

## Introduction

The Altera® PCI-to-DDR SDRAM reference design, which you can download to the Stratix PCI development board, provides an interface between the Altera `pci_mt64` MegaCore® function and a 64-bit, 256-MByte DDR SDRAM module. The reference design has the following features:

- Supports 32- and 64-bit PCI master and target transactions
- Includes a DMA engine
- Uses the dual-port FIFO buffer function from the library of parameterized modules (LPM)
- Uses the DDR SDRAM Controller MegaCore function.
- Interfaces PCI to flash memory
- Includes PCI throughput performance measurement logic

The Stratix PCI development board allows designers to evaluate, demonstrate, and develop system-level designs with PCI, PCI-X, DDR SDRAM, HyperTransport, SPI-4 level 2, and RapidIO interfaces. Combined with intellectual property (IP) from Altera and Altera Megafunction Partners Program (AMPP™) partners, users can solve design problems that formerly required custom solutions.



For more information on the Stratix PCI development board, refer to the *Stratix PCI Development Board Data Sheet*.

## General Description

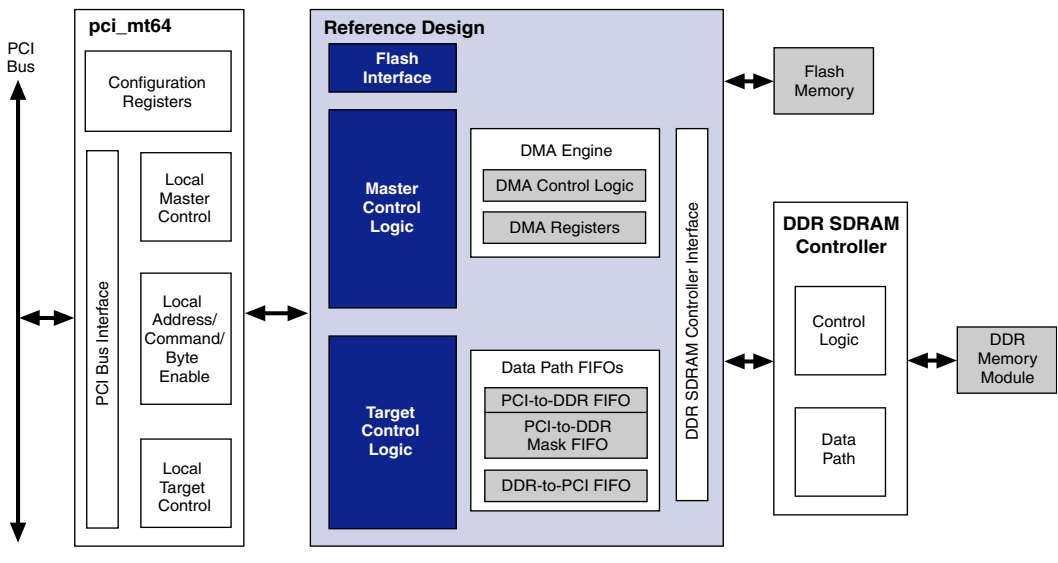
The Altera PCI-to-DDR SDRAM reference design is an example of a typical user application that interfaces to the local side of the Altera `pci_mt64` MegaCore function. This reference design provides interface logic between the `pci_mt64` and DDR SDRAM Controller MegaCore functions, enabling DDR SDRAM accesses via the PCI bus. The PCI core can act as a target or PCI bus master in a DDR SDRAM access. When the PCI core operates as a PCI target or bus master, the DMA engine initiates the transaction, monitors the status, and manages the progress of the data transfers.

The reference design also includes a flash memory interface to allow access to the flash memory device via the PCI bus. Flash memory is an effective solution for storing multiple device configuration files.

Figure 1 shows a high-level block diagram of the reference design. The reference design consists of the following elements:

- Master control logic
- Target control logic
- DMA engine
- Data path FIFO buffers
- DDR SDRAM memory interface
- Flash memory interface
- Miscellaneous registers

Figure 1. Block Diagram



## Master Control Logic

The master control logic controls the operation of the `pci_mt64` function in master mode. It interacts with the DMA engine to control the PCI master transactions. During a PCI master write operation, data flows from the DDR memory to the PCI bus. The master control logic performs the following functions:

- Provides status of the PCI bus to the DMA engine
- Interacts with the `pci_mt64` function to execute PCI master write cycles
- Transfers the data from the DDR-to-PCI FIFO buffer to the `pci_mt64` function

During a PCI master read operation, the data flows from the PCI bus to the DDR memory. The master control logic performs the following functions:

- Provides the status of the PCI bus to the DMA engine
- Interacts with the `pci_mt64` function to execute PCI master read cycles
- Writes the data read by the `pci_mt64` function into the PCI-to-DDR FIFO buffer

## Target Control Logic

The target control logic controls the operation of the `pci_mt64` function in target mode. During a PCI target write, the data flows from the PCI bus to the DDR memory. The target control logic performs the following functions:

- Interacts with the `pci_mt64` function to execute a PCI target write cycle
- Writes the data—written by the host or the master on the PCI bus—into the PCI-to-DDR FIFO buffer
- Interacts with the DDR SDRAM controller interface to read data from the PCI-to-DDR FIFO buffer and write it into the DDR memory

During a PCI target read, the data flows from the DDR memory to the PCI bus. The target control logic performs the following functions:

- Interacts with the `pci_mt64` function to execute a PCI target read cycle
- Interacts with the DDR SDRAM controller interface to fetch the data from the DDR memory and writes it to the DDR-to-PCI FIFO buffer
- Transfers the data from the DDR-to-PCI FIFO buffer to the `pci_mt64` function

## DMA Engine

The DMA engine interfaces with the master control logic, the data path FIFO buffers, and the DDR SDRAM controller interface to coordinate DMA transfers to and from the DDR memory. The DMA engine consists of:

- DMA control logic
- DMA registers

### *DMA Control Logic*

The DMA control logic:

- Provides control signals to the master control logic to prompt it to request the PCI bus when needed
- Initiates a new access to the DDR memory
- Monitors the data path FIFO buffers and the current DDR memory access
- Monitors the DMA registers in order to initiate a new transaction.
- Updates the interrupt status register (ISR) and control and status register (CSR) in the DMA registers

### *DMA Registers*

Setting up the DMA registers in the DMA engine initiates DMA transactions. These registers are memory-mapped to `BAR0` of the `pci_mt64` function; they can be accessed with a target transaction to their memory-mapped addresses. The registers must be written by another master on the PCI bus. The DMA registers consist of:

- Control and status register (CSR)
- PCI address register (PAR)
- Byte counter register (BCR)
- Interrupt status register (ISR)
- Local address register (LAR)

“DMA Engine” on page 17 provides a detailed description of these registers.

### **Data Path FIFO Buffers**

The data path FIFO buffers serve as the buffer space for the data transferred between the DDR memory and PCI bus. The FIFO buffers synchronize the data crossing PCI and DDR clock domains. They are also used to align the data path from the PCI bus, which can be 32- or 64-bits, and the DDR memory, which is always 64 bits. The reference design implements the following FIFO buffers:

- PCI-to-DDR FIFO buffer (128 x 64)
- PCI-to-DDR mask FIFO buffer (128 x 8)
- DDR-to-PCI FIFO buffer (128 x 64)

### PCI-to-DDR FIFO Buffer

The PCI-to-DDR FIFO buffer stores data transferred from the PCI bus to the DDR memory. During target write or master read transactions, data written into the PCI-to-DDR FIFO buffer is controlled by the target and master control logic, respectively. The DDR SDRAM controller interface module reads the data from the PCI-to-DDR FIFO buffer.

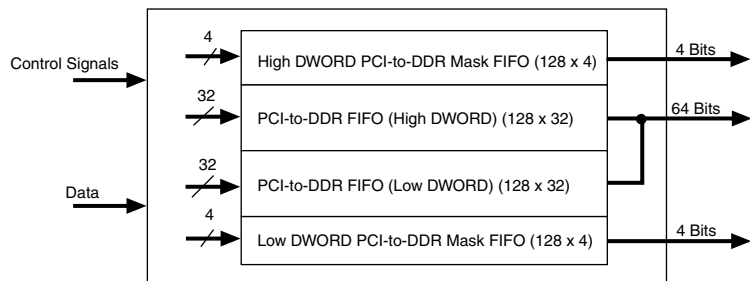
The PCI-to-DDR FIFO buffer consists of two independent  $128 \times 32$  FIFO buffers, which allows a low DWORD to be written independently from the high DWORD when the PCI bus operates in 32-bit mode. The read operation from the PCI-to-DDR FIFO buffer is always 64 bits wide.

### PCI-to-DDR Mask FIFO Buffer

The PCI-to-DDR mask FIFO buffer validates the data stored in the PCI-to-DDR FIFO buffer. Each bit stored in this buffer corresponds to a different byte in the PCI-to-DDR FIFO buffer. This buffer stores the byte enables from the PCI bus during target write transactions and masks the invalid DWORDs when the PCI bus is in 32-bit mode. When the PCI bus performs a 32-bit transaction, the starting or ending DWORD might be invalid due to misalignment. In this case, the FIFO control logic detects these conditions and writes the appropriate mask bit value into the PCI-to-DDR mask FIFO buffer. The DDR memory controller interface module then reads the data along with the mask value and writes only the valid data into the DDR memory.

For example, if there is a target write to the DDR memory starting at high DWORD (F0000004h), the first low DWORD (F0000000h) of the FIFO buffer is invalid and should not be written into the memory. Therefore, the DWORD at address F0000000h is masked by writing all 1s to the first 4 bits of the data mask FIFO buffer. See [Figure 2](#).

**Figure 2. Data Flow in the PCI-to-DDR & PCI-to-DDR Mask FIFO Buffers**



### *DDR-to-PCI FIFO*

The DDR-to-PCI FIFO buffer provides the buffer space for the data transferred from the DDR memory to the PCI bus. During the target read and master write transactions, the DDR memory controller controls data written into this FIFO buffer. The target or master control modules control the read operations from this buffer.

The DDR-to-PCI FIFO buffer is 128 x 64. Unlike the PCI-to-DDR FIFO buffer, this FIFO buffer is always treated as a 64-bit wide buffer, i.e., the read or write from this FIFO buffer is 64 bits at a time. During a 32-bit PCI transaction, data is transferred from the DDR memory to the `pci_mt64` function in 64-bit mode. The function then multiplexes the low and high DWORD to its output registers.

## **DDR SDRAM Controller Interface**

The DDR SDRAM controller interface provides the interface logic to the user side of the DDR SDRAM Controller MegaCore function. It translates the PCI cycles to the DDR access cycles based on the control signals from the master control, target control, and FIFO buffers' status.

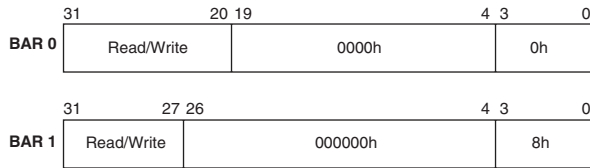
The DDR memory module is 64 bits wide and requires the user interface to the DDR SDRAM controller to be 128 bits wide. However, the reference design interfaces to a 64-bit PCI bus; therefore, only the lower 32-bit data pins of the DDR memory module are used. The upper 32 bits of the DDR memory pins are not connected. Effectively, the reference design only uses 128 MBytes of the 256-MByte DDR memory module.

The following sections provide a detailed description of the operation of the various modules in the PCI-to-DDR memory reference design.

## **Functional Description**

### **Memory Mapping**

The `pci_mt64` function used in this reference design implements two base address registers (BARs): `BAR0` and `BAR1`. `BAR0` reserves 1 MByte of the system address space (see [Figure 3](#)); the 12 most significant bits of the PCI address are decoded by the `pci_mt64` function to claim a target transaction. All of the internal registers and the flash control register are mapped to this address space.

**Figure 3. Base Address Register 0 & 1 Settings**

BAR1 reserves 128 MBytes of the system address space; the 7 most significant bits (MSB) of the PCI address are decoded by the `pci_mt64` function to claim a target transaction. Table 1 describes the `pci_mt64` address mapping.

**Table 1. Memory Address Space**

Memory Region	Block Size	Address Offset	Description
BAR0	1 MByte	00000h-FFFFFFh	Internal reference design configuration registers.
BAR1	128 MBytes	0000000h-7FFFFFFh	256-MByte DDR memory module.

Table 2 describes the internal registers memory-mapped addresses.

**Table 2. Internal Registers Memory Maps**

Range Reserved	Read/Write	Mnemonic	Register Name
00000h-00003h	Read/Write	CSR[8:0]	DMA Control/Status
00004h-00007h	Read/Write	PAR[31:0]	DMA PCI Address Register
00008h-0000Bh	Read/Write	BCR[16:0]	DMA Byte Counter Register
0000Ch-0000Fh	Read	ISR[6:0]	DMA Interrupt/Status Register
00010h-00013h	Write	LAR[25:0]	DMA Local Address Register
00080h-00083h	Read	TARG_PERF_REG	Target Performance Register
00084h-00087h	Read	MSTR_PERF_REG	Master Performance Register
0030h-0033h	Write	FLASH_CMD	Flash Interface Command
0034h-0037h	Write	FLASH_ADR	Flash Interface Address
0038h-003Bh	Write	FLASH_DATA	Flash Write Data
00028h-0002Bh	Read	FLASH_STAT	Flash Interface Status
0002Ch-0002Fh	Read	FLASH_RD_DATA	Flash Read Data

## Target Transactions

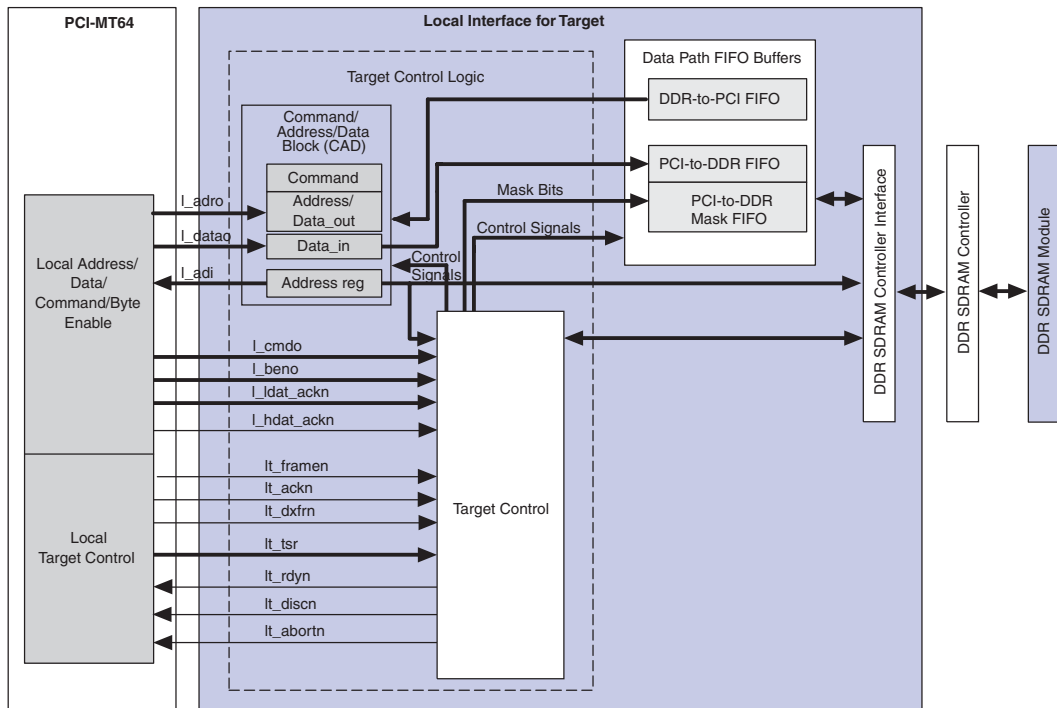
Table 3 describes the local signals of the `pci_mt64` function used in this reference design. For more information on the local signals of the `pci_mt64` function, refer to the *PCI MegaCore Function User Guide*.

<b>Table 3. Local Signals Used in Target Transactions (Part 1 of 2)</b>		
<b>Type</b>	<b>Signal</b>	<b>Description</b>
Data path input	<code>l_adi</code>	Local address/data input. This signal is driven with data by the local side during PCI bus-initiated target read transactions.
Data path output	<code>l_dato</code>	Local data output. The PCI function drives data on this bus during target write transactions.
	<code>l_adro</code>	Local address output. The PCI function drives address on this bus during target read and write transactions.
	<code>l_beno</code>	Local byte-enable output. The PCI function drives a byte enable on this bus during target read and write transactions.
	<code>l_cmndo</code>	Local command output. The PCI function drives commands on this bus during target operation.
64-bit support signals	<code>l_ldat_ackn</code>	Local low data acknowledge. When asserted, this signal indicates that valid data is present on the <code>l_dato[31..0]</code> bus. <code>lt_ackn</code> must be used to qualify valid data.
	<code>l_hdat_ackn</code>	Local high data acknowledge. When asserted, this signal indicates that valid data is present on the <code>l_dato[63..32]</code> bus. <code>lt_ackn</code> must be used to qualify valid data.
Local target control inputs	<code>lt_rdyn</code>	Local target ready. The local side asserts this signal to indicate valid data input during a target read, or to indicate that it is ready to accept data input during a target write.
	<code>lt_discn</code>	Local target disconnect request. This signal requests the PCI function to issue a retry or a disconnect depending on when the signal is asserted during a transaction.
	<code>lt_abortn</code>	Local target abort request. This signal requests the PCI function to issue a target abort to the PCI master.

**Table 3. Local Signals Used in Target Transactions (Part 2 of 2)**

Type	Signal	Description
Local target control outputs	lt_framen	Local target frame request. This signal is asserted while the PCI function is requesting access to the local side. It is asserted one clock cycle before the functions asserts <code>devseln</code> , and it is released after the last data phase of the transaction is transferred to/from the local side.
	lt_ackn	Local target acknowledge. The PCI function asserts this signal to indicate valid data output during a target write, or to indicate that it is ready to accept data during a target read.
	lt_dxfrn	Local target data transfer. The PCI function asserts this signal when a data transfer on the local side is successful during a target transaction.
	lt_tsr	Local target transaction status register. The <code>lt_tsr</code> bus carries several signals, which can be monitored for the transaction status.

Figure 4 provides a block diagram of the target reference design.

**Figure 4. Target Reference Design**

During the address phase of a PCI target transaction, the `pci_mt64` function compares the PCI address with the BAR0 and BAR1 address ranges; if the address decoded matches the address space reserved for BAR0 or BAR1, the function asserts the `devseln` signal on the PCI bus to claim the transaction.

After the target transaction has been claimed by the `pci_mt64`, the function asserts the `lt_framen` signal and the appropriate `lt_tsr[11..0]` signal on the local side to indicate to the target control logic that a target transaction has been requested.

When the user logic detects a target transaction, it decodes the command and the BAR hit information from the `l_cmdo` and `lt_tsr` outputs of the `pci_mt64` core to determine the type and destination of the current target access (DDR memory or internal registers).

The command/address/data (CAD) block in the target control logic uses the data path input from the function to provide relevant data to target control and data path FIFO buffers. The CAD uses data path outputs from the target control and the data path FIFO buffers to provide relevant data to the local side of the `pci_mt64` function. The CAD block is common to both the target control logic and master control logic.

### *PCI Target Memory Read*

When a target memory read is decoded:

- A retry is signaled immediately by asserting the `lt_discn` signal to the `pci_mt64` function, because a read from the DDR memory requires more than 16 clock cycles to provide the first data phase on the PCI side. All memory read cycles are treated as delayed transactions.
- All memory target transactions to BAR1 are retried until the number of words in the DDR-to-PCI FIFO buffer reaches a predetermined threshold. Target transactions to internal registers (BAR0) are accepted with no interruption while the target transaction to BAR1 is retried.
- If the threshold has been reached in the DDR-to-PCI FIFO when the same memory read cycle is retried, the local side completes the read by asserting `lt_rdyn` to transfer the data from the DDR-to-PCI FIFO to the `pci_mt64` function.
- The data from the DDR-to-PCI FIFO buffer is transferred to the PCI bus until the `lt_framen` signal is deasserted by the `pci_mt64` function.
- If the number of data words in the FIFO buffer reaches a lower threshold, `lt_discn` is asserted to end the current target read and

prevent a FIFO buffer overflow. Subsequently, the master has to reread the rest of the data.

- On deassertion of `lt_framen` signal, the remaining data in the DDR-to-PCI FIFO buffer is discarded and the FIFO buffer is flushed.

### *PCI Target Memory Write*

When a target memory write is decoded:

- The local control logic writes the data to the PCI-to-DDR FIFO buffer until `lt_framen` is deasserted by the `pci_mt64` function.
- On the read side of the PCI-to-DDR FIFO buffer, the DDR SDRAM controller interface reads the data from the FIFO buffer and writes it into the DDR memory until the FIFO buffer is empty.
- If the PCI-to-DDR FIFO buffer is almost full, the target control logic requests a disconnect to prevent a FIFO overflow by asserting `lt_discn`. If the PCI master has more data to transfer, it must re-initiate a write cycle to complete writing all of the data from where it left off.
- On termination of a target write transaction, all target memory transactions are retried until the DDR SDRAM controller has completed writing all of the data from the FIFO buffer into the DDR memory.
- If the PCI transaction is in 32-bit mode, the low and high DWORD of the FIFO buffer is written alternately. A write to the low DWORD FIFO buffer occurs when the `lt_dxfrn` and `l_ldat_ackn` signals are asserted by the `pci_mt64` function and a write to the high DWORD FIFO buffer occurs when the `pci_mt64` function asserts the `lt_dxfrn` and `l_hdat_ackn` signals.

Because the reference design's local side is always a 64-bit data path, while the PCI bus is either 32- or 64-bits wide, additional logic performs data alignment when the PCI bus performs a 32-bit transaction. In general, the data alignment logic writes the data from the PCI bus during a target write into the PCI-to-DDR FIFO—starting from the low DWORD followed by the high DWORD—along with the mask bits, except when there is a data misalignment. A data misalignment can occur at the beginning or end of the PCI target write transaction in the case of a burst write.

If the starting address is not QWORD aligned,  $AD[2..0] = 4h$ , the first DWORD transferred on the PCI bus must be written to the high DWORD and the low DWORD must be masked. The data alignment logic writes the low DWORD of the PCI-to-DDR FIFO and `1s (Fh)` into the 4 least significant bits of the PCI-to-DDR mask FIFO to indicate that the low DWORD should not be written to the DDR memory. The data written into the low DWORD of the PCI-to-DDR FIFO is don't care.

Data misalignment occurs during a target burst write when the last DWORD received from the PCI bus is intended for the low DWORD in the PCI-to-DDR FIFO. This situation leaves the last high DWORD empty when writing to the DDR memory. In this case, the alignment logic detects the condition and writes in the high DWORD PCI-to-DDR FIFO and 1s (Fh) in the 4 most significant bits of the PCI-to-DDR mask FIFO to indicate that the high DWORD should not be written to the DDR memory. The data written into the high DWORD is don't care.

## DMA Register Access

The DMA register can be accessed via target transactions with the appropriate memory-mapped address. Refer to [“Memory Mapping” on page 6](#) for more information. All PCI target transactions to the DMA registers are single-cycle transactions.

## Master Transactions

[Table 4](#) describes the local signals of the `pci_mt64` function used in this reference design. For more information on the local signals of the `pci_mt64` function, refer to the *PCI MegaCore Function User Guide*.

Type	Signal	Description
Data path input	<code>l_adi</code>	Local address/data input. This signal is a local-side time multiplexed address/data bus. During master transaction the local side must provide address on this bus when <code>lm_adr_ackn</code> is asserted. Data is driven on this bus from the local side during master write transaction.
	<code>l_cbeni</code>	Local command/byte enable input. This bus is a local-side time multiplexed command/byte enable bus. During the master transactions, the local side must provide the command on <code>l_cben</code> when <code>lm_adr_ackn</code> is asserted. The local master device must provide the byte-enable value on <code>l_cbeni</code> during the next clock after <code>lm_adr_ackn</code> is asserted.
Data path output	<code>l_dato</code>	Local data output. The PCI function drives data on this bus during local-side initiated master read transaction.
64-bit support signals	<code>l_ldat_ackn</code>	Local low data acknowledge. When asserted, this signal indicates that valid data is present on the <code>l_dato[31..0]</code> bus. <code>lm_ackn</code> must be used to qualify valid data.
	<code>l_hdat_ackn</code>	Local high data acknowledge. When asserted, this signal indicates that valid data is present on the <code>l_dato[63..32]</code> bus. <code>lm_ackn</code> must be used to qualify valid data.

Type	Signal	Description
Local master control inputs	lm_req32n	Local master request 32-bit data transaction. The local side asserts this signal to request ownership of the PCI bus for a 32-bit master transaction.
	lm_req64n	Local master request 64-bit data transaction. The local side asserts this signal to request ownership of the PCI bus for a 64-bit master transaction.
	lm_rdyn	Local master ready. The local side asserts the lm_rdyn signal to indicate a valid data input during a master write, or ready to accept data during a master read.
	lm_lastn	Local master last. This signal is driven by the local side to request the function master interface to end the current transaction. When the local side asserts this signal, the megafunction master interface deasserts <i>framen</i> as soon as possible and asserts <i>lt_rdyn</i> to indicate that the last data phase has begun.
Local master control outputs	lm_adr_ackn	Local master address acknowledge. The PCI function asserts lm_adr_ackn to the local side to acknowledge the requested master transaction. During the same clock cycle lm_adr_ackn is asserted low, the local side must provide the transaction address on the l_adi bus and the transaction command on the l_cbeni.
	lm_ackn	Local master acknowledge. The PCI function asserts this signal to indicate valid data output during a master read, or ready to accept data during a master write.
	lm_dxfrn	Local master data transfer. The PCI function asserts this signal when a data transfer on the local side is successful during a master transaction.
	lm_tsr	Local master transaction status register. These signals inform the local interface the progress of the transaction

Figure 5 shows a block diagram of the master portion of the reference design.



*PCI Master Read*

To initiate a master read transaction:

- The master control logic requests the bus by asserting the `lm_req64n` or `lm_req32n` signal to the local side of the `pci_mt64` function for a 64- or 32-bit transaction, respectively.
- At the same time, the DMA engine signals the DDR memory interface to request a DDR memory write access.
- The `pci_mt64` function outputs `reqn` to the PCI bus arbiter to request the PCI bus and simultaneously asserts `lm_tsr[0]` from the local side to indicate that the master is requesting the PCI bus.
- Upon receiving `gntn` from the PCI bus arbiter the `pci_mt64` function asserts the `lm_adr_ackn` and `lm_tsr[1]` local side signals, indicating the bus has been granted. The master control provides the PCI address on `l_adi` and the PCI command on `l_cbeni` during the same clock cycle that the `lm_adr_ackn` and `lm_tsr[1]` signals are asserted.
- The master control asserts the `lm_rdyn` input to the `pci_mt64` function to indicate that it is ready to accept data.
- The `pci_mt64` function asserts `lm_ackn`, indicating to the master control that it has registered data from the PCI side on the previous clock cycle and is ready to send data to the local side master interface. Because `lm_rdyn` was asserted in the previous clock cycle and `lm_ackn` is asserted in the current cycle, the function asserts `lm_dxfrn` to indicate a valid data transfer on the local side.
- The `pci_mt64` function also asserts `lm_tsr[8]`, indicating to the master control that a data phase was completed successfully on the PCI bus during the previous clock.
- If the master transaction is in 32-bit mode, the low and high DWORD of the PCI-to-DDR FIFO buffer are written alternately. A write to the low DWORD of the buffer occurs when the `lm_dxfrn` and `l_ldat_ackn` signals are asserted and the write to the high DWORD of the buffer occurs when the `lm_dxfrn` and `l_hdat_ackn` signals are asserted.
- Because the DDR control logic clears the PCI-to-DDR FIFO buffer faster than the PCI bus fills it, the PCI-to-DDR FIFO can have an underflow very quickly. Before the PCI-to-DDR FIFO buffer reaches the empty state, the master control logic ends the current master transaction to prevent reading invalid data from the PCI-to-DDR FIFO. It also signals the DDR interface logic to stop the current read cycle after emptying the PCI-to-DDR FIFO. After the FIFO buffer is empty, the DMA engine re-initiates a master transaction where it left off.
- If the latency timer expires, the `pci_mt64` function stops the current master transaction and asserts `lm_tsr[4]` to indicate a premature termination of the current transfer. The master control logic and the

DMA engine monitor this signal and initiate a new master read transaction to read the rest of the data where it left off.

The DDR memory interface logic reads one QWORD (64 bits) of data at a time from the PCI-to-DDR FIFO buffer and writes it into the DDR memory. During a master read transaction, if the `pci_mt64` function must release the bus prematurely, the master control logic asserts the `stop` signal to the DMA and the DDR interface as well as `lm_lastn` to the local side of the `pci_mt64` function. When the `lm_lastn` signal is asserted to the local side of the `pci_mt64` function, the function deasserts `framen` as soon as possible and asserts `irdyn` to indicate the last data phase has begun. Additionally, the DDR interface transfers all valid data from PCI-to-DDR FIFO buffer and stops the operation when it completes. The master control logic and the DMA engine must restart a new transaction and read the rest of the data.

PCI master transactions always start at a QWORD-aligned address; However, the intended target may be a 32-bit target. In this case, a 32-bit PCI transaction will take place. If an odd number of DWORDs are written into the PCI-to-DDR FIFO buffer during a 32-bit master read transaction, the master control logic marks the last high DWORD invalid with the corresponding masking bits in the PCI-to-DDR mask FIFO buffer. Additionally, if a 32-bit master transaction is interrupted at the high DWORD, the transaction restarts at the previous low DWORD address.

### *PCI Master Write*

To initiate a master write transaction:

- The DMA sends a signal to the DDR interface to request a DDR memory access.
- The master control logic waits for the DDR-to-PCI FIFO buffer to be filled with a predetermined number of QWORDS (threshold) before requesting the PCI bus. This action ensures that the master does not violate PCI latency rules due to DDR memory read latency.
- On receiving `gntn`, the `pci_mt64` function asserts `lm_tsr[1]` and `lm_adr_ackn` to indicate to the local side that the bus has been granted. The master control must provide a valid address and command on the same clock cycle.
- During the address phase on the PCI bus, the `pci_mt64` function asserts `lm_tsr[2]`. The master control logic must provide the byte enables for the transaction on `l_cbeni`.
- The master control logic asserts `lm_rdyn` to the `pci_mt64` function to indicate that it is ready to transfer the data from DDR-to-PCI FIFO buffer.

- One QWORD is read at a time from the DDR-to-PCI FIFO buffer to the `pci_mt64` function even if the current PCI transaction is in 32-bit mode.
- The `pci_mt64` function asserts `lm_ackn` during the first data phase on the local side, even if the PCI side is not ready to transfer data. During subsequent data phases, the `pci_mt64` function does not assert `lm_ackn` unless the PCI side is ready to accept data. Because the master control asserted `lm_rdyn` during the previous clock cycle and `lm_ackn` is asserted in the current cycle, the PCI function asserts `lm_dxfrn` to indicate a valid data transfer on the local side.
- Premature termination may occur if the latency timer is expired, the target disconnects, or the target retries.
- During a master write transaction, the DDR-to-PCI FIFO buffer can overflow very quickly because the DDR memory can fill up the FIFO buffer faster than the PCI bus can empty it. Before the DDR-to-PCI FIFO can be overflowed, the master control logic ends the current transaction and flushes the DDR-to-PCI FIFO buffer.

If the PCI transaction is in 32-bit mode, the appropriate high/low DWORD is multiplexed to the output registers of the `pci_mt64` function.

If a master write transaction is terminated prematurely, the master control logic aborts the current DDR memory read access, stops the DMA, and flushes the FIFO. The DMA engine restarts a new master read transaction to transfer the rest of the data.

PCI master transactions always start at a QWORD aligned address. If a 32-bit master transaction is interrupted at the high DWORD, the transaction will restart at the previous low DWORD address. For example, the master plans to write to system address `F000_0000h` through `F000_0080h`, but the transaction is terminated early after transferring data at address `F000_007Ch` (interrupted at high DWORD). The master restarts the process to request a PCI write transaction and begins transferring data from address `F000_0078h` (low DWORD).

## DMA Engine

This section describes the functionality of the DMA engine.

### DMA Registers

The DMA reference design implements the following registers

- Control and status register (CSR)
- PCI address register (PAR)
- Byte counter register (BCR)
- Interrupt status register (ISR)
- Local address register (LAR)

These registers are memory-mapped to BAR0 of the `pci_mt64` function. Another PCI master/host on the PCI bus must configure the DMA before initiating a transfer.

## Control Status Register (CSR)

The control status register is a 9-bit register and is used to configure the DMA engine. The CSR directs the DMA operation and provides the status of the current memory transfer. The CSR can be written/read by another master on the PCI bus. Table 5 describes the format of the CSR.

Data Bit	Mnemonic	Read/Write	Definition
0	<code>int_ena</code>	Read/Write	PCI interrupt enable.
1	<code>flush</code>	Write	Flush the buffer. When high, it resets <code>dma_tc</code> and <code>ad_loaded</code> (bits 3 and 4 of the ISR). The <code>flush</code> bit resets itself; therefore, it is always zero when read. The <code>flush</code> bit should never be set when the <code>dma_on</code> bit is set because a DMA transfer is in progress.
2	-	-	Reserved.
3	<code>write</code>	Read/Write	Memory read/write. The <code>write</code> bit determines the direction of the DMA transfer. When <code>write</code> is high, the data flows from the SDRAM to PCI bus (master write); when low, data flows from the PCI bus to the SDRAM (master read).
4	<code>dma_ena</code>	Read/Write	DMA enable. When high, <code>dma_ena</code> allows the DMA engine to respond to DMA requests as long as the PCI bus activity is not stopped due to a pending interrupt.
5	<code>tci_dis</code>	Read/Write	Transfer complete interrupt disable. When high, <code>tci_int</code> prevents <code>dma_tc</code> (bit 3 of the ISR) from generating PCI bus interrupts.
6	<code>dma_on</code>	Read/Write	DMA on. When high, <code>dma_on</code> indicates that the DMA engine can request ownership of the PCI bus. The <code>dma_on</code> bit is high when: <ul style="list-style-type: none"> <li>■ The PAR is loaded (bit 4 of ISR).</li> <li>■ The DMA is enabled.</li> <li>■ There is no error pending.</li> <li>■ The DMA transfer sequence begins when <code>dma_on</code> is set. Under normal conditions (i.e., DMA is enabled and no error is pending), <code>dma_on</code> is set when a write transaction to the PAR occurs via a target write.</li> </ul>
7	-	-	Reserved.
8	-	-	Reserved.

## PCI Address Register (PAR)

The PCI address register is a 32-bit register. The PAR is implemented with a 29-bit counter and the 3 least significant bits are tied to ground. The PAR contains the PCI bus address for the current memory transfer and is incremented after every data phase on the PCI bus. The PCI bus memory transfer initiated by the DMA engine must begin at the QWORD boundary. The PAR can be written/read by the PCI bus master. [Table 6](#) shows the format of the PAR. The `ad_loaded` bit triggers the beginning of the DMA operation because it sets the `dma_on` bit in the CSR. It is automatically set when the write to the PAR occurs. Therefore, the PAR must be written last when setting up the DMA register.

**Table 6. DMA PCI Address Register Format**

Data Bit	Name	Read/Write	Definition
2..0	PAR	Read	Ground.
31..3	PAR	Read/write	29-bit counter.

## DMA Byte Counter Register (BCR)

The DMA byte counter register is a 17-bit register. The BCR is implemented with a 14-bit counter with the 3 least significant bits tied to ground. The BCR holds the byte count for the current DMA memory transfer and decrements (by 8 bytes) after every data transfer on the PCI bus. PCI bus memory transfer initiated by the DMA engine must be a QWORD transfer. The BCR can be written/read by the PCI bus master. [Table 7](#) shows the format of the BCR.

**Table 7. DMA Byte Counter Register Format**

Data Bit	Name	Read/Write	Definition
2..0	BCR	Read	Ground.
16..3	BCR	Read/Write	14-bit down counter.
31..17	Unused	-	-

## Interrupt Status Register (ISR)

The interrupt and status register is a 6-bit register. The ISR provides all interrupt source status signals to the interrupt handler. The ISR is a read-only register and can be read by another master on the PCI bus. [Table 8](#) shows the format of the ISR.

Data Bit	Mnemonic	Read/Write	Definition
0	int_pend	Read	Interrupt pending. The DMA engine asserts <code>int_pend</code> to indicate that the DMA interrupt is pending. The possible interrupt signals are <code>err_pend</code> and <code>dma_tc</code> .
1	err_pend	Read	Error pending. When high, <code>err_pend</code> indicates that an error has occurred during the DMA memory transfer. The interrupt handler must read the PCI configuration status register and clear the appropriate bits. Any one of the following PCI status register bits can assert <code>err_pend</code> : <code>mstr_abrt</code> , <code>tar_abrt</code> , and <code>det_par_err</code> .
2	int_irq	Read	Interrupt request. When high, <code>int_irq</code> indicates that the local logic is requesting an interrupt, i.e., the <code>l_irqn</code> signal to the <code>pci_mt64</code> function is asserted.
3	dma_tc	Read	DMA terminal count. When high, <code>dma_tc</code> indicates that the DMA transfer is complete. The <code>dma_tc</code> bit is reset with any of the following three conditions: <ul style="list-style-type: none"> <li>■ A read transaction to the ISR.</li> <li>■ A write transaction to the CSR.</li> <li>■ A write transaction to the PAR.</li> </ul>
4	ad_loaded	Read	Address loaded. When high <code>ad_loaded</code> indicates that the address has been loaded in the PAR. This bit is cleared if any of the following conditions occur: <ul style="list-style-type: none"> <li>■ The DMA operation completes and the <code>dma_tc</code> bit is set.</li> <li>■ The <code>flush</code> bit is set.</li> <li>■ The PCI bus is reset with <code>rstn</code> going low.</li> <li>■ The <code>ad_loaded</code> bit triggers the beginning of the DMA operation because it sets the <code>dma_on</code> bit in the CSR. It is automatically set when the write to the PAR occurs; therefore, the PAR must be written last when setting up the DMA register.</li> </ul>
5	-	-	Reserved.

## Local Address Register (LAR)

The local address register (LAR) is a 25-bit register. The LAR holds the SDRAM address from which the data will be transferred to/from the SDRAM. It is implemented with a 22-bit counter and the 3 least significant bits are tied to ground. This register decrements after every data phase transfer. The LAR is a write-only register. Table 9 shows the format of the LAR.

<i>Table 9. DMA Local Address Register Format</i>			
Data Bit	Name	Read/Write	Definition
2..0	LAR	Write	Ground
27..3	LAR	Write	25-bit down counter
31..28	Unused	-	-

## DMA Operation

To activate the DMA, the DMA registers must be written in the following sequence:

1. Write the CSR with the appropriate value.
2. Write the LAR with the DDR memory starting address of the transaction.
3. Write the BCR with the number of bytes to be transferred.
4. Write the PAR with the starting PCI address for the current transaction.

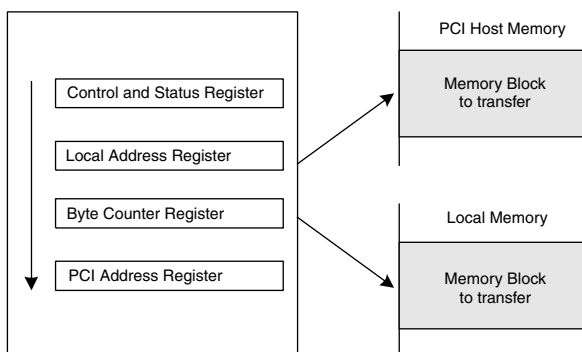
After the PAR is loaded with the PCI address, the `ad_loaded` bit in the ISR register is set and the DMA state machine is triggered. In the subsequent clock cycle, the `dma_on` bit of the CSR register is set to indicate that the DMA transfer is in progress. The DMA state machine then sends the request to signal the master control logic to request a PCI master transaction; the master control logic forwards the request to the `pci_mt64` interface to request the PCI bus. The DMA state machine also asserts the `local_start` signal to request a DDR access through the DDR controller.

Once the bus has been granted to the `pci_mt64` master, data transfers can take place if data is available in the appropriate data FIFO buffer. If data is not available wait state(s) are inserted on the PCI bus. The BCR counts down after every data transfer on the PCI bus for DMA write and on the local side for DMA read until it reaches zero. The DMA control logic then sets the `dma_tc` and `int_pend` bits and resets the `ad_loaded` and `dma_on` bits to return the DMA to the idle state.

In the event of an abnormal termination of a DMA transaction where the byte counter has not expired, the DMA state machine waits for the master and DDR control logic to return to the idle state before requesting a new master transaction to transfer the remaining data.

Figure 6 shows the register set up sequence for DMA operation.

**Figure 6. DMA Setup Sequence**



## Interrupt Request

The local side logic requests an interrupt when one of the following conditions occurs:

- The DMA transaction has been completed successfully and the `dma_tc` bit of the ISR has been set.
- An error has been detected and the `err_pend` bit of the ISR has been set.

The interrupt request signal is asserted until the appropriate bit causing the interrupt is cleared.

## Flash Interface

The reference design includes a PCI-to-flash memory interface used to program flash memory sections with device configuration files. The flash device on the Stratix PCI development board is divided into sections. Section 0 contains the factory default configuration files and is not accessible by the user.

Device configuration data is written into the flash memory device via a 32- or 64-bit PCI bus. The interface between the Altera `pci_mt64` MegaCore function and the flash memory interface is 32 bits and the interface between the flash memory interface and the flash memory device is 8 bits. The PCI-to-flash interface only implements read, erase, and write functionality. Figure 7 shows the flash memory interface block diagram.

**Figure 7. Flash Memory Controller Block Diagram**

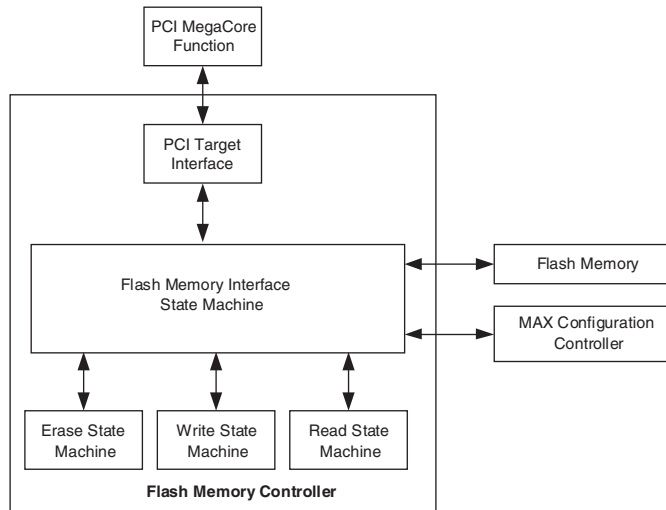


Table 10 describes the flash memory interface signals.

<b>Table 10. Flash Memory Interface Signals (Part 1 of 2)</b>			
<b>Name</b>	<b>Type</b>	<b>Polarity</b>	<b>Description</b>
<code>clk</code>	Input	-	Clock.
<code>rstn</code>	Input	Low	Reset.
<code>s_adri[25..0]</code>	Input	-	Register address. This signal selects the command, status, address, or data register in the flash memory interface.
<code>config_in[31..0]</code>	Input	-	Register data. This signal contains the data to be written to the register selected by <code>s_adri</code> .
<code>lt_tsr_0</code>	Input	High	Register transaction enable. The PCI MegaCore function asserts this signal when a target transaction is made to BAR0.

**Table 10. Flash Memory Interface Signals (Part 2 of 2)**

Name	Type	Polarity	Description
lt_dxfrn	Input	Low	The PCI MegaCore function asserts this signal when a data transfer on the local side is successful.
f_data_in[7..0]	Input	-	Flash data input. <code>f_data_in[7]</code> provides status from the flash memory during flash erase and write operations.
lt_rdyn	Output	Low	Local target ready. The flash memory interface asserts this signal when a flash memory operation is complete.
lt_discn	Output	Low	Local target disconnect. The flash memory interface asserts this signal when a flash memory operation is incomplete.
f_resetn	Output	Low	Flash memory reset.
f_cen	Output	Low	Flash memory chip enable.
f_wen	Output	Low	Flash memory write enable.
f_oen	Output	Low	Flash memory output enable.
f_addr_out_oe	Output	High	Flash memory address output enable. This signal enables the flash memory interface to drive addresses to the flash memory.
f_addr_out[22..0]	Output	-	Flash memory address.
f_data_out_oe	Output	Low	Flash memory data output enable. This signal enables the flash memory interface to drive data to the flash memory.
f_data_out[7..0]	Output	-	Flash memory interface write data.
flash_status[31..0]	Output	-	Flash memory interface status. During flash memory erase, write, and read operations, a high on <code>flash_status[0]</code> indicates the flash memory interface is performing an erase, write, or read operation.
flash_rd_data[31..0]	Output	-	Flash memory interface read data.
reconfig	Output	Low	Reconfigure the Stratix device. A low on this signal reconfigures the Stratix device from the section of memory specified by <code>reconfig_addr</code> .
reconfig_addr[3..0]	Output	-	Reconfigure section. This signal defines the section of memory to be used during Stratix device reconfiguration.

The flash memory interface implements the following five registers:

- *Command Register*—This register holds the command value for the flash memory interface. Depending on the value written into this register, the flash memory interface triggers the erase state machine or the write state machine.
- *Address Register*—This register holds the address of the flash memory device for the operation defined in the command register.
- *Write Data Register*—This register holds the data to be written into the flash memory device during a flash memory write.
- *Read Data Register*—This register holds the data that has been read from the flash memory device during a flash memory read.
- *Status Register*—This register indicates the status of the flash memory interface during flash memory erase, write, and read operations. [Table 11](#) describes the status register values.

Status Register	Flash Memory Operation	Description
00h	Erase, write, or read	The flash memory interface is idle.
01h	Erase, write, or read	The flash memory interface is erasing, writing, or reading to the flash memory device.

[Table 12](#) shows the PCI address space for the flash memory interface.

Memory Region	Range Reserved	Read/Write	Mnemonic	Register Name
BAR0	0030h-0033h	Write	Cmd[31..0]	Command
	0034h-0037h	Write	Addr[21..0]	Address
	0038h-003Bh	Write	WrData[31..0]	Write Data
	0028h-002Bh	Read	Status[31..0]	Status
	002Ch-002Fh	Read	RdData[31..0]	Read Data

## Flash Memory Interface State Machine

The flash memory interface state machine responds to specific PCI transactions and triggers the erase, write, or read state machines. [Figure 8](#) shows the state diagram for the flash memory interface.

Figure 8. Flash Memory Interface State Diagram

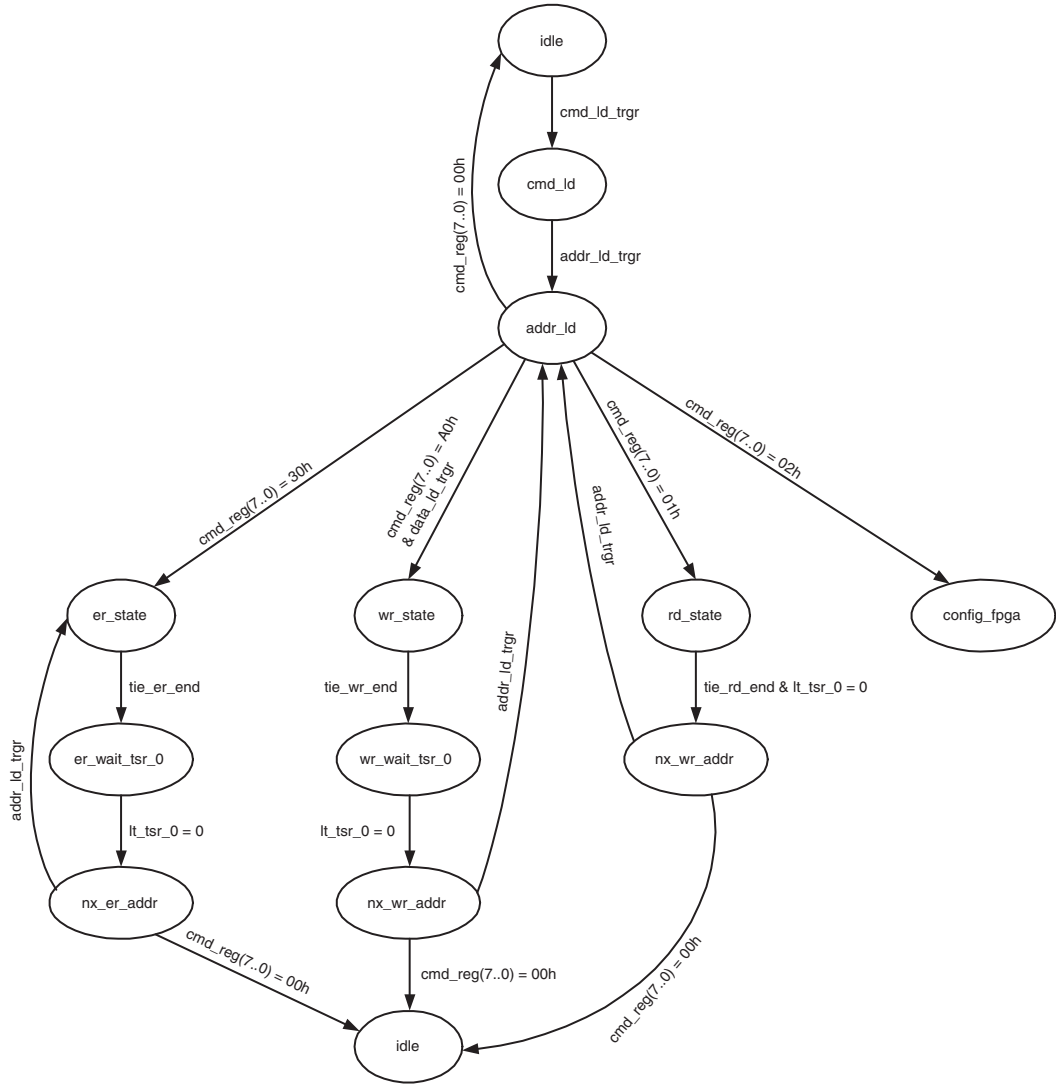


Table 13 describes the input signals to the flash memory interface state machine.

<b>Name</b>	<b>Polarity</b>	<b>Description</b>
cmd_ld_trgr	High	Command load trigger. This signal is high when Cmd[31..0] is written.
addr_ld_trgr	High	Address load trigger. This signal is high when Addr[31..0] is written.
data_ld_trgr	High	Data load trigger. This signal is high when Data[31..0] is written.
lt_tsr_0	High	Local target TSR 0. This signal is active during a BAR 0 access.
tie_er_end	High	Erase end. The erase state machine asserts this signal at the completion of an erase operation.
tie_wr_end	High	Write end. The write state machine asserts this signal at the end of a write operation.
tie_rd_end	High	Read end. The read state machine asserts this signal at the end of a read operation.

## Erase State Machine

The erase state machine is triggered when the flash memory interface state machine enters the `er_state` state. The value in the address register is used as the sector address to determine which sector of flash memory is erased. The erase state machine erases one 64-KByte sector with a single command.

The flash memory device requires a specific sequence on the flash memory interface signals to perform a sector erase. Table 14 shows the signal sequence that performs a flash memory device sector erase.

**Table 14. Flash Memory Device Sector Erase Signal Sequence**

Flash Memory Signals	Command Sequence (1)													
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
f_cen	0	1	0	1	0	1	0	1	0	1	0	1	1	1
f_wen	0	1	0	1	0	1	0	1	0	1	0	1	1	1
f_oen	1	1	1	1	1	1	1	1	1	1	1	1	0	1
f_addr_out_oe	1	0	1	0	1	0	1	0	1	0	1	0	1	0
f_addr_out[21..0]	AAAh	-	555h	-	AAAh	-	AAAh	-	555h	-	SA (2)	-	SA	-
f_data_out_oe	1	0	1	0	1	0	1	0	1	0	1	0	0	0
f_data_out[7..0]	AAh	-	55h	-	80h	-	AAh	-	55h	-	30h	-	-	-
f_data_in[7]	-	-	-	-	-	-	-	-	-	-	-	-	RDY (3)	-

**Notes to Table 14:**

- (1)  $T_n$  is the time interval for which the signals must be valid.
- (2) SA is the 23-bit flash memory sector address of the sector to be erased
- (3) The erase state machine remains in this state until the rdy signal on f\_data\_in[7] is 1.

After the erase state machine has issued the signals to time interval T11, the flash memory device drives f\_data\_in[7] to 0. When the sector erase operation completes, the flash memory device sets f\_data\_in[7] to 1.

The flash memory interface flash\_status[0] signal shows a 1 while the flash memory performs the sector erase.

## Write State Machine

The write state machine is triggered when the flash memory interface enters the wr\_state state. Four byte write command sequences are issued to the flash memory device. The first sequence writes the low eight bits of the data register, WrData[7..0], to the address contained in the address register. The next three command sequences write successive bytes from the data register, WrData[15..8], WrData[23..16], and WrData[31..24], to sequential addresses of the flash memory device.

The flash memory device requires a specific sequence on the flash memory interface signals to perform byte writes. Table 15 shows the sequence of signals that performs a flash memory device byte write.

**Table 15. Flash Memory Device Byte Write Signal Sequence**

Flash Memory Signals	Command Sequence									
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
f_cen	0	1	0	1	0	1	0	1	1	1
f_wen	0	1	0	1	0	1	0	1	1	1
f_oen	1	1	1	1	1	1	1	1	0	1
f_addr_out_oe	1	0	1	0	1	0	1	0	1	0
f_addr_out[21..0]	AAAh	-	555h	-	AAAh	-	WA (1)	-	WA	-
f_data_out_oe	1	0	1	0	1	0	1	0	0	0
f_data_out[7..0]	AAh	-	55h	-	A0h	-	WD (2)	-	-	-
f_data_in[7]	-	-	-	-	-	-	-	-	WD[7] (3)	-

**Notes to Table 15:**

- (1) WA is the 23-bit flash memory address of the byte of data to be written.
- (2) WD is the 8-bit byte of data to be written.
- (3) The write state machine remains in this state until the value read from f\_data\_in[7] is equal to WD[7], the high bit that was written to the flash memory device for the current byte write command.

After the write state machine has issued the signals to time interval T7, the flash memory device drives f\_data\_in[7] to the complement of the value written to f\_data\_out[7]. When the byte write operation completes, the flash memory device sets f\_data\_in[7] to the value written to f\_data\_out[7].

The flash memory interface flash\_status[0] signal shows a 1 while the flash memory is performing the four byte write commands.

## Read State Machine

The read state machine is triggered when the flash memory interface enters the rd\_state state. Four byte reads are performed from sequential addresses of the flash memory device. The first byte is placed in the low eight bits of the read data register, RdData[7..0]. The next three bytes are placed in successive bytes of the read data register, RdData[15..8], RdData[23..16], and RdData[31..24].

## Flash Memory Configuration

Flash memory configuration is triggered when the flash memory interface enters the `fpga_config` state. Address register bits `Addr[22..20]` are decoded to flash memory sections according to [Table 16](#).

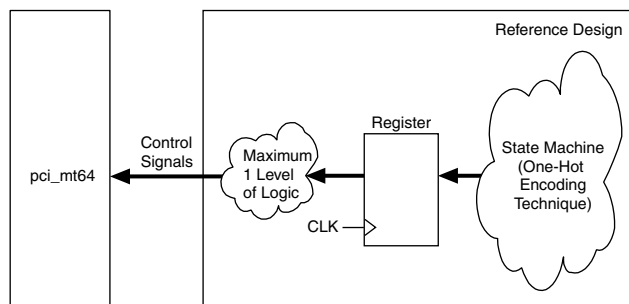
Addr[22..20]	Flash Memory Section	Description
010b	0	Factory-programmed configuration image
011b	1	User configuration image 1
100b	2	User configuration image 2
101b	3	User configuration image 3
Others	0	Factory-programmed configuration image

## Designing for 66-MHz Operation

When designing for 66-MHz operation, follow the guidelines below.

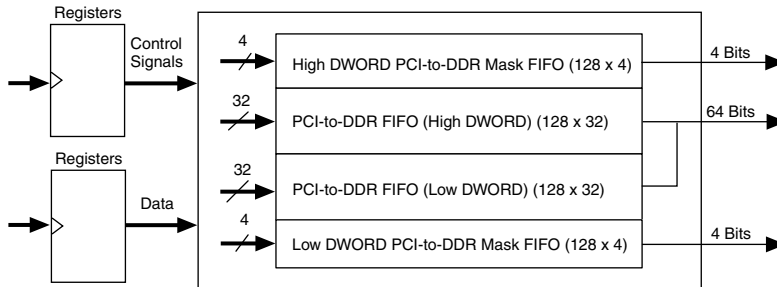
- All the input control signals to the PCI function must be registered with—at the most—one level of logic between the register and the function. See [Figure 9](#).

**Figure 9. Input Control Signals**



- Use one-hot encoding for all state machines.
- When using the reference design, control signals and the data written into the FIFO buffer must be registered. See [Figure 10](#).

Figure 10. Register Control Signals & Data



## Design File Structures

This section explains the file structure and modules used in the reference design. Figure 11 shows the file structure.

Figure 11. Reference Design File Structure

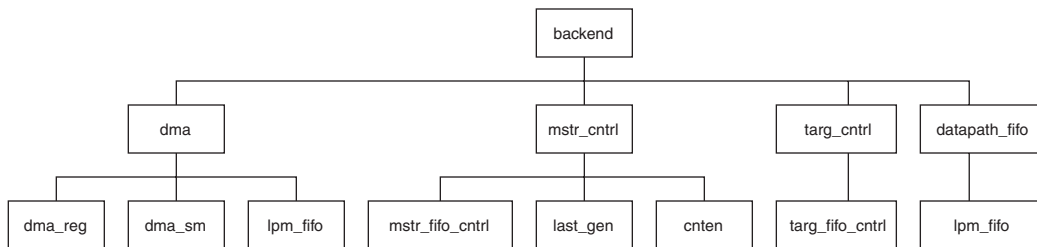


Table 17 describes the modules shown in Figure 11.

Module	Description
dma_reg	Write and read control. This module includes all of the DMA registers such as CSR, BCR, PAR, ISR, and LAR.
dma_sm	DMA state machines. This module provides DMA control signals.
lpm_fifo	Altera parameterizable LPM FIFO buffer.
dma	Top level of the DMA engine.
mstr_fifo_cntrl	Master FIFO buffer control. This module provides PCI-to-DDR FIFO buffers write control, DDR-to-PCI FIFO buffers read control, and masking of invalid data for master transactions.
last_gen	lm_last signal generation. This module generates the lm_lastn signal for 32-bit and 64-bit master transactions.

Module	Description
cnten	Count enable generation. This module provides the count enable signals for BCR, PAR, and LAR counters.
mstr_cntrl	Master control. This module drives all the master local signals of the <code>pci_mt64</code> function. It also interfaces to the DMA engine for the status of the current master transaction.
targ_fifo_cntrl	Target FIFO buffer control. This module provides PCI-to-DDR FIFO buffers write control, DDR-to-PCI FIFO buffers read control, and masking of invalid data for target transactions.
targ_cntrl	Target control. This module drives all the target local signals of the <code>pci_mt64</code> function. It also connects to the DDR interface module to provide DDR memory access control.
datapath_fifo	Data path FIFO buffer. This module includes all of the FIFO buffers that buffer the data transfer between the PCI function and the DDR memory. It also includes the PCI-to-DDR masking FIFO buffer.
backend	The top level of the reference design.

Figure 12 shows the structure of the PCI-to-flash interface.

**Figure 12. PCI-to-Flash Interface**

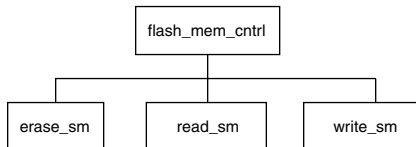


Table 18 describes the modules shown in Figure 12.

Module	Description
erase_sm	Flash erase state machine. This module performs the erase functionality of the user flash sections of the external flash memory.
read_sm	Flash read state machine. This module performs the read functionality to the user flash sections of the external flash memory.
write_sm	Flash write state machine. This module performs the write functionality to the user flash section of the external memory.

**Table 18. PCI-to-Flash Submodule Description (Part 2 of 2)**

Module	Description
flash_mem_cntrl	Top level of the flash interface. This module provides the interface between the <code>pci_mt64</code> MegaCore function and the external flash memory. It also includes the interface logic to the Altera MAX device to configure the Stratix device with the configuration data stored in one of the flash sections.

Figure 13 shows the structure of the DDR memory interface.

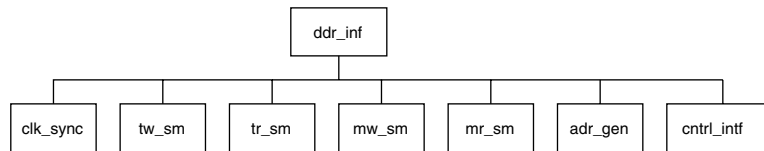
**Figure 13. DDR SDRAM Controller Interface File Structure**

Table 19 describes the modules shown in Figure 13.

**Table 19. DDR SDRAM Controller Interface Submodule Description**

Module	Description
clk_sync	Clock domain synchronization. This module synchronizes all of the signals that crossing from the DDR memory clock domain to the PCI clock domain.
tw_sm	Target write control state machine. This module interfaces with the target functionality of the reference design and directs the DDR memory write accesses.
tr_sm	Target read control state machine. This module interfaces with the target functionality of the reference design and directs the DDR memory read accesses.
mw_sm	Master write state machine. This module interfaces with the master functionality of the reference design and directs the DDR memory write accesses.
mr_sm	Master read state machine. This module interfaces with the master functionality of the reference design and directs the DDR memory read accesses.
adr_gen	Address generation. This module generates the DDR memory addresses for the DDR memory interface.
cntrl_intf	DDR SDRAM Controller interface. This module takes directions from all of the control state machines and translate them into the signal interface protocol on the user side of the DDR SDRAM Controller
ddr_intf	Top level of the DDR memory interface module that instantiate all of the submodules.

## Design Limitations

This reference design has the following limitations:

- The DMA is not designed to transfer data to a target that is only capable of completing single-cycle, 32-bit PCI transactions.
- The DMA cannot read from non-prefetchable memory space because the low DWORD needs to be reread if there is an odd number of DWORDs transferred.

## References

For more information refer to the following documents:

- *Altera PCI MegaCore Function User Guide*
- *Altera DDR SDRAM MegaCore Function User Guide*
- Micron Technology DDR SDRAM data sheet
- AMD 64-Mbit DL-type boot-block flash data sheet



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
[www.altera.com](http://www.altera.com)  
**Applications Hotline:**  
(800) 800-EPLD  
**Literature Services:**  
[lit\\_req@altera.com](mailto:lit_req@altera.com)

Copyright © 2003 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This reference design file, and your use thereof, is subject to and governed by the terms and conditions of the applicable Altera Reference Design License Agreement (found at [www.altera.com](http://www.altera.com)). By using this reference design file, you indicate your acceptance of such terms and conditions between you and Altera Corporation. In the event that you do not agree with such terms and conditions, you may not use the reference design file and please promptly destroy any copies you have made.

This reference design file being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed. By making this reference design file available, Altera expressly does not recommend, suggest or require that this reference design file be used in combination with any other product not provided by Altera.



I.S. EN ISO 9001