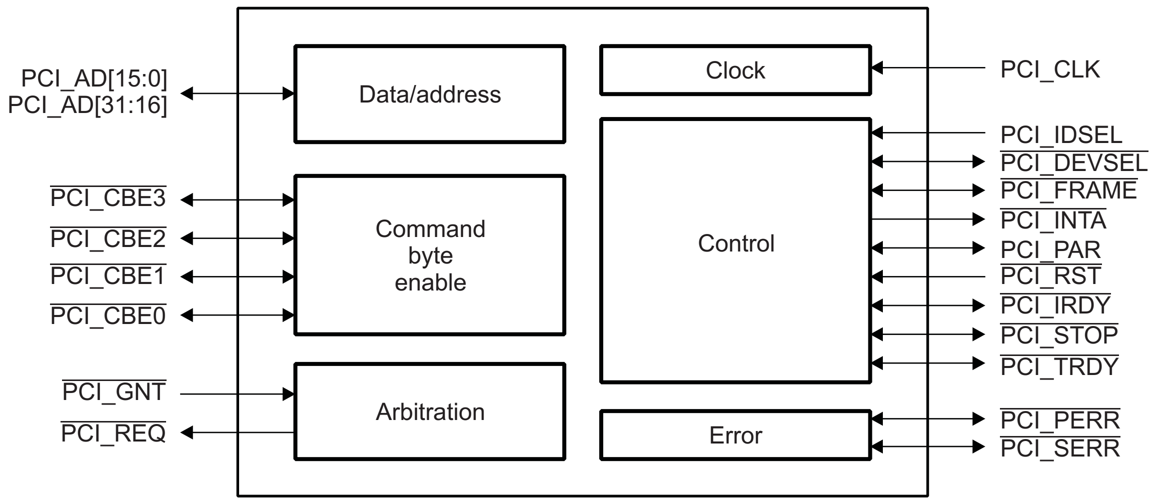


**Table 1. PCI Pin Description**

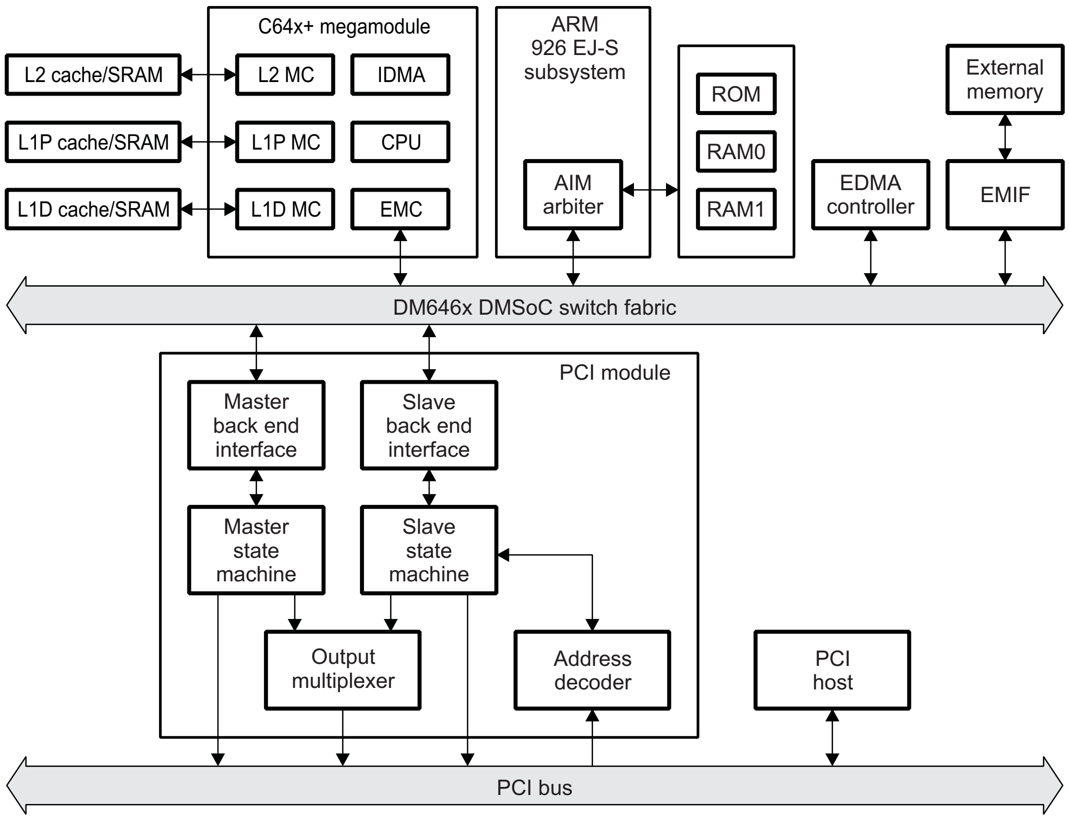
| Pin Name | Type [(1)] | Description |
|---|---|---|
| PCI_FRAME | I/O/Z | PCI Cycle Frame |
| PCI_DEVSEL | I/O/Z | PCI Device Select |
| PCI_STOP | I/O/Z | PCI Transaction Stop Indicator |
| PCI_CLK | I | PCI Clock |
| PCI_CBE[3:0] | I/O/Z | PCI Command/Byte Enables |
| PCI_PAR | I/O/Z | PCI Parity |
| PCI_PERR | I/O/Z | PCI Parity Error |
| PCI_SERR | I/O/Z | PCI System Error |
| PCI_IRDY | I/O/Z | PCI Initiator Ready |
| PCI_TRDY | I/O/Z | PCI Target Ready |
| PCI_REQ | O/Z | PCI Bus Request |
| PCI_INTA | O/Z | PCI Interrupt A |
| PCI_RST | I | PCI Reset |
| PCI_GNT | I | PCI Bus Grant |
| PCI_IDSEL | I | PCI Initialization Device Select |
| PCI_AD[31:16] | I/O/Z | PCI Address/Data bus [31:16] |
| PCI_AD[15:0] | I/O/Z | PCI Address/Data bus [16:0] |

(1) I = Input, O = Output, Z = High impedance.

### 2.2.1 Connecting a Local PCI to an External PCI Device

Figure 3 shows a simplified block diagram of how the PCI module interfaces local ARM master modules (EDMA controller, CPU, etc.) and other DM646x DMSoC resources (EMIF, DSP and ARM internal memory, etc.) to external PCI memory and external PCI masters.

**Figure 3. PCI to External PCI device**



(1) EDMA: Enhanced Direct Memory Access Controller
EMIF: External Memory Interface
EMC: Extended Memory Controller
L1P MC: L1 P Memory Controller
L1D MC: L1D Memory Controller
L2 MC: L2 Memory Controller
IDMA: Internal Direct Memory Access Controller

The following steps show how the PCI module interfaces local DM646x DMSoC master modules (EDMA controller, CPU, ARM, etc.) to external PCI memory:

1. An ARM master initiates a transaction aimed at external PCI memory through the DM646x DMSoC switch fabric.
2. The address is decoded by the PCI master back end interface.
3. The PCI master back end interface claims the transaction if the DM646x DMSoC address falls within the master memory map (described in Section 2.6.1).
4. PCI master back end interface translates the DM646x DMSoC address into a PCI address and generates a request to the master state machine.
5. The master state machine initiates a transaction on the PCI bus using the PCI address.
6. The request is received by the external PCI host, which responds accordingly.

The following steps show how the PCI module interfaces external PCI masters to DM646x DMSoC resources (EMIF, DSP and ARM internal memory, etc.):

1. External PCI master initiates a transaction on the PCI bus.
2. The PCI address decoder decodes the PCI address of the transaction and instructs the PCI slave state machine to claim the transaction if the PCI address falls within the slave memory map (described in Section 2.5.1) assigned to the DM646x DMSoC.
3. The PCI slave state machine forwards the request to the PCI slave back end Interface.
4. PCI slave back end Interface translates the PCI address into a DM646x DMSoC address and places the DM646x DMSoC address on the DM646x DMSoC switch fabric.
5. All DM646x DMSoC slaves decode the address to determine if they are being accessed. If so, they respond accordingly. For example, in the case of an external memory access, the EMIF accesses external memory using the DM646x DMSoC address.

## 2.3 Pin Multiplexing

On the DM646x DMSoC, the PCI peripheral is pin multiplexed with the asynchronous external memory interface (EMIF) to accommodate multiple peripheral functions in a smaller possible package. Pin multiplexing is controlled by using a combination of hardware configuration at device reset and software programmable register settings. Refer to the device-specific data manual to determine how pin multiplexing affects the PCI module.

## 2.4 Byte Addressing

The PCI interface is byte-addressable. It can read and write 8-bit bytes, 16-bit half words, 24-bit words, and 32-bit words. Words are aligned on an even four-byte boundary, and always start at a byte address where the two LSBs are 00. Halfwords always start at a byte address where the last LSB is 0. PCI slave transactions are fully byte-addressable, but PCI master transactions must start on a word-aligned address.

## 2.5 PCI as Slave

### 2.5.1 Slave Memory-Map

The PCI module on the DM646x DMSoC provides full visibility for a PCI host into the DM646x DMSoC memory through six sets of PCI slave base address translation registers (PCIBAR0TRL-PCIBAR5TRL) and PCI base address mask registers (PCIBAR0MSK-PCIBAR5MSK). The ARM can use any of these sets of registers to map any memory region or memory-mapped registers (MMRs) to the PCI memory-map. These registers can be configured by software at any time. The default values of these registers provide the mapping shown in Table 2.

Section 2.5.1.1 and Section 2.5.1.2 explain how to map a region in the PCI host address space to a region in the DM646x DMSoC memory space by setting up a slave window.

**Table 2. PCI Base Addresses**

| Base Address | Window Size (1) | Prefetchable (1) | Memory Space |
|---|---|---|---|
| 0 | 16 KB | Yes | ARM TCM RAM 0 |
| 1 | 32 KB | No | DDR2 Control Registers |
| 2 | 4 MB | No | Chip-Level MMRs(2) |
| 3 | 128 KB | Yes | GEM L2 RAM |
| 4 | 8 MB | Yes | DDR2 Memory |
| 5 | 8 MB | Yes | DDR2 Memory |

(1) The default values can be changed by the device's PCI bootloader code. Refer to the PCI bootloader code documentation for more information.

(2) Only certain regions within this 4MB (01C0 0000h-0200 0000h) are accessible. The regions accessible for read and write include MMRs of McASP0, McASP1, ATA, UART0, UART1, UART2, GPIO, PWM0, PWM1, I2C, SPI, Timer0, Timer1, Timer2, USB, HPI, EMAC, MDIO, PLLC0, PLLC1, PSC, CRGEN0, CRGEN1, PCI, and System Module Registers. Access to the MMRs of other peripherals is not supported.

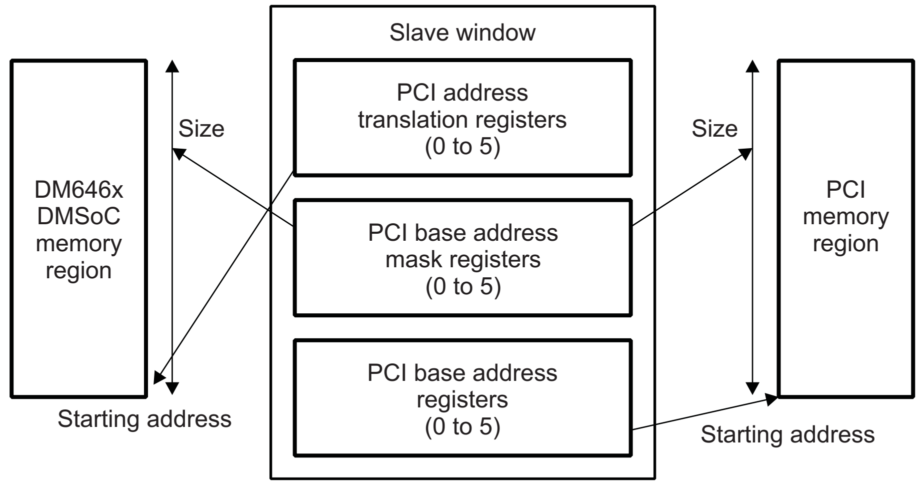
#### *2.5.1.1 Configuring Slave Window Registers*

Figure 4 displays a slave window configuration. A slave window maps a region in the DM646x DMSoC memory space to a region in the PCI address space. This allows a PCI host to access the DM646x DMSoC memory through the PCI address space. A slave window is configured with the following registers:

- PCI slave base address translation register (PCIBAR*n*TRL): Configures the starting address of the window in the DM646x DMSoC address space
- PCI base address register (PCIBAR*n*): Configures the starting address of the slave window in the PCI address space
- PCI base address mask register (PCIBAR*n*MSK): Configures the size of the window and prefetchability of the DM646x DMSoC memory region being mapped

PCI supports six slave window configurations with the support of these registers. For more information on slave access address translation, see Section 2.5.1.2.

Figure 4. Slave Window Configuration


Slave window
PCI address translation registers (0 to 5)
PCI base address mask registers (0 to 5)
PCI base address registers (0 to 5)
DM646x DMSoC memory region
Size
Starting address
PCI memory region
Size
Starting address

#### 2.5.1.1.1 Configuration of Base Address Registers (PCIBAR*n*) by PCI Host

The PCI base address registers (PCIBAR0-PCIBAR5) allow the PCI host to map the DM646x DMSoC address space into the host memory or I/O address space. The base address registers reside in configuration space and a PCI host normally configures them. The PCI host can access the base address registers (PCIBAR0-PCIBAR5) by performing a TYPE 0 access in the PCI bus.

The base address registers (PCIBAR*n*) contain the following bit fields:

- ADDR (bits 31-4): These bits specify the base address of the slave window on the PCI address space.
- PREFETCH (bit 3): This bit specifies the prefetchability of the memory space controlled by the base address register.
- TYPE (bits 2:1): These bits specify whether the base address maps to PCI I/O address space or memory address space. The DM646x PCI supports only mapping into PCI memory space.
- IOMEM_SP_IND (bit 0): The size of the base address register, either 32 or 64 bits. The DM646x PCI only supports 32-bit addressing.

Normally, a PCI host configures the ADDR bits of the base address registers during its boot time when it enumerates all the PCI devices. The write-access of the PCI host to each of the ADDR bits is determined by the corresponding bit in the address mask (ADDRMASK) bits of the base address mask register (PCIBAR*n*MSK). A bit in ADDR is read-only to the PCI host when its corresponding bit in ADDRMASK is cleared. Conversely, a bit in ADDR can be both read and written by the PCI host when its corresponding bit in ADDRMASK is set. The DM646x DMSoC is required to complete the configuration of the address mask register before the host attempts to configure the base address registers.

#### 2.5.1.1.2 Configuration of Slave Base Address Translation Registers (PCIBAR*n*TRL) by ARM

The PCI slave base address translation registers (PCIBAR0TRL-PCIBAR5TRL) configure the DM646x DMSoC side parameters of a slave window. There are six slave base address translation registers (PCIBAR0TRL-PCIBAR5TRL) that allow six slave windows to be set up. The slave base address translation registers are configured by the ARM. The slave base address translation registers control the translation of transaction addresses as they flow from the external PCI bus to the DM646x DMSoC. Section 2.5.1.2 explains the translation of PCI addresses to DM646x DMSoC addresses.

#### 2.5.1.1.3 Configuration of PCI Base Address Mask Registers (PCIBAR*n*MSK) by ARM

The PCI base address mask registers (PCIBAR0MSK-PCIBAR5MSK) configure the size and prefetchability of a slave window. The base address mask registers are configured by the ARM. The ARM can access the PCI base address mask registers directly, as they are mapped to the DM646x DMSoC memory space.

The base address mask registers (PCIBAR*n*MSK) contain the following bit fields:

- ADDRMASK (bits 31-4): These bits control the PCI host write-access of the corresponding bits in the corresponding PCI configuration base address registers.
- PREFETCH_EN (bit 3): This bit specifies whether or not the memory space controlled by the corresponding PCI configuration base address register is prefetchable. This bit is reflected in bit 3 of the corresponding PCI configuration base address register.

The ARM should configure this register before the host attempts to program the base address register.

### 2.5.1.2 Slave Access Address Translations

Window configurations control the translation of transaction addresses as they flow from the external PCI bus to the DM646x DMSoC. This translation process uses the contents of the corresponding PCI base address mask register (PCIBAR*n*MSK) to determine which of the bits in the PCI address should be modified. Bits 31 to 4 (ADDRMASK) are replaced in the address where the corresponding bit in the base address mask register is set by the corresponding bit in the slave base address translation register.

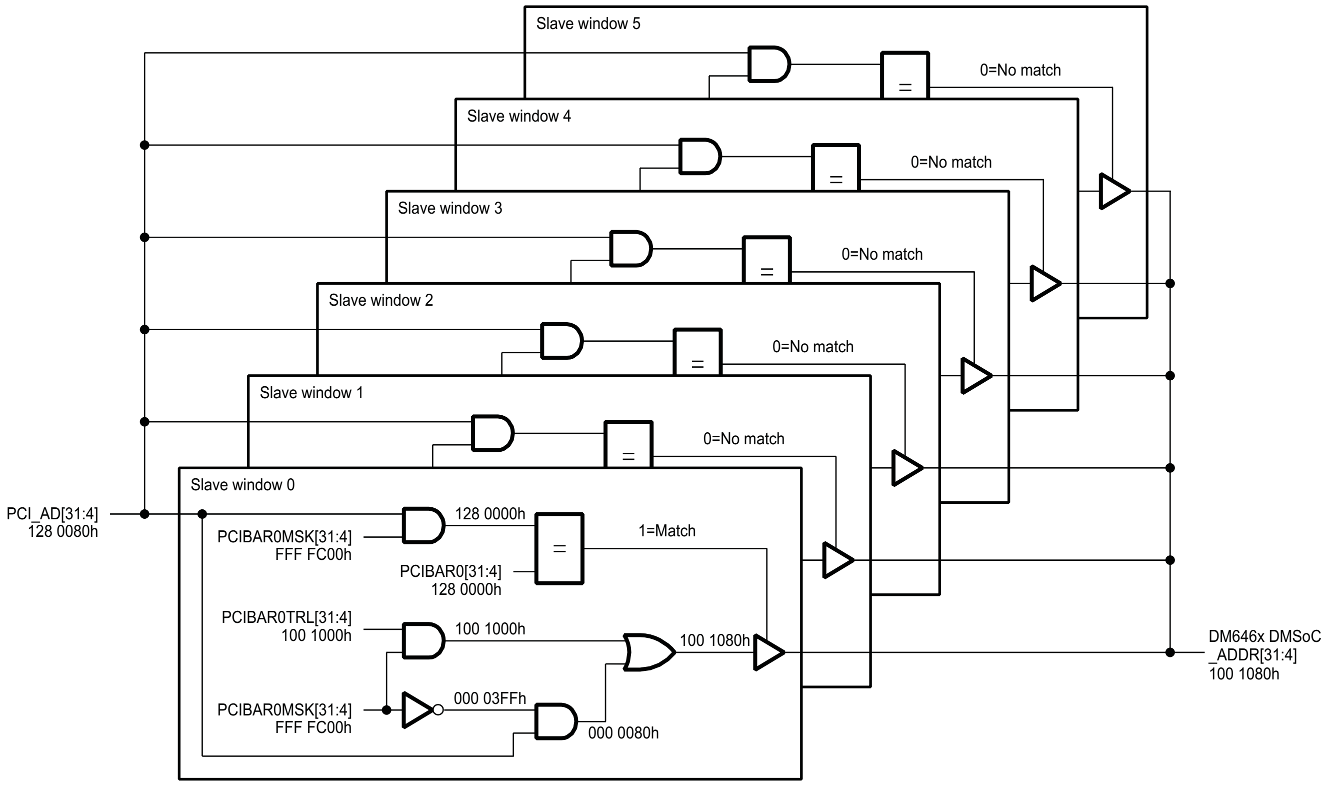
The following steps occur during a PCI-to-DM646x DMSoC address translation:

1. During the address phase, the external PCI master places the PCI address on the address bus PCI_AD[31:0].
2. The PCI finds the appropriate slave window for the address by comparing the address bits given on PCI_AD[31:N] with the corresponding bits in the base address register of all the slave windows one by one. The value of N is the number of bits set in the corresponding base address mask register of the slave window. The minimum value of N is 4. The value of N indicates the number of significant bits in the PCI address that needs to be decoded. If the address on PCI_AD[31:N] matches the corresponding bits of the base address register of any one of the slave windows, the PCI claims the PCI transaction. Otherwise, it ignores the transaction.
3. If the PCI claims the transaction, it generates a DM646x DMSoC address by replacing bits 31:N in the PCI address with the corresponding bits in the base address translation register of the previously selected slave window.

Figure 5 gives an example of a PCI-to-DM646x DMSoC address translation using a PCI address of 1280 0800h. In this example, slave window 0 is created using this configuration: PCIBAR0 = 1280 0000h, PCIBAR0MSK = FFFF C000h, and PCIBAR0TRL = 1001 0000h. With these settings, slave window 0 translates PCI addresses from 1280 0000h to 1280 3FFFh (16KB) to DM646x DMSoC addresses 1001 0000h to 1001 3FFFh. The following is the sequence of events for generating the DM646x DMSoC address:

1. The external PCI master places the PCI address 1280 0800h on the address bus PCI_AD[31:0].
2. The PCI finds the slave window corresponding to the PCI address by comparing the address given on PCI_AD[31:4] with the corresponding bits in the PCI base address registers. The PCI base address mask registers indicate bits in the PCI address that need to be compared.
3. Slave window 0 has PCIBAR0[31:4] set to 1280 000h and PCIBAR0MSK[31:4] = FFFF C00h. Therefore, the PCI matches the PCI address with slave window 0 and claims transaction.
4. The value of the PCIBAR0MSK[31:4] bits is inverted and ANDed with the PCI address (PCI_AD[31:4]). The PCIBAR0TRL[31:4] bits are also ANDed with the value of the PCIBAR0MSK[31:4] register. The resulting values are ORed together to form the DM646x DMSoC address (1001 0800h).

**Figure 5. PCI-to-DM646x DMSoC Address Translation**



### 2.5.2 Slave Operations

The PCI slave operates in response to transfer requests that are presented on the PCI bus. The PCI slave was intended to enable high performance read and write performance through the use of delayed transactions combined with prefetching for reads and posting for writes. The PCI slave supports two FIFOS/buffers (a read and write) for efficient data transfer. Each buffer holds 16 32-bit words of read or write data.

#### *2.5.2.1 Slave Configuration Operations*

##### *2.5.2.1.1 Configuration Write Transactions*

The decoding of and response to a configuration write transaction by the PCI slave depends on the following:

- PCI_CBE[3:0] must be Bh during the address phase
- PCI_IDSEL must be asserted during the address phase
- PCI_AD[1:0] must be 00b during the address phase

If the above conditions are met, the transaction is decoded as a hit and the PCI slave will assert PCI_DEVSEL using medium decode timing. PTRDY will be asserted coincident with the assertion of PCI_DEVSEL. If the master on the PCI bus intends to perform more than a single data phase transaction (as determined by the state of PCI_FRAME), PCI_STOP will also be asserted coincident with the assertion of PCI_DEVSEL and PTRDY to signal a disconnect. PCI_STOP will continue to be asserted until PCI_FRAME is deasserted and PCI_IRDY is asserted in accordance with the PCI specification.

Configuration write transactions will never result in a retry. Because the configuration registers are included within the PCI, no transactions will occur on any of the back end interfaces as a result of a configuration write transaction.

##### *2.5.2.1.2 Configuration Read Transactions*

Decoding of and response to a configuration read transaction by the PCI slave depends on the following:

- PCI_CBE[3:0] must be Ah during the address phase.
- PCI_IDSEL must be asserted during the address phase.
- PCI_AD[1:0] must be 00b during the address phase.

If the above conditions are met, the transaction is decoded as a hit and the PCI slave will assert PCI_DEVSEL using medium decode timing. PTRDY will be asserted one cycle following the assertion of PCI_DEVSEL. If the master on the PCI bus intends to perform more than a single data phase transaction (as determined by the state of PCI_FRAME), PCI_STOP will also be asserted coincident with the assertion of PTRDY to signal disconnect. PCI_STOP will continue to be asserted until PCI_FRAME is deasserted and PCI_IRDY is asserted in accordance with the PCI specification.

Configuration read transactions will never result in a retry. Because the configuration registers are included within the PCI, no transactions will occur on any of the PCI blocks as a result of a configuration read transaction.

#### 2.5.2.2 Slave Memory Operations

##### 2.5.2.2.1 Memory Write or Memory Write and Invalidate Transactions

The PCI slave treats memory write and memory write and invalidate transactions identically and is therefore not intended to support cacheable memory spaces.

The decoding of and response to a memory write or a memory write and invalidate transaction by the PCI slave depends on the following conditions:

- $\overline{\text{PCI\_CBE[3:0]}}$ must be 7h (memory write) or Fh (memory write and invalidate) during the address phase.
- At least one of the 6 base address registers (PCIBAR0-PCIBAR5) must have bit 0 (IOMEM_SP_IND) cleared to 0.
- The address which is given on PCI_AD[31:N] during the address phase must match the value in the corresponding bits of one of the memory base address registers. The value of N is based on the base address mask registers (PCIBAR0MSK-PCIBAR5MSK) and indicates the number of significant bits in the address to be decoded. The minimum value of N is 4.
- The PCI slave write buffer is empty.
- The address that is given on PCI_AD[1:0] during the address phase must be 00b (linear addressing).

If only the first three of the previous conditions are met, a retry will be issued because the PCI slave write buffer is not empty. If only the first four conditions are met, a single data phase transfer will be completed on the PCI bus. This transfer will be identical in behavior to a configuration write transaction. If the byte enables have at least one byte asserted, a write transaction will initiate on the DM646x DMSoC as soon as any pending read burst requests have completed. If no byte enables are asserted, the transaction will be terminated internally in the PCI slave and no DM646x DMSoC transactions will occur. If all of the above conditions are met, a multi-data phase transfer will be completed. No wait states will be inserted by the PCI slave via the target ready indicator ($\overline{\text{PTRDY}}$), but wait states inserted by the master will be properly handled. The burst will be allowed to continue until one of the following conditions occurs:

- The PCI slave write buffer is almost full.
- A data phase with no asserted byte enables is encountered.
- The burst is about to extend beyond the address boundary of the PCI slave.
- The master ends the transaction.

##### 2.5.2.2.2 Memory Read Transactions

The decoding of and response to a memory read transaction by the PCI slave depends on the following conditions:

- $\overline{\text{PCI\_CBE[3:0]}}$ must be 6h during the address phase.
- At least one of the 6 base address registers (PCIBAR0-PCIBAR5) must have bit 0 (IOMEM_SP_IND) cleared to 0.
- The address on PCI_AD[31:N] during the address phase must match the value in the corresponding bits of one of the memory base address registers. The value of N is based on the base address mask registers (PCIBAR0MSK-PCIBAR5MSK) and indicates the number of significant bits in the address to be decoded. The minimum value of N is 4.
- A delayed read is not outstanding, or this transaction is a re-request of a pending delayed read request and the read data is available. A delayed read is a transaction that must complete on the destination bus before completing on the originating bus.
- The PCI slave write buffer is empty.

If only the first three of the previous conditions are met, a retry will be issued because the delayed transaction is not ready to complete or the PCI slave write buffer is not empty. If the first four or all of the conditions are met, a single data phase transfer will be completed on the PCI bus. This transfer will be similar in behavior to a configuration transaction.

If the transaction is not a delayed completion and the byte enables have at least one byte asserted, a single word read transaction will initiate on the DM646x DMSoC as soon as any pending write burst requests have completed on the interface. If no byte enables are asserted and the addressed memory region is not prefetchable, the transaction will be terminated internally in the PCI slave and no DM646x DMSoC transactions will occur. The byte enables which were presented for the first data phase on the PCI bus will be inverted and presented as the byte enables for the transfer. The PCI slave can wait up to 12 PCI_CLK cycles for data to be returned from the slave back end interface. If data does not return within this time, a retry will be issued and the transaction will be tagged as a delayed read. Alternatively, for performance reasons, if the FORCE_DEL_READ bit (bit 2) in the PCI Slave Control Register (PCISLVCNTL) is asserted, a retry and delayed read can be immediately forced without waiting for the 12 PCI_CLK cycles.

If the transaction is a delayed completion and the data is available, the transaction will complete immediately with $\overline{\text{PTRDY}}$ being asserted coincident with $\overline{\text{PCI\_DEVSEL}}$. Only a single-word prefetch is supported for the memory read command even when it is used within prefetchable memory regions.

#### 2.5.2.2.3 Memory Read Line Transactions

The decoding of and response to a memory read line transaction by the PCI slave depends on the following conditions:

- $\overline{\text{PCI\_CBE[3:0]}}$ must be Eh during the address phase.
- At least one of the 6 base address registers (PCIBAR0-PCIBAR5) must have bit 0 (IOMEM_SP_IND) cleared to 0.
- The address on PCI_AD[31:N] during the address phase must match the value in the corresponding bits of one of the memory base address registers. The value of N is based on the base address mask registers (PCIBAR0MSK-PCIBAR5MSK) and indicates the number of significant bits in the address to be decoded. The minimum value of N is 4.
- A delayed read is not outstanding, or this transaction is a re-request of a pending delayed read line request and the read data is available. A delayed read is a transaction that must complete on the destination bus before completing on the originating bus.
- The PCI slave write buffer is empty.

If only the first three previous conditions are met, a retry will be issued because the delayed transaction is not ready to complete or the PCI slave buffer is not empty. If the first four or all of the conditions are met, a burst read operation will be initiated on the DM646x DMSoC as soon as any pending write burst requests have completed on the interface. The length of the transfer will be the number of words from the requested address to the end of the cache line, unless this value exceeds the size of the PCI slave read data FIFO (16 words). If the burst size is larger than the PCI slave read data FIFO, the transfer will be broken up into 8 word transfers in the same way as the memory read multiple transfers. Since memory read line transactions are prefetchable, all byte enables are asserted internally during the burst. The PCI slave can wait up to 12 PCI_CLK cycles for data to be returned from the slave memory slave back end interface. If data does not return within this time, a retry will be issued and the transaction will be tagged as a delayed read. Alternatively, for performance reasons, if the FORCE_DEL_READ_LN bit (bit 3) in the PCI Slave Control Register (PCISLVCNTL) is asserted, a retry and delayed read can be immediately forced without waiting for the 12 PCI_CLK cycles. If the transaction is a delayed completion and the data is available, the first data phase of the transaction will complete immediately with $\overline{\text{PTRDY}}$ being asserted coincident with $\overline{\text{PCI\_DEVSEL}}$. $\overline{\text{PTRDY}}$ will continue to be asserted until all of the prefetched data has been read or the burst is terminated by the master. If the master attempts to burst beyond the current cache line, the PCI will assert $\overline{\text{PCI\_STOP}}$ on the next to the last data phase in the cache line.

#### 2.5.2.2.4 Memory Read Multiple Transactions

The decoding of and response to a memory read multiple transaction by the PCI slave depends on the following conditions:

- $\overline{\text{PCI\_CBE[3:0]}}$ must be Ch during the address phase.
- At least one of the 6 base address registers (PCIBAR0-PCIBAR5) must have bit 0 (IOMEM_SP_IND) cleared to 0.
- The address on PCI_AD[31:N] during the address phase must match the value in the corresponding bits of one of the memory base address registers. The value of N is based on the base address mask registers (PCIBAR0MSK-PCIBAR5MSK) and indicates the number of significant bits in the address to be decoded. The minimum value of N is 4.
- A delayed read is not outstanding, or this transaction is a re-request of a pending delayed read line request and the read data is available. A delayed read is a transaction that must complete on the destination bus before completing on the originating bus.
- The PCI slave write buffer is empty.

If only the first three previous conditions are met, a retry will be issued because the delayed transaction is not ready to complete or the PCI slave buffer is not empty. If the first four or all of the conditions are met, a burst read operation will be initiated on the DM646x DMSoC as soon as any pending write burst request have completed on the interface. The length of the transfer will be 16 words for the initial transfer of a burst. As memory read multiple transactions are prefetchable, all byte enables are asserted internally during the burst. The PCI slave can wait up to 12 PCI_CLK cycles for data to be returned from the slave memory slave back end interface. If data does not return within this time, a retry will be issued and the transaction will be tagged as a delayed read. Alternatively, for performance reasons, if the FORCE_DEL_READ_MUL bit (bit 4) in the PCI Slave Control (PCISLVCNTL) is asserted, a retry and delayed read can be immediately forced without waiting for the 12 PCI_CLK cycles.

If the transaction is a delayed completion and the data is available, the first data phase of the transaction will complete immediately with $\overline{\text{PTRDY}}$ being asserted coincident with $\overline{\text{PCI\_DEVSEL}}$. $\overline{\text{PTRDY}}$ will continue to be asserted as long as data is available in the read prefetch buffer or the burst is terminated by the master. As data is transferred from the read prefetch buffer on to the PCI bus, additional 8 word read fetches will be performed on the slave memory slave back end interface as buffer space becomes available.

## 2.6 PCI as Master

### 2.6.1 Master Memory-Map

The PCI enables the DM646x DMSoC to access the PCI memory through the master memory-map. There is 256MB of space dedicated for PCI memory in the DM646x DMSoC memory-map. This 256MB space is divided into 32 windows of 8MB fixed size. These windows are called master windows or PCI address windows. Each master window can be configured individually to map 8MB of PCI memory to the DM646x DMSoC address space.

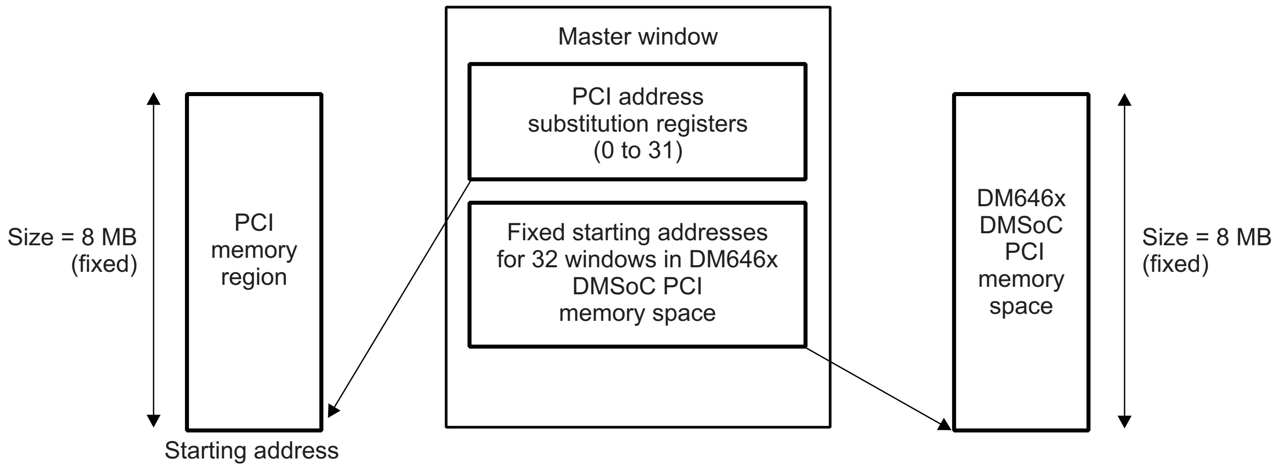
**Table 3. PCI Master Windows**

| Master Window Number | Base Address Register | Window Size | DM646x DMSoC Memory Range |
|---|---|---|---|
| 0 | PCIADDSUB0 | 8MB | 3000 0000h - 307F FFFFh |
| 1 | PCIADDSUB1 | 8MB | 3080 0000h - 30FF FFFFh |
| 2 | PCIADDSUB2 | 8MB | 3100 0000h - 317F FFFFh |
| 3 | PCIADDSUB3 | 8MB | 3180 0000h - 31FF FFFFh |
| 4 | PCIADDSUB4 | 8MB | 3200 0000h - 327F FFFFh |
| 5 | PCIADDSUB5 | 8MB | 3280 0000h - 32FF FFFFh |
| 6 | PCIADDSUB6 | 8MB | 3300 0000h - 337F FFFFh |
| 7 | PCIADDSUB7 | 8MB | 3380 0000h - 33FF FFFFh |
| 8 | PCIADDSUB8 | 8MB | 3400 0000h - 347F FFFFh |
| 9 | PCIADDSUB9 | 8MB | 3480 0000h - 34FF FFFFh |
| 10 | PCIADDSUB10 | 8MB | 3500 0000h - 357F FFFFh |
| 11 | PCIADDSUB11 | 8MB | 3580 0000h - 35FF FFFFh |
| 12 | PCIADDSUB12 | 8MB | 3600 0000h - 367F FFFFh |
| 13 | PCIADDSUB13 | 8MB | 3680 0000h - 36FF FFFFh |
| 14 | PCIADDSUB14 | 8MB | 3700 0000h - 377F FFFFh |
| 15 | PCIADDSUB15 | 8MB | 3780 0000h - 37FF FFFFh |
| 16 | PCIADDSUB16 | 8MB | 3800 0000h - 387F FFFFh |
| 17 | PCIADDSUB17 | 8MB | 3880 0000h - 38FF FFFFh |
| 18 | PCIADDSUB18 | 8MB | 3900 0000h - 397F FFFFh |
| 19 | PCIADDSUB19 | 8MB | 3980 0000h - 39FF FFFFh |
| 20 | PCIADDSUB20 | 8MB | 3A00 0000h - 3A7F FFFFh |
| 21 | PCIADDSUB21 | 8MB | 3A80 0000h - 3AFF FFFFh |
| 22 | PCIADDSUB22 | 8MB | 3B00 0000h - 3B7F FFFFh |
| 23 | PCIADDSUB23 | 8MB | 3B80 0000h - 3BFF FFFFh |
| 24 | PCIADDSUB24 | 8MB | 3C00 0000h - 3C7F FFFFh |
| 25 | PCIADDSUB25 | 8MB | 3C80 0000h - 3CFF FFFFh |
| 26 | PCIADDSUB26 | 8MB | 3D00 0000h - 3D7F FFFFh |
| 27 | PCIADDSUB27 | 8MB | 3D80 0000h - 3DFF FFFFh |
| 28 | PCIADDSUB28 | 8MB | 3E00 0000h - 3E7F FFFFh |
| 29 | PCIADDSUB29 | 8MB | 3800 0000h - 3EFF FFFFh |
| 30 | PCIADDSUB30 | 8MB | 3F00 0000h - 3F7F FFFFh |
| 31 | PCIADDSUB31 | 8MB | 3F80 0000h - 3FFF FFFFh |

### 2.6.1.1 Configuring Master Windows Using Address Substitution Registers (PCIADDSUB*n*)

Each master window corresponds to 8MB of the DM646x DMSoC PCI memory address space. For example, window 0 corresponds to DM646x DMSoC addresses 3000 000h-307F FFFFh, and window 1 corresponds to the next 8 MB, 3080 0000h–30FF FFFFh. Each window can map an 8MB of PCI memory to its corresponding DM646x DMSoC PCI memory address space through its address substitution register. Figure 6 displays a master window configuration.

**Figure 6. Master Window Configuration**


Master window
PCI address substitution registers (0 to 31)
Fixed starting addresses for 32 windows in DM646x DMSoC PCI memory space
Size = 8 MB (fixed)
PCI memory region
Starting address
DM646x DMSoC PCI memory space
Size = 8 MB (fixed)

There are 32 address substitution registers (PCIADDSUB0-PCIADDSUB31) available in the PCI. Each of these registers corresponds to a master window. These registers reside in the PCI interface and are normally programmed by the ARM. Figure 7 shows an example of an address substitution register.

**Figure 7. PCI Address Substitution Registers (PCIADDSUB0-PCIADDSUB31)**

| 31 – 23 | 22 – 0 |
|---|---|
| ADD_SUBS | Reserved |
| R/W-0 | R-0 |

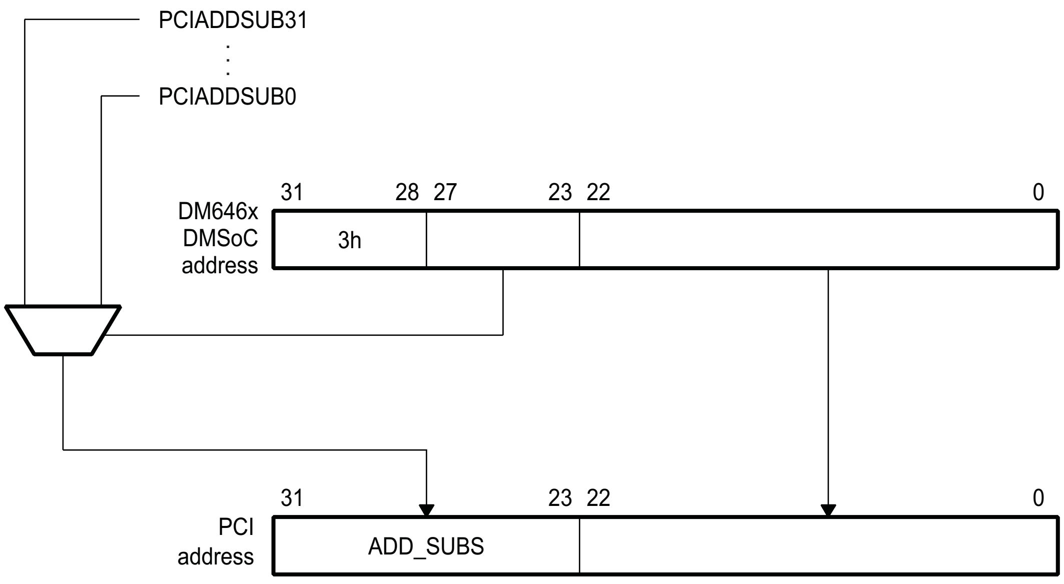
LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Bits 31-23 (ADD_SUBS) of the register contain the MSBs of the PCI addresses within the corresponding window. The remaining reserved 23 bits are the size of the window and these bits have no function. Reads of this field will return 0s. Section 2.6.1.2 explains the translation of addresses as a transaction flow from the DM646x DMSoC domain to the PCI domain.

#### 2.6.1.2 *Master Address Translation*

Address translation from the DM646x DMSoC to the PCI domain is done using the address substitution registers 0 to 31. Figure 8 shows the address just prior to the address translation.

**Figure 8. DM646x DMSoC to PCI Address Translation**



During the address translation, the upper 4 bits are don't cares, as they have already been used for the address decode to reach the DM646x DMSoC PCI memory space. The next 5 bits decide which PCI address window corresponds to the DM646x DMSoC address. Once the window is determined, the 23 LSBs of the DM646x DMSoC address are allowed to pass through to the PCI address and the upper 9 bits are taken from the ADD_SUBS field of the corresponding PCIADDSUB*n* register. Figure 9 illustrates this process.

**Figure 9. Example of DM646x DMSoC to PCI Address Translation**

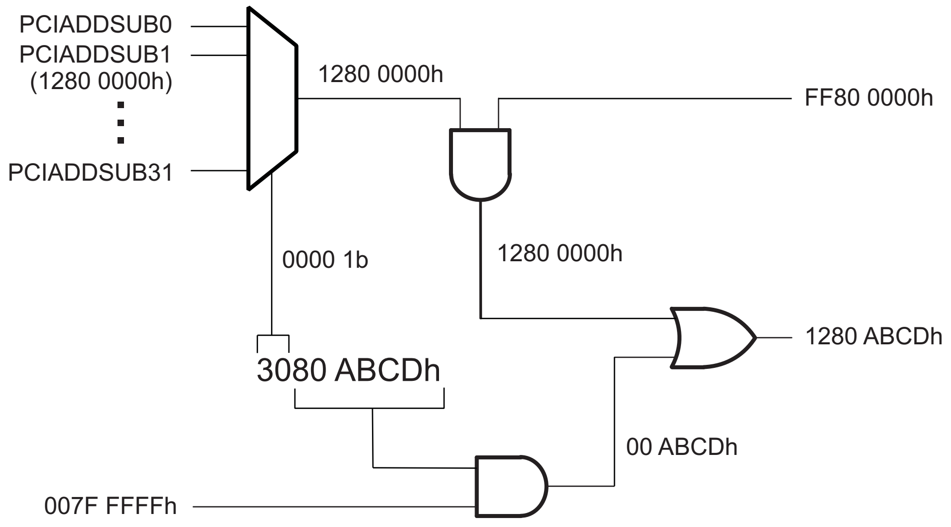


Figure 9 shows an example of a DM646x DMSoC-to-PCI address translation. In this example, the DM646x DMSoC address is 3080 ABCDh and the PCIADDSUB1 register is set to 1280 0000h.

1. The 4 MSBs of the address (3h) indicate the DM646x DMSoC PCI memory space is being accessed.
2. The next 5 bits (00001b) determine that the destination is the second 8MB PCI window, signifying that PCIADDSUB1 will be used. Note that bits 27-3 directly correspond to the PCIADDSUB*n* used.
3. The upper 9 bits of the address substitution register 1 and the lower 23 bits of the DM646x DMSoC address are concatenated to form the PCI address (1280 ABCDh).
4. This address is used by the PCI module.

Example 1 details how to perform an EDMA transfer from a DM646x DMSoC source address (for example, L2 or EMIF) to PCI memory. For the EDMA configuration, see the *TMS320DM646x DMSoC Enhanced Direct Memory Access (EDMA3) Controller User's Guide* (SPRUEQ5).

**Example 1. Master Memory Write**

```
/* Configure master window */
    /* Map external PCI memory (0x12800000) to its corresponding 8MB DM646x DMSoC
     * PCI memory address space using following address substitution register PCIADDSUB1
     */
     PCIADDSUB1 = 0x12800000;   /* PCIADDSUBn, n = 0 to 31 */
   ...
     /* Setup EDMA module and Enable the DMA Region */
     /* Setup EDMA PARAM */
     /* The OPT register contains following the bit fields
    ITCCHEN = 0x0;TCCHEN = 0x0; ITCINTEN = 0x0; TCINTEN = 0x1;
    WIMODE = 0x0; TCC = 0x0; TCCMODE = 0x0; FIFOWID = 0x2;
    STATIC = 0x0; SYNCDIM = 0x0; DAM = 0x1; SAM = 0x1; */
     PARAMSET [paramEntry]. OPT = 0x100203; /* paramEntry can be 0 to 511 */
     PARAMSET [paramEntry]. srcAddr = (Uint32)srcAddr; //srcAddr can be L2 or EMIF
     PARAMSET [paramEntry]. dstAddr = (Uint32)0x30800000;
     PARAMSET [paramEntry]. aCntbCnt = 0x00100004;
     PARAMSET [paramEntry]. srcDstBidx = 0x00000004;
     PARAMSET [paramEntry]. linkBcntrld =  0x0;
     PARAMSET [paramEntry]. srcDstCidx = 0x0;
     PARAMSET [paramEntry]. cCnt = 1;
     /* Setup EDMA Channel */
     /* Enable the Channel */
     /* Wait for a transmit completion */
     /* Example End */
```

Example 2 details how to perform an EDMA transfer from PCI memory to a DM646x DMSoC destination, such as L2 or EMIF. For the EDMA configuration, see the *TMS320DM646x DMSoC Enhanced Direct Memory Access (EDMA3) Controller User's Guide* (SPRUEQ5).

***Example 2. Master Memory Read***

```
/* Configure master window */
    /* Map external PCI memory (0x12800000) to its corresponding 8MB DM646x DMSoC
     * PCI memory address space using following address substitution register PCIADDSUB1
     */
     PCIADDSUB1 = 0x12800000;   /* PCIADDSUBn, n = 0 to 31 */
     ...
       /* Setup EDMA module and Enable the DMA Region */
       /* Setup EDMA PARAM */
       /* The OPT register contains following the bit fields
         ITCCHEN = 0x0;TCCHEN = 0x0; ITCINTEN = 0x0; TCINTEN = 0x1;
         WIMODE = 0x0; TCC = 0x0; TCCMODE = 0x0; FIFOWID = 0x2;
         STATIC = 0x0; SYNCDIM = 0x0; DAM = 0x1; SAM = 0x1; */
     PARAMSET [paramEntry]. OPT = 0x00100203; /* paramEntry can be 0 to 511 */
     PARAMSET [paramEntry]. srcAddr = (Uint32) 0x30800000;
     PARAMSET [paramEntry]. dstAddr = (Uint32) dstAddr; //dstAddr can be L2 or EMIF
     PARAMSET [paramEntry]. aCntbCnt = 0x00100004;
     PARAMSET [paramEntry]. srcDstBidx = 0x00040000;
     PARAMSET [paramEntry]. linkBcntrld =  0x0;
     PARAMSET [paramEntry]. srcDstCidx = 0x0;
     PARAMSET [paramEntry]. cCnt = 1;
     /* Setup EDMA Channel */
     /* Enable the Channel */
     /* Wait for a transmit completion */
     /* Example End */
```

### 2.6.2 Master Operations

The PCI master operates in response to memory transfer requests that are presented on the master back-end interface, and to indirect IO and configuration requests that are presented via the master configuration/IO transaction proxy registers. For memory transactions, the master can be programmed to only use the basic memory read or write transactions, or can also perform burst transfers as efficiently as possible by automatically selecting the proper command for memory transactions based on the transaction length and the cache line size. The PCI master supports two FIFOS/buffers (a read and write) for efficient data transfer. Each buffer holds 16 32-bit words of read or write data.

#### *2.6.2.1 Master Configuration Operations*

During a configuration space access, PCI_AD[31:2] is used to address the PCI device, the function within the device and a DWORD in the function’s configuration space. PCI_AD[1:0] is ignored. However, you can set PCI_AD[1:0] to 00 for TYPE 0 access, or set it to 01 for TYPE 1 access.

The byte enables select the bytes within the addressed DWORD. The byte enables allow access to a byte, word, DWORD, or non-contiguous bytes in the addressed DWORD. In the addressed DWORD, BE0 enables byte 0, BE1 enables the byte 1, and so on.

#### 2.6.2.1.1 *Configuration Write Transactions*

The back-end application causes the PCI master to perform a configuration write operation by executing the following steps:

1. Reading the PCI master configuration/IO access command register (PCIMCFGCMD) and ensuring that the READY bit (bit 31) is asserted.
2. Writing the data for the configuration write to the PCI master configuration/IO access data register (PCIMCFGDAT) through the DM646x DMSoC PCI interface.
3. Writing the address for the configuration write to the PCI master configuration/IO access address register (PCIMCFGADR).
4. Writing to PCIMCFGCMD with the TYPE bit cleared to 0 (configuration transaction), the RD_WR bit cleared to 0 (write), and the BYTE_EN field set to the desired value.

On the next cycle, $\overline{\text{PCI\_REQ}}$ is asserted. Once $\overline{\text{PCI\_GNT}}$ is sampled asserted, the master asserts $\overline{\text{PCI\_FRAME}}$ and outputs the write address onto the PCI_AD[31:0] pins and the Configuration Write command (Bh) onto the $\overline{\text{PCI\_CBE[3:0]}}$ pins. On the next cycle, the master asserts $\overline{\text{PCI\_IRDY}}$, deasserts $\overline{\text{PCI\_FRAME}}$, and outputs the data to be written. The master will never insert wait states during a transfer but will respond to wait states as controlled by $\overline{\text{PTRDY}}$. As is required, the master continually checks the bus for exceptions (master abort, target abort, retry, disconnect, latency timeout, parity error, system error) while the transfer is ongoing. A ready signal is returned to the DM646x DMSoC PCI Master interface through the READY bit in PCIMCFGCMD when the transfer is complete.

#### 2.6.2.1.2 *Configuration Read Transactions*

The back-end application can request that the PCI master to perform a configuration read operation by doing the following:

1. Reading the PCI master configuration/IO access command register (PCIMCFGCMD) and ensuring that the READY bit (bit 31) is asserted.
2. Writing the address for the configuration write to the PCI master configuration/IO access address register (PCIMCFGADR).
3. Writing to PCIMCFGCMD with the TYPE bit cleared to 0 (configuration transaction), the RD_WR bit set to 1 (read), and the BYTE_EN field set to the desired value.

On the next cycle, $\overline{\text{PCI\_REQ}}$ is asserted. Once $\overline{\text{PCI\_GNT}}$ is sampled asserted, the master asserts $\overline{\text{PCI\_FRAME}}$ and outputs the read address onto the PCI_AD[31:0] pins and the Configuration Read command (Ah) onto the $\overline{\text{PCI\_CBE[3:0]}}$ pins. On the next cycle, the master asserts $\overline{\text{PCI\_IRDY}}$, and deasserts $\overline{\text{PCI\_FRAME}}$. The master will never insert wait states during a transfer but will respond to wait states as controlled by $\overline{\text{PTRDY}}$. As is required, the master continually checks the bus for exceptions (master abort, target abort, retry, disconnect, latency timeout, parity error, system error) while the transfer is ongoing. When the transfer is complete, the data is returned to the PCI Master Configuration/IO Access Data Register (PCIMCFGDAT) and the READY bit in PCIMCFGCMD is set to 1.

### 2.6.2.2 *Master I/O Operations*

In the I/O address space, all 32 PCI_AD lines are used to provide a full byte address. The master that initiates an I/O transaction is required to ensure that PCI_AD[1:0] indicates the least significant valid byte for the transaction.

The byte enables indicate the size of the transfer and the affected bytes within the DWORD and must be consistent with PCI_AD[1:0]. Table 4 lists the valid combinations for PCI_AD[1:0] and the byte enables for the initial data phase. Byte enables are asserted when 0.

**Table 4. Byte Enables and PCI_AD[1:0] Encodings**

| PCI_AD[1:0] | Starting Byte | Valid BE#[3:0] Combinations |
|---|---|---|
| 00 | Byte 0 | xxx0 or 1111 |
| 01 | Byte 1 | xx01 or 1111 |
| 10 | Byte 2 | x011 or 1111 |
| 11 | Byte 3 | 0111 or 1111 |

#### 2.6.2.2.1 I/O Write Transactions

The back-end application can request for the PCI master to perform an I/O write operation by doing the following:

1. Reading the PCI master configuration/IO access command register (PCIMCFGCMD) and ensuring that the READY bit (bit 31) is asserted.
2. Writing the data for the configuration write to the PCI master configuration/IO access data register (PCIMCFGDAT), through the PCI back-end registers interface.
3. Writing the address for the configuration write to the PCI master configuration/IO access address register (PCIMCFGADR).
4. Writing to PCIMCFGCMD with the TYPE bit cleared to 0 (I/O transaction), the RD_WR bit cleared to 0 (write), and the BYTE_EN field set to the desired value.

When a request is made for I/O write operation, the DM646x DMSoC PCI master interface asserts $\overline{\text{PCI\_REQ}}$ on the PCI bus. Once $\overline{\text{PCI\_GNT}}$ is sampled asserted, the master asserts $\overline{\text{PCI\_FRAME}}$ and outputs the write address onto the PCI_AD[31:0] pins and the I/O Write command (3h) onto the $\overline{\text{PCI\_CBE[3:0]}}$ pins. On the next cycle, the master asserts $\overline{\text{PCI\_IRDY}}$, deasserts $\overline{\text{PCI\_FRAME}}$, and outputs the data to be written. The master will never insert wait states during a transfer but will respond to wait states as controlled by $\overline{\text{PTRDY}}$. As is required, the master continually checks the bus for exceptions (master abort, target abort, retry, disconnect, latency timeout, parity error, system error) while the transfer is ongoing. A ready signal is returned to the DM646x DMSoC PCI Master interface through the READY bit in PCIMCFGCMD when the transfer is complete.

#### 2.6.2.2.2 I/O Read Transactions

The back-end application can request for the PCI master to perform an I/O read operation by doing the following:

1. Reading the PCI master configuration/IO access command register (PCIMCFGCMD) and ensuring that the READY bit (bit 31) is asserted.
2. Writing the address for the configuration write to the PCI master configuration/IO access address register (PCIMCFGADR).
3. Writing to PCIMCFGCMD with the TYPE bit cleared to 0 (configuration transaction), the RD_WR bit set to 1 (read), and the BYTE_EN field set to the desired value.

When a request is made for I/O read operation, the DM646x DMSoC PCI master interface asserts $\overline{\text{PCI\_REQ}}$ on the PCI bus, $\overline{\text{PCI\_REQ}}$ is asserted. Once $\overline{\text{PCI\_GNT}}$ is sampled asserted, the master asserts $\overline{\text{PCI\_FRAME}}$ and outputs the read address onto the PCI_AD[31:0] pins and the I/O Read command (2h) onto the $\overline{\text{PCI\_CBE[3:0]}}$ pins. On the next cycle, the master asserts $\overline{\text{PCI\_IRDY}}$, and deasserts $\overline{\text{PCI\_FRAME}}$. The master will never insert wait states during a transfer but will respond to wait states as controlled by $\overline{\text{PTRDY}}$. As is required, the master continually checks the bus for exceptions (master abort, target abort, retry, disconnect, latency timeout, parity error, system error) while the transfer is ongoing. When the transfer is complete, the data is returned to the PCI Master Configuration/IO Access Data Register (PCIMCFGDAT) and the READY bit in PCIMCFGCMD is set.

### 2.6.2.3 *Master Memory Operations*

In the memory access, bits PCI_AD[31:2] are used to address the PCI device, the function within the device and a DWORD in the function’s memory space. Bits PCI_AD[1:0] are ignored. However, bits PCI_AD[1:0] indicate the order in which the master is requesting the data to be transferred.

#### 2.6.2.3.1 *Memory Write Transactions*

The back end application can request for the PCI master to perform a memory write operation by making a memory write access to PCI master memory map. When a request is made for a memory write operation, the PCI master asserts $\overline{\text{PCI\_REQ}}$ on the PCI bus. Once $\overline{\text{PCI\_GNT}}$ is sampled asserted; the master asserts $\overline{\text{PCI\_FRAME}}$ and outputs the write address onto the PCI_AD[31:0] pins and the Memory Write command (7h) onto the $\overline{\text{PCI\_CBE[3:0]}}$ pins. On the next cycle, the master asserts $\overline{\text{PCI\_IRDY}}$ and outputs the first word of data to be written. If the transfer only consists of a single data phase, $\overline{\text{PCI\_FRAME}}$ is deasserted as required by the PCI specification coincident with the assertion of $\overline{\text{PCI\_IRDY}}$. Otherwise, $\overline{\text{PCI\_FRAME}}$ continues to be asserted until the next to the last data phase completes. The master will never insert wait states during a burst but will respond to wait states as controlled by $\overline{\text{PTRDY}}$. As is required, the master continually checks the bus for exceptions (master abort, target abort, retry, disconnect, latency timeout, parity error, system error) while the transfer is ongoing. As the burst progresses, an internal ready signal is returned from the master back end interface for each successful data phase completion until the entire burst is finished.

Section 2.6.1.2 provides an example that describes how to program the EDMA to perform a master memory write operation.

#### 2.6.2.3.2 *Memory Read Transactions*

The back end application can request for the PCI master to perform a memory read operation by making a memory read access to PCI master memory-map.

When a request is made for a memory read operation, $\overline{\text{PCI\_REQ}}$ is asserted and the read burst is performed following the same behavior as for the write burst (no wait states, etc.). During the address phase, the Memory Read (6h), Memory Read Line (Eh), or Memory Read Multiple command (Ch) is output on the $\overline{\text{PCI\_CBE[3:0]}}$ pins depending on the length of the transfer and the cache line size. The byte enables that were registered from the master back end interface are inverted and output during the data phases. Unlike the write burst, the byte enables on the PCI bus will not change as the transaction progresses. This will not cause problems since bursts are only performed to prefetchable regions of memory. As the burst progresses, an internal ready signal is returned from the master back end interface along with the read data for each successful data phase completion until the entire burst is finished.

Section 2.6.1.2 provides an example that describes how to program the EDMA to perform a master memory read operation.

## 2.7 Exceptions, Status Reporting, and Interrupts

### 2.7.1 PCI Exceptions

The PCI supports the detection of the error conditions listed in Table 5. The PCI can generate an interrupt to the Host and ARM for a particular set of error conditions. The status set register (PCISTATSET), host interrupt enable set register (PCIHINTSET), and back-end application interrupt enable set register (PCIBINTSET) enable interrupts to the Host and ARM. The PCI also provides the status of these PCI errors, as described in Section 2.7.2.

**Table 5. PCI Exceptions**

| Exception Name | Description |
|---|---|
| PERR_DET | Data parity error. Detected during a read transaction of the PCI bus master and write transaction of a PCI bus target. |
| SERR_DET | System error. Detected when the PCI has received a target abort while mastering the bus or when an address parity error is detected on the PCI bus. |
| MS_ABRT_DET | Master Abort. Generated by the PCI master unit in the PCI to indicate that it terminated a transaction with a master abort. |
| TGT_ABRT_DET | Target Abort. Generated by the PCI slave unit in the PCI to indicate that it has initiated a target abort. |

#### *2.7.1.1 Parity Error*

If the PCI master is mastering the bus, the master data parity reported (MS_DPAR_REP) bit in the PCI configuration space command/status register (PCICSR) will be set under either of the following conditions:

- If it detects a parity error during the data phase of a read transaction.
- If it detects that $\overline{\text{PCI\_PERR}}$ has been asserted by the target during the data phase of a write transaction.

The detected parity error (DET_PAR_ERR) bit in PCICSR will be set under any of the following conditions:

- If it is acting as the PCI bus master and it detects a data parity error during a read transaction.
- If it is acting as a PCI bus target and it detects a data parity error during a write transaction.
- If it detects an address parity error.

The PCI will assert $\overline{\text{PCI\_PERR}}$ if the parity error response (PAR_ERR_RES) bit in PCICSR is set and the DET_PAR_ERR bit is set. The assertion of $\overline{\text{PCI\_PERR}}$ will remain valid until the second clock after the cycle in which the error occurred

If a parity error is detected during a transfer involving the PCI, the transaction will be allowed to complete unless the PCI is the master and a target disconnect is detected (that is, the PCI will not master abort due to a parity error).

#### *2.7.1.2 System Error*

The PCI will set an internal system error flag under any of the following conditions:

- If an address parity error is detected on the PCI bus (even if the PCI is not the target of the transaction) and the parity error response (PAR_ERR_RES) bit in the PCI configuration space command/status register (PCICSR) is set.
- If the PCI detected $\overline{\text{PCI\_PERR}}$ asserted while mastering the bus.
- If the PCI received a target abort (disconnect without retry) while mastering the bus.

The PCI will assert $\overline{\text{PCI\_SERR}}$ if the $\overline{\text{PCI\_SERR}}$ enable bit (SERR_N_EN) in PCICSR is set and the internal system error flag is set. The PCI will halt and wait for software or hardware reset after $\overline{\text{PCI\_SERR}}$ has been asserted. The PCI will set the signaled system error (SIG_SYS_ERR) bit in PCICSR whenever $\overline{\text{PCI\_SERR}}$ is asserted.

#### 2.7.1.3 Master Abort Protocol

If a master abort occurs while the PCI is the master, the current transfer will be gracefully terminated on both the PCI (by de-asserting $\overline{\text{PCI\_FRAME}}$ and asserting $\overline{\text{PCI\_IRDY}}$) and back-end buses (by supplying ready signals through the back-end interface until the burst is completed). Both the received master abort signal in the PCI command/status register (PCICSR) and the received master abort (RCV_MS_ABRT) bit in the PCI back-end command/status mirror register (PCICSRMIR) will be set.

#### 2.7.1.4 Target Abort Protocol

If a target abort occurs while the PCI is the master, the current transfer will be gracefully terminated on both the PCI and back-end buses (in the same way as for the master abort). Both the received target abort signal in the PCI command/status register (PCICSR) and the received target abort (RCV_TGT_ABRT) bit in the PCI back-end command/status mirror register (PCICSRMIR) will be set.

#### 2.7.1.5 Retry /Disconnect Protocol

If a transaction is disconnected or retried, the master will unconditionally repeat the transaction starting at the location of the first remaining uncompleted word. The back-end has no knowledge of retry or disconnections on the bus.

### 2.7.2 Status Reporting

The PCI module provides the status of various PCI errors generated or detected by it in the command/status register (PCICSR) and an internal status register. The PCICSR is in the PCI configuration register space. An external host can access this register through the TYPE0 configuration space access. The ARM can access this register through the command/status mirror register (PCICSRMIR).

The PCICSR provides the status of the following error conditions:

- Detected Parity Error (DET_PAR_ERR)
- Signaled System Error (SIG_SYS_ERR)
- Received Master Abort Error (RCV_MS_ABRT)
- Received Target Abort Error (RCV_TGT_ABRT)
- Signaled Target Abort (SIG_TGT_ABRT)
- Master Data Parity Reported (MS_DPAR_REP)

Status bits in PCICSR cannot be set manually. They are set only by the PCI module. A status bit in this register can be cleared by writing a 1 to that bit.

The internal status register provides the status of the following PCI interrupt and errors:

- Software interrupts (SOFT_INT)
- Parity Error Detected (PERR_DET)
- System Error Detected (SERR_DET)
- Master Abort Error Detected (MS_ABRT_DET)
- Target Abort Error Detected (TGT_ABRT_DET)

The PCI provides the status set register (PCISTATSET) and the status clear register (PCISTATCLR) to set and clear the bits in the internal status register. Reading of both these registers returns the value of the internal status register. The internal status register is internal to the PCI module and is not directly accessible to an external host. It is also not directly accessible to the ARM.

To clear an error bit in the command/status register the corresponding bit in the internal status register also needs to be cleared first. The bits in the internal status register can be cleared through the status clear register (PCISTATCLR). Setting or clearing a bit in the internal status register does not affect the corresponding bit in the command/status register. Similarly, clearing the command/status register does not affect the corresponding bit in the internal status register. An interrupt can be generated to an external host (through the $\overline{\text{PCI\_INTA}}$ pin) and to the ARM through the bits of the internal status register provided the corresponding bit is enabled in the internal host/DM646x DMSoC interrupt enable register.

### 2.7.3 PCI Interrupts

The PCI can generate an interrupt (Table 6) for the status conditions listed in Table 7. When a status condition is set, PCI can raise an interrupt to the ARM or PCI host or both based on what interrupts have been enabled for host and back end interface. See Section 2.7.3.1 and Section 2.7.3.2 for enabling interrupts to host and ARM for a particular set of status conditions.

**Table 6. PCI Interrupt**

| ARM Event | Acronym | Source |
|---|---|---|
| 15 | PCIINT | PCI |

**Table 7. PCI Interrupt Status Conditions**

| Interrupt Name | Description |
|---|---|
| INTA | A level-sensitive active-low interrupt is generated to the host on the $\overline{\text{PCI\_INTA}}$ pin if a bit in the internal PCI Status Register is asserted and the corresponding bit in the internal Host Interrupt Enable Register is also asserted, as long as the PCI is in the D0 power state. |
| PERR_DET | A parity error is detected during a read transaction of PCI bus master and write transaction of a PCI bus target. |
| SERR_DET | A system error is detected when the PCI has received a target abort while mastering the bus, or when an address parity error is detected on the PCI bus. |
| MS_ABRT_DET | A Master Abort is generated by the PCI master unit in the PCI to indicate that it terminated a transaction with a master abort. |
| TGT_ABRT_DET | A Target Abort is generated by the PCI slave unit in the PCI to indicate that it has initiated a target abort. |

#### *2.7.3.1 DM646x DMSoC-to-Host Interrupts*

PCI can raise an interrupt to the host for various status conditions, as described in Table 7. The PCI includes an internal host interrupt enable register that specifies which status conditions will generate interrupts to the host. The PCI host interrupt enable register is not directly accessible by the host or the back-end application. Two registers are provided to set or clear bits in the internal PCI host interrupt enable register: PCI host interrupt enable set register (PCIHINTSET) and PCI host interrupt enable clear register (PCIHINTCLR).

An interrupt for a particular status condition can be enabled by setting the corresponding bit in PCIHINTSET. The interrupt for INTA is enabled by default when PCI is in D0 power state. Reading PCIHINTSET returns the contents of the internal host interrupt enable register.

An interrupt for a particular status condition can be disabled by setting the corresponding bit in PCIHINTCLR. Reading PCIHINTCLR returns the bitwise ANDing of the internal PCI status register and the internal PCI host interrupt enable register. PCIHINTCLR is typically read by the host to determine the source of an interrupt when the $\overline{\text{PCI\_INTA}}$ pin is asserted.

A level-sensitive active-low interrupt is generated to the host on the $\overline{\text{PCI\_INTA}}$ pin if a bit in the internal status register is asserted and the corresponding bit in the internal host interrupt enable register is also asserted. When an interrupt is raised on $\overline{\text{PCI\_INTA}}$ pin, the host can read PCIHINTCLR to determine the status condition that caused the interrupt.

Software can also use the SOFT_INT bits to interrupt the host via the $\overline{\text{PCI\_INTA}}$ pin. Software interrupts are enabled and disabled by writing to the PCI host interrupt enable register. Setting the corresponding bit in the internal status register will assert a level sensitive active low interrupt on the $\overline{\text{PCI\_INTA}}$ pin. The ARM or host can clear this interrupt condition by clearing applicable bit in the internal status register.

Interrupt generation on the $\overline{\text{PCI\_INTA}}$ pin is enabled only when the PCI is in D0 power state. It is disabled in the D1, D2, or D3 power states.

#### 2.7.3.2 Host-to-DM646x DMSoC Interrupts

PCI can raise an interrupt to back end (ARM) for various status conditions described in Table 7. The PCI includes an internal PCI back-end interrupt enable register that specifies which status conditions will generate interrupts to the back end. The PCI back-end interrupt enable register is not directly accessible by the host or the back end application. Two registers are provided to set or clear bits in the internal PCI back-end interrupt enable register: back-end application interrupt enable set register (PCIBINTSET) and back-end application interrupt enable clear register (PCIBINTCLR).

An interrupt for a particular status condition can be enabled by setting the corresponding bit in the back-end interrupt enable set register (PCIBINTSET). Reading PCIBENTSET returns the contents of the internal back-end interrupt enable register.

An interrupt for a particular status condition can be disabled by setting the corresponding bit in PCIBINTCLR. Reading PCIBINTCLR returns the bitwise ANDing of the internal PCI status register and the internal PCI back-end interrupt enable register. PCIBINTCLR is typically read by the back-end application to determine the source of an interrupt.

An interrupt signal is generated to the ARM if a bit in the internal status register (PCISTATSET) is asserted and the corresponding bit in the internal back-end application interrupt enable register (PCIBINTSET) is also asserted. When an interrupt is raised to the ARM (back-end), the ARM can read PCISTATCLR to know the status condition that caused the interrupt.

The interrupt request to the ARM is generated through the PCI-to-ARM interrupt (PCIINT) line. This interrupt can also be forced by setting any of the SOFT_INT bits of the PCI status set register (PCISTATSET). Note that the SOFT_INT interrupts are independently enabled through the internal PCI back-end application interrupt enable register.

## 2.8 PCI Reset Information

### 2.8.1 PCI Pin Reset

The PCI reset pin ($\overline{\text{PCI\_RST}}$) is the main PCI hardware reset. This resets most of the PCI logic within the main PCI clock domain. This reset brings PCI-specific registers, sequencers, and signals to a consistent state.

### 2.8.2 PCI Register Reset Values

The reset values for some PCI registers are specified in the PCI configuration hook registers (Section 3). The values in the PCI configuration hook registers are latched to their corresponding PCI registers following a PCI hardware reset (through the $\overline{\text{PCI\_RST}}$ pin). This functionality is implemented mainly to support PCI autoinitialization, which is described in more detail in Section 2.9.4.

## 2.9 PCI Configuration

The operation of the PCI is configured through the configuration space registers and the back-end registers.

### 2.9.1 Programming the PCI Configuration Space Registers

Configuration space registers can be programmed both from the external PCI host and ARM. Normally, the PCI host programs a part of configuration space registers and ARM programs the remaining part.

A PCI host can control modes and options in PCI by programming the configuration space registers. For example, the PCI host can program the command/status register (PCICSR) to enable PCI bus master capability for the DM646x DMSoC and to enable the memory access to DM646x DMSoC. It can program the base address registers (PCIBAR0-PCIBAR5) to map the DM646x DMSoC memory regions into PCI address space.

The PCI host can access the configuration space registers by performing a TYPE 0 access in the PCI bus.

The ARM needs to program a set of registers in the configuration space before the PCI host system software scans the PCI, as part of enumerating the PCI devices. For example, it needs to program the vendor ID/device ID register (PCIVENDEV) and the class code/revision ID register (PCICLREV) so that the PCI host system software can identify the device and load the respective host driver if required. This can be done automatically using the I2C EEPROM initialization method, as described in Section 2.9.4.

The ARM cannot access the configuration space registers directly. To facilitate access to the configuration space registers, mirror registers are supported. Updating the mirror registers updates the corresponding configuration space registers.

### 2.9.2 Programming the PCI Back-End Registers

The back-end registers include configuration space mirror registers, master and slave address translation registers, and other miscellaneous PCI control registers. Back-end registers can be programmed from both the PCI host and the ARM. Normally, the PCI host programs a part of the back-end registers and the ARM programs the remaining part of the registers. For example, the PCI host may want to program the host interrupt enable set register (PCIHINTSET) and the host interrupt enable clear register (PCIHINTCLR) to selectively enable host interrupts.

The PCI host can access the back-end registers through the slave interface supported by PCI, provided the ARM has mapped those registers to PCI through the slave window base address registers.

The ARM may program the configuration space mirror registers, slave and master address translation registers, and other miscellaneous configuration registers. The ARM can access all the back-end registers directly, as these registers are either mapped to DM646x DMSoC memory space or implemented as MMRs.

The back-end configuration registers have a default value as described in Table 8. These values can be overwritten by the application software, and, in some cases, by the software routine located in the internal ROM of the DM646x DMSoC, as explained in Section 2.9.4.

### 2.9.3 PCI Configuration Hook Registers

The reset values of all the PCI back-end configuration registers, listed in Table 8, can be specified through a set of memory-mapped registers called the configuration hook registers. See Section 3.4 for a list of configuration hook registers supported. The values in the configuration hook registers are latched to the actual PCI module registers on a PCI reset (through $\overline{\text{PCI\_RST}}$). The default values in the configuration hook registers can be overwritten by software. The configuration hook registers are implemented mainly to support PCI I2C EEPROM autoinitialization, as explained in Section 2.9.4.

**Table 8. PCI Back-End Configuration Registers Default Values**

| Register | Default Value |
|---|---|
| Vendor ID/Device ID Mirror Register | DEV_ID = B002h; VEN_ID = 104Ch |
| Command/Status Mirror Register | 0000 0000h |
| Class Code/Revision ID Mirror Register | 1180 0001h |
| Subsystem Vendor ID/Subsystem ID Mirror Register | 0000 0000h |
| Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register | 0000 0100h |
| Base Address 0 Mask Register | ADDRMASK = FFF FC00h |
| Base Address 1 Mask Register | ADDRMASK = FFF F800h |
| Base Address 2 Mask Register | ADDRMASK = FFC 0000h |
| Base Address 3 Mask Register | ADDRMASK = FFF E000h |
| Base Address 4 Mask Register | ADDRMASK = FF8 0000h |
| Base Address 5 Mask Register | ADDRMASK = FF8 0000h |
| Base Address 0 Mirror Register | PREFETCH = 1 |
| Base Address 1 Mirror Register | PREFETCH = 0 |
| Base Address 2 Mirror Register | PREFETCH = 0 |
| Base Address 3 Mirror Register | PREFETCH = 1 |
| Base Address 4 Mirror Register | PREFETCH = 1 |
| Base Address 5 Mirror Register | PREFETCH = 1 |
| Slave Control Register | BASE*n*_EN = 11 1111b |
| Slave Base Address 0 Translation Register | 100 1000h |
| Slave Base Address 1 Translation Register | 200 0000h |
| Slave Base Address 2 Translation Register | 01C 0000h |
| Slave Base Address 3 Translation Register | 118 1800h |
| Slave Base Address 4 Translation Register | 800 0000h |
| Slave Base Address 5 Translation Register | 808 0000h |

### 2.9.4 PCI I2C EEPROM Auto-Initialization

Normally, at boot time, a PCI host provides a PCI reset to all the PCI devices. A host can start accessing a PCI device once the device completes the PCI reset. So, if any PCI configuration registers need to be programmed before a host starts accessing the PCI, it should be done before PCI reset is completed. Table 9 lists the default values for some of the PCI configuration registers. These default values can be changed by enabling PCI I2C EEPROM auto-initialization.

**Table 9. PCI Configuration Registers Default Values**

| Register | Default Value |
|---|---|
| Vendor ID/Device ID Register | DEV_ID = B002h; VEN_ID = 104Ch |
| Class Code/Revision ID Register | CL_CODE = 11 8000h; REV_ID = 01h |
| Subsystem Vendor ID/Subsystem ID Register | 0000 0000h |
| Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register | 0000 0100h |

#### 2.9.4.1 PCI Auto-Initialization from I2C EEPROM

When auto-initialization is used, the PCI configuration registers are programmed by the on-chip ROM Boot Loader (RBL) with the values stored in an I2C EEPROM.

PCI I2C EEPROM auto-initialization is enabled when in the System Module BOOTCFG register, BOOTMODE[3:0] = 0011b and PCIEN = 1. If auto-initialization is not enabled, the PCI configuration registers are left with their default values and the I2C EEPROM is not accessed for PCI configuration purposes. The function of the BOOTMODE[3:0] and PCIEN pins and the BOOTCFG register is fully described in the device data manual, refer to that document for more details.

When auto-initialization is enabled, the CONFIG_DONE bit in the configuration done register (PCICFGDONE) takes a default value of 0. This prevents the PCI from responding to any requests. When auto-initialization is completed, the RBL sets the CONFIG_DONE bit to 1 to allow the PCI to respond to requests.

#### 2.9.4.2 I2C EEPROM Memory Map

The on-chip ROM Boot Loader requires big-endian format for the data stored in the I2C EEPROM. Byte addresses 400h through 41Bh of the I2C EEPROM are reserved for auto-initialization of PCI configuration registers. The remaining locations are not used for auto-initialization and can be used for storing other data. Table 10 summarizes the I2C EEPROM memory layout, as required for PCI auto-initialization.

**Table 10. I2C EEPROM Memory Layout**

| Byte Address | Contents |
|---|---|
| 400h | Vendor ID [15:8] |
| 401h | Vendor ID [7:0] |
| 402h | Device ID [15:8] |
| 403h | Device ID [7:0] |
| 404h | Class code [7:0] |
| 405h | Revision ID [7:0] |
| 406h | Class code [23:16] |
| 407h | Class code [15:8] |
| 408h | Subsystem vendor ID [15:8] |
| 409h | Subsystem vendor ID [7:0] |
| 40Ah | Subsystem ID [15:8] |
| 40Bh | Subsystem ID [7:0] |
| 40Ch | Max_Latency |
| 40Dh | Min_Grant |
| 40Eh-419h | Reserved (use 00h) |
| 41Ah | Checksum [15:8] |
| 41Bh | Checksum [7:0] |

### 2.9.4.3 I2C EEPROM Checksum

The PCI configuration data contained in the I2C EEPROM is checked against a checksum. The configuration data bytes are treated as an array of 16-bit words (little-endian format). The checksum is a 16-bit cumulative exclusive-OR (XOR) of the configuration data words, starting with an initial value of AAAAh. You must ensure that the proper 16-bit checksum value is written to address 419h and 41Ah when programming the I2C EEPROM.

Checksum = AAAAh XOR Byte0[400h] XOR Byte1[401h] ...... XOR Byte25[419h]

If the I2C EEPROM is not accessed for PCI configuration purposes, then the checksum is not performed. If the checksum fails, the on-chip ROM bootloader defaults to the UART boot and it does not set the CONFIG_DONE bit in the configuration done register (PCICFGDONE).

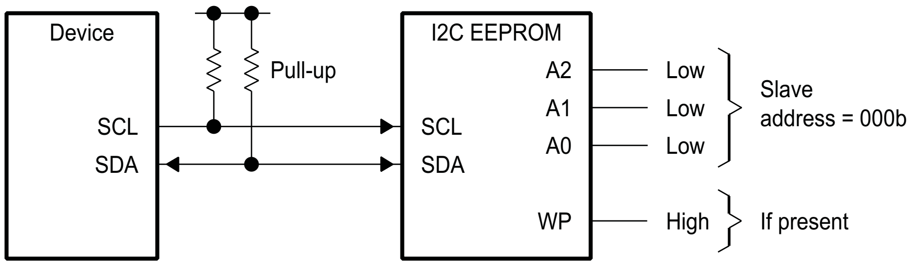
### 2.9.4.4 ARM I2C EEPROM Interface

For PCI auto-initialization, the ARM supports I2C EEPROMs or devices operating as I2C slaves with the following features:

- The memory device complies with Philips I2C Bus Specification v 2.1
- The memory device uses two bytes for internal addressing; that is, the read/write bit followed by two bytes for addressing

During PCI auto-initialization, the ARM acts as the master and the I2C EEPROM acts as the slave. Figure 10 shows the minimum connection required between the ARM and one I2C EEPROM. The required pull-ups must be placed on SDA and SCL to ensure that the I2C EEPROM interface works correctly. The slave address of the I2C EEPROM slave address must be set to 50h.

**Figure 10. Signal Connections for I2C EEPROM Boot Mode**



Some I2C EEPROMs have a write-protect (WP) feature that prevents unauthorized writes to memory. This feature is not needed for auto-initialization because the ARM will only read data from the I2C EEPROM. The write protect feature can be enabled or disabled.

For PCI auto-initialization purposes only byte address 400-401h are used. The remaining locations in the I2C EEPROM can be used for other purposes.

For detailed information on the I2C, see the *TMS320DM646x DMSoC Inter-Integrated Circuit (I2C) Module User's Guide* (SPRUER0).

## 3 Registers

There are four types of PCI registers:

- PCI configuration registers - Configuration space registers can be programmed both from the PCI host and DM646x DMSoC.
- PCI back-end configuration registers - Back-end registers can be programmed both from the PCI host and the DM646x DMSoC. Normally, the PCI host programs a part of the back-end registers and the ARM programs the remaining part of the registers.
- DM646x DMSoC-to-PCI address translation registers – These registers are located in the same address space as the PCI back-end configuration registers, so no new memory space is needed. These registers are programmed from the ARM.
- PCI configuration hook registers - These registers are located in the same address space as the PCI back-end configuration registers. These registers can be used to specify the reset value of other PCI registers and are implemented mainly to support PCI I2C EEPROM autoinitialization.

The following sections describe the various software-accessible registers that are contained within the PCI.

### *3.1 PCI Configuration Registers*

The DM646x DMSoC supports all standard PCI configuration registers. These registers, which can be directly accessed by the external PCI host through Type 0 configuration read and write transactions, contain the standard PCI configuration information (vendor identification, device identification, class code, revision number, base addresses, etc.). The DM646x DMSoC can access these registers indirectly through the PCI memory-mapped registers. Depending on the boot and device configuration settings at device reset, some of the PCI configuration registers can be auto-loaded from an I2C EEPROM at device reset or can be initialized with default values. If I2C EEPROM auto-initialization is not used, the PCI configuration registers are initialized with their default values. If auto-initialization is used, the PCI configuration registers cannot be properly accessed by the host until they are fully read from the I2C EEPROM. PCI host access to the PCI configuration registers before the completion of auto-initialization results in a disconnect with retry. For more details, see Section 2.9.4.

**Table 11. PCI Configuration Registers**

| Offset | Register [(1)] | | | | Section |
|---|---|---|---|---|---|
| | Byte 3 | Byte 2 | Byte 1 | Byte 0 | |
| 0h | Device ID | | Vendor ID | | Section 3.1.1 |
| 4h | PCI Status | | PCI Command | | Section 3.1.2 |
| 8h | Class Code | | | Revision ID | Section 3.1.3 |
| Ch | Built-In Self-Test | Header Type | Latency Timer | Cache Line Size | Section 3.1.4 |
| 10h | Base Address 0 (8 Mbyte prefetchable) | | | | Section 3.1.5 |
| 14h | Base Address 1 (64 Kbyte prefetchable) | | | | Section 3.1.5 |
| 18h | Base Address 2 (64 Kbyte prefetchable) | | | | Section 3.1.5 |
| 1Ch | Base Address 3 (8 Mbyte prefetchable) | | | | Section 3.1.5 |
| 20h | Base Address 4 (8 Mbyte prefetchable) | | | | Section 3.1.5 |
| 24h | Base Address 5 (8 Mbyte prefetchable) | | | | Section 3.1.5 |
| 2Ch | Subsystem ID | | Subsystem Vendor ID | | Section 3.1.6 |
| 34h | Reserved | | | Capabilities Pointer | Section 3.1.7 |
| 3Ch | Maximum Latency | Minimum Grant | Interrupt Pin | Interrupt Line | Section 3.1.8 |

(1) Shaded registers can be autoloaded from an I2C EEPROM.

### 3.1.1 Vendor ID/Device ID Register (PCIVENDEV)

The vendor ID/device ID register (PCIVENDEV) is shown in Figure 11 and described in Table 12.

**Figure 11. Vendor ID/Device ID Register (PCIVENDEV)**

| 31 | 16 |
|---|---|
| DEV_ID | |
| R/W-B002h | |

| 15 | 0 |
|---|---|
| VEN_ID | |
| R/W-104Ch | |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 12. Vendor ID/Device ID Register (PCIVENDEV) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-16 | DEV_ID | B002h | Device ID bits. Identifies a specific device from the manufacturer. |
| 15-0 | VEN_ID | 104Ch | Vendor ID bits. Uniquely identifies the manufacturer of the device. |

### 3.1.2 PCI Command/Status Register (PCICSR)

The PCI command/status register (PCICSR) is shown in Figure 12 and described in Table 13.

**Figure 12. PCI Command/Status Register (PCICSR)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DET_PAR_ERR | SIG_SYS_ERR | RCV_MS_ABRT | RCV_TGT_ABRT | SIG_TGT_ABRT | DEVSEL_TIM | | MS_DPAR_REP |
| R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R-1 | | R/W1C-0 |

| 23 | 22 | 21 | 20 | 19 | 18 | | 16 |
|---|---|---|---|---|---|---|---|
| FAST_BTOB_CAP | Reserved | 66MHZ_CAP[1] | CAP_LIST_IMPL | INT_STAT | Reserved | | |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | | |

| 15 | | | | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | INT_DIS | FAST_BTOB_EN | SERR_N_EN |
| R-0 | | | | | R/W-0 | R-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| WAITCYCLECNTL | PAR_ERR_RES | VGA_PAL_SNP | MEM_WRINV_EN | SP_CYCL | BUS_MS | MEM_SP | IO_SP |
| R-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1C = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

[1] This bit is supported on DM6467T devices only; on other DM646x devices, this bit is hardwired to 0.

**Table 13. PCI Command/Status Register (PCICSR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | DET_PAR_ERR | 0-1 | Detected parity error bit. This bit is set by the PCI to indicate that it detected a parity error, which was not necessarily reported on $\overline{\text{PCI\_PERR}}$ if parity reporting is disabled. |
| 30 | SIG_SYS_ERR | 0-1 | Signaled system error bit. This bit is set by the PCI to indicate that it signaled a system error on the $\overline{\text{PCI\_SERR}}$ pin. |
| 29 | RCV_MS_ABRT | 0-1 | Received master abort bit. This bit is set by the PCI master unit in the PCI to indicate that it terminated a transaction with a master abort. |
| 28 | RCV_TGT_ABRT | 0-1 | Received target abort bit. This bit is set by the PCI master unit in the PCI to indicate that it has received a target abort when acting as a bus master. |
| 27 | SIG_TGT_ABRT | 0-1 | Signaled target abort bit. This bit is set by the PCI slave unit in the PCI to indicate that it has initiated a target abort. |
| 26-25 | DEVSEL_TIM | 1 | DEVSEL timing bits. This bit indicates the decode response time capability of the device. The PCI decode logic supports medium $\overline{\text{DEVSEL}}$ timing; therefore, these bits are hardwired to 01. |
| 24 | MS_DPAR_REP | 0-1 | Master data parity reported bit. This bit is set by the PCI master unit in the PCI when all of the following conditions are met. 1. The PCI asserted $\overline{\text{PCI\_PERR}}$ or observed $\overline{\text{PCI\_PERR}}$ asserted. 2. The PCI master unit was the bus master during the observed $\overline{\text{PCI\_PERR}}$ assertion. 3. The PAR_ERR_RES bit is set. |
| 23 | FAST_BTOB_CAP | 0 | Fast back-to-back capable bit. This bit indicates that the device is capable of performing fast back-to-back transactions. The PCI does not support fast back-to-back transactions; therefore, this bit is hardwired to 0. |
| 22 | Reserved | 0 | Reserved |
| 21 | 66MHZ_CAP | | 66MHz capable bit. This bit indicates whether or not the interface is capable of meeting the 66 MHz PCI timing requirements. This bit is supported on DM6467T devices only. For other DM646x devices, 66 MHz operation is not supported and this bit is hardwired to 0. |
| | | 0 | Not capable of operating in 66 MHz mode. |
| | | 1 | Capable of operating in 66 MHz mode. |
| 20 | CAP_LIST_IMPL | 0 | Capabilities list implemented bit. This bit indicates whether or not the interface provides at least one capabilities list. |
| 19 | INT_STAT | 0 | Interrupt status bit. This bit indicates the current interrupt status for the function as generated by the interrupt registers in the back end interface. If this bit is set and INT_DIS is cleared, the $\overline{\text{PCI\_INTA}}$ pin will be asserted low. INT_DIS has no effect on the value of this bit. |
| 18-11 | Reserved | 0 | Reserved. |

**Table 13. PCI Command/Status Register (PCICSR) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 10 | INT_DIS | | $\overline{\text{PCI\_INTA}}$ disable bit. Controls whether or not the device can assert the $\overline{\text{PCI\_INTA}}$ pin. This bit is a disable for the output driver on the $\overline{\text{PCI\_INTA}}$ pin. |
| | | 0 | Interrupt condition appears on the external $\overline{\text{PCI\_INTA}}$ pin. |
| | | 1 | Interrupt condition does not appear on the external $\overline{\text{PCI\_INTA}}$ pin. |
| 9 | FAST_BTOB_EN | 0 | Fast back-to-back enable bit. Controls whether or not the device is allowed to perform back-to-back writes to different targets. The PCI will not perform fast back-to-back transactions; therefore, this bit is hardwired to 0. |
| 8 | SERR_N_EN | | $\overline{\text{PCI\_SERR}}$ enable bit. This bit is an enable for the output driver on the $\overline{\text{PCI\_SERR}}$ pin. |
| | | 0 | If this bit is cleared and a system error condition is set inside the PCI, the error signal does not appear on the $\overline{\text{PCI\_SERR}}$ pin. |
| | | 1 | The error signal appears on the $\overline{\text{PCI\_SERR}}$ pin. |
| 7 | WAITCYCLECNTL | 0 | Waite cycle control bit. Indicates whether or not the device performs address stepping. The PCI does not support address stepping; therefore this bit is hardwired to 0. |
| 6 | PAR_ERR_RES | | Parity error response bit. Controls whether or not the device responds to detected parity errors. |
| | | 0 | The PCI sets the detected parity error bit (DET_PAR_ERR) when an error is detected, but does not assert $\overline{\text{PCI\_PERR}}$ and continues normal operation. |
| | | 1 | The PCI responds normally to parity errors. |
| 5 | VGA_PAL_SNP | 0 | VGA palette snoop bit. This bit is not applicable for the PCI and is hardwired to 0. |
| 4 | MEM_WRINV_EN | | Memory write and invalidate enable bit. This bit enables the device to use the Memory Write and Invalidate command. |
| | | 0 | The PCI does not attempt to use the Memory Write and Invalidate command. |
| | | 1 | The PCI uses the Memory Write and Invalidate command. |
| 3 | SP_CYCL | | Special cycle bit. Controls the device's response to special cycle commands. |
| | | 0 | The device ignores all special cycle commands. |
| | | 1 | The device monitors special cycle commands. |
| 2 | BUS_MS | | Bus master bit. This bit enables the device to act as a PCI bus master. |
| | | 0 | The device does not act as a master on the PCI bus. |
| | | 1 | The device acts as a master on the PCI bus. |
| 1 | MEM_SP | | Memory access bit. This bit enables the device to respond to memory accesses within its address space. |
| | | 0 | The PCI does not respond to memory-mapped accesses. |
| | | 1 | The PCI responds to memory-mapped accesses. |
| 0 | IO_SP | | IO access bit. This bit enables the device to respond to I/O accesses within its address space. The PCI does not support I/O accesses as a slave; therefore, this bit is hardwired to 0. |

### 3.1.3 Class Code/Revision ID Register (PCICLREV)

The class code/revision ID register (PCICLREV) is shown in Figure 13 and described in Table 14.

**Figure 13. Class Code/Revision ID Register (PCICLREV)**

| 31 24 | 23 16 |
|---|---|
| BASE_CLASS | SUB_CLASS |
| R-11h | R-80h |

| 15 8 | 7 0 |
|---|---|
| REG_LVL | REV_ID |
| R-0 | R-1 |

LEGEND: R = Read only; -*n* = value after reset

**Table 14. Class Code/Revision ID Register (PCICLREV) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | BASE_CLASS | 0-FFh | Base class bits. |
| 23-16 | SUB_CLASS | 0-FFh | Sub-class bits. |
| 15-8 | REG_LVL | 0-FFh | Register-level programming interface bits. |
| 7-0 | REV_ID | 0-FFh | Revision ID bits. Identifies a revision of the specific device. |

### 3.1.4 BIST/Header Type/Latency Timer/Cacheline Size Register (PCICLINE)

The built-in self-test/header type/latency timer/cache line size register (PCICLINE) is register is shown in Figure 14 and described in Table 15.

**Figure 14. BIST/Header Type/Latency Timer/Cacheline Size Register (PCICLINE)**

| 31 — 24 | 23 — 16 |
|---|---|
| BIST | HDR_TYPE |
| R-0 | R-0 |

| 15 — 8 | 7 — 0 |
|---|---|
| LAT_TMR | CACHELN_SIZ |
| R/W-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 15. BIST/Header Type/Latency Timer/Cacheline Size Register (PCICLINE) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | BIST | 0 | Built-in self test bits. Hardwired to 0. |
| 23-16 | HDR_TYPE | 0-FFh | Header type bits. Identifies the layout of bytes 10 through 3F and if the device is single or multi-function. |
| 15-8 | LAT_TMR | 0-FFh | Latency timer bits. These bits are provided so that the host can restrict the continued usage of the PCI bus by a master involved in a multiple data cycle transaction after its $\overline{\text{PCI\_GNT}}$ has been removed. The host is required to write a value into this register indicating the maximum number of PCI cycles for which the master can hold the bus (beginning from the assertion of $\overline{\text{PCI\_FRAME}}$). If $\overline{\text{PCI\_GNT}}$ is never removed during the transaction, the value in the latency timer value will not be used. Since the PCI will support transactions with multiple data cycles, the latency timer register is implemented. The latency timer register is initialized with all zeroes at reset. This register is not cleared on software reset. |
| 7-0 | CACHELN_SIZ | 0-FFh | Cache line size bits. These bits are provided so that the host can inform the device of the cache line size in units of 32-bit words. This value is used by the PCI as a master device to determine whether to use Memory Write, Memory Write and Invalidate, Read, Read Line, or Read Multiple commands for accessing memory. This value is also used by the slave state machine to determine the size of prefetches that are performed on the Slave Back End Interface. Supported values for these bits are as follows; writing an unsupported value to these bits results in the value cleared to 0, as specified in the *PCI Local Bus Specification*. |
| | | 0h | Disabled |
| | | 1h-3h | Unsupported value |
| | | 4h | Cache Line is 16 bytes |
| | | 5h-7h | Unsupported value |
| | | 8h | Cache Line is 32 bytes |
| | | 9h-Fh | Unsupported value |
| | | 10h | Cache Line is 64 bytes |
| | | 11h-1Fh | Unsupported value |
| | | 20h | Cache Line is 128 bytes |
| | | 21h-FFh | Unsupported value |

### 3.1.5 Base Address Registers (PCIBAR0-PCIBAR5)

The base address *n* register (PCIBAR*n*) is shown in Figure 15 and described in Table 16.

**Figure 15. Base Address *n* Register (PCIBAR*n*)**

| 31 – 16 |
|---|
| ADDR |
| R/W-0 |

| 15 – 4 | 3 | 2 – 1 | 0 |
|---|---|---|---|
| ADDR | PREFETCH | TYPE | IOMEM_SP_IND |
| R/W-0 | R-0 or 1[(1)] | R-0 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

[(1)] Reflects the value that is input from the PREFETCH_EN bit in the base address *n* mask register (PCIBAR*n*MSK).

**Table 16. Base Address *n* Register (PCIBAR*n*) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-4 | ADDR | 0-FFF FFFFh | Address bits. These bits can be written by the host to allow initialization of the base address at startup. The writeability of individual bits is determined by the corresponding bit in the base address *n* mask register (PCIBAR*n*MSK). |
| 3 | PREFETCH | 0-1 | Prefetchable bit. Specifies whether or not the memory space controlled by this base address register is prefetchable. This bit reflects the value that is input from the PREFETCH_EN bit in the base address *n* mask register (PCIBAR*n*MSK). |
| 2-1 | TYPE | 0 | Type bits. Indicates the size of the base address register/decoder. This version of the PCI only supports 32-bit addressing, so these bits are hardwired to 0. |
| 0 | IOMEM_SP_IND | 0 | IO/Memory space indicator bit. Indicates whether the base address maps into the host's memory or I/O space. This version of the PCI only supports memory-mapped base address registers, so this bit is hardwired to 0. |

### 3.1.6 Subsystem Vendor ID/Subsystem ID Register (PCISUBID)

The subsystem vendor ID/subsystem ID register (PCISUBID) is shown in Figure 16 and described in Table 17.

**Figure 16. Subsystem Vendor ID/Subsystem ID Register (PCISUBID)**

| 31 — 16 |
|---|
| SUBSYS_ID |
| R-0 |

| 15 — 0 |
|---|
| SUBSYS_VEN_ID |
| R-0 |

LEGEND: R = Read only; -*n* = value after reset

**Table 17. Subsystem Vendor ID/Subsystem ID Register (PCISUBID) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-16 | SUBSYS_ID | 0-FFFFh | Subsystem ID bits. Identifies the board level device. The subsystem ID is specified by the board level manufacturer. The actual value is usually loaded through a serial EEPROM interface. |
| 15-0 | SUBSYS_VEN_ID | 0-FFFFh | Subsystem vendor ID bits. Identifies the board level manufacturer. The subsystem vendor ID is specified by the PCI Special Interest Group. The actual value is usually loaded through a serial EEPROM interface. |

### 3.1.7 Capabilities Pointer Register (PCICPBPTR)

The capabilities pointer register (PCICPBPTR) is shown in Figure 17 and described in Table 18.

**Figure 17. Capabilities Pointer Register (PCICPBPTR)**

| 31 — 16 |
|---|
| Reserved |
| R-0 |

| 15 — 8 | 7 — 0 |
|---|---|
| Reserved | CAP |
| R-0 | R-40h |

LEGEND: R = Read only; -*n* = value after reset

**Table 18. Capabilities Pointer Register (PCICPBPTR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-8 | Reserved | 0 | Reserved |
| 7-0 | CAP | 0-FFh | Capabilities pointer bits. Specifies the address in configuration space where the first entry in the capabilities list is located. |

### 3.1.8 Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Register (PCILGINT)

The maximum latency/minimum grant/interrupt pin/interrupt line register (PCILGINT) is shown in Figure 18 and described in Table 19.

**Figure 18. Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Register (PCILGINT)**

| 31–24 | 23–16 |
|---|---|
| MAX_LAT | MIN_GRNT |
| R-0 | R-0 |

| 15–8 | 7–0 |
|---|---|
| INT_PIN | INT_LINE |
| R-1 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 19. Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Register (PCILGINT) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | MAX_LAT | 0-FFh | Maximum latency bits. Specifies how often the device needs to gain access to the PCI bus in 0.25 μsec units. This field reflects the value which is input on the maximum latency port on the PCI module. |
| 23-16 | MIN_GRNT | 0-FFh | Minimum grant bits. Specifies the length of the burst period for the device needs in 0.25 μsec units. This field reflects the value which is input from the minimum grant port on the PCI module. |
| 15-8 | INT_PIN | 1 | Interrupt pin bits. Specifies the interrupt pin the device uses. This bit is hardwired to 1h in the PCI to indicate that interrupt A will be used. |
| 7-0 | INT_LINE | 0-FFh | Interrupt line bits. This value is written by the host and indicates to which input of the system interrupt controller the PCI interrupt pin is connected. These bits are not cleared on software reset. |

## 3.2 PCI Back-End Configuration Registers

The back-end configuration registers provide the back-end application access to control and status information within the PCI. The ARM can access all of the PCI configuration registers through the PCI memory-mapped registers. An external PCI host can indirectly access the PCI memory-mapped registers through base address registers. Portions of the back-end configuration registers are located inside the peripheral clock domain and the remaining registers are located inside the PCI clock (PCI_CLK) domain. Table 20 lists the registers that are located in the peripheral clock domain. Registers in the PCI clock domain can only be accessed if both the PCI and peripheral clocks are running. Registers in the peripheral clock domain can be accessed even if the PCI clock is not running.

**Table 20. PCI Back-End Configuration Registers**

| Offset | Acronym | Register Description | Section |
|---|---|---|---|
| 10h | PCISTATSET[1] | Status Set Register | Section 3.2.1 |
| 14h | PCISTATCLR[1] | Status Clear Register | Section 3.2.2 |
| 20h | PCIHINTSET[1] | Host Interrupt Enable Set Register | Section 3.2.3 |
| 24h | PCIHINTCLR[1] | Host Interrupt Enable Clear Register | Section 3.2.4 |
| 30h | PCIBINTSET[1] | Back End Application Interrupt Enable Set Register | Section 3.2.5 |
| 34h | PCIBINTCLR[1] | Back End Application Interrupt Enable Clear Register | Section 3.2.6 |
| 100h | PCIVENDEVMIR | Vendor ID/Device ID Mirror Register | Section 3.2.7 |
| 104h | PCICSRMIR | Command/Status Mirror Register | Section 3.2.8 |
| 108h | PCICLREVMIR | Class Code/Revision ID Mirror Register | Section 3.2.9 |
| 10Ch | PCICLINEMIR | BIST/Header Type/Latency Timer/Cacheline Size Mirror Register | Section 3.2.10 |
| 110h | PCIBAR0MSK | Base Address 0 Mask Register | Section 3.2.11 |
| 114h | PCIBAR1MSK | Base Address 1 Mask Register | Section 3.2.11 |
| 118h | PCIBAR2MSK | Base Address 2 Mask Register | Section 3.2.11 |
| 11Ch | PCIBAR3MSK | Base Address 3 Mask Register | Section 3.2.11 |
| 120h | PCIBAR4MSK | Base Address 4 Mask Register | Section 3.2.11 |
| 124h | PCIBAR5MSK | Base Address 5 Mask Register | Section 3.2.11 |
| 12Ch | PCISUBIDMIR | Subsystem Vendor ID/Subsystem ID Mirror Register | Section 3.2.12 |
| 134h | PCICPBPTRMIR | Capabilities Pointer Mirror Register | Section 3.2.13 |
| 13Ch | PCILGINTMIR | Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register | Section 3.2.14 |
| 180h | PCISLVCNTL | Slave Control Register | Section 3.2.15 |
| 1C0h | PCIBAR0TRL | Slave Base Address 0 Translation Register | Section 3.2.16 |
| 1C4h | PCIBAR1TRL | Slave Base Address 1 Translation Register | Section 3.2.16 |
| 1C8h | PCIBAR2TRL | Slave Base Address 2 Translation Register | Section 3.2.16 |
| 1CCh | PCIBAR3TRL | Slave Base Address 3 Translation Register | Section 3.2.16 |
| 1D0h | PCIBAR4TRL | Slave Base Address 4 Translation Register | Section 3.2.16 |
| 1D4h | PCIBAR5TRL | Slave Base Address 5 Translation Register | Section 3.2.16 |
| 1E0h | PCIBAR0MIR | Base Address 0 Mirror Register | Section 3.2.17 |
| 1E4h | PCIBAR1MIR | Base Address 1 Mirror Register | Section 3.2.17 |
| 1E8h | PCIBAR2MIR | Base Address 2 Mirror Register | Section 3.2.17 |
| 1ECh | PCIBAR3MIR | Base Address 3 Mirror Register | Section 3.2.17 |
| 1F0h | PCIBAR4MIR | Base Address 4 Mirror Register | Section 3.2.17 |
| 1F4h | PCIBAR5MIR | Base Address 5 Mirror Register | Section 3.2.17 |
| 300h | PCIMCFGDAT | Master Configuration/IO Access Data Register | Section 3.2.18.1 |
| 304h | PCIMCFGADR | Master Configuration/IO Access Address Register | Section 3.2.18.2 |
| 308h | PCIMCFGCMD | Master Configuration/IO Access Command Register | Section 3.2.18.3 |
| 310h | PCIMSTCFG | Master Configuration Register | Section 3.2.19 |

(1) These PCI Back End Configuration Registers are located in the peripheral clock domain. Registers in the peripheral clock domain can be accessed even if the PCI clock (PCI_CLK) is not running. Other registers in the PCI Back End Configuration Registers are located in the PCI clock domain, and can only be accessed if both the PCI and the peripheral clocks are running.

### 3.2.1 Status Set Register (PCISTATSET)

The PCI includes an internal status register that is not directly writable by the DM646x DMSoC for software simplification. As a result, two registers, the status set register (PCISTATSET) and the status clear register (PCISTATCLR), are provided to set or clear bits in the internal status register.

PCISTATSET and PCISTATCLR both return the contents of the internal status register during reads. Writing a 1 to any of the bits in PCISTATSET sets the corresponding bit in the internal status register. Writing a 1 to any of the bits in PCISTATCLR clears the corresponding bit in the internal status register as long as the corresponding condition that sets that bit is not also asserted.

Bits in the status register cannot be cleared unless the underlying condition that caused the bit to be set has also been cleared. This is important for the $\overline{\text{PCI\_INTA}}$, $\overline{\text{PCI\_PERR}}$, and $\overline{\text{PCI\_SERR}}$ interrupts that are generated and accessed on a level-sensitive external pin. Additionally, there is a synchronization delay of 3 peripheral clocks present between the time that a level-sensitive status condition is cleared in the PCI clock domain and when that condition will be cleared in the peripheral clock domain. Interrupt service routines need to be designed to include this delay.

The PCISTATSET is shown in Figure 19 and described in Table 21.

**Figure 19. Status Set Register (PCISTATSET)**

| 31 | 30 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|
| Reserved | Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 |

| 23 8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1S-0 | R/W1S-0 | R-0 | R/W1S-0 | R/W1S-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1S = Write 1 to set, write of 0 has no effect; -*n* = value after reset

**Table 21. Status Set Register (PCISTATSET) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | Reserved | 0 | Reserved |
| 30-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt set bits. Writing a 1 to these bits sets the corresponding software interrupt. If the corresponding bit in the PCI host interrupt enable register is also set to 1, an interrupt is generated to the host via the $\overline{\text{PCI\_INTA}}$ pin. The host-to-ARM interrupt (INT) is also generated if the corresponding bit in the PCI back end application interrupt enable register is set to 1. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Generate software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect bit. Writing a 1 to this bit sets a parity error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Set parity error. |
| 5 | SERR_DET | | System error detect bit. Writing a 1 to this bit sets a system error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Set the system error. |
| 4-3 | Reserved | 0 | Reserved |

**Table 21. Status Set Register (PCISTATSET) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 2 | MS_ABRT_DET | | Master abort detect bit. Writing a 1 to this bit sets master abort. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Set the master abort error. |
| 1 | TGT_ABRT_DET | | Target abort detect bit. Writing a 1 to this bit sets target abort. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Set the target abort error. |
| 0 | Reserved | 0 | Reserved |

### 3.2.2 Status Clear Register (PCISTATCLR)

The PCI includes an internal status register that is not directly writable by the DM646x DMSoC for software simplification. As a result, two registers, the status set register (PCISTATSET) and the status clear register (PCISTATCLR), are provided to set or clear bits in the internal status register.

PCISTATSET and PCISTATCLR both return the contents of the internal status register during reads. Writing a 1 to any of the bits in PCISTATSET sets the corresponding bit in the internal status register. Writing a 1 to any of the bits in PCISTATCLR clears the corresponding bit in the internal status register as long as the corresponding condition that sets that bit is not also asserted.

Bits in the status register cannot be cleared unless the underlying condition that caused the bit to be set has also been cleared. This is important for the $\overline{\mathrm{PCI\_INTA}}$, $\overline{\mathrm{PCI\_PERR}}$, and $\overline{\mathrm{PCI\_SERR}}$ interrupts that are generated and accessed on a level-sensitive external pin. Additionally, there is a synchronization delay of 3 peripheral clocks present between the time that a level-sensitive status condition is cleared in the PCI clock domain and when that condition will be cleared in the peripheral clock domain. Interrupt service routines need to be designed to include this delay.

The PCISTATCLR is shown in Figure 20 and described in Table 22.

**Figure 20. Status Clear Register (PCISTATCLR)**

| 31 | 30–28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|
| INT | Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 |

| 23–8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4–3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1C-0 | R/W1C-0 | R-0 | R/W1C-0 | R/W1C-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1C = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

**Table 22. Status Clear Register (PCISTATCLR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | Reserved | 0 | Reserved. |
| 30-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt clear bits. Writing a 1 to these bits clears the corresponding software interrupt. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Clear software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect bit. Writing 1 to this bit clears the parity error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Clears the parity error. |
| 5 | SERR_DET | | System error detect bit. Writing 1 to this bit clears the system error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Clears the system error. |
| 4-3 | Reserved | 0 | Reserved |
| 2 | MS_ABRT_DET | | Master abort detect bit. Writing 1 to this bit clears the master abort error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Clears the master abort error. |

**Table 22. Status Clear Register (PCISTATCLR) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 1 | TGT_ABRT_DET | | Target abort detect bit. Writing 1 to this bit clears the target abort error. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Clears the target abort error. |
| 0 | Reserved | 0 | Reserved |

### 3.2.3 Host Interrupt Enable Set Register (PCIHINTSET)

The PCI includes an internal host interrupt enable register that for software simplification is not directly writable by the host. As a result, two registers, the host interrupt enable set register (PCIHINTSET) and the host interrupt enable clear register (PCIHINTCLR), are provided to set or clear bits in the internal host interrupt enable register.

The host uses the internal host interrupt enable register to configure on which status conditions an interrupt will be generated to the host. A level-sensitive active-low interrupt is generated to the host on the $\overline{\text{PCI\_INTA}}$ pin if a bit in the internal status register is asserted and the corresponding bit in the internal host interrupt enable register is also asserted, as long as the PCI is in the D0 power state. Interrupt generation on the $\overline{\text{PCI\_INTA}}$ pin is disabled whenever the PCI is in the D1, D2, or D3 power states.

Writing a 1 to any of the bits in PCIHINTSET sets the corresponding bit in the internal host interrupt enable register. Writing a 1 to any of the bits in PCIHINTCLR clears the corresponding bit in the internal host interrupt enable register.

Reading PCIHINTSET returns the internal host interrupt enable register contents. Reading from PCIHINTCLR returns the masked host status that is the bitwise ANDing of the internal status register and the internal host interrupt enable register. PCIHINTCLR is typically read by the host to determine the interrupt source when the $\overline{\text{PCI\_INTA}}$ pin is asserted. An external PCI host can indirectly access the PCI memory-mapped registers through base address registers.

The PCIHINTSET is shown in Figure 21 and described in Table 23.

**Figure 21. Host Interrupt Enable Set Register (PCIHINTSET)**

| 31-28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|
| Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 |

| 23-8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4-3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1S-0 | R/W1S-0 | R-0 | R/W1S-0 | R/W1S-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1S = Write 1 to set, write of 0 has no effect; -*n* = value after reset

**Table 23. Host Interrupt Enable Set Register (PCIHINTSET) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt enable bits. Writing a 1 to these bits enables the corresponding software interrupt. When the corresponding bit in the internal Status Register is set to 1, an interrupt is generated to the host via the $\overline{\text{PCI\_INTA}}$ pin. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect enable bit. Writing 1 to this bit enables the parity error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the parity error. |
| 5 | SERR_DET | | System error detect enable bit. Writing 1 to this bit enables the system error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the system error. |
| 4-3 | Reserved | 0 | Reserved |

**Table 23. Host Interrupt Enable Set Register (PCIHINTSET) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 2 | MS_ABRT_DET | | Master abort detect enable bit. Writing 1 to this bit enables the master abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the master abort detection. |
| 1 | TGT_ABRT_DET | | Target abort detect enable bit. Writing 1 to this bit enables the target abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the target abort detection. |
| 0 | Reserved | 0 | Reserved |

### 3.2.4 Host Interrupt Enable Clear Register (PCIHINTCLR)

The PCI includes an internal host interrupt enable register that for software simplification is not directly writable by the host. As a result, two registers, the host interrupt enable set register (PCIHINTSET) and the host interrupt enable clear register (PCIHINTCLR), are provided to set or clear bits in the internal host interrupt enable register.

The host uses the internal host interrupt enable register to configure on which status conditions an interrupt will be generated to the host. A level-sensitive active-low interrupt is generated to the host on the $\overline{\text{PCI\_INTA}}$ pin if a bit in the internal status register is asserted and the corresponding bit in the internal host interrupt enable register is also asserted, as long as the PCI is in the D0 power state. Interrupt generation on the $\overline{\text{PCI\_INTA}}$ pin is disabled whenever the PCI is in the D1, D2, or D3 power states.

Writing a 1 to any of the bits in PCIHINTSET sets the corresponding bit in the internal host interrupt enable register. Writing a 1 to any of the bits in PCIHINTCLR clears the corresponding bit in the internal host interrupt enable register.

Reading PCIHINTSET returns the internal host interrupt enable register contents. Reading from PCIHINTCLR returns the masked host status that is the bitwise ANDing of the internal status register and the internal host interrupt enable register. PCIHINTCLR is typically read by the host to determine the interrupt source when the $\overline{\text{PCI\_INTA}}$ pin is asserted. An external PCI host can indirectly access the PCI memory-mapped registers through base address registers.

The PCIHINTCLR is shown in Figure 22 and described in Table 24.

**Figure 22. Host Interrupt Enable Clear Register (PCIHINTCLR)**

| 31-28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|
| Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 |

| 23-8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4-3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1C-0 | R/W1C-0 | R-0 | R/W1C-0 | R/W1C-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1C = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

**Table 24. Host Interrupt Enable Clear Register (PCIHINTCLR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt disable bits. Writing a 1 to these bits disables the corresponding software interrupt. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect disable bit. Writing 1 to this bit disables the parity error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the parity error. |
| 5 | SERR_DET | | System error detect disable bit. Writing 1 to this bit disables the system error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the system error. |
| 4-3 | Reserved | 0 | Reserved |
| 2 | MS_ABRT_DET | | Master abort detect disable bit. Writing 1 to this bit disables the master abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the master abort detection. |

**Table 24. Host Interrupt Enable Clear Register (PCIHINTCLR) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 1 | TGT_ABRT_DET | | Target abort detect disable bit. Writing 1 to this bit disables the target abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the target abort detection. |
| 0 | Reserved | 0 | Reserved |

### 3.2.5 Back-End Application Interrupt Enable Set Register (PCIBINTSET)

The PCI includes an internal back-end application interrupt enable register that is not directly writable by the back-end application for software simplification. As a result, two registers, the back-end application interrupt enable set register (PCIBINTSET) and the back-end application interrupt enable clear register (PCIBINTCLR) are provided to set or clear bits in the internal back-end application interrupt enable register.

Writing a 1 to any of the bits in PCIBINTSET sets the corresponding bit in the internal back-end application interrupt enable register. Writing a 1 to any of the bits in PCIBINTCLR clears the corresponding bit in the internal back-end application interrupt enable register.

Reading PCIBINTSET returns the internal back-end application interrupt enable register contents. Reading from PCIBINTCLR returns the masked back-end application status that is the bitwise ANDing of the internal status register and the internal back-end application interrupt enable register. PCIBINTCLR is typically read by the back-end application to determine the source of a back-end interrupt.

**NOTE:** Clearing the INT bit of the internal back-end application interrupt enable register will not gate all interrupts to the ARM. An interrupt will be generated to the ARM via the INT interrupt line, if any bit in the internal back-end application interrupt enable is set to 1 and the corresponding bit in the internal status register is also set to 1.

The PCIBINTSET is shown in Figure 23 and described in Table 25.

**Figure 23. Back-End Application Interrupt Enable Set Register (PCIBINTSET)**

| 31 | 30 - 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|
| Reserved | Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 | R/W1S-0 |

| 23 - 8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4 - 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1S-0 | R/W1S-0 | R-0 | R/W1S-0 | R/W1S-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1S = Write 1 to set, write of 0 has no effect; -*n* = value after reset

**Table 25. Back-End Application Interrupt Enable Set Register (PCIBINTSET) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | Reserved | 0 | Reserved. |
| 30-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt enable bits. Writing a 1 to these bits enables the corresponding software interrupt. When the corresponding bit in the internal Status Register is set to 1, an interrupt is generated to the ARM via the INT interrupt line. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect enable bit. Writing 1 to this bit enables the parity error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the parity error. |
| 5 | SERR_DET | | System error detect enable bit. Writing 1 to this bit enables the system error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the system error. |

**Table 25. Back-End Application Interrupt Enable Set Register (PCIBINTSET) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 4-3 | Reserved | 0 | Reserved |
| 2 | MS_ABRT_DET | | Master abort detect enable bit. Writing 1 to this bit enables the master abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the master abort detection. |
| 1 | TGT_ABRT_DET | | Target abort detect enable bit. Writing 1 to this bit enables the target abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Enables the target abort detection. |
| 0 | Reserved | 0 | Reserved |

### 3.2.6 Back-End Application Interrupt Enable Clear Register (PCIBINTCLR)

The PCI includes an internal back-end application interrupt enable register that is not directly writable by the back-end application for software simplification. As a result, two registers, the back-end application interrupt enable set register (PCIBINTSET) and the back-end application interrupt enable clear register (PCIBINTCLR) are provided to set or clear bits in the internal back-end application interrupt enable register.

Writing a 1 to any of the bits in PCIBINTSET sets the corresponding bit in the internal back-end application interrupt enable register. Writing a 1 to any of the bits in PCIBINTCLR clears the corresponding bit in the internal back-end application interrupt enable register.

Reading PCIBINTSET returns the internal back-end application interrupt enable register contents. Reading from PCIBINTCLR returns the masked back-end application status that is the bitwise ANDing of the internal status register and the internal back-end application interrupt enable register. PCIBINTCLR is typically read by the back-end application to determine the source of a back-end interrupt.

**NOTE:** Clearing the INT bit of the internal back-end application interrupt enable register will not gate all interrupts to the ARM. An interrupt will be generated to the ARM via the INT interrupt line, if any bit in the internal back-end application interrupt enable is set to 1 and the corresponding bit in the internal status register is also set to 1.

The PCIBINTCLR is shown in Figure 24 and described in Table 26.

**Figure 24. Back-End Application Interrupt Enable Clear Register (PCIBINTCLR)**

| 31 | 30 – 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|
| Reserved | Reserved | SOFT_INT3 | SOFT_INT2 | SOFT_INT1 | SOFT_INT0 |
| R-0 | R-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 |

| 23 – 8 |
|---|
| Reserved |
| R-0 |

| 7 | 6 | 5 | 4 – 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Reserved | PERR_DET | SERR_DET | Reserved | MS_ABRT_DET | TGT_ABRT_DET | Reserved |
| R-0 | R/W1C-0 | R/W1C-0 | R-0 | R/W1C-0 | R/W1C-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1C = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

**Table 26. Back-End Application Interrupt Enable Clear Register (PCIBINTCLR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | Reserved | 0 | Reserved. |
| 30-28 | Reserved | 0 | Reserved |
| 27-24 | SOFT_INT*n* | | Software interrupt disable bits. Writing a 1 to these bits disables the corresponding software interrupt. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables software interrupt. |
| 23-7 | Reserved | 0 | Reserved |
| 6 | PERR_DET | | Parity error detect disable bit. Writing 1 to this bit disables the parity error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the parity error. |
| 5 | SERR_DET | | System error detect disable bit. Writing 1 to this bit disables the system error detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the system error. |

Table 26. Back-End Application Interrupt Enable Clear Register (PCIBINTCLR) Field Descriptions (continued)

| Bit | Field | Value | Description |
|---|---|---|---|
| 4-3 | Reserved | 0 | Reserved |
| 2 | MS_ABRT_DET | | Master abort detect disable bit. Writing 1 to this bit disables the master abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the master abort detection. |
| 1 | TGT_ABRT_DET | | Target abort detect disable bit. Writing 1 to this bit disables the target abort detection. |
| | | 0 | A write of 0 is ignored. |
| | | 1 | Disables the target abort detection. |
| 0 | Reserved | 0 | Reserved |

## 3.2.7 Vendor ID/Device ID Mirror Register (PCIVENDEVMIR)

The vendor ID/device ID mirror register (PCIVENDEVMIR) is used to initialize the vendor ID and/or device ID in the configuration space vendor ID/device ID register (PCIVENDEV) prior to enabling configuration accesses. The PCIVENDEVMIR is typically written by either an EEPROM controller or an on-chip CPU. The PCIVENDEVMIR is shown in Figure 25 and described in Table 27.

Figure 25. Vendor ID/Device ID Mirror Register (PCIVENDEVMIR)

| 31 - 16 |
|---|
| DEV_ID |
| R/W-B002h |

| 15 - 0 |
|---|
| VEN_ID |
| R/W-104Ch |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 27. Vendor ID/Device ID Mirror Register (PCIVENDEVMIR) Field Descriptions

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-16 | DEV_ID | B002h | Device ID bits. Identifies a specific device from the manufacturer. |
| 15-0 | VEN_ID | 104Ch | Vendor ID bits. Uniquely identifies the manufacturer of the device. |

### 3.2.8 Command/Status Mirror Register (PCICSRMIR)

The command/status mirror register (PCICSRMIR) initializes certain bits in the configuration space command/status register (PCICSR) prior to enabling configuration accesses. The PCICSRMIR is typically written by either an EEPROM controller or an on-chip CPU. The PCICSRMIR can also be used by the back-end application to directly query the status of the PCI. The PCICSRMIR is shown in Figure 26 and described in Table 28.

**Figure 26. Command/Status Mirror Register (PCICSRMIR)**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| DET_PAR_ERR | SIG_SYS_ERR | RCV_MS_ABRT | RCV_TGT_ABRT | SIG_TGT_ABRT | DEVSEL_TIM | | MS_DPAR_REP |
| R/W1C-0 | R/W1C-0 | R/W1C-0 | R/W1C-0 | R-0 | R-1 | | R/W1C-0 |

| 23 | 22 | 21 | 20 | 19 | 18 | 16 |
|---|---|---|---|---|---|---|
| FAST_BTOB_CAP | Reserved | 66MHZ_CAP[1] | CAP_LIST_IMPL | INT_STAT | Reserved | |
| R-0 | R-0 | R/W-0 | R/W-0 | R-0 | R-0 | |

| 15 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|
| Reserved | | INT_DIS | FAST_BTOB_EN | SERR_N_EN |
| R-0 | | R/W-0 | R-0 | R/W-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | PAR_ERR_RES | VGA_PAL_SNP | MEM_WRINV_EN | SP_CYCL | BUS_MS | MEM_SP | IO_SP |
| R-0 | R/W-0 | R-0 | R/W-0 | R-0 | R/W-0 | R/W-0 | R-0 |

LEGEND: R = Read only; R/W = Read/Write; W1C = Write 1 to clear, write of 0 has no effect; -*n* = value after reset

[1] This bit is supported on DM6467T devices only; on other DM646x devices, this bit is hardwired to 0.

**Table 28. Command/Status Mirror Register (PCICSRMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | DET_PAR_ERR | 0-1 | Detected parity error bit. This bit is set by the PCI to indicate that it detected a parity error, which was not necessarily reported on $\overline{\mathrm{PCI\_PERR}}$ if parity reporting is disabled. |
| 30 | SIG_SYS_ERR | 0-1 | Signaled system error bit. This bit is set by the PCI to indicate that it signaled a system error on the $\overline{\mathrm{PCI\_SERR}}$ pin. |
| 29 | RCV_MS_ABRT | 0-1 | Received master abort bit. This bit is set by the PCI master unit in the PCI to indicate that it terminated a transaction with a master abort. |
| 28 | RCV_TGT_ABRT | 0-1 | Received target abort bit. This bit is set by the PCI master unit in the PCI to indicate that it has received a target abort when acting as a bus master. |
| 27 | SIG_TGT_ABRT | 0-1 | Signaled target abort bit. This bit is always 0 because the PCI cannot issue a target abort. |
| 26-25 | DEVSEL_TIM | 1 | DEVSEL timing bits. This bit indicates the decode response time capability of the device. The PCI decode logic supports medium $\overline{\mathrm{DEVSEL}}$ timing; therefore, these bits are hardwired to 01. |
| 24 | MS_DPAR_REP | 0-1 | Master data parity reported bit. This bit is set by the PCI master unit in the PCI when all of the following conditions are met: 1. The PCI asserted $\overline{\mathrm{PCI\_PERR}}$ or observed $\overline{\mathrm{PCI\_PERR}}$ asserted 2. The PCI master unit was the bus master during the observed $\overline{\mathrm{PCI\_PERR}}$ assertion 3. The Parity Error Response bit (PAR_ERR_RES) is set |
| 23 | FAST_BTOB_CAP | 0 | Fast back to back capable bit. This bit indicates that the device is capable of performing fast back to back transactions. The PCI does not support fast back to back transactions; therefore, this bit is hardwired to 0. |
| 22 | Reserved | 0 | Reserved |
| 21 | 66MHZ_CAP | | 66MHz capable bit. This bit indicates whether or not the interface is capable of meeting the 66 MHz PCI timing requirements. This bit is supported on DM6467T devices only. For other DM646x devices, 66 MHz operation is not supported and this bit is hardwired to 0. |
| | | 0 | Not capable of operating in 66 MHz mode. |
| | | 1 | Capable of operating in 66 MHz mode. |
| 20 | CAP_LIST_IMPL | 0-1 | Capabilities list implemented bit. This bit indicates whether or not the interface provides at least one capabilities list. |

**Table 28. Command/Status Mirror Register (PCICSRMIR) Field Descriptions (continued)**

| Bit | Field | Value | Description |
|---|---|---|---|
| 19 | INT_STAT | 0 | Interrupt status bit. This bit indicates the current interrupt status for the function as generated by the interrupt registers in the back end interface. If this bit is set and INT_DIS is cleared, the $\overline{\text{PCI\_INTA}}$ pin will be asserted low. INT_DIS has no effect on the value of this bit. |
| 18-11 | Reserved | 0 | Reserved |
| 10 | INT_DIS | 0-1 | Interrupt disable bit. This bit controls whether or not the device can assert the $\overline{\text{PCI\_INTA}}$ pin. This bit is a disable for the output driver on the $\overline{\text{PCI\_INTA}}$ pin. If this bit is set, the interrupt condition will not appear on the external $\overline{\text{PCI\_INTA}}$ pin. |
| 9 | FAST_BTOB_EN | 0 | Fast back to back enable bit. This bit controls whether or not the device is allowed to perform back to back writes to different targets. The PCI will not perform fast back to back transactions; therefore, this bit is hardwired to a 0. |
| 8 | SERR_N_EN | 0-1 | SERR_N enable bit. This bit is an enable for the output driver on the $\overline{\text{PCI\_SERR}}$ pin. If this bit is cleared, and a system error condition is set inside the PCI, the error signal will not appear on the external $\overline{\text{PCI\_SERR}}$ pin. |
| 7 | Reserved | 0 | Reserved |
| 6 | PAR_ERR_RES | 0-1 | Parity error response bit. This bit controls whether or not the device responds to detected parity errors. If this bit is set, the PCI will respond normally to parity errors. If this bit is cleared, the PCI will set its Detected Parity Error status bit (DET_PAR_ERR) when an error is detected, but does not assert $\overline{\text{PCI\_PERR}}$ and continues normal operation. |
| 5 | VGA_PAL_SNP | 0 | VGA Palette Snoop bit. This bit is not generally applicable for the PCI and is hardwired to a 0. |
| 4 | MEM_WRINV_EN | 0-1 | Memory write and invalidate enable bit. This bit enables the device to use the Memory Write and Invalidate command. If this bit is cleared, the PCI will not attempt to use the Memory Write and Invalidate command. |
| 3 | SP_CYCL | 0-1 | Special cycle bit. This bit controls the device's response to special cycle commands. If this bit is cleared, the device will ignore all special cycle commands. If this bit is set to 1, the device can monitor special cycle commands. |
| 2 | BUS_MS | 0-1 | Bus master bit. This bit enables the device to act as a PCI bus master. If this bit is cleared, the device will not act as a master on the PCI bus. |
| 1 | MEM_SP | 0-1 | Memory access bit. This bit enables the device to respond to memory accesses within its address space. If this bit is cleared, the PCI will not respond to memory mapped accesses. |
| 0 | IO_SP | 0 | IO access bit. This bit enables the device to respond to I/O accesses within its address space. The PCI does not support IO accesses as a slave; therefore, this bit is hardwired to 0. |

### 3.2.9 Class Code/Revision ID Mirror Register (PCICLREVMIR)

The class code/revision ID mirror register (PCICLREVMIR) is used to initialize the class code and revision ID in the configuration space class code/revision ID register (PCICLREV) prior to enabling configuration accesses. The PCICLREVMIR is typically written by either an EEPROM controller or an on-chip CPU. The PCICLREVMIR is shown in Figure 27 and described in Table 29.

**Figure 27. Class Code/Revision ID Mirror Register (PCICLREVMIR)**

| 31 24 | 23 16 |
|---|---|
| BASE_CLASS | SUB_CLASS |
| R/W-11h | R/W-80h |

| 15 8 | 7 0 |
|---|---|
| REG_LVL | REV_ID |
| R/W-0 | R/W-1 |

LEGEND: R = Read only; -*n* = value after reset

**Table 29. Class Code/Revision ID Mirror Register (PCICLREVMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | BASE_CLASS | 0-FFh | Base class bits. |
| 23-16 | SUB_CLASS | 0-FFh | Sub-class bits. |
| 15-8 | REG_LVL | 0-FFh | Register-level programming interface bits. |
| 7-0 | REV_ID | 0-FFh | Revision ID bits. Identifies a revision of the specific device. |

### 3.2.10 BIST/Header Type/Latency Timer/Cacheline Size Mirror Register (PCICLINEMIR)

The built-in self-test/header type/latency timer/cache line size mirror register (PCICLINEMIR) is used to set latency timer and cacheline size. The PCICLINEMIR is shown in Figure 28 and described in Table 30.

**Figure 28. BIST/Header Type/Latency Timer/Cacheline Size Mirror Register (PCICLINEMIR)**

| 31-24 | 23-16 |
|---|---|
| BIST | HDR_TYPE |
| R-0 | R-0 |

| 15-8 | 7-0 |
|---|---|
| LAT_TMR | CACHELN_SIZ |
| R/W-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 30. BIST/Header Type/Latency Timer/Cacheline Size Mirror Register (PCICLINEMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | BIST | 0 | Built-in self test bits. Hardwired to 0. |
| 23-16 | HDR_TYPE | 0-FFh | Header type bits. Identifies the layout of bytes 10 through 3F and if the device is single or multi-function. |
| 15-8 | LAT_TMR | 0-FFh | Latency timer bits. These bits are provided so that the host can restrict the continued usage of the PCI bus by a master involved in a multiple data cycle transaction after its $\overline{\text{PCI\_GNT}}$ has been removed. The host is required to write a value into this register indicating the maximum number of PCI cycles for which the master can hold the bus (beginning from the assertion of $\overline{\text{PCI\_FRAME}}$). If $\overline{\text{PCI\_GNT}}$ is never removed during the transaction, the value in the latency timer value will not be used. Since the PCI will support transactions with multiple data cycles, the latency timer register is implemented. The latency timer register is initialized with all zeroes at reset. This register is not cleared on software reset. |
| 7-0 | CACHELN_SIZ | 0-FFh | Cache line size bits. These bits are provided so that the host can inform the device of the cache line size in units of 32-bit words. This value is used by the PCI as a master device to determine whether to use Memory Write, Memory Write and Invalidate, Read, Read Line, or Read Multiple commands for accessing memory. This value is also used by the slave state machine to determine the size of prefetches that are performed on the Slave Back End Interface. Supported values for these bits are as follows; writing an unsupported value to these bits results in the value cleared to 0, as specified in the *PCI Local Bus Specification*. |
| | | 0h | Disabled |
| | | 1h-3h | Unsupported value |
| | | 4h | Cache Line is 16 bytes |
| | | 5h-7h | Unsupported value |
| | | 8h | Cache Line is 32 bytes |
| | | 9h-Fh | Unsupported value |
| | | 10h | Cache Line is 64 bytes |
| | | 11h-1Fh | Unsupported value |
| | | 20h | Cache Line is 128 bytes |
| | | 21h-FFh | Unsupported value |

### 3.2.11 Base Address Mask Registers (PCIBAR0MSK-PCIBAR5MSK)

The base address *n* mask register (PCIBAR*n*MSK) controls the size and prefetchability of the PCI configuration base address *n* register (PCIBAR*n*). The base address mask registers are shown in Figure 29 through Figure 34 and described in Table 31.

**Figure 29. Base Address 0 Mask Register (PCIBAR0MSK)**

| 31–16 |
|---|
| ADDRMASK |
| R/W-FFF FC00h |

| 15–4 | 3 | 2–0 |
|---|---|---|
| ADDRMASK | PREFETCH_EN | Reserved |
| R/W-FFF FC00h | R/W-1 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 30. Base Address 1 Mask Register (PCIBAR1MSK)**

| 31–16 |
|---|
| ADDRMASK |
| R/W-FFF F800h |

| 15–4 | 3 | 2–0 |
|---|---|---|
| ADDRMASK | PREFETCH_EN | Reserved |
| R/W-FFF F800h | R/W-1 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 31. Base Address 2 Mask Register (PCIBAR2MSK)**

| 31–16 |
|---|
| ADDRMASK |
| R/W-FFC 0000h |

| 15–4 | 3 | 2–0 |
|---|---|---|
| ADDRMASK | PREFETCH_EN | Reserved |
| R/W-FFC 0000h | R/W-1 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 32. Base Address 3 Mask Register (PCIBAR3MSK)**

| 31–16 |
|---|
| ADDRMASK |
| R/W-FFF E000h |

| 15–4 | 3 | 2–0 |
|---|---|---|
| ADDRMASK | PREFETCH_EN | Reserved |
| R/W-FFF E000h | R/W-1 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 33. Base Address 4 Mask Register (PCIBAR4MSK)**

| 31 | 16 |
|---|---|
| ADDRMASK | |
| R/W-FF8 0000h | |

| 15 | 4 | 3 | 2 | 0 |
|---|---|---|---|---|
| ADDRMASK | | PREFETCH_EN | Reserved | |
| R/W-FF8 0000h | | R/W-1 | R-0 | |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 34. Base Address 5 Mask Register (PCIBAR5MSK)**

| 31 | 16 |
|---|---|
| ADDRMASK | |
| R/W-FF8 0000h | |

| 15 | 4 | 3 | 2 | 0 |
|---|---|---|---|---|
| ADDRMASK | | PREFETCH_EN | Reserved | |
| R/W-FF8 0000h | | R/W-1 | R-0 | |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 31. Base Address *n* Mask Register (PCIBAR*n*MSK) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-4 | ADDRMASK | 0-FFF FFFFh | Address mask bits. These bits control the writeability of the corresponding bits in the corresponding base address *n* mirror register (PCIBAR*n*MIR). |
| 3 | PREFETCH_EN | 0-1 | Prefetchable bit enable. Specifies whether or not the memory space controlled by the corresponding base address *n* mirror register (PCIBAR*n*MIR) is prefetchable. This bit is reflected in the PREFETCH bit in the corresponding PCIBAR*n*MIR. |
| 2-0 | Reserved | 0 | Reserved |

### 3.2.12 Subsystem Vendor ID/Subsystem ID Mirror Register (PCISUBIDMIR)

The subsystem vendor ID/subsystem ID mirror register (PCISUBIDMIR) is shown in Figure 35 and described in Table 32.

**Figure 35. Subsystem Vendor ID/Subsystem ID Mirror Register (PCISUBIDMIR)**

| 31 | 16 |
|---|---|
| SUBSYS_ID | |
| R/W-0 | |

| 15 | 0 |
|---|---|
| SUBSYS_VEN_ID | |
| R/W-0 | |

LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 32. Subsystem Vendor ID/Subsystem ID Mirror Register (PCISUBIDMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-16 | SUBSYS_ID | 0-FFFFh | Subsystem ID bits. Identifies the board level device. |
| 15-0 | SUBSYS_VEN_ID | 0-FFFFh | Subsystem Vendor ID bits. Identifies the board level manufacturer. |

### 3.2.13 Capabilities Pointer Mirror Register (PCICPBPTRMIR)

The capabilities pointer mirror register (PCICPBPTRMIR) is shown in Figure 36 and described in Table 33.

**Figure 36. Capabilities Pointer Mirror Register (PCICPBPTRMIR)**

| 31 | 16 |
|---|---|
| Reserved | |
| R-0 | |

| 15 | 8 | 7 | 0 |
|---|---|---|---|
| Reserved | | CAP | |
| R-0 | | R-40h | |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 33. Capabilities Pointer Mirror Register (PCICPBPTRMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-8 | Reserved | 0 | Reserved |
| 7-0 | CAP | 0-FFh | Capabilities pointer bits. Specifies the address in configuration space where the first entry in the capabilities list is located. |

### 3.2.14 Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register (PCILGINTMIR)

The maximum latency/minimum grant/interrupt pin/interrupt line mirror register (PCILGINTMIR) is shown in Figure 37 and described in Table 34.

**Figure 37. Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register (PCILGINTMIR)**

| 31 24 | 23 16 |
|---|---|
| MAX_LAT | MIN_GRNT |
| R/W-0 | R/W-0 |

| 15 8 | 7 0 |
|---|---|
| INT_PIN | INT_LINE |
| R-1 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 34. Maximum Latency/Minimum Grant/Interrupt Pin/Interrupt Line Mirror Register (PCILGINTMIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-24 | MAX_LAT | 0-FFh | Maximum latency bits. Specifies how often the device needs to gain access to the PCI bus in 0.25 μsec units. |
| 23-16 | MIN_GRNT | 0-FFh | Minimum grant bits. Specifies the length of the burst period for the device needs in 0.25 μsec units. |
| 15-8 | INT_PIN | 1 | Interrupt pin bits. Specifies the interrupt pin the device uses. This bit is hardwired to 1h in the PCI to indicate that interrupt A will be used. |
| 7-0 | INT_LINE | 0-FFh | Interrupt line bits. This value is written by the host and indicates to which input of the system interrupt controller the PCI interrupt pin is connected. |

### 3.2.15 Slave Control Register (PCISLVCNTL)

The slave control register (PCISLVCNTL) is shown in Figure 38 and described in Table 35.

**Figure 38. Slave Control Register (PCISLVCNTL)**

| 31–24 |
|---|
| Reserved |
| R-0 |

| 23–22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|
| Reserved | BASE5_EN | BASE4_EN | BASE3_EN | BASE2_EN | BASE1_EN | BASE0_EN |
| R-0 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |

| 15–8 |
|---|
| Reserved |
| R-0 |

| 7–5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Reserved | FORCE_DEL_READ_MUL | FORCE_DEL_READ_LN | FORCE_DEL_READ | DIS_SLV_TOUT | CFG_DONE |
| R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 35. Slave Control Register (PCISLVCNTL) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-22 | Reserved | 0 | Reserved |
| 21-16 | BASE*n*_EN | | Base address enable bits. This bit enables/disables the corresponding base address *n* register. When BASE*n*_EN is cleared to 0 (disabled) any host accesses targeted at the slave memory window covered by the base address register is ignored. Host writes to the base address registers are not affected by BASE*n*_EN. |
| | | 0 | Disable base address *n* register. |
| | | 1 | Enable base address *n* register. |
| 15-5 | Reserved | 0 | Reserved |
| 4 | FORCE_DEL_READ_MUL | | Force Delayed Read Multiple bit. |
| | | 0 | Slave should respond with normal 16 clock cycle timeout and retry mechanism for Memory Read Multiple transactions |
| | | 1 | Slave should immediately respond with a retry whenever a Memory Read Multiple transaction is decoded for this slave. This bit overrides the DIS_SLV_TOUT bit for Memory Read Multiple transactions. |
| 3 | FORCE_DEL_READ_LN | | Force Delayed Read Line bit. |
| | | 0 | Slave should respond with normal 16 clock cycle timeout and retry mechanism for Memory Read Line transactions |
| | | 1 | Slave should immediately respond with a retry whenever a Memory Read Line transaction is decoded for this slave. This bit overrides the DIS_SLV_TOUT bit for Memory Read Line transactions. |
| 2 | FORCE_DEL_READ | | Force Delayed Read bit. |
| | | 0 | Slave should respond with normal 16 clock cycle timeout and retry mechanism for Memory Read transactions |
| | | 1 | Slave should immediately respond with a retry whenever a Memory Read transaction is decoded for this slave. This bit overrides DIS_SLV_TOUT for Memory Read transactions. |
| 1 | DIS_SLV_TOUT | | Disable Slave timeout bit. |
| | | 0 | Slave responds with normal 16 clock cycle timeout mechanism |
| | | 1 | Slave will insert wait states on the PCI bus indefinitely until the access is ready to complete |
| 0 | CFG_DONE | | Configuration done bit. Indicates if the configuration registers have been loaded with their proper reset values. |
| | | 0 | Configuration registers are being loaded. No access allowed into PCI interface. |
| | | 1 | Configuration registers loading is complete. PCI interface will accept accesses. |

## 3.2.16 Slave Base Address Translation Registers (PCIBAR0TRL-PCIBAR5TRL)

The slave base address *n* translation register (PCIBAR*n*TRL) controls the translation of transaction addresses as they flow from the PCI bus to the back-end interface. The translation registers are programmed with a value that replaces the most significant portion of the PCI address as it is converted to a DM646x DMSoC address. The slave base address translation registers are shown in Figure 39 through Figure 44 and described in Table 36.

**Figure 39. Slave Base Address 0 Translation Register (PCIBAR0TRL)**

| 31–16 |
|---|
| TRANS_ADDR |
| R/W-100 1000h |

| 15–4 | 3–0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-100 1000h | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 40. Slave Base Address 1 Translation Register (PCIBAR1TRL)**

| 31–16 |
|---|
| TRANS_ADDR |
| R/W-200 0000h |

| 15–4 | 3–0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-200 0000h | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 41. Slave Base Address 2 Translation Register (PCIBAR2TRL)**

| 31–16 |
|---|
| TRANS_ADDR |
| R/W-W-01C 0000h |

| 15–4 | 3–0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-W-01C 0000h | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 42. Slave Base Address 3 Translation Register (PCIBAR3TRL)**

| 31–16 |
|---|
| TRANS_ADDR |
| R/W-118 1800h(1) |

| 15–4 | 3–0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-118 1800h(1) | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

(1) This is not a supported memory address range in DM646x devices and needs to be reprogrammed before being used.

**Figure 43. Slave Base Address 4 Translation Register (PCIBAR4TRL)**

| 31 … 16 |
|---|
| TRANS_ADDR |
| R/W-800 0000h |

| 15 … 4 | 3 … 0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-800 0000h | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Figure 44. Slave Base Address 5 Translation Register (PCIBAR5TRL)**

| 31 … 16 |
|---|
| TRANS_ADDR |
| R/W-808 0000h |

| 15 … 4 | 3 … 0 |
|---|---|
| TRANS_ADDR | Reserved |
| R/W-808 0000h | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 36. Slave Base Address *n* Translation Register (PCIBAR*n*TRL) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-4 | TRANS_ADDR | 0-FFF FFFFh | Translation address bits. These address bits replace the address bits of a PCI slave transaction. The ADDRMASK bits of the base address *n* mask registers (PCIBAR*n*MSK) specify which bits are replaced in the original PCI address. |
| 3-0 | Reserved | 0 | Reserved |

### 3.2.17 Base Address Mirror Registers (PCIBAR0MIR-PCIBAR5MIR)

The base address *n* mirror register (PCIBAR*n*MIR) is shown in Figure 45 and described in Table 37.

**Figure 45. Base Address *n* Mirror Register (PCIBAR*n*MIR)**

| 31–16 |
|---|
| ADDR[1] |
| R/W-0 |

| 15–4 | 3 | 2–1 | 0 |
|---|---|---|---|
| ADDR[1] | PREFETCH | TYPE | IOMEM_SP_IND |
| R/W-0 | R-0 or 1[2] | R-0 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

(1) The access type of this field depends on the address mask bits (ADDRMASK) in the base address *n* mask register (PCIBAR*n*MSK).
(2) Reflects the value that is input from the PREFETCH_EN bit in the base address *n* mask register (PCIBAR*n*MSK).

**Table 37. Base Address *n* Mirror Register (PCIBAR*n*MIR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-4 | ADDR | 0-FFF FFFFh | Address bits. These bits can be written by the host to allow initialization of the base address at startup. The writeability of individual bits is determined by the corresponding bit in the base address *n* mask register (PCIBAR*n*MSK). |
| 3 | PREFETCH | 0-1 | Prefetchable bit. Specifies whether or not the memory space controlled by this base address register is prefetchable. This bit reflects the value that is input from the PREFETCH_EN bit in the base address *n* mask register (PCIBAR*n*MSK). |
| 2-1 | TYPE | 0 | Type bits. Indicates the size of the base address register/decoder. This version of the PCI only supports 32-bit addressing, so these bits are hardwired to 0. |
| 0 | IOMEM_SP_IND | 0 | IO/Memory Space Indicator bit. Indicates whether the base address maps into the host's memory or I/O space. This version of the PCI only supports memory-mapped base address registers, so this bit is hardwired to 0. |

### 3.2.18 Master Configuration/IO Transaction Proxy Registers

The master configuration/IO access registers cause the PCI Master to generate configuration or IO transactions. By using an indirect access method (or proxy), the entire configuration and IO space can be addressed, which otherwise would be impossible. For write transactions, the software should write to the data register, then the address register, and then the command register. For read transactions, the software should write to the address register and then the command register. The transaction will start when the command register is written.

#### *3.2.18.1 Master Configuration/IO Access Data Register (PCIMCFGDAT)*

When requesting a configuration or IO access, software either writes the data into the master configuration/IO access data register (PCIMCFGDAT) for a write or reads the data from PCIMCFGDAT for a read. The PCIMCFGDAT is shown in Figure 46 and described in Table 38.

**Figure 46. Master Configuration/IO Access Data Register (PCIMCFGDAT)**

| 31 ... 0 |
|---|
| DATA |
| R/W-0 |

LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 38. Master Configuration/IO Access Data Register (PCIMCFGDAT) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-0 | DATA | 0-FFFF FFFFh | Data bits. Software writes the data into this register for a configuration/IO write or reads the data from this register for a configuration/IO read. |

#### *3.2.18.2 Master Configuration/IO Access Address Register (PCIMCFGADR)*

When requesting a configuration or IO access, software writes the desired address for the transaction into the master configuration/IO access address register (PCIMCFGADR). The PCIMCFGADR is shown in Figure 47 and described in Table 39.

**Figure 47. Master Configuration/IO Access Address Register (PCIMCFGADR)**

| 31 ... 0 |
|---|
| ADDR |
| R/W-0 |

LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 39. Master Configuration/IO Access Address Register (PCIMCFGADR) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-0 | ADDR | 0-FFFF FFFFh | Address bits. The address bits provide the address for configuration/IO transactions. Bits 1-0 are ignored for a configuration transaction. Software must ensure that bits 1-0 of the address correspond to the BYTE_EN bits in the master configuration/IO access command register (PCIMCFGCMD) for IO transactions, thus ensuring that the access is valid on the PCI bus. |

### 3.2.18.3 Master Configuration/IO Access Command Register (PCIMCFGCMD)

The back-end application will program the master configuration/IO access command register (PCIMCFGCMD) to start a configuration read or write. The READY bit should be checked to determine if the previous transaction is complete. The PCIMCFGCMD is shown in Figure 48 and described in Table 40.

**Figure 48. Master Configuration/IO Access Command Register (PCIMCFGCMD)**

| 31 | 30–16 |
|---|---|
| READY | Reserved |
| R-1 | R-0 |

| 15–8 | 7–4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Reserved | BYTE_EN | Rsvd | TYPE | Rsvd | RD_WR |
| R-0 | R/W-0 | R-0 | R/W-0 | R-0 | R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 40. Master Configuration/IO Access Command Register (PCIMCFGCMD) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31 | READY | | Ready bit. The READY bit should be checked to determine if the previous transaction is complete. |
| | | 0 | Register is not ready to accept a new command. |
| | | 1 | Register is ready to accept a new command. |
| 30-8 | Reserved | 0 | Reserved |
| 7-4 | BYTE_EN | 0-Fh | Byte enable bits. Determines which bytes within the addressed DWORD are being accessed. Byte enables indicate the size of the transfer and must be consistent with bits 1-0 of the address that is specified in the master configuration/IO access address register (PCIMCFGADR). |
| 3 | Reserved | 0 | Reserved |
| 2 | TYPE | | Type bit. Sets configuration or IO transaction. |
| | | 0 | Configuration transaction |
| | | 1 | IO transaction |
| 1 | Reserved | 0 | Reserved |
| 0 | RD_WR | | Read/write bit. Set read or write operation. |
| | | 0 | Write |
| | | 1 | Read |

### 3.2.19 Master Configuration Register (PCIMSTCFG)

The master configuration register (PCIMSTCFG) is shown in Figure 49 and described in Table 41.

**Figure 49. Master Configuration Register (PCIMSTCFG)**

| 31–16 |
|---|
| Reserved |
| R-0 |

| 15–11 | 10 | 9 | 8 |
|---|---|---|---|
| Reserved | CFG_FLUSH_IF_NOT_ENABLED | IO_FLUSH_IF_NOT_ENABLED | MEM_FLUSH_IF_NOT_ENABLED |
| R-0 | R/W-0 | R/W-0 | R/W-0 |

| 7–3 | 2 | 1 | 0 |
|---|---|---|---|
| Reserved | SW_MEM_RD_MULT_EN | SW_MEM_RD_LINE_EN | SW_MEM_RD_WRINV_EN |
| R-0 | R/W-1 | R/W-1 | R/W-1 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 41. Master Configuration Register (PCIMSTCFG) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-11 | Reserved | 0 | Reserved |
| 10 | CFG_FLUSH_IF_NOT_ENABLED | | Configuration flush bit. Controls whether or not the Master will flush configuration transactions from the proxy registers, if the Master is not enabled (BUS_MS = 0 in command/status register). |
| | | 0 | Master does not flush configuration transactions from the proxy registers. |
| | | 1 | Master flushes configuration transactions from the proxy registers. |
| 9 | IO_FLUSH_IF_NOT_ENABLED | | IO flush bit. Controls whether or not the Master will flush I/O transactions from the proxy registers, if the Master is not enabled (BUS_MS = 0 in command/status register). |
| | | 0 | Master does not flush I/O transactions from the proxy registers. |
| | | 1 | Master flushes I/O transactions from the proxy registers. |
| 8 | MEM_FLUSH_IF_NOT_ENABLED | | Memory flush bit. Controls whether or not the Master will flush transactions on the PCIM interface, if the Master is not enabled (BUS_MS = 0 in command/status register). |
| | | 0 | Master does not flush transactions on the PCIM interface. |
| | | 1 | Master flushes transactions on the PCIM interface. |
| 7-3 | Reserved | 0 | Reserved |
| 2 | SW_MEM_RD_MULT_EN | | Memory read multiple enable bit. Controls whether or not the Master command generation logic is permitted to use the Memory Read Multiple command. |
| | | 0 | Master will not generate Memory Read Multiple transactions. |
| | | 1 | Master is enabled to generate Memory Read Multiple transactions for appropriate length bursts. |
| 1 | SW_MEM_RD_LINE_EN | | Memory read line enable bit. Controls whether or not the Master command generation logic is permitted to use the Memory Read Line command. |
| | | 0 | Master will not generate Memory Read Line transactions. |
| | | 1 | Master is enabled to generate Memory Read Line transactions for appropriate length bursts. |
| 0 | SW_MEM_WRINV_EN | | Memory write invalid enable bit. Controls whether or not the Master command generation logic is permitted to use the Memory Write and Invalidate command. |
| | | 0 | Master will not generate Memory Write and Invalidate transactions. |
| | | 1 | Master is enabled to generate Memory Write and Invalidate transactions for appropriate length bursts. |

## 3.3 DM646x DMSoC-To-PCI Address Translation Registers

The DM646x DMSoC-to-PCI address translation registers (Table 42) are used for address translation from the DM646x DMSoC to PCI domain. Using these 32 registers, the master windows can be configured. Each of these 32 registers correspond to 1/32 of the 256 MB addressable PCI space.

**Table 42. DM646x DMSoC-to-PCI Address Translation Registers**

| Offset | Acronym | Register Description | Section |
|---|---|---|---|
| 314h-390h | PCIADDSUB[0-31] | Address Substitution [0-31] Registers | Section 3.3.1 |

### 3.3.1 PCI Address Substitution Registers (PCIADDSUB0-PCIADDSUB31)

The PCI address substitution register (PCIADDSUB*n*) is shown in Figure 50 and described in Table 43.

**Figure 50. PCI Address Substitution *n* Registers (PCIADDSUB*n*)**

| 31 - 23 | 22 - 16 |
|---|---|
| ADD_SUBS | Reserved |
| R/W-0 | R-0 |

| 15 - 0 |
|---|
| Reserved |
| R-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 43. PCI Address Substitution *n* Registers (PCIADDSUB*n*) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-23 | ADD_SUBS | 0-1FFh | Address substitution bits. Substitutes the 9 MSBs of the DM646x DMSoC address during DM646x DMSoC-to-PCI transactions. |
| 22-0 | Reserved | 0 | Reserved |

## 3.4 PCI Configuration Hook Registers

All the PCI back-end configuration registers (Section 3.2) are hooked up to a set of memory-mapped registers called the configuration hook registers, listed in Table 44. The values in these hook registers are latched to the actual PCI registers on PCI reset. The default values in these hook registers can be overwritten by software. These registers are implemented mainly to support PCI I2C EEPROM autoinitialization, as discussed in Section 2.9.4.

**Table 44. PCI Configuration Hook Registers**

| Offset | Acronym | Register Description | Section |
|---|---|---|---|
| 394h | PCIVENDEVPRG | Vendor ID/Device ID Program Register | Section 3.4.1 |
| 39Ch | PCICLREVPRG | Class Code/Revision ID Program Register | Section 3.4.2 |
| 3A0h | PCISUBIDPRG | Subsystem Vendor ID/Subsystem ID Program Register | Section 3.4.3 |
| 3A4h | PCIMAXLGPRG | Maximum Latency/Minimum Grant Program Register | Section 3.4.4 |
| 3ACh | PCICFGDONE | Configuration Done Register | Section 3.4.5 |

### 3.4.1 Vendor ID/Device ID Program Register (PCIVENDEVPRG)

The vendor ID/device ID program register (PCIVENDEVPRG) provides the default values for the vendor ID/device ID mirror register (PCIVENDEVMIR). PCIVENDEVPRG is shown in Figure 51 and described in Table 45.

**Figure 51. Vendor ID/Device ID Program Register (PCIVENDEVPRG)**

| 31 … 0 |
|---|
| VENDOR_DEVICE_ID_PROG |
| R/W-B002 104Ch |

LEGEND: R/W = Read/Write; *-n* = value after reset

**Table 45. Vendor ID/Device ID Program Register (PCIVENDEVPRG) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-0 | VENDOR_DEVICE_ID_PROG | 0-FFFF FFFFh | Vendor device ID program bits. Default values for the VEN_ID and DEV_ID bits in the vendor ID/device ID mirror register (PCIVENDEVMIR). |

### 3.4.2 Class Code/Revision ID Program Register (PCICLREVPRG)

The class code/revision ID program register (PCICLREVPRG) provides default values for the class code/revision ID mirror register (PCICLREVMIR). The PCICLREVPRG is shown in Figure 52 and described in Table 46.

**Figure 52. Class Code/Revision ID Program Register (PCICLREVPRG)**

| 31 … 0 |
|---|
| CLASS_CODE_REV_ID_PROG |
| R/W-1180 0001h |

LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 46. Class Code/Revision ID Program Register (PCICLREVPRG) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-0 | CLASS_CODE_REV_ID_PROG | 0-FFFF FFFFh | Class code revision ID program bits. Default values for the CL_CODE and REV_ID bits in the class code/revision ID mirror register (PCICLREVMIR). |

### 3.4.3 Subsystem Vendor ID/Subsystem ID Program Register (PCISUBIDPRG)

The subsystem vendor ID/subsystem ID program register (PCISUBIDPRG) provides default values for the subsystem vendor ID/subsystem ID mirror register (PCISUBIDMIR). The PCISUBIDPRG is shown in Figure 53 and described in Table 47.

**Figure 53. Subsystem Vendor ID/Subsystem ID Program Register (PCISUBIDPRG)**

| 31 … 0 |
|---|
| SUBSYS_VENDOR_ID_SUBSYS_ID_PROG |
| R/W-0 |

LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 47. Subsystem Vendor ID/Subsystem ID Program Register (PCISUBIDPRG) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-0 | SUBSYS_VENDOR_ID_SUBSYS_ID_PROG | 0-FFFF FFFFh | Subsystem vendor ID and subsystem ID program bits. Default values for the SUBSYS_VEN_ID and SUBSYS_ID bits in the subsystem vendor ID/subsystem ID mirror register (PCISUBIDMIR). |

### 3.4.4 Maximum Latency/Minimum Grant Program Register (PCIMAXLGPRG)

The maximum latency/minimum grant program register (PCIMAXLGPRG) provides the default values for the maximum latency/minimum grant/interrupt pin/interrupt line mirror register (PCILGINTMIR). The PCIMAXLGPRG is shown in Figure 54 and described in Table 48.

**Figure 54. Maximum Latency/Minimum Grant Program Register (PCIMAXLGPRG)**

| 31–16 |
|---|
| Reserved |
| R-0 |

| 15–0 |
|---|
| MAX_LAT_MIN_GRANT_PROG |
| R/W-0 |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

**Table 48. Maximum Latency/Minimum Grant Program Register (PCIMAXLGPRG) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-16 | Reserved | 0 | Reserved |
| 15-0 | MAX_LAT_MIN_GRANT_PROG | 0-FFFFh | Maximum latency and minimum grant program bits. Default values for the MAX_LAT and MIN_GRNT bits in the maximum latency/minimum grant/interrupt pin/interrupt line mirror register (PCILGINTMIR). |

### 3.4.5 Configuration Done Register (PCICFGDONE)

Some of the PCI registers can be autoinitialized by the DM646x DMSoC ROM code after reset. When autoinitialization is selected, accesses to the PCI are held off until the CONFIG_DONE bit in the PCI configuration done register (PCICFGDONE) is 1. After initialization is done, the software will set this bit to 1. The PCICFGDONE is shown in Figure 55 and described in Table 49.

**Figure 55. Configuration Done Register (PCICFGDONE)**

| 31–8 |
|---|
| Reserved |
| R-0 |

| 7–1 | 0 |
|---|---|
| Reserved | CONFIG_DONE |
| R-0 | R/W-0[1] |

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

(1) This bit defaults to 0 and the boot code needs to set this bit to 1.

**Table 49. Configuration Done Register (PCICFGDONE) Field Descriptions**

| Bit | Field | Value | Description |
|---|---|---|---|
| 31-1 | Reserved | 0 | Reserved |
| 0 | CONFIG_DONE | | Configuration done bit. Holds off accesses to the PCI until configuration by the DM646x DMSoC ROM code is complete. |
| | | 0 | Configuration in progress, accesses to the PCI are not permitted. |
| | | 1 | Configuration complete, accesses to the PCI are allowed. |

# Appendix A  Revision History

Table 50 lists the changes made since the previous version of this document.

**Table 50. Document Revision History**

| Reference | Additions/Modifications/Deletions |
|---|---|
| Section 1.1 | Changed paragraph. |
| Section 1.2 | Changed fourth bullet in first paragraph. |
| Section 1.3 | Changed fifth bullet. |
| Section 2.1 | Added second subbullet to first bullet. |
| Figure 12 | Added footnote to 66MHZ_CAP bit. |
| | Added footnote. |
| Table 13 | Changed Description of 66MHZ_CAP bit. |
| Figure 26 | Added footnote to 66MHZ_CAP bit. |
| | Added footnote. |
| Table 28 | Changed Description of 66MHZ_CAP bit. |