pci_mt64 MegaCore Function

Reference Design

September 2003, ver. 1.2 Functional Specification 10

Features

- Provides an interface between the Altera `pci_mt64` MegaCore® function and a 64-bit, 32-MByte SDRAM module
- Supports 32- and 64-bit PCI master and target transactions
- Supports chaining and non-chaining mode DMA
- Uses the dual-port FIFO buffer function from the library of parameterized modules (LPM)
- Uses the SDR SDRAM controller MegaCore function

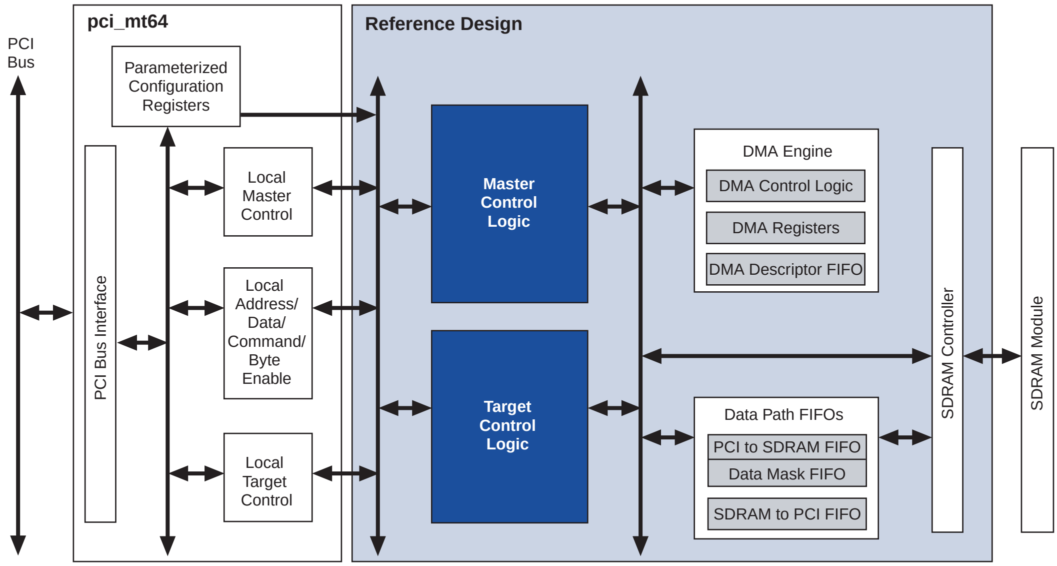
General Description

This reference design shows how to connect the local-side signals of the Altera® `pci_mt64` MegaCore function to local-side applications when the MegaCore function is used as a master or target on the PCI bus. 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 buffer functions
- SDRAM interface

FS-10-1.2

Figure 1. Block Diagram



Master Control Logic

When the `pci_mt64` function is acts as a master, the master control logic interacts with the DMA engine to control the PCI master transactions. During a PCI master write, the data flows from the local master to the PCI bus. The master control logic:

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

During a PCI master read, the data flows from the PCI bus to the local master. The master control logic:

- Provides the status of the PCI bus to the DMA engine
- Interacts with the PCI function to execute the PCI master read cycle
- Buffers the data read by the `pci_mt64` function into the PCI-to-SDRAM FIFO buffer

Target Control Logic

When the PCI function acts as a target, the `pci_mt64` function prompts the target control logic to execute target transactions. During a PCI target write, the data flows from PCI bus to the local target. The target control logic:

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

During a PCI target read, the data flows from local target to the PCI bus. The target control logic:

- Interacts with the `pci_mt64` function to execute a PCI target read
- Interacts with the SDRAM controller to fetch the data from the SDRAM
- Transfers the data from the SDRAM-to-PCI FIFO buffer to the `pci_mt64` function

DMA Engine

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

- DMA control logic
- DMA registers
- DMA descriptor FIFO buffer s

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
- Triggers a new access to the SDRAM
- Monitors the data path FIFO buffer's and the current SDRAM access
- Monitors the DMA registers in order to initiate a new transaction.
- Loads the address counter register (ACR) and byte counter register (BCR) in the DMA registers when DMA is in chaining mode
- Updates the interrupt status register (ISR) and control and status register (CSR) in the DMA registers (chaining and non-chaining mode)

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 consists of:

- Control and status register (CSR)
- Address counter register (ACR)
- Byte counter register (BCR)
- Interrupt status register (ISR)
- Local address counter

Detailed description of these registers is provided in "DMA Engine" on page 18.

DMA Descriptor FIFO Buffer

The DMA descriptor FIFO buffer provides the storage area for the series of byte count and PCI address pairs when the DMA is programmed to operate in chaining mode. The size of the descriptor FIFO buffer is 256 x 32, and it is capable of holding up to 128 DMA transactions in a chain. This FIFO buffer must be written with byte count and address pairs by another master/host on the PCI bus before starting the DMA in chaining mode. The descriptor FIFO buffer is read by the DMA control logic to fetch the current byte count to the BCR and address to the ACR before executing the next DMA transaction in a chain.

Data Path FIFO Buffers

The data path FIFO buffers serve as the buffer space for the data flowing between the SDRAM and PCI bus. The FIFO buffers are needed to resolve the SDRAM's high data-access latency. The reference design implements the following FIFO buffer's:

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

PCI-to-SDRAM FIFO Buffer

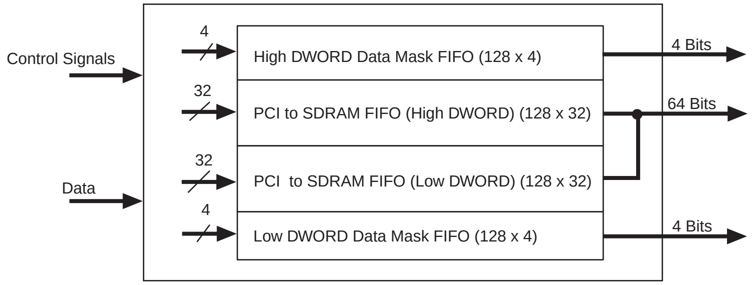
This FIFO buffer stores data transferred from the PCI bus to the SDRAM. It is used during PCI target write and master read transactions. During these transactions, data written into the FIFO buffer is controlled by the target and master control logic, respectively. The SDRAM interface module reads the data from the FIFO buffer. The PCI-to-SDRAM FIFO buffer is a combination of two 128 x 32 buffers, one used for the low DWORD and the other used for the high DWORD. The write to the low DWORD FIFO buffer and the high DWORD FIFO buffer can be controlled independently, giving the backend the flexibility to support both 32- and 64 bit transactions. The reference design is a pure 64-bit design, however, by using the separate high and low DWORD read/write controls, a 32-bit PCI transaction can be accepted and data can be written one DWORD at a time into the 64-bit FIFO buffer. SDRAM interface logic reads one QWORD word (64-bits) at a time from this buffer.

Data Mask FIFO Buffer

This FIFO buffer is used to mask invalid data from being written into the SDRAM when the address of the DWORD transferred is not QWORD aligned. This FIFO buffer can be thought as the width extension of the PCI-to-SDRAM FIFO buffer. This FIFO buffer is 128 x 8 in size and has separate read and write controls, which are the same as the PCI-to-SDRAM FIFO buffer except when there is a need to mask the unaligned DWORD.

For example, if there is a target write to the SDRAM starting at high DWORD (`xf0000004`), the first low DWORD (`xf0000000`) of the FIFO buffer is invalid and should not be written into the SDRAM. Therefore, the DWORD at address `xf0000000` 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-SDRAM & Data Mask FIFO Buffers



SDRAM-to-PCI FIFO

The SDRAM-to-PCI FIFO buffer provides the buffer for the data transferred from the SDRAM to the PCI bus. This FIFO buffer is used during PCI target read and master write transactions. During the target read and master write transactions, the SDRAM controller controls data written into this FIFO buffer. The target or master control modules control reading data from this buffer.

The SDRAM-to-PCI FIFO buffer is 128 x 64 and has a single read/write control. 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 local side to the `pci_mt64` function in 64-bit mode. The function then multiplexes the low and high DWORD to its output registers.

SDRAM Controller

The SDRAM controller controls the read and write operation from the SDRAM module. The SDRAM controller logic consists of an Altera SDR SDRAM controller and interface logic to connect the DMA engine, data path FIFO buffers, master control logic, and target control logic to the SDRAM controller.

Functional Description

The `pci_mt64` function used in this reference design implements two base address registers (BARs): `BAR0` and `BAR1`. `BAR0` is set to reserve 1 MBytes of system address space to map all the local read/write registers and the descriptor FIFO buffer. `BAR1` is set to reserve 16 MBytes of the system address space to map the to the 16-MByte SDRAM module.

Memory Mapping

`BAR0` reserves 1 MByte of the system address space; the 12 most significant bits of the PCI address are decoded by the `pci_mt64` function to claim a target transaction. All of the DMA registers and the descriptor FIFO buffer are mapped to this address space. This reserved space is divided into two equal regions of 512 KBytes: the upper memory region is used for the DMA FIFO buffer and the lower region is used to map the DMA registers.

BAR1 reserves 16 MBytes of the system address space; the 8 most significant bits of the PCI address are decoded by the `pci_mt64` function to claim a target transaction. This memory space is mapped to the lower 16 MBytes of the 32-MByte SDRAM module. The upper 16 MBytes of the SDRAM module are not used in target operation.

Table 1 describes the pci_mt64 BAR mapping.

Table 1. Memory Address Space

Memory Region	Block Size	Address Offset	Description
BAR0	512 KBytes	00000h-7FFFFh	Internal SDRAM configuration registers.
	512 KBytes	80000h-FFFFFh	DMA descriptor FIFO buffer for chaining DMA operation.
BAR1	16 MBytes	000000h-FFFFFFh	First 16 MBytes of the 32 MBytes SDRAM module.

Table 2 describes the internal registers and descriptor FIFO buffer memory-mapped addresses.

Table 2. Internal Registers & DMA FIFO buffer Memory Maps

Range Reserved	Read/Write	Mnemonic	Register Name
00000h-00003h	Read/Write	CSR[8:0]	DMA Control/Status
00004h-00007h	Read/Write	ACR[31:0]	DMA Address Counter
00008h-0000Bh	Read/Write	BCR[16:0]	DMA Byte Counter
0000Ch-0000Fh	Read	ISR[6:0]	DMA Interrupt/Status
00010h-00013h	Write	LAR[25:0]	Local Address
00018h-0001Bh	Write	SDRAM_BASE_ADDR	SDRAM Configuration
0001Ch-0001Fh	Write	SDRAM_CFG_REG0	SDRAM Configuration
00020h-00023h	Write	SDRAM_CFG_REG1	SDRAM Configuration
00024h-00027h	Write	SDRAM_CFG_REG2	SDRAM Configuration
80000h-FFFFFh	Write	DMA_FIFO	DMA Descriptor FIFO Buffer

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***.

Table 3. Local Signals Used in Target Transactions (Part 1 of 2)

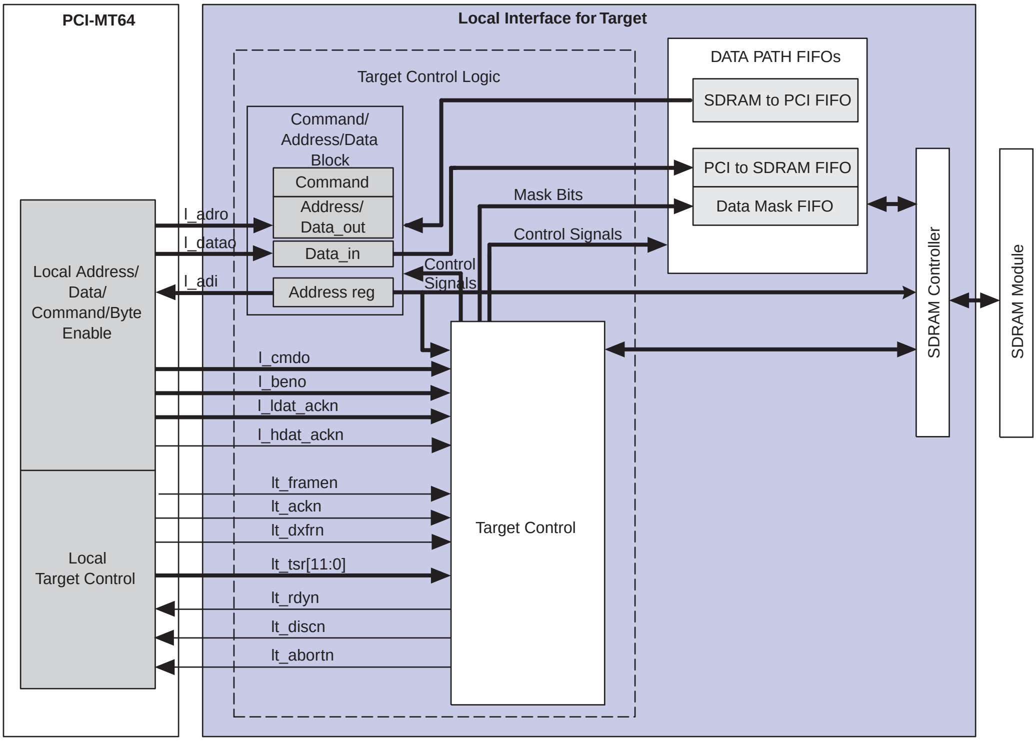
Type	Signal	Description
Data path input	`l_adi`	Local address/data input. This signal is driven with data by the local side during PCI bus-initiated target read transactions.
Data path output	`l_dato`	Local data output. The PCI function drives data on this bus during target write transactions.
	`l_adro`	Local address output. The PCI function drives address on this bus during target read and write transactions.
	`l_beno`	Local byte-enable output. The PCI function drives a byte enable on this bus during target read and write transactions.
	`l_cmdo`	Local command output. The PCI function drives commands on this bus during target operation.
64-bit support signals	`l_ldat_ackn`	Local low data acknowledge. When asserted, this signal indicates that valid data is present on the `l_dato[31..0]` bus. `lt_ackn` must be used to qualify valid data.
	`l_hdat_ackn`	Local high data acknowledge. When asserted, this signal indicates that valid data is present on the `l_dato[63..32]` bus. `lt_ackn` must be used to qualify valid data.
Local target control inputs	`lt_rdyn`	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.
	`lt_discn`	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.
	`lt_abortn`	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 devseln, 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[11..0]	Local target transaction status register. The lt_tsr[11..0] bus carries several signals, which can be monitored for the transaction status.

Figure 3 provides a block diagram of the target reference design.

Figure 3. 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.

The reference design uses `l_adro[19]` and the `lt_tsr[1..0]` signals from the `pci_mt64` function to determine if the current transaction is for SDRAM, the DMA registers, or the DMA FIFO buffer. The reference design decodes `l_cmdo[3..0]` to determine whether the transaction is a memory read or write.

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, due to the SDRAM requirement of more than 16 clock cycles to provide the initial data to the SDRAM-to-PCI FIFO buffer. All memory read cycles are treated as delayed transactions.
- While waiting for the SDRAM-to-PCI FIFO buffer to be filled to a pre-determined number of QWORDS (threshold), all memory target transactions are retried and only target access to the internal registers is accepted.
- After the threshold has been reached, the local side asserts `lt_rdyn` to accept a memory read cycle with the same address as that of the read cycle for which the retry was terminated.
- The data from the SDRAM-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 falls below a predetermined value, lt_discn is asserted to end the current target read since the SDRAM can not keep supplying the read data on time. Subsequently, the master has to re-read the rest of the data.
- On deassertion of `lt_framen` signal, the remaining data in the SDRAM-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-SDRAM FIFO buffer until `lt_framen` is deasserted by the `pci_mt64` function.
- On the other side of the PCI-to-SDRAM FIFO buffer, the SDRAM interface reads the data from the PCI-to-SDRAM FIFO buffer and writes it into the SDRAM until the FIFO buffer becomes empty or the target transaction ends.
- If the FIFO buffer becomes almost full during the target write transaction, lt_discn is asserted to stop the data being written into the

FIFO buffer. The master has to re-initiate a write cycle to complete writing the rest of the data from where it left off.

- On termination of a target write transaction, all target memory transactions is retried until the SDRAM controller has completed writing the data from the PCI-to-SDRAM FIFO buffer into the SDRAM.
- If the PCI bus is 32 bits wide, the low and high DWORD of the FIFO buffer is written alternately, starting at the low DWORD FIFO buffer. The write to the low DWORD FIFO buffer occurs when the `lt_dxfrn` and `l_ldat_ackn` signals are asserted by the `pci_mt64` function. The write to the high DWORD FIFO buffer occurs when the `pci_mt64` function asserts the `lt_dxfrn` and `l_hdat_ackn` signals.

To support 32-bit target write transactions, the PCI-to-SDRAM FIFO buffer is divided into two 32-bit FIFO buffers for low and high DWORDS. The start address or the end address can be QWORD dealigned. During the 32-bit target write transaction, the high and low SDRAM-to-PCI FIFO buffers are written alternately starting at the low FIFO buffer and ending at the high FIFO buffer. Depending on the starting address QWORD alignment, the data written into the PCI-to-SDRAM FIFO buffers is marked valid or invalid by writing 0 or 1 in the data mask FIFO buffer. The following cases may occur.

- The starting address and number of DWORDs transferred are QWORD aligned. In this case, all the data written into the high and low FIFO buffer are valid. Masking bits for low and high DWORDs are set to zero and written into the byte enable FIFO buffer simultaneously.
- The starting address is QWORD aligned, but the number of DWORDs transferred is not QWORD aligned. In this case, the last DWORD written into the high FIFO buffer is marked invalid with the corresponding masking bits in the data mask FIFO buffer.
- The starting address is not QWORD aligned but number of DWORDs transferred are QWORD aligned. In this case, the first 32-bit data written into the low FIFO buffer is marked invalid with the corresponding masking bits in the data mask FIFO buffer.
- The starting address and number of DWORDs transferred are not QWORD aligned. In this case, the first DWORD in the low DWORD FIFO buffer and the last data in the high DWORD FIFO buffer is marked invalid with the corresponding masking bits in the data mask FIFO buffer.

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. The DMA descriptor FIFO buffer can accept single-cycle or burst writes.

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*.

Table 4. Signals Used in Master Transactions (Part 1 of 2)

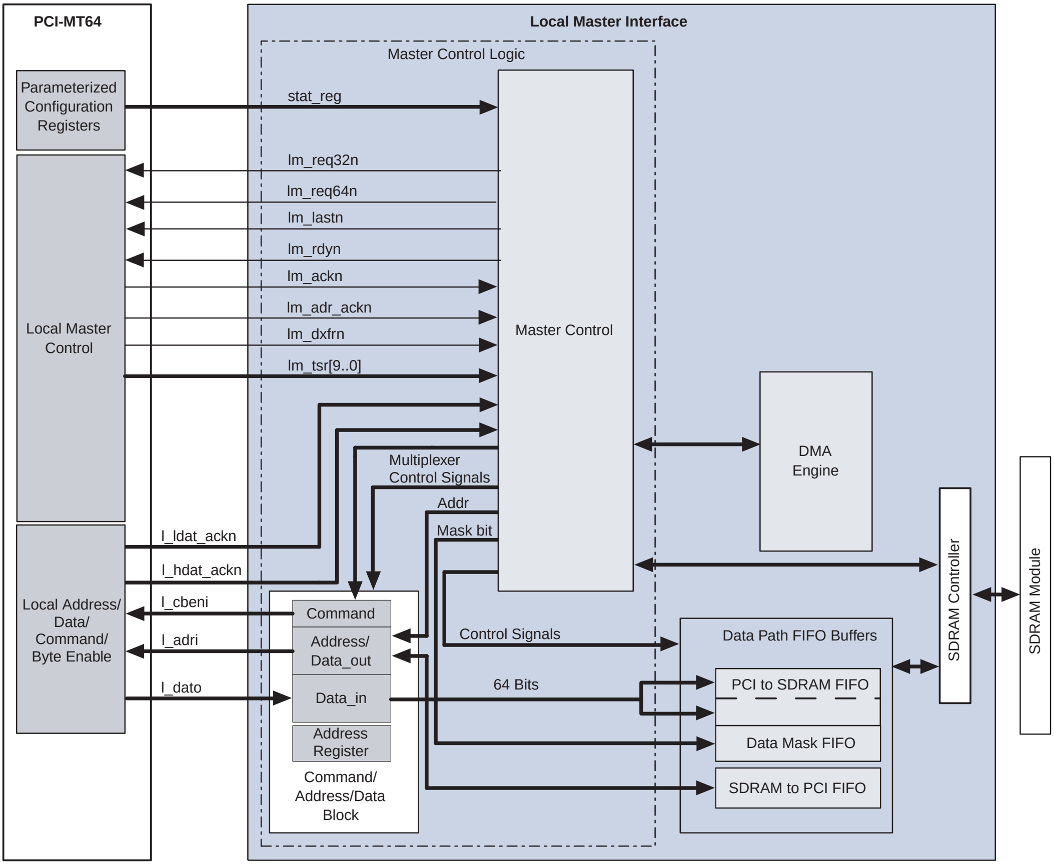
Type	Signal	Description
Data path input	`l_adi`	Local address/data input. This signal is a local-side time multiplexed address/data bus. During master transaction the local side must provide address on this bus when `lm_adr_ackn` is asserted. Data is driven on this bus from the local side during master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

Table 4. Signals Used in Master Transactions (Part 2 of 2)

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 `framen` as soon as possible and asserts `lt_rdyn` 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[9..0]`	Local master transaction status register. These signals inform the local interface the progress of the transaction

Figure 4 shows a block diagram of the master portion of the reference design.

Figure 4. Master Reference Design



The DMA engine sends control signals to the master control logic to request a master transaction. The master control logic interacts with the local master signals of the PCI function to execute the master transactions, provide status signals of the ongoing transaction to the DMA engine, and interact with the data path FIFO buffer to transfer the data from the SDRAM to the PCI function or vice versa.

The command/address/data (CAD) block in the master control logic uses the data path input from the function to provide relevant data to the master control and data path FIFO buffers. The CAD uses data path outputs from the master 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 target control logic and master control logic.

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-bit transaction or 32-bit transaction, respectively.
- At the same time, the DMA engine signals the SDRAM interface to request an SDRAM write access.
- The PCI function outputs `reqn` on 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.
- On receiving the `gntn` signal from the arbiter the `pci_mt64` function asserts the `lm_adr_ackn` and `lm_tsr[1]` local side signals, indicating the bus has been granted and the local side must supply the starting address and command. 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 FIFO buffer are written alternately, starting at the low DWORD FIFO buffer. The write to the low DWORD FIFO buffer occurs when the `lm_dxfrn` and `l_ldat_ackn` signals are asserted; the write to the high DWORD FIFO buffer occurs when the `lm_dxfrn` and `l_hdat_ackn` signals are asserted.
- IF the SDRAM cannot empty the FIFO buffer fast enough and the FIFO buffer is almost full, the master control logic asserts the stop signal to request the DMA and the PCI-mt64 master function to end the current transaction to prevent FIFO-buffer overflow.
- If the latency timer expires, the `pci_mt64` function will stop the current master transaction and assert the lm_tsr[4] to indicate a premature termination of the current transfer. The master control logic and the DMA engine monitor this signal and react accordingly.

The SDRAM interface logic reads one QWORD (64 bits) of data at a time from the PCI-to-SDRAM FIFO buffer and writes it into the SDRAM memory. During a master read transaction, the master control logic asserts the `stop` signal to the DMA and `lm_lastn` signal to the `pci_mt64` function and the SDRAM interface if the master must release the bus prematurely. 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 SDRAM interface transfers valid data from PCI-to-SDRAM FIFO buffer and stops the operation. If the master has not read all of the data from the target, the master control logic and the DMA engine must restart the entire transaction.

PCI master transactions always start at a QWORD-aligned address. If an odd number of DWORDs are written into the PCI-to-SDRAM 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 data 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 SDRAM interface to request an SDRAM access.
- The master control logic waits for the SDRAM-to-PCI FIFO buffer to be filled with a predetermined number of QWORDs before requesting the PCI bus. This action ensures that the master does not violate PCI latency protocol because of slow SDRAM reads.
- If the transfer count is less than 32 QWORDs, the master control logic waits for all of the data to be ready in the SDRAM-to-PCI FIFO buffer before requesting the bus.
- If the transfer count is more than or equal to 32 QWORDs, the master control logic requests the bus by asserting `lm_req32n` or `lm_req64n`.
- One QWORD is read at a time from the SDRAM-to-PCI FIFO buffer to the `pci_mt64` function.
- 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 and the local side must provide a valid address and command.
- The master control must provide PCI address on `l_adi` and PCI command on `l_cbeni`.

- 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 ready to transfer the data from SDRAM-to-PCI FIFO buffer.
- 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.

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

If a master write transaction is terminated prematurely, the master control logic asserts the `lm_lastn` signal to the `pci_mt64` function local side and the DMA and the `abort` signal to the SDRAM interface. The reference design continues to flush the SDRAM-to-PCI FIFO buffer until the SDRAM interface completes the current write cycle by writing the threshold number of QWORDs into the FIFO buffer. If the master did not complete the transaction to the target, the master control logic and the DMA engine restart the process to request a PCI write.

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 `xf000_0000` through `xf000_0080`, but the transaction is terminated early after transferring data at address `xf0000_0078` (interrupted at high DWORD). The master restarts the process to request a PCI write transaction and begins transferring data from address `xf000_0070` (low DWORD).

DMA Engine

DMA Registers

The DMA reference design implements the following registers

- Control and status register (CSR)
- Address counter register (ACR)
- Byte counter register (BCR)
- Interrupt status register (ISR)
- Local address counter

These registers are memory-mapped to BAR0 of the pci_mt64 function and must be configured by another master/host on the PCI bus.

Control & Status Register (CSR)

The control and 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.

Table 5. DMA Control & Status Register Format

Data Bit	Mnemonic	Read/Write	Definition
0	int_ena	Read/Write	PCI interrupt enable.
1	flush	Write	Flush the buffer. When high, it flushes the DMA FIFO buffer to empty and resets DMA_tc and ad_loaded (bits 3 and 4 of the ISR). The flush bit resets itself; therefore, it always zero when read. The flush bit should never be set when the DMA_on bit is set because a DMA transfer is in progress.
2	-	-	Reserved.
3	write	Read/Write	Memory read/write. The write bit determines the direction of the DMA transfer. When write 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	DMA_ena	Read/Write	DMA enable. When high, DMA_ena 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	tci_dis	Read/Write	Transfer complete interrupt disable. When high, tci_int disables DMA_tc (bit 3 of the ISR) from generating PCI bus interrupts.
6	DMA_on	Read/Write	DMA on. When high, DMA_on indicates that the DMA engine can request ownership of the PCI bus. The DMA_on bit is high when: ■ The ACR is loaded (bit 4 of ISR). ■ The DMA is enabled. ■ There is no error pending. ■ The DMA transfer sequence begins when DMA_on is set. Under normal conditions (i.e., DMA is enabled and no error is pending), DMA_on is set when a write transaction to the ACR occurs via a target write.
7	-	-	Reserved.
8	chain_ena	Read/Write	Chaining mode enable. When high, chain_ena indicates that the current DMA request is in chaining mode and the DMA descriptor FIFO buffer has the information for the DMA transfers.

DMA Address Counter (ACR)

The address counter is a 32-bit register. The ACR is implemented with a 29-bit counter and the 3 least significant bits are tied to ground. The ACR 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 ACR can be written/read by the PCI bus master. Table 6 shows the format of the ACR. 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 ACR occurs. Therefore, the ACR must be written last when setting up the DMA register.

Table 6. DMA Address Counter Format

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

DMA Byte Counter (BCR)

The DMA byte counter is a 17-bit register. The BCR is implemented with a 14-bit counter and the 3 least significant bits are 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 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.

Table 8. Interrupt Status Register Format (Part 1 of 2)

Data Bit	Mnemonic	Read/Write	Definition
0	`int_pend`	Read	Interrupt pending. The DMA engine asserts `int_pend` to indicate that the DMA interrupt is pending. The possible interrupt signals are `err_pend` and `DMA_tc`.
1	`err_pend`	Read	Error pending. When high, `err_pend` 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 `err_pend`: `mstr_abrt`, `tar_abrt`, and `det_par_err`.
2	`Int_irq`	Read	Interrupt request. When high, `int_irq` indicates that the local logic is requesting an interrupt, i.e., the `l_irqn` signal to the `pci_mt64` function is asserted.
3	`Dma_tc`	Read	DMA terminal count. When high, `DMA_tc` indicates that the DMA transfer is complete. The `DMA_tc` bit is reset with any of the following three conditions: ■ A read transaction to the ISR. ■ A write transaction to the CSR. ■ A write transaction to the ACR.
4	`ad_loaded`	Read	Address loaded. When high `ad_loaded` indicates that the address has been loaded in the ACR. This bit is cleared if any of the following conditions occur: ■ The DMA operation completes and the `DMA_tc` bit is set. ■ The flush bit is set. ■ The PCI bus is reset with `rstn` going low. ■ 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 ACR occurs; therefore, the ACR must be written last when setting up the DMA register.

Table 8. Interrupt Status Register Format (Part 2 of 2)

Data Bit	Mnemonic	Read/Write	Definition
5	`start_chain`	Read	Start chaining mode. This bit is used in DMA chaining mode only. When high, it indicates that the CSR has been loaded and the `chain_enable` bit of the CSR is set. This bit is cleared if one of the following conditions occurs: ■ The DMA operation completes and the `DMA_tc` bit is set. ■ The flush bit is set. ■ The PCI bus is reset with `rstn` going low. ■ `start_chain` triggers the beginning of the chaining DMA by setting the `DMA_on` bit in the CSR register.

Local Address Counter Register (LAR)

The local address counter 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.

Table 9. DMA Local Address Counter Register Format

Data Bit	Name	Read/Write	Definition
2..0	LAR	Write	Ground
24..3	LAR	Write	22-bit down counter
31..25	Unused	-	-

DMA Operation

This section describes the operation of the DMA in non-chaining and chaining mode.

Non-Chaining Mode

To operate the DMA in non-chaining mode, the DMA registers must be written in the following sequence:

1. Write the CSR register with the appropriate value (`chain_enable` bit = 0).

2. Write the LAR register with the SDRAM starting address of the transaction.

3. Write the BCR with the number of bytes to be transferred.

4. Write the ACR with the starting PCI address for the transaction.

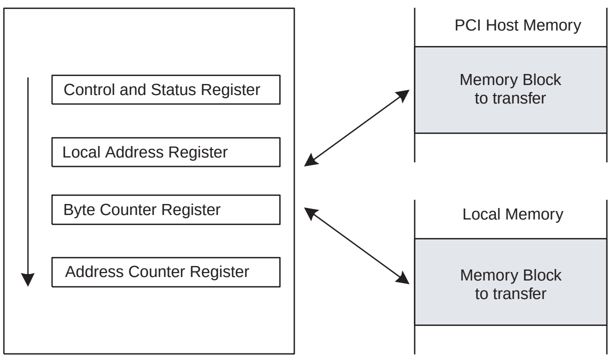
After the ACR is loaded with the PCI address, the `ad_loaded` bit in the ISR register is set and the DMA state machine is triggered. In the subsequent clock cycle, the `DMA_on` bit of the CSR register is set to indicate that the DMA transfer is in progress. The DMA state machine then sends the request 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 an SDRAM access through the SDRAM 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) will be 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 SDRAM control logic to return to the idle state before requesting a new DMA transaction to transfer the remaining data.

Figure 5 shows the register set up sequence for a non-chaining mode DMA operation.

Figure 5. Non-Chaining DMA Setup Sequence



Chaining Mode

To operate the DMA in the chaining mode, the descriptor FIFO buffer and the DMA registers must be written in the following sequence:

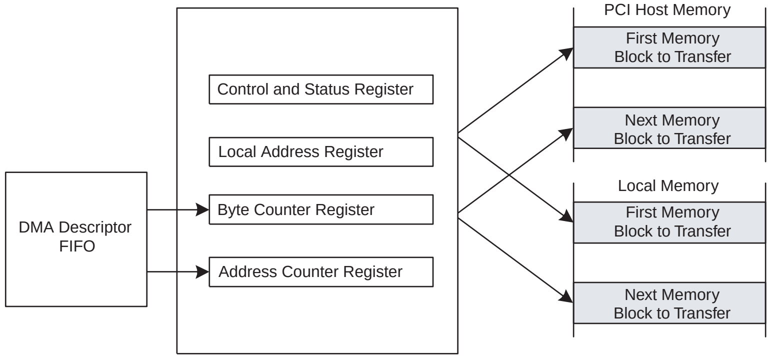
1. Target burst write to the DMA descriptor FIFO buffer with the series of DMA transfer information. Each DMA transfer information in the DMA descriptor FIFO buffer contains PCI address and byte count to be transferred.
2. Write the LAR with the SDRAM starting address of the transaction.
3. Write the CSR with the appropriate control data bits. (`chain_enable` bit = 1).

When the `chain_enable` bit is set, the `start_chain` bit of the ISR will be set to indicate that the DMA is in the chain operation mode. This action triggers the loading of the first set of DMA data from the descriptor FIFO buffer to the ACR and BCR. After the ACR and BCR are loaded with valid data, the DMA state machine requests a DMA transaction. The DMA then operates similar to the non-chaining mode.

After the first DMA transfer has completed and the next set of DMA data is loaded from the descriptor FIFO buffer into the ACR and BCR, a new DMA transfer takes place. This process is repeated until the DMA FIFO buffer becomes empty, i.e. the DMA chain has ended. 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.

Figure 6 shows the set up sequence for a chaining DMA mode operation.

Figure 6. Chaining 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.

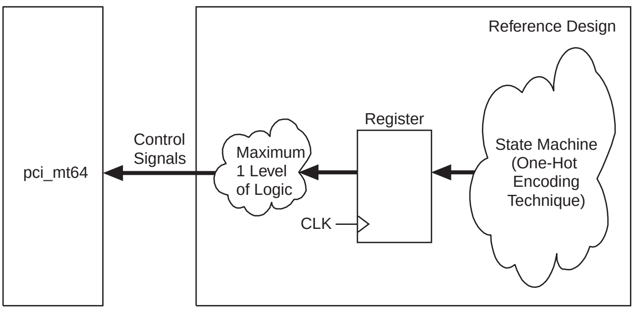
The interrupt request signal will be asserted until the appropriate bit causing the interrupt is cleared.

Designing for 66-Mhz Operation

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

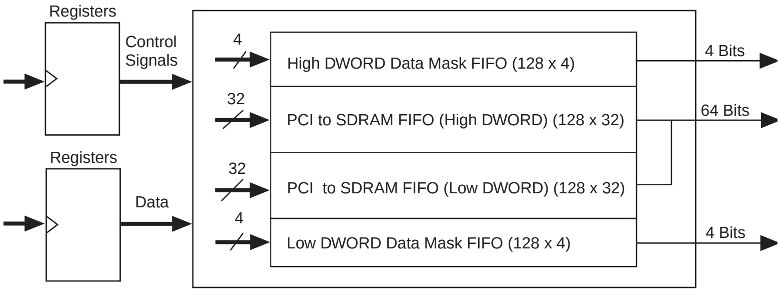
- All the input control signals to the PCI function must be registered with—at the most—one level of logic between the register and the function. See Figure 7.

Figure 7. 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 8.

Figure 8. Register Control Signals & Data



Design File Structures

This section explains the file structure and modules used in the reference design. Figure 9 shows the file structure.

Figure 9. Reference Design File Structure

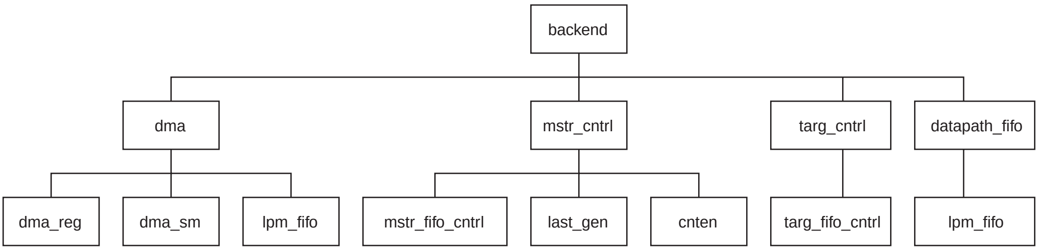


Table 10 describes the modules shown in Figure 9.

Table 10. Reference Design Sub-Module Description (Part 1 of 2)

Module	Description
`dma_reg`	Write and read control. This module includes all of the DMA registers such as CSR, BCR, ACR, ISR, and LAR.
`dma_sm`	DMA state machines. This module provides DMA control in chaining DMA and non-chaining DMA mode.
`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-SDRAM FIFO buffers write control, SDRAM-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.

Table 10. Reference Design Sub-Module Description (Part 2 of 2)

Module	Description
`cnten`	Count enable generation. This module provides the count enable signals for BCR, ACR, and LAR counters.
`mstr_cntrl`	Master control. This module drives all the master local signals of the `pci_mt64` 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-SDRAM FIFO buffers write control, SDRAM-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 `pci_mt64` function. It also connects to the SDRAM interface module to provide SDRAM 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 SDRAM. It also includes the data masking FIFO buffer.
`backend`	The top level of the reference design.

Table 10 shows the structure of the SDRAM interface.

Figure 10. SDRAM Interface File Structure

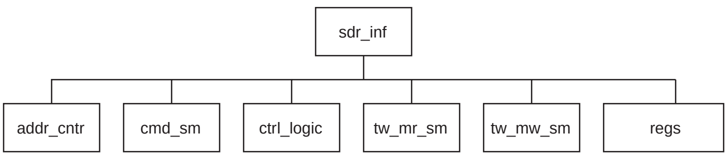


Table 11 describes the modules shown in Figure 10.

Table 11. SDRAM Interface Sub-Module Description

Module	Description
`addr_cntr`	Address counter. This module generates the address for the SDRAM controller to store data.
`cmd_sm`	Command state machine. This module generates commands to the SDRAM controller.
`ctrl_logic`	Control logic. This module controls the read and write of the data path FIFO buffer.
`tw_mr_sm`	Target write or master read state machine. This module provides control signals for the data to be written into the SDRAM (target write/master read).
`tr_mw_sm`	Target read or master write state machine. This module provides control signals for the data to be read from the SDRAM (target read/master write).
`regs`	This module outputs the data that has to be loaded into the address counter, which is output as the address to the SDRAM controller during the configuration or load mode register access.

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 SDR SDRAM MegaCore Function Beta User Guide*
- Micron Technology 64-Mb SDRAM data sheet

Revision History

The information contained in *Functional Specification 10: pci_mt64 MegaCore Function Reference Design* version 1.1 supersedes information published in previous versions.

Version 1.1

Functional Specification 10: pci_mt64 MegaCore Function Reference Design version 1.1 contains the following changes:

- Updated text throughout for 2.2.0 release of the PCI Compiler software.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
lit_req@altera.com