

Core Overview

The PCI Lite core is a protocol interface that translates PCI transactions to Avalon® Memory-Mapped (Avalon-MM) transactions with low latency and high throughput. The PCI Lite core uses the PCI-Avalon bridge to connect the PCI bus to the interconnect fabric, allowing you to easily create simple PCI systems that include one or more SOPC Builder components. This core has the following features:

- SOPC Builder ready
- PCI complexities, such as retry and disconnect are handled by the PCI/Avalon Bridge logic and transparent to the user
- Run-time configurable (dynamic) Avalon-to-PCI address translation
- Separate Avalon Memory-Mapped (Avalon-MM) slave ports for PCI bus access (PBA) and control register access (CRA)
- Support for Avalon-MM burst mode
- Common PCI and Avalon clock domains
- Option to increase PCI read performance by increasing the number of pending reads and maximum read burst size.

This chapter contains the following sections:

- “Performance and Resource Utilization”
- “Functional Description” on page 11–2
- “Instantiating the Core in SOPC Builder” on page 11–11
- “Device Support” on page 11–14
- “Simulation Considerations” on page 11–14

Performance and Resource Utilization

This section lists the resource utilization and performance data for supported devices when operating in the PCI Target-Only, and PCI Master/Target device modes for each of the application-specific performance settings.

The estimates are obtained by compiling the core using the Quartus® II software. Performance results vary depending on the parameters that you specify for the system module.

Table 11–1 shows the resource utilization and performance data for a Stratix® III device (EP3SE50F780C2). The performance of the MegaCore function in the Stratix IV family is similar to the Stratix III family.

Table 11-1. Memory Utilization and Performance Data for the Stratix III Family

PCI Device Mode	PCI Target	PCI Master	ALUTs (2)	Logic Register	M9K Memory Blocks	I/O Pins
Min (1)	Enabled	N/A	715	517	2	48
Max (1)	Enabled	Enabled	1,347	876	5	50

Notes to Table 11-1:

- (1) **Min** = One BAR with minimum settings for each parameter.
Max = Three BARs with maximum settings for each parameter.
- (2) The logic element (LE) count for the Stratix III family is based on the number of adaptive look-up tables (ALUTs) used for the design as reported by the Quartus II software.

Table 11-2 lists the resource utilization and performance data for a Cyclone III device (EP3C40F780C6).

Table 11-2. Memory Utilization and Performance Data for the Cyclone III Family

PCI Device Mode	PCI Target	PCI Master	Logic Elements	Logic Register	M4K Memory Blocks	I/O Pins
Min (1)	Enabled	N/A	1,057	511	2	48
Max (1)	Enabled	Enabled	2,027	878	5	50

Note to Table 11-2:

- (1) **Min** = One BAR with minimum settings for each parameter.
Max = Three BARs with maximum settings for each parameter.

Functional Description

The following sections provide a functional description of the PCI Lite Core.

PCI-Avalon Bridge Blocks

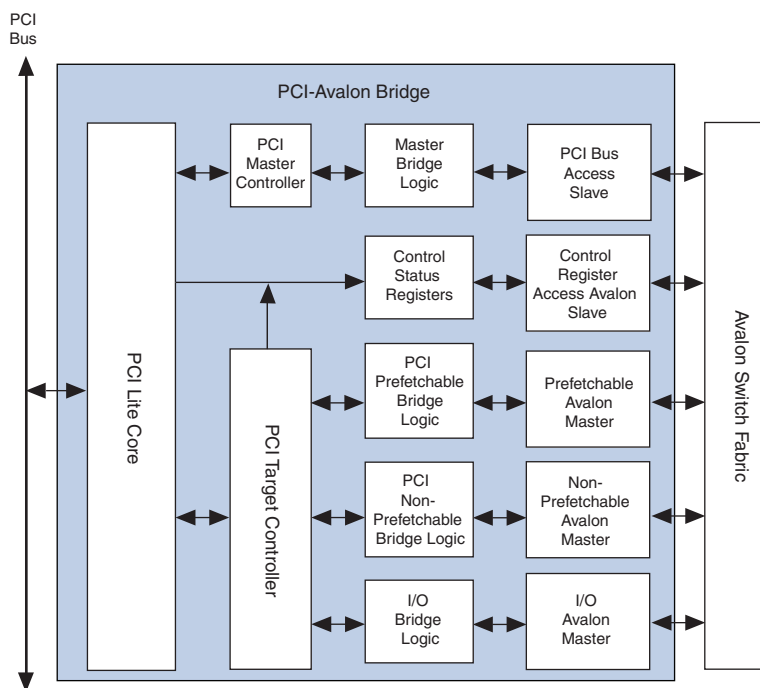
The PCI-Avalon bridge's blocks manage the connectivity for the following PCI operational modes:

- PCI Target-Only Peripheral
- PCI Master/Target Peripheral
- PCI Host-Bridge Device

Depending on the operational mode, the PCI-Avalon bridge uses some or all of the predefined Avalon-MM ports. Figure 11-1 shows a generic PCI-Avalon bridge block diagram, which includes the following blocks:

- Five predefined Avalon-MM ports
- Control registers
- PCI master controller (when applicable)
- PCI target controller

Figure 11-1. Generic PCI-Avalon Bridge Block Diagram



Avalon-MM Ports

The Avalon bridge comprises up to five predefined ports to communicate with the interconnect (depending on device operating mode).

This section discusses the five Avalon-MM ports:

- Prefetchable Avalon-MM master
- Non-Prefetchable Avalon-MM master
- I/O Avalon-MM master
- PCI bus access slave
- Control register access (CRA) Avalon-MM slave

Prefetchable Avalon-MM Master

The prefetchable Avalon-MM master port provides a high bandwidth PCI memory request access to Avalon-MM slave peripherals. This master port is capable of generating Avalon-MM burst transactions for PCI requests that hit a prefetchable base address register (BAR). You should only connect prefetchable Avalon-MM slaves to this port, typically RAM or ROM memory devices.

This port is optimized for high bandwidth transfers as a PCI target and it does not support single cycle transactions.

Non-Prefetchable Avalon-MM Master

The Non-Prefetchable Avalon-MM master port provides a low latency PCI memory request access to Avalon-MM slave peripherals. Burst operations are not supported on this master port. Only the exact amount of data needed to service the initial data phase is read from the interconnect. Therefore, the PCI byte enables (for the first data phase of the PCI read transaction) are passed directly to the interconnect.

This Avalon-MM master port is optimized for low latency access from PCI-to-Avalon-MM slaves. This is optimal for providing PCI target access to simple Avalon-MM peripherals.

I/O Avalon-MM Master

The I/O Avalon-MM master port provides a low latency PCI I/O request access to Avalon-MM slave peripherals. Burst operations are not supported on this master port. As only the exact amount of data needed to service the initial data phase is read from the interconnect, the PCI byte enables (for the first data phase of the PCI read transaction) are passed directly to the interconnect.

This Avalon-MM master port is also optimized for I/O access from PCI-to-Avalon-MM slaves for providing PCI target access to simple Avalon-MM peripherals.

PCI Bus Access Slave

This Avalon-MM slave port propagates the following transactions from the interconnect to the PCI bus:

- Single cycle memory read and write requests
- Burst memory read and write requests
- I/O read and write requests
- Configuration read and write requests

Burst requests from the interconnect are the only way to create burst transactions on the PCI bus.

This slave port is not implemented in the PCI Target-Only Peripheral mode.

Control Register Access (CRA) Avalon-MM Slave

This Avalon-MM slave port is used to access control registers in the PCI-Avalon bridge. To provide external PCI master access to these registers, one of the bridge's master ports must be connected to this port. There is no internal access inside the bridge from the PCI bus to these registers. You can only write to these registers from the interconnect. The Control Register Access Avalon Slave port is only enabled on Master/Target selection. The range of values supported by PCI CRA is 0x1000 to 0x1FFF. Depending on the system design, these values can be accessed by PCI processors, Avalon processors or both.

Table 11-3 on page 11-5 shows the instructions on how to use these values. The address translation table is writable via the Control Register Access Avalon Slave port. If the **Number of Address Pages** field is set to the maximum of **512**, 0x1FF8 contains A2P_ADDR_MAP_LO511 and 0x1FFC contains A2P_ADDR_MAP_HI511.

Each entry in the PCI address translation table is always 8 bytes wide. The lower order address bits that are treated as a pass through between Avalon-MM and PCI, and the number of pass-through bits, are defined by the size of page in the address translation table and are always forced to 0 in the hardware table. For example, if the page size is 4 KBytes, the number of pass-through bits is $\log_2(\text{page size}) = \log_2(4 \text{ KBytes}) = 12$.

Refer to “Avalon-to-PCI Address Translation” on page 11-6 for more details.

Table 11-3. Avalon-to-PCI Address Translation Table – Address Range: 0x1000-0x1FFF


Address	Bit	Name	Access Mode	Description
0x1000	1:0	A2P_ADDR_SPACE0	W	Address space indication for entry 0. See Table 11-4 on page 11-7 for the definition of these bits.
	31:2	A2P_ADDR_MAP_LO0	W	Lower bits of Avalon-to-PCI address map entry 0. The pass through bits are not writable and are forced to 0.
0x1004	31:0	A2P_ADDR_MAP_HI0	W	Reserved.
0x1008	1:0	A2P_ADDR_SPACE1	W	Address Space indication for entry 1. See Table 11-4 on page 11-7 for the definition of these bits.
	31:2	A2P_ADDR_MAP_LO1	W	Lower bits of Avalon-to-PCI address map entry 1. Pass through bits are not writable and are forced to 0. This entry is only implemented if the number of pages in the address translation table is greater than 1.
0x100C	31:0	A2P_ADDR_MAP_HI1	W	Reserved.

Master and Target Performance

The performance of the PCI Lite core is designed to provide low-latency single-cycle and burst transactions.

Master Performance

The master provides high throughput for transactions initiated by Avalon-MM master devices to PCI target devices via the PCI bus master interface. Avalon-MM read transactions are implemented as latent read transfers. The PCI master device issues only one read transaction at a time.

 The PCI bus access (PBA) handles the Avalon master transaction system interconnect hold state for 6 clock cycles. This is the maximum number of cycles supported by the PCI specification.

Target Performance

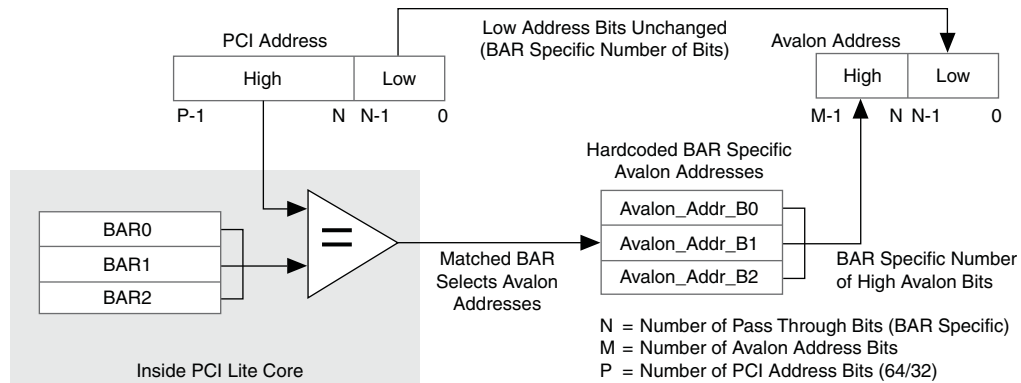
The target allows high throughput read/write operations to Avalon-MM slave peripherals. Read/write accesses to prefetchable base address registers (BARs) use dual-port buffers to enable burst transactions on both the PCI and Avalon-MM sides. This profile also allows access to the PCI BARs (Prefetchable, Non-Prefetchable, and I/O) to use their respective Avalon-MM master ports to initiate transfers to Avalon-MM slave peripherals. Prefetchable handles burst transaction and Non-Prefetchable and I/O handles only single-cycle transaction.

All PCI read transactions are completed as delayed reads. However, only one delayed read is accepted and processed at a time.

PCI-to-Avalon Address Translation

Figure 11-2 shows the PCI-to-Avalon address translation. The bits in the PCI address that are used in the BAR matching process are replaced by an Avalon-MM base address that is specific to that BAR.

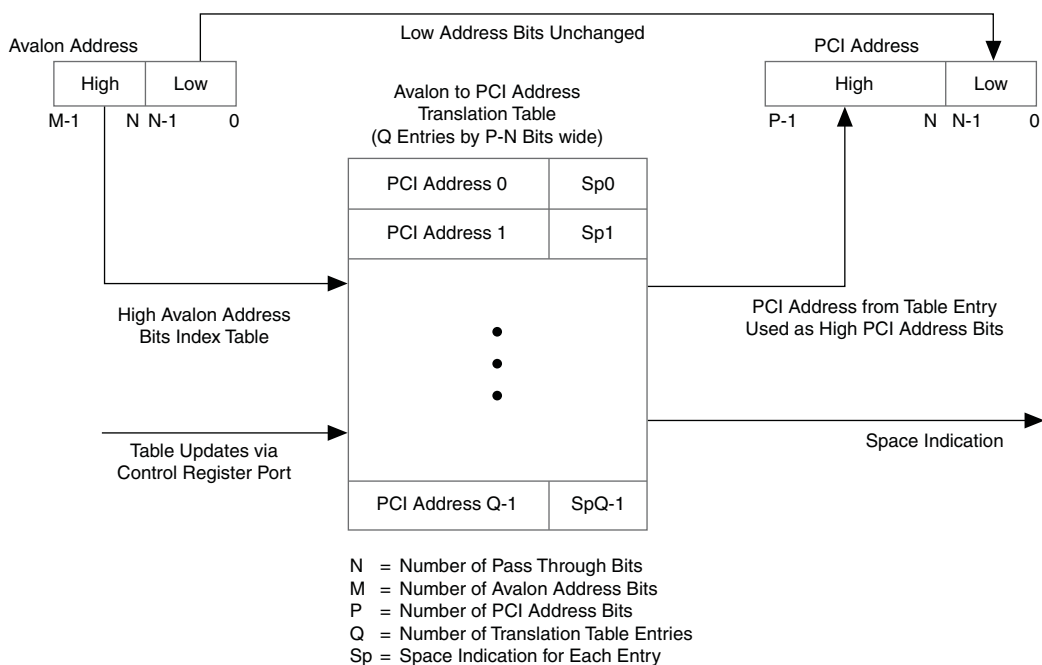
Figure 11-2. PCI-to-Avalon Address Translation



Avalon-to-PCI Address Translation

Avalon-to-PCI address translation is done through a translation table. Low order Avalon-MM address bits are passed to PCI unchanged; higher order Avalon-MM address bits are used to index into the address translation table. The value found in the table entry is used as the higher order PCI address bits. Figure 11-3 depicts this process.

Figure 11-3. Avalon-to-PCI Address Translation



The address size selections in the translation table determine both the number of entries in the Avalon-to-PCI address translation table, and the number of bits that are passed through the transaction table unchanged.

Each entry in the address translation table also has two address space indication bits, which specify the type of address space being mapped. If the type of address space being mapped is memory, the bits also indicate the resulting PCI address is a 32-bit address.

Table 11-4 shows the address space field's format of the address translation table entries.

Table 11-4. Address Space Bit Encodings

Address Space Indicator (Bits 1:0)	Description
00	Memory space, 32-bit PCI address. Address bits 63:32 of the translation table entries are ignored.
01	Reserved.
10	I/O space. The address from the translation table process is modified as described in Table 11-5.
11	Configuration space. The address from the translation table process is treated as a type 1 configuration address and is modified as described in Table 11-5.

If the space indication bits specify configuration or I/O space, subsequent modifications to the PCI address are performed. See [Table 11-5](#).

Table 11-5. Configuration and I/O Space Address Modifications

Address Space	Modifications Performed
I/O	<ul style="list-style-type: none"> Address bits 2:0 are set to point to the first enabled byte according to the Avalon byte enables. (Bit 2 only needs to be modified when a 64-bit data path is in use.) Address bits 31:3 are handled normally.
Configuration address bits 23:16 == 0 (bus number == 0)	<ul style="list-style-type: none"> Address bits 1:0 are set to 00 to indicate a type 0 configuration request. Address bits 10:2 are passed through as normal. Address bits 31:11 are set to be a one-hot encoding of the device number field (15:11) of the address from the translation table. For example, if the device number is 0x00, address bit 11 is set to 1 and bits 31:12 are set to 0. If the device number is 0x01, address bit 12 is set to 1 and bits 31:13, 11 are set to 0. Address bits 31:24 of the original PCI address are ignored.
Configuration address bits 23:16 > 0 (bus number > 0)	<ul style="list-style-type: none"> Address bits 1:0 are set to 01 to indicate a type 1 configuration request. Address bits 31:2 are passed through unchanged.

Avalon-To-PCI Read and Write Operation

The PCI Bus Access Slave port is a burst-capable slave that attempts to create PCI bursts that match the bursts requested from the interconnect.

The PCI-Avalon bridge is capable of handling bursts up to 512 bytes with a 32-bit PCI bus. In other words, the maximum supported Avalon-MM burst count is 128.

Bursts from Avalon-MM can be received on any boundary. However, when internal PCI-Avalon bridge bursts cross the Avalon-to-PCI address page boundary, they are broken into two pieces. Two bursts are used because the address translation can change at that boundary, requiring a different PCI address for the second portion of the burst with a burst count greater than 1.



Avalon-MM burst read requests are treated as if they are going to prefetchable PCI space. Therefore, if the PCI target space is non-prefetchable, you should not use read bursts.

Several factors control how Avalon-MM transactions (bursts or single cycle) are translated to PCI transactions. These cases are discussed in [Table 11-6](#).

Table 11-6. Translation of Avalon Requests to PCI Requests

Data Path Width	Avalon Burst Count	Type of Operation	Avalon Byte Enables	Resulting PCI Operation and Byte Enables
32	1	Read or Write	Any value	Single data phase read or write, PCI byte enables identical to Avalon byte enables
32	>1	Read	Any value	Attempt to burst on PCI. All data phases have all PCI bytes enabled.
32	>1	Write	Any value	Attempt to burst on PCI. All data phases have PCI byte enables identical to the Avalon byte enables.

Avalon-to-PCI Write Requests

For write requests from the interconnect, the write request is pushed onto the PCI bus as a configuration write, I/O write, or memory write. When the Avalon-to-PCI command/write data buffer either has enough data to complete the full burst or 8 data phases (32 bytes on a 32-bit PCI bus) are exceeded, the PCI master controller issues the PCI write transaction.

The PCI write is issued to configuration, I/O, or memory space based on the address translation table. See [“Avalon-to-PCI Address Translation” on page 11-6](#).

A PCI write burst can be terminated for various reasons. [Table 11-7](#) describes the resulting action for the PCI master write request termination condition.

Table 11-7. PCI Master Write Request Termination Conditions

Termination condition	Resulting Action
Burst count satisfied	Normal master-initiated termination on PCI bus, command completes, and the master controller proceeds to the next command.
Latency timer expiring during configuration, I/O, or memory write command	Normal master-initiated termination on PCI bus, the continuation of the PCI write is requested from the master controller arbiter.
Avalon-to-PCI command/write data buffer running out of data	Normal master-initiated termination on the PCI bus. Master controller waits for the buffer to reach 8 <code>DWORDS</code> on a 32-bit PCI or 16 <code>DWORDS</code> on a 64-bit PCI, or there is enough data to complete the remaining burst count. Once enough data is available, the master controller arbiter continues with the PCI write.
PCI target disconnect	The master controller arbiter attempts to initiate the PCI write until the transaction is successful.
PCI target retry	
PCI target-abort	The rest of the write data is read from the buffer and discarded.
PCI master-abort	

Avalon-to-PCI Read Requests

For read requests from the interconnect, the request is pushed on the PCI bus by a configuration read, I/O read, memory read, memory read line, or memory read multiple command. The PCI read is issued to configuration, I/O, or memory space based on the address translation table entry. See [“Avalon-to-PCI Address Translation” on page 11-6](#).

If a memory space read request can be completed in a single data phase, it is issued as a memory read command. If the memory space read request spans more than one data phase but does not cross a cacheline boundary (as defined by the cacheline size register), it is issued as a memory read line command. If the memory space read request crosses a cache line boundary, it is issued as multiple memory read commands.

Read requests on PCI may initially be retried. Retries depend on the response time from the target. The master continues to retry until it gets the required data.

Table 11-8 shows PCI master read request termination conditions.

Table 11-8. PCI Master Read Request Termination Conditions

Termination Condition	Resulting Action
Burst count satisfied	Normal master initiated termination on the PCI bus. Master controller proceeds to the next command.
Latency timer expired	Normal master initiated termination on PCI bus. The continuation of the PCI read is made pending as a request from the master controller arbiter.
PCI target disconnect	The continuation of the PCI read is requested from the master controller arbiter.
PCI target retry	
PCI target-abort	Dummy data is returned to complete the Avalon-MM read request. The next operation is then attempted in a normal fashion.
PCI master-abort	

Ordering of Requests

The PCI-Avalon bridge handles the following types of requests:

- PMW—Posted memory write.
- DRR—Delayed read request.
- DWR—Delayed write request. DWRs are I/O or configuration write operation requests. The PCI-Avalon bridge does not handle DWRs as delayed writes.
 - As a PCI master, I/O or configuration writes are generated from posted Avalon-MM writes. If required to verify completion, you must issue a subsequent read request to the same target.
 - As a PCI target, configuration writes are the only requests accepted, which are never delayed. These requests are handled directly by the PCI core.
- DRC—Delayed read completion.
- DWC—Delayed write completion. These are never passed through to the core in either direction. Incoming configuration writes are never delayed. Delayed write completion status is not passed back at all.

Every single transaction that is initiated, locks the core until it is completed. Only then can a new transaction be accepted.

PCI Interrupt

When Avalon-MM asserts the `IRQ` signal, an interrupt on the PCI bus occurs. The Avalon-MM `IRQ` input causes a bit to be set in the PCI interrupt status register.

Instantiating the Core in SOPC Builder

Table 11-9 describes the parameters that can be configured in SOPC Builder for the PCI Lite core.

Table 11-9. Parameters for PCI Lite Core (Part 1 of 2)

Parameters	Legal Values	Description
Enable Master/Target Mode	On or Off	Turning this option On enables Master/Target mode. This option enables allows Avalon-MM master devices to access PCI target devices via the PCI bus master interface, and PCI bus master devices to access Avalon-MM slave devices via the PCI bus target interface. Turning this option Off means you have selected Target Only mode, which allows PCI bus mastering devices to access Avalon-MM slave devices via the PCI bus target interface.
Enable Host Bridge Mode	On or Off	Turning this option On enables this mode. In addition to the same features provided by the PCI Master/Target mode, Host Bridge Mode provides host bridge functionality including hardwiring the master enable bit to 1 in the PCI command register and allowing self-configuration. This value can only be set if the Enable Master/Target Mode option is turned On .
Number of Address Pages	2, 4, 8, or 16	The number of translation/pages supported by the device for Avalon to PCI address translation.
Size of Address Pages	12-27	The supported address size (in bits) that can be assigned to each map number entries.
Prefetchable BAR	On or Off	Turning this option On invokes a Prefetchable Master (PM) Bar in the PCI system. This option allows PCI-Avalon Bridge Lite to accept and process PM transactions.
Prefetchable BAR Size	10-31	The allowed reserved address range supported by the PM BAR. The reserved memory space is 1 KByte (10 bits) to 4 GBytes (31 bits).
Prefetchable BAR Avalon Address Translation Offset	<BAR translation value>	The direct translation of the value that hits the BAR and modified to a fixed address in the Avalon space. Refer to “ PCI-to-Avalon Address Translation ” on page 11-6.
Non-Prefetchable BAR	On or Off	Turning this option On invokes a Non-Prefetchable Master (NPM) Bar in the PCI system. This option allows the PCI-Avalon Bridge Lite to accept and process NPM transactions.
Non-Prefetchable BAR Size	10-31	Specifies the allowed reserved address range supported by the NPM BAR. The reserved memory space is 1 KByte (10 bits) to 4 GBytes (31 bits).
Non-Prefetchable BAR Avalon Address Translation Offset	<BAR translation value>	The direct translation of the value that hits the BAR and modified to a fixed address in the Avalon space. Refer to “ PCI-to-Avalon Address Translation ” on page 11-6.
I/O BAR	On or Off	Turning this option On enables an I/O BAR in the system. This option allows PCI-Avalon Bridge Lite to accept and process I/O type transactions.
I/O BAR Size	2-8	The allowed reserved address range supported by the I/O BAR. The reserved memory space is 4 bytes (2 bits) to 256 bytes (8 bits).
I/O BAR Avalon Address Translation Offset	<BAR translation value>	The direct translation of the value that hits the BAR and modified to a fixed address in the Avalon address space. Refer to “ PCI-to-Avalon Address Translation ” on page 11-6.

Table 11-9. Parameters for PCI Lite Core (Part 2 of 2)

Parameters	Legal Values	Description
Maximum Target Read Burst Size	1, 2, 4, 8, 16, 32, 64, or 128	Specifies the maximum FIFO depth that is used for reading. Larger values allow more reads to be read in a single transaction but also require more time to clear the FIFO content.
Device ID	<register value>	Device ID register. This parameter is a 16-bit hexadecimal value that sets the device ID register in the configuration space.
Vendor ID	<register value>	Vendor ID register. This parameter is a 16-bit read-only register that identifies the manufacturer of the device. The value of this register is assigned by the PCI Special Interest Group (SIG).
Class Code	<register value>	Class code register. This parameter is a 24-bit hexadecimal value that sets the class code register in the configuration space. The value entered for this parameter must be valid PCI SIG-assigned class code register value.
Revision ID	<register value>	Revision ID register. This parameter is an 8-bit read-only register that identifies the revision number of the device. The value of this register is assigned by the manufacturer.
Subsystem ID	<register value>	Subsystem ID register. This parameter is a 16-bit hexadecimal value that sets the subsystem ID register in the PCI configuration space. Any value can be entered for this parameter.
Subsystem Vendor ID	<register value>	Subsystem vendor ID register. This parameter is a 16-bit hexadecimal value that sets the subsystem vendor ID register in the PCI configuration space. The value for this parameter must be a valid PCI SIG-assigned vendor ID number.
Maximum Latency	<register value>	Maximum latency register. This parameter is an 8-bit hexadecimal value that sets the maximum latency register in the configuration space. This parameter must be set according to the guidelines in the PCI specifications. Only meaningful when the Enable Master/Target Mode option is turned On .
Minimum Grant	<register value>	Minimum grant register. This parameter is an 8-bit hexadecimal value that sets the minimum grant register in the PCI configuration space. This parameter must be set according to the guidelines in the PCI specifications. Only meaningful when the Enable Master/Target Mode option is turned On .

PCI Timing Constraint Files

The PCI Lite core supplies a Tcl timing constraint file for your target device family.


When run, the constraint file automatically sets the PCI Lite core assignments for your design such as PCI Lite core hierarchy, device family, density and package type used in your Quartus II project.

To run a PCI constraint file, perform the following steps:

1. Copy `pci_constraints.tcl` from
`<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite.`

2. Update the pin list in the Tcl constraint file. Edit the `get_user_pin_name` procedure in the Tcl constraint file to match the default pin names. To edit the PCI constraint file, follow these steps:
 - a. Locate the `get_user_pin_name` procedure. This procedure maps the default PCI pin names to user PCI pin names. The following lines are the first few lines of the procedure:

```
proc get_user_pin_name { internal_pin_name } {
    #----- Do NOT change ----- Change -----
    array set map_user_pin_name_to_internal_pin_name {ad          ad          }
    b. Edit the pin names under the Change header in the file to match the PCI pin
       names used in your Quartus II project. In the following example, the name ad
       is changed to pci_ad:
    #----- Do NOT change ----- Change -----
    array set map_user_pin_name_to_internal_pin_name { ad          pci_ad      }
```

 The Tcl constraint file uses the default PCI pin names to make assignments. When overwriting existing assignments, the Tcl constraint file checks the new assignment pin names against the default PCI pin names. You must update the assignment pin names if there is a mismatch between the assignment pin names and the default PCI pin names.

3. Source the constraint file by typing the following in the Quartus II Tcl Console window:


```
source pci_constraints.tcl ←
```

4. Add the PCI constraints to your project by typing the following command in the Quartus II Tcl Console window:

```
add_pci_constraints ←
```

See [“Additional Tcl Option” on page 11-13](#) for the option supported by the `add_pci_constraints` command.

When you add the timing constraints file as described in Step 4 above, the Quartus II software generates a Synopsys Design Constraints (`.sdc`) file with the file name format, `<variation name>.sdc`. The Quartus II TimeQuest timing analyzer uses the constraints specified in this file.

 For more information about `.sdc` files or TimeQuest timing analyzer, refer to Quartus II Help.

Additional Tcl Option

If you do not want to compile your project and prefer to skip analysis and synthesis, you can use the `-no_compile` option:

```
add_pci_constraints [-no_compile]
```

By default, the `add_pci_constraints` command performs analysis and synthesis in the Quartus II software to determine the hierarchy of your PCI Lite core design. You should only use this option if you have already performed analysis and synthesis or fully compiled your project prior to using this script.

Device Support

The PCI Lite core supports the Arria® GX, Arria II, Cyclone® III, Hardcopy® II, Stratix® III, and Stratix IV device families.

Simulation Considerations

The PCI Lite core includes a testbench that facilitates the design and verification of systems that implement the Altera PCI-Avalon bridge. The testbench only works for master systems and is provided in Verilog HDL only.

To use the PCI testbench, you must have a basic understanding of PCI bus architecture and operations. This section describes the features and applications of the PCI testbench to help you successfully design and verify your design.

Features

The PCI testbench includes the following features:

- Easy to use simulation environment for any standard Verilog HDL simulator
- Open source Verilog HDL files
- Flexible PCI bus functional model to verify your application that uses any PCI Lite core
- Simulates all basic PCI transactions including memory read/write operations, I/O read/write transactions, and configuration read/write transactions
- Simulates all abnormal PCI transaction terminations including target retry, target disconnect, target abort, and master abort
- Simulates PCI bus parking

Master Transactor (mstr_tranx)

The master transactor simulates the master behavior on the PCI bus. It serves as an initiator of PCI transactions for PCI testbench. The master transactor has three main sections:

- TASKS (Verilog HDL)
- INITIALIZATION
- USER COMMANDS

TASKS Sections

The TASKS (Verilog HDL) sections define the events that are executed for the user commands supported by the master transactor. The events written in the TASKS sections follow the phases of a standard PCI transaction as defined by the *PCI Local Bus Specification, Revision 3.0*, including:

- Address phase
- Turn-around phase (read transactions)
- Data phases
- Turn-around phase

The master transactor terminates the PCI transactions in the following cases:

- The PCI transaction has successfully transferred all the intended data.
- The PCI target terminates the transaction prematurely with a target retry, disconnect, or abort as defined in the *PCI Local Bus Specification, Revision 3.0*.
- A target does not claim the transaction resulting in a master abort.

The bus monitor informs the master transactor of a successful data transaction or a target termination. Refer to the source code, which shows you how the master transactor uses these termination signals from the bus monitor.

The PCI testbench master transactor TASKS sections implement basic PCI transaction functionality. If your application requires different functionality, modify the events to change the behavior of the master transactor. Additionally, you can create new procedures or tasks in the master transactor by using the existing events as an example.

INITIALIZATION Section

This user-defined section defines the parameters and reset length of your PCI bus on power-up. Specifically, the system should reset the bus and write the configuration space of the PCI agents. You can modify the master transactor INITIALIZATION section to match your system requirements by changing the time that the system reset is asserted and by modifying the data written in the configuration space of the PCI agents.

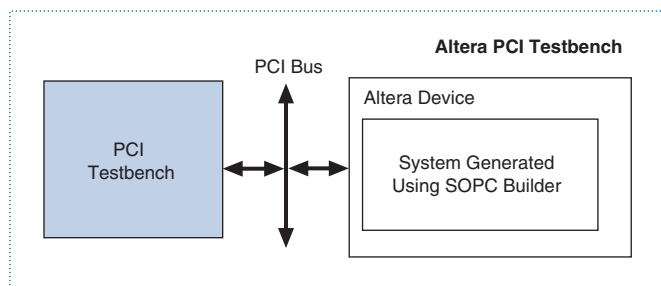
USER COMMANDS Section

The master transactor USER COMMANDS section contains the commands that initiate the PCI transactions you want to run for your tests. The list of events that are executed by these commands is defined in the TASKS sections. Customize the USER COMMANDS section to execute the sequence of commands needed to test your design.

Simulation Flow

This section describes the simulation flow using Altera PCI testbench. [Figure 11-4](#) shows the block diagram of a typical verification environment using the PCI testbench.

Figure 11-4. Typical Verification Environment Using the PCI Testbench



The simulation flow using Altera PCI testbench comprises the following steps.

1. Use SOPC Builder to create your system. SOPC creates the *<variation name_system>_sim* folder in your project directory.
2. Source **pci_constraints.tcl**.
3. Copy **<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/verilog/pci_lite/trgt_tranx_mem_init.dat** to **<project_directory>/<variation name_system>_sim** folder.
4. Edit the top level HDL verilog files in the testbench. Insert the following lines just before `module test_bench`.

```
`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/pci_tb.v"

`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/clk_gen.v"

`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/arbiter.v"

`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/pull_up.v"

`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/monitor.v"

`include
"<quartus_root>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/
verilog/pci_lite/trgt_tranx.v"

`include "mstr_tranx.v"
```



Modify **mstr_tranx.v** in your project directory to add the PCI transactions to your system. If you regenerate your system, SOPC Builder overwrites the testbench files in the **<sopc_system>_sim** directory. If you want the default testbench files, regenerate the system. Then resource **pci_constraints.tcl** or simply copy the **mstr_tranx.v** from **<quartus_ip>/ip/sopc_builder_ip/altera_avalon_pci_lite/pci_sim/verilog/pci_lite** into your project folder and repeat steps 3 and 4.

5. Set the initialization parameters, which are defined in the master transactor model source code. These parameters control the address space reserved by the target transactor model and other PCI agents on the PCI bus.
6. The master transactor defines the tasks (Verilog HDL) needed to initiate PCI transactions in your testbench. Add the commands that correspond to the transactions you want to implement in your tests to the master transactor model source code. At a minimum, you must add configuration commands to set the BAR for the target transactor model and write the configuration space of the PCI Lite core. Additionally, you can add commands to initiate memory or I/O transactions to the PCI Lite core.

7. Compile the files in your simulator, including the testbench modules and the files created by SOPC Builder.
8. Simulate the testbench for the desired time period.

Referenced Documents

This chapter references *Avalon Interface Specifications*.

Document Revision History

Table 11-10 shows the revision history for this chapter.

Table 11-10. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	No change from previous release.	—
March 2009 v9.0.0	No change from previous release.	—
November 2008 v8.1.0	Changed to 8-1/2 x 11 page size. Edited the command errors in the Simulation Flow section.	—
May 2008 v8.0.0	Initial release.	—

