HB0089
Handbook
CorePCIF_AHB v4.3

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 11.0

Added PolarFire® SoC support.

1.2 Revision 10.0

Updated changes related to CorePCIF_AHB v4.2.

1.3 Revision 9.0

Updated changes related to CorePCIF_AHB v4.1.

1.4 Revision 8.0

Updated changes related to CorePCIF_AHB v4.0.

1.5 Revision 7.0

Updated changes related to CorePCIF_AHB v3.7.

1.6 Revision 6.0

Updated changes related to CorePCIF_AHB v3.6.

1.7 Revision 5.0

Updated changes related to CorePCIF_AHB v3.5.

1.8 Revision 4.0

Updated changes related to CorePCIF_AHB v3.4.

1.9 Revision 3.0

Updated changes related to CorePCIF_AHB v3.3.

1.10 Revision 2.0

Updated changes related to CorePCIF_AHB v3.2.

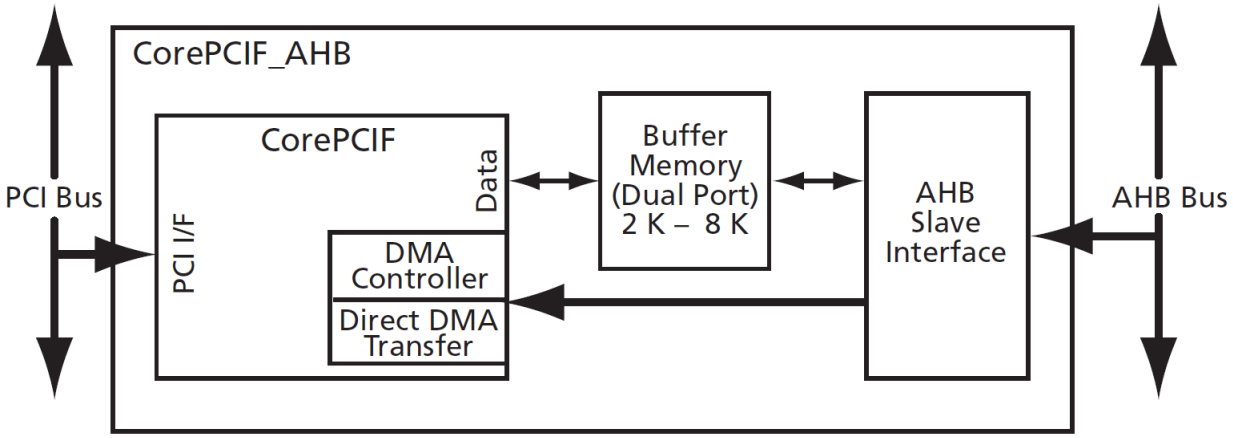
1.11 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CorePCIF_AHB v3.1.

2 Introduction

CorePCIF_AHB allows an AHB bus system to be connected to a PCI bus. It allows for both the AHB bus and PCI bus to initiate data transfers between the two buses. CorePCIF_AHB is built on top of the CorePCIF core. An AHB Slave interface and memory buffer are added as shown in Figure 1.

Figure 1 • **CorePCIF_AHB Architecture**



The AHB slave interface allows the AHB bus to access the buffer memory and the DMA controller included in CorePCIF. The core can be configured in several ways, allowing different system architectures to be supported.

Note: CorePCIF_AHB supports only 32-bit operation.

2.1 PCI Target with Memory Buffer (PTB)

In this configuration, the core allows both the AHB and PCI buses to read and write the buffer memory.

2.2 PCI Target and Master with Memory Buffer (PTMBP)

In this configuration, the core allows both the AHB and PCI buses to read and write the buffer memory. A DMA engine is also included that allows the CorePCIF controller to initiate PCI transfers moving data between the buffer memory and PCI bus. The DMA engine can only be configured from the PCI. There is no access to the DMA controller from the AHB interface.

2.3 PCI Target and Master with Memory Buffer (PTMBPA)

In this configuration, the core allows both the AHB and PCI buses to read and write the buffer memory. A DMA engine is also included that allows the CorePCIF controller to initiate PCI transfers moving data between the buffer memory and PCI bus. The DMA engine can be configured from either the PCI or AHB buses.

2.4 PCI Master and Direct DMA (PMDA)

In this configuration, the memory buffer is not included, and the PCI core is configured as a master only with Direct DMA operation enabled. This allows the AHB bus to initiate single-cycle PCI read and write operations on the PCI bus.

2.5 PCI Master with Memory Buffer and Direct DMA (PMBDA)

In this configuration, the core allows the AHB bus to read and write the buffer memory. A DMA engine is also included that allows the CorePCIF controller to initiate PCI transfers moving data between the buffer memory and PCI bus. The DMA engine can be started only from the AHB bus. Additionally, the Direct DMA mode is available, allowing the AHB bus to directly initiate single-cycle PCI read and write operations without using the memory buffer.

2.6 PCI Target and DMA Master with Memory Buffer and Direct DMA (PTMBDPA)

In this configuration, the core allows both the AHB and PCI buses to read and write the buffer memory. A DMA engine is also included that allows the CorePCIF controller to initiate PCI transfers moving data between the buffer memory and PCI bus. The DMA engine can be started from either the PCI or AHB buses. Additionally, the Direct DMA mode is available, allowing the AHB bus to directly initiate single cycle PCI read and write operations without using the memory buffer.

The following naming scheme is used above: PTMBDPA, where

- T = PCI Target operation enabled
- M = PCI Master operation enabled
- B = Memory Buffer enabled
- D = Direct DMA enabled
- P = DMA engine can be started from PCI interface
- A = DMA engine can be started from AHB interface

2.7 Supported Device Families

- PolarFire® SoC
- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2
- SmartFusion
- Fusion®
- IGLOO®
- IGLOOe
- ProASIC®3L
- ProASIC3
- ProASIC3E

2.8 Core Versions

This handbook applies to CorePCIF_AHB v4.3. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

2.9 CorePCIF_AHB Device Requirements

2.9.1 Utilization Statistics

Table 1 gives the CorePCIF_AHB utilization. The numbers in these tables are typical and will vary based on the synthesis tools used. Additionally, device utilization and performance varies, depending on how additional features are configured.

Table 1 • **Device Utilization Information all Families**

		Cell or Tiles						Frequency (MHz)	
Family	**Configuration**	**Comb.**	**Seq.**	**Total**	**Memory Block**	**Device**	**Utilization (%)**	**PCI**	**HCLK**
Fusion	PTB	618	174	817	4	AFS600	9	53	100
IGLOO	PTMBP	1503	389	1918	6	AGL600	21	44	100
IGLOOe	PTMBPA	1687	446	2163	6	AGLE600	23	44	100
ProASIC3L ProASIC3	PMDA	1158	295	1479	0	A3P600 A3PE600	16	53	100
ProASIC3E	PMBDA	1350	376	1756	6	A3PL600	19	46	100
SmartFusion	PTMBDPA	1753	572	2388	6	A2F500M3G	25	41	100

Table 2 • **CorePCIF_AHB Device Utilization and Performance for SmartFusion2 and IGLOO2**

Family	**Configuration**	**Logic Elements**			**Memory Block (RAM1K18)**	**Total %**	**Performance (MHz)**	
		4LUT	**DFF**	**Total**			**PCI**	**HCLK**
SmartFusion2 (M2S050) / IGLOO2 (M2GL050)	PTB	858	483	1341	4	1.21	135	305
	PTMBA	1247	642	1889	5	1.67	125	295
	PTMBPA	1427	698	2125	5	1.88	109	271
	PMDA	771	417	1188	0	1.05	153	286
	PMBDA	1116	619	1735	5	1.54	165	274
	PTMBDPA	1441	824	2265	5	2.01	114	268

Note: The data in Table 2 was achieved using Verilog RTL with typical synthesis and layout settings. Frequency (in MHz) was set to 100 and speed grade was STD. The data given in Table 2 is indicative only. The overall device utilization and performance of the core is system dependent.

Table 3 • **CorePCIF_AHB Device Utilization and Performance for RTG4**

Family	Configuration	Logic Elements			Memory Block (RAM1K18)	Total %	Performance (MHz)	
		4LUT	DFF	Total			PCI	HCLK
RTG4 (RT4G150)	PTB	853	475	1328	4	0.43	119	248
	PTMBA	1252	642	1894	5	0.62	103	243
	PTMBPA	1478	698	2176	5	0.71	100	222
	PMDA	778	417	1195	0	0.39	130	236
	PMBDA	1127	616	1743	5	0.57	125	232
	PTMBDPA	1432	824	2274	5	0.74	104	223

Note: The data in Table 3 was achieved using Verilog RTL with typical synthesis and layout settings. Frequency (in MHz) was set to 100 and speed grade was STD. The data given in Table 3 is indicative only. The overall device utilization and performance of the core is system dependent.

Table 4 • **CorePCIF_AHB Device Utilization and Performance for PolarFire**

Family	Configuration	Logic Elements			Memory Block (RAM1K18)	Total %	Performance (MHz)	
		4LUT	DFF	Total			PCI	HCLK
PolarFire (MPF300T_ES)	PTB	770	470	1240	4	0.21	186	362
	PTMBA	1212	641	1853	5	0.30	168	362
	PTMBPA	1293	693	1986	5	0.33	136	362
	PMDA	720	419	1139	0	0.21	191	432
	PMBDA	1139	617	1756	5	0.29	167	362
	PTMBDPA	1467	821	2288	5	0.38	165	362

The data in Table 4 was achieved using Verilog RTL with typical synthesis and layout settings. Frequency (in MHz) was set to 100 and speed grade was STD. The data given in Table 4 is indicative only. The overall device utilization and performance of the core is system dependent.

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

2.10 Performance Statistics

Table 5 and Table 6 give the device speed grades required to meet either 33 MHz or 66 MHz PCI operation for the 32-bit mode for the three operating environments supported by Microsemi. Not all families support 66 MHz operation.

Table 5 • **Device Speed Grade Requirements**

	Family	Commercial	Industrial
33 MHz 32-bit	Fusion	STD	STD
	IGLOO/e	STD	STD
	ProASIC3/E/L	STD	STD
	SmartFusion	STD	STD
66 MHz 32-bit	Fusion	–2	–2
	IGLOO/e	N/A	N/A
	ProASIC3/E	–2	–2
	SmartFusion	–1	–1

The PCI specification timing requirements are given in Table 6.

Table 6 • **PCI Bus Timing**

	Setup		Hold		Clock to Out	
Signals	**33 MHz**	**66 MHz**	**33 MHz**	**66 MHz**	**33 MHz**	**66 MHz**
Bussed Signals	7 ns	3 ns	0 ns	0 ns	11 ns	6 ns
Non-Bussed Signals (e.g., GNTN)	10 ns	5 ns	0 ns	0 ns	11 ns	6 ns

2.11 I/O Requirements

The FPGA I/O requirements of CorePCIF_AHB are very dependent on the system configuration. In most cases, only the PCI bus will be connected to I/O pins, with the AHB bus connected to internal logic in the FPGA. The PCI bus requires approximately 50 pins.

2.12 Electrical Requirements

CorePCIF_AHB supports both the 3.3 V and 5.0 V PCI specifications when operating at 33 MHz; at 66 MHz, the PCI bus must operate at 3.3 V. 5.0 V PCI environments may require external voltage level translator devices, or may not be supported. See Table 7 for details.

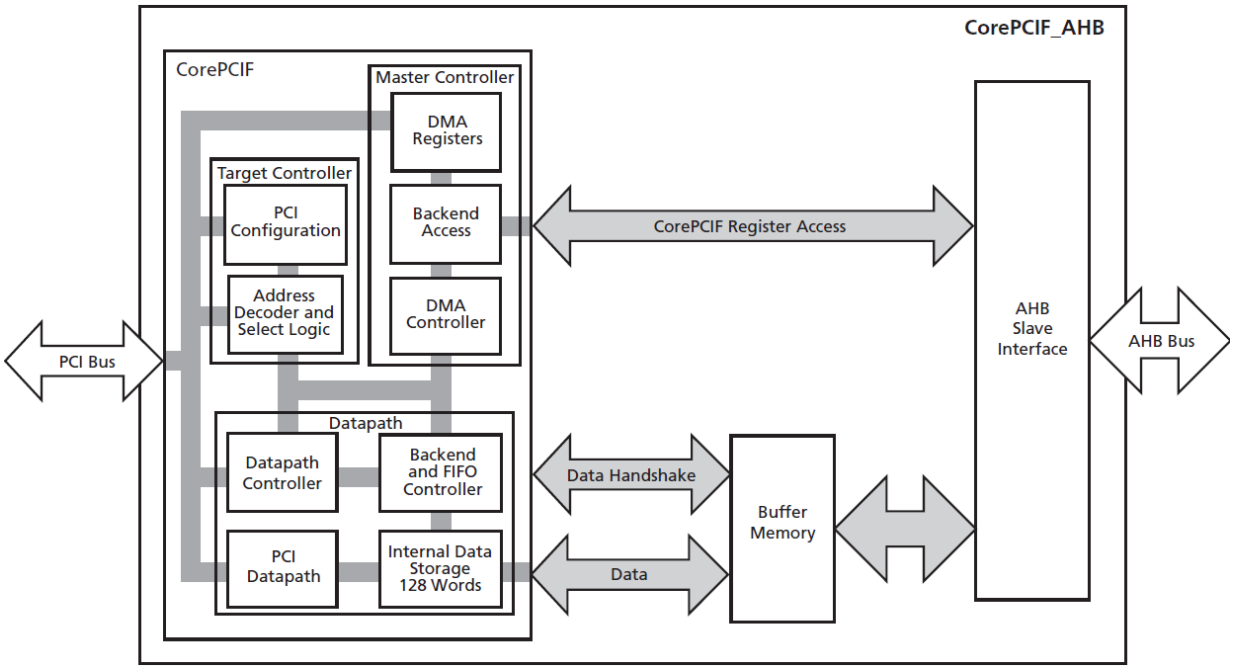
Table 7 • **Supported Electrical Environments**

Clock Speed	Family	PCI Voltage with Direct FPGA Connection	PCI Voltage with Level Translators
33 MHz	Fusion	3.3	3.3 and 5.0
	IGLOO/e	3.3	3.3 and 5.0
	ProASIC3/E/L	3.3	3.3 and 5.0
	SmartFusion2	3.3 and 5.0	3.3 and 5.0
66 MHz	Fusion	3.3	3.3
	IGLOO/e	3.3	3.3
	ProASIC3/E	3.3	3.3
	SmartFusion2	3.3	3.3 and 5.0

3 Functional Description

CorePCIF_AHB consists of three major functional blocks, shown in Figure 2. These blocks are the base CorePCIF core, the buffer memory and AHB slave interface.

Figure 2 • **CorePCIF_AHB Block Diagram**



3.1 AHB Slave Interface

The AHB slave interface allows the AHB to access the buffer memory, the CorePCIF configuration space through the backend interface of CorePCIF, and some local control and status registers for handling AHB interrupts, etc. This block also performs the cross-clock-domain control logic when the AHB clock and PCI clock are asynchronous.

3.2 Buffer Memory

This is at least a 2-kbyte (512×32) dual-port memory used as a memory buffer between the PCI bus and the AHB. The memory uses the true dual-port memory blocks within the FPGA, allowing both the AHB and PCI bus to access the memory at the same time. The memory supports zero-wait-state access from both sides.

3.3 CorePCIF – Target Controller

The Target controller implements the PCI Target function. It contains two sub-blocks: the PCI configuration space and the address decoder logic. The configuration block implements a "type 0" PCI configuration space, supporting up to six base address registers and the Expansion ROM register.

The actual registers implemented are described in Table 23.

The address decoder block monitors the PCI bus for address cycles and compares the address with the base address registers configured in the configuration space. A match signals the datapath controller to start a PCI cycle.

3.4 CorePCIF – Master Controller

The Master controller implements the PCI Master function. It contains three sub-blocks: the DMA registers, DMA controller, and backend access logic. The DMA register block implements the four 32-bit registers that control the DMA controller. These registers can be programmed either from the PCI bus or from the backend.

The DMA controller implements a PCI-compliant Master function that can burst up to 2^{32} bytes of data without intervention. CorePCIF_AHB uses a dual-port RAM (memory buffer) of configurable size 2KB, 4KB, or 8KB using parameter "MEMORY_SIZE". CorePCIF_AHB has DPRAM between PCIF and AHB bus, CorePCIF_AHB reads data from DPRAM and sends it on PCI bus. STOP_MASTER feature of CorePCIF is discontinued in CorePCIF_AHB, as the AHB master and also the buffer in between AHB bus and PCIF bus will be used for DMA data flow control.

The backend access block allows a processor connected to the core backend to access the DMA registers and initiate a DMA transfer.

3.5 CorePCIF – Datapath

The datapath block provides the data control and storage path between the backend and the PCI bus. It contains four sub-blocks: the PCI datapath, the PCI datapath controller, the backend and FIFO controller, and the internal data storage memory.

The PCI datapath controller is responsible for controlling the PCI control signals and coordinating the data transfers with the backend controller for both Target and Master operations.

The PCI datapath block selects which data should be routed to the PCI bus. Data may come from the PCI configuration block, the DMA register block, or the internal data storage. The datapath block also generates and verifies the PCI parity signals.

The backend controller implements the FIFO control logic. This interfaces to the user's backend logic and moves data from the backend interface into the internal storage. It also includes logic that monitors how much data is actually transferred on the PCI bus. The backend controller can recover data that has not actually been transferred, such as when a Master transfer is terminated with a disconnect without data.

3.6 CorePCIF – Internal Data Storage

CorePCIF includes a 64-word internal memory block that is used to store data being moved from the backend to the PCI bus. Data being transferred from the PCI bus to the backend is not stored internally in the core.

This data storage performs two functions. First, it implements a four-word FIFO that decouples the PCI data transfers from the backend data transfers, thereby increasing throughput. Second, it provides storage for the FIFO recovery logic used to prevent data loss when the backend is connected to a standard FIFO.

Each of the seven supported BARs (six BARs and the Expansion ROM) is allocated eight words of memory. BAR 0 is allocated locations 0–7, BAR 1 is allocated 8–15, and so on. The Expansion ROM is allocated locations 48–55, and the remaining eight locations are not used. Each word is 32 bits wide for 32-bit implementations and 64 bits wide for 64-bit implementations.

Each BAR will require at least 256 logic modules to implement the storage. Storage is only required for the enabled BARs.

When the SLOW_READ parameter is set, the internal data storage is not implemented, eliminating the need for FPGA memory resources. Instead, the data throughput rate is reduced to prevent data loss.

3.7 CorePCIF Target Function

The CorePCIF Target function acts as a slave on the PCI bus. The Target controller monitors the bus and checks for hits to the configuration space or the address space defined in its BARs. When a hit is detected, the Target controller notifies the backend and then acts to control the flow of data between the PCI bus and the backend.

3.7.1 Supported Target Commands

Table 8 lists the PCI commands supported in the CorePCIF Target implementation.

Table 8 • **Supported PCI Target Commands**

CBEN[3:0]	Command Type	Supported
0000	Interrupt Acknowledge	No
0001	Special Cycle	No
0010	I/O Read	Yes
0011	I/O Write	Yes
0100	Reserved	–
0101	Reserved	–
0110	Memory Read	Yes
0111	Memory Write	Yes
1000	Reserved	–
1001	Reserved	–
1010	Configuration Read	Yes
1011	Configuration Write	Yes
1100	Memory Read Multiple	Yes
1101	Dual Address Cycle	No
1110	Memory Read Line	Yes
1111	Memory Write and Invalidate	Yes

3.7.2 I/O Read (0010) and Write (0011)

The I/O Read command is used to read data mapped into I/O address space. The I/O Write command is used to write data mapped into I/O address space. In this case, the write is qualified by the byte enables.

3.7.3 Memory Read (0110) and Write (0111)

The Memory Read command is used to read data in memory-mapped address space. The Memory Write command is used to write data mapped into memory address space. In this case, the write is qualified by the byte enables.

3.7.4 Memory Read Multiple (1100) and Memory Read Line (1110)

The Memory Read Multiple and Memory Read Line commands are treated in the same manner as a Memory Read command. Typically, the bus master will use these commands when data is prefetchable.

3.7.5 Memory Write and Invalidate (1111)

The Memory Write and Invalidate command is treated in the same manner as a Memory Write command.

3.7.6 Configuration Read (1010) and Write (1011)

The Configuration Read command is used to read the configuration space of each device. The Configuration Write command is used to write information into the configuration space. The device is selected if its IDSEL signal is asserted and AD[1:0] are set to '00'. Additional address bits are defined as follows:

- AD[7:2] contain one of 64 DWORD addresses for the configuration registers.
- AD[10:8] indicate which device of a multi-function agent is addressed. The core does not support multi-function devices, and these bits should be '000'.
- AD[31:11] are ignored.

The core supports burst configuration read and write cycles.

3.7.7 Disconnects and Retries

The CorePCIF Target will perform either single-DWORD or burst transactions, depending on the request from the system Master. If the backend is unable to deliver data quickly enough, the Target will respond with either a PCI retry or disconnect, with or without data. If the system Master requests a transfer that the backend is not able to perform, a Target abort can be initiated by the backend.

3.8 CorePCIF Master Function

The Master function in CorePCIF is designed to do the following:

- Arbitrate for the PCI bus
- Initiate a PCI cycle
- Pass dataflow control to the Target controller
- End the transfer when the DMA count has been exhausted
- Allow the backend hardware to stop and start DMA cycles

Master transfers can be initiated directly from the backend interface, or another PCI device may program the DMA engine to initiate a PCI transfer.

3.8.1 Backend Interface

Through the backend interface (BE_REQ, BE_GNT, BE_ADDRESS, and so on), an external processor through the backend interface, the AHB interface can access the DMA Master control registers and initiate a Master transfer. This interface also allows the complete PCI configuration space to be accessed so the core can be self-configured by a backend processor. This is required when the core is used to implement the PCI device responsible for configuring the PCI bus. A hardware lock (BE_CFGLOCK) is provided for safety reasons to prevent the backend from changing the values in the PCI configuration space.

3.8.2 Supported Master Commands

The CorePCIF Master controller is capable of performing configuration, I/O, memory, and interrupt acknowledge cycles. Data transfers can be up to 2^{32} bytes.

The Master controller will attempt to complete the transfer using a maximum-length PCI burst unless the maximum burst length bits are set in the control register. If the addressed Target is unable to complete the transfer and performs a retry or disconnect, the Master control will restart the transfer and continue from the last known good transfer. If a Target does not respond (no DEVSELn asserted) or responds with a Target abort cycle, the Master controller will abort the current transaction and report it as an error in the control register.

3.8.3 DMA Master Registers

There are four 32-bit registers used to control the function of the CorePCIF Master. The first register is the PCI address register. The second register is the RAM or backend address register. These two registers provide the source/destination addressing for all data transfers. The third register contains the number of words to be transferred, and the final control register defines the type and status of a Master transfer. These registers are cleared on reset. They are defined in detail in Table 34 through Table 39.

The DMA registers can be accessed from either the PCI or the backend interface. The address locations for the DMA registers are listed in Table 9. When these registers are accessible from the PCI bus, they can be memory-, I/O-, or configuration-mapped. The DMA_REG_LOC, DMA_REG_BAR, and BACKEND parameters control access to these registers are accessible though BAR 1, a 256-byte memory-mapped BAR.

The complete configuration space can be read when BAR access to these registers is enabled, but writing can be done only to the four DMA control registers.

When the BACKEND parameter is set, the four registers and the complete PCI configuration space can be accessed through the backend (Table 9).

Table 9 • **DMA Register Addresses**

Register Name	Address
PCI address	50h
RAM address or data register	54h
DMA transfer length	58h
DMA control register	5Ch

3.8.4 Master Transfers

The CorePCIF Master function supports full DMA transfers to and from the backend interface and initiates direct PCI transfers.

When normal DMA transfers are used, CorePCIF writes each data word to or fetches it from memory through its backend interface. This allows data to be transferred directly from the PCI bus to or from backend memory blocks. In some circumstances, this is inefficient, especially if a processor connected to the backend simply wants to carry out a single-word PCI read or write. In this case, the processor writes the data word to a known location in its memory map. It then programs the DMA controller to perform a single-word DMA transfer. The DMA controller accesses the memory location to obtain the data value; this may require the processor to stop operating while the PCI core accesses the memory to complete the PCI transfer.

When direct DMA transfers are enabled, the processor simply writes the PCI address and data into the core and starts the transfer by writing to the control register, setting the DMA_BAR value to '111'. The core then fetches the data value or writes it to the internal register during the PCI transfer. Access to the backend memory is not required to complete the DMA transfer.

Direct DMA transfer supports only 32-bit transfers. When using 64-bit versions of the core, the 64-bit transfer mode select bit in the DMA control register should not be set if Direct DMA mode is enabled.

3.8.5 Master Byte Commands

CorePCIF can either transfer multiple whole DWORDs (QWORDs for 64-bit transfers) or perform a single DWORD or QWORD transfer with one or more byte enables active.

When multiple words are to be transferred—the DMA transfer length register is greater than four bytes (eight bytes for 64-bit)—the byte enable bits in the DMA control register should be programmed to all ones. All four or eight (64-bit) bytes will be transferred in each data cycle.

If a partial-word read or write is required, the DMA transfer length register should be programmed to four bytes (or eight for 64-bit) and the correct bits set in the byte enable bits in the DMA control register. The DMA engine will transfer a single word, setting the appropriate byte enable bits on both the backend and the PCI interface.

If a non-aligned DMA transfer is required, three separate DMA operations should be performed. The first DMA transfer should be configured to transfer a single DWORD with just the initial bytes enabled. The second DMA should transfer the remaining complete DWORDs with all bytes enabled. A third DMA transfer should transfer the final DWORD with just the remaining bytes enabled. For example, a transfer starting at address 3 and ending at address 12 would require three operations. The first DMA transfer would enable byte 3 only, the second transfer would transfer two DWORD addresses to bytes 4 through 11, and the third DMA transfer would enable byte 0 and transfer address 12.

3.9 CardBus Support

CorePCIF directly supports CardBus functional requirements. Two top-level parameters, CIS_UPPER and CIS_LOWER, specify the 32-bit configuration space setting for the CIS pointer. CIS_UPPER sets the upper 16 bits, and CIS_LOWER sets the lower 16 bits.

The CIS address space must be mapped to one of the BARs or the Expansion ROM. It may not be mapped to configuration space, which means the lower three bits of the CIS pointer (that is, the lower three bits of CIS_LOWER) must not be set to '000'. This allows the user to implement the CIS address space as one of the external backend BAR memory spaces.

When CardBus support is enabled, the IDSEL core input is disabled. CardBus does not require IDSEL to be active for configuration cycles.

3.10 CompactPCI Hot-Swap Support

CorePCIF supports the CompactPCI Hot-Swap PICMG 2.1 R2.0 standard; additional inputs and outputs are provided to support this standard. When enabled, the core includes the hot-swap capabilities register in the configuration space and a state machine that implements the hardware connection process defined in the PICMG Hot-Swap specification.

4 Core Structure

This chapter describes the internal core structure and associated source files.

CorePCIF_AHB is built on top of the CorePCIF core. A wrapper is created around the base CorePCIF code to provide the AHB interface.

As shown in Figure 3, the unshaded modules are common to all FPGA families; the shaded modules are specific to each family. This is required to consistently meet the PCI timing constraints. The shaded modules are optimized specifically for each of the supported logic families. The low-level technology cells are used in the higher-level modules.

Figure 3 • **CorePCIF_AHB Structure**

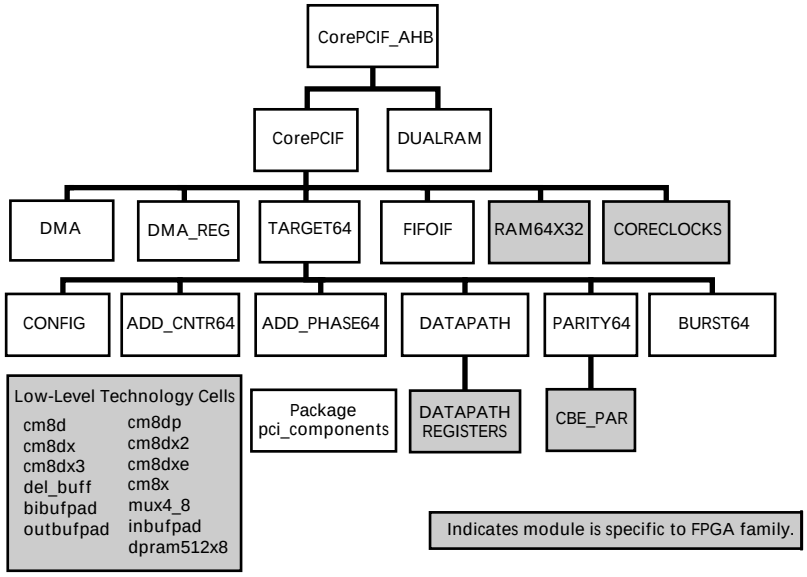


Table 10 provides details for each of the common source modules and the functions they implement. Table 11 provides the details of the FPGA technology files. Table 11 provides details of a few high-level source files not directly used by the core but provided to create test databases.

Table 10 • **CorePCIF_AHB Common Source Files**

Common Files	Description
corepcif_ahb	This is the top level of the core. It includes all the top-level ports.
dualram	This is a dual-port RAM block used as the memory buffer.
corepcif	This is the top level of the core. It includes all the top-level ports. Parameters will enable and disable the Target, Master, and backend functions. This is the CorePCIF top level upon which this core is built.
pci_components	This is the VHDL package that contains the component declarations used within the core along with some common type conversion functions.
fifoif	This is the control logic that manages internal data storage and performs FIFO recovery cycles.
target64	This is the top level of the main PCI Target function.
burst64	This is the main control logic used to handle the PCI protocol and manage data transfers to and from the PCI bus.
add_cntr64	This is the main address counter that tracks both the current PCI and backend addresses.

Table 10 • **CorePCIF_AHB Common Source Files *(continued)***

Common Files	Description
add_phase64	This block compares the PCI address during an address phase to detect whether the configuration space or one of the BARs on the Target is being addressed. It also contains the logic to detect BAR overflows so a Target disconnect can be triggered.
config	This block contains the registers required to implement PCI configuration space.
datapath	This block routes the data between the backend interface or memory buffer and the PCI bus. It provides a data storage register used to recover when transferred data are stalled.
parity64	This block creates the top-level structure for parity generation and checking. The parity generation and checking is done using the cbe_par module.
dma	This is the main Master control logic. It contains state machines and counters that control the DMA engine and initiate PCI cycles.
dma_reg	This module contains the four DMA control registers and the main DMA transfer counters.

Table 11 • **Technology-Specific Source Files**

FPGA Specific Files	Description
bibufpad	This module contains a bidirectional I/O pad.
cbe_par	This block implements a PCI parity generator and checker. The 36-input parity tree is hand-optimized to obtain the most efficient implementation for each FPGA family.
cm8d	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with clear.
cm8dp	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with preset.
cm8dx	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with clear.
cm8dx2	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with clear, with some inputs tied or shared.
cm8dx3	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with clear, with some inputs tied or shared.
cm8dxe	This is a low-level FPGA technology cell implementing a four-input multiplexer and register with enable and clear.
cm8x	This is a low-level FPGA technology cell implementing a four-input multiplexer.
coreclocks	This module contains the global and clock buffers.
datapath_registers	This module implements the datapath registers used to interface to the PCI bus.
del_buff	This module contains a delay element used to insert delays to control the PCI hold times. The amount of inserted delay for all critical PCI input paths can be adjusted in this file.
family	This is a VHDL package or Verilog include file that sets the FPGA family to enable some family-specific optimizations.
inbufpad	This module contains an input I/O pad.
mux4_8	This is a low-level FPGA technology cell implementing eight four-input multiplexers.
outputpad	This module contains an output I/O pad.
ram64x32	This module contains a 64×32 RAM using the appropriate FPGA memory blocks.
dpram512x8	This module contains a 512×8 dual-port RAM using the appropriate FPGA memory blocks.

Table 12 • **CorePCIF_AHB Miscellaneous Source Files**

Miscellaneous Files	Description
components	This is a VHDL components package that contains a declaration for CorePCIF_AHB for use in the user's top level, if required.
pcicore_components	This is a VHDL components package that contains the PCI core component declaration.
amba_components	This is a VHDL components package that contains component declarations specific to CorePCIF_AHB.

5 Tool Flows

5.1 Licensing

CorePCIF_AHB is licensed in two ways: Obfuscated and RTL. Tool flow functionality may be limited, depending on your license.

5.1.1 Obfuscated

Complete RTL code is provided for the core, enabling the core to be instantiated, configured, and generated within SmartDesign. Simulation, Synthesis, and Layout can be performed with Libero Integrated Design Environment (IDE). The RTL code for the core is obfuscated.

5.1.2 RTL

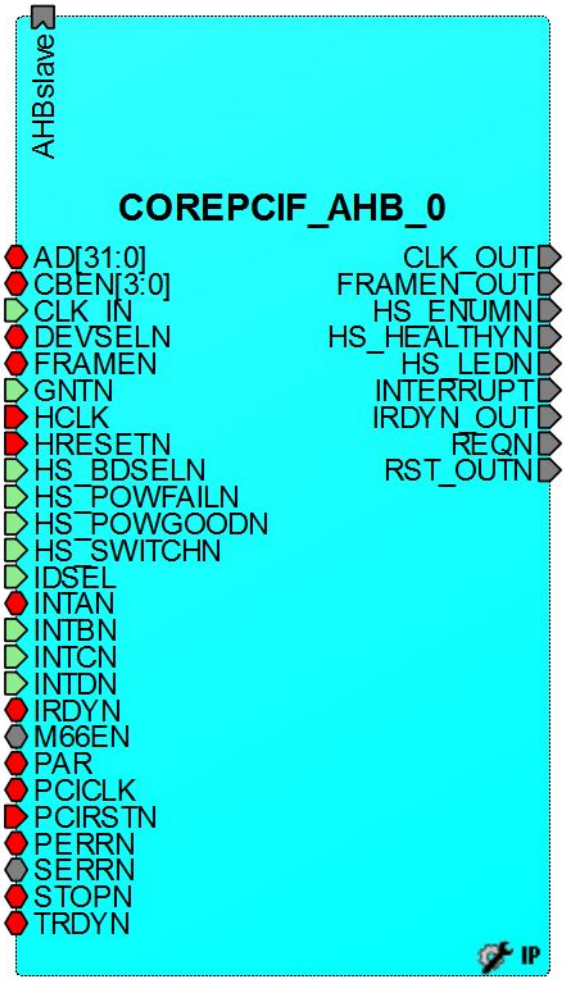
Complete RTL source code is provided for the core.

5.2 SmartDesign

CorePCIF_AHB is pre-installed in the SmartDesign IP deployment design environment or downloaded from the online repository. Figure 4 shows an example instantiated.

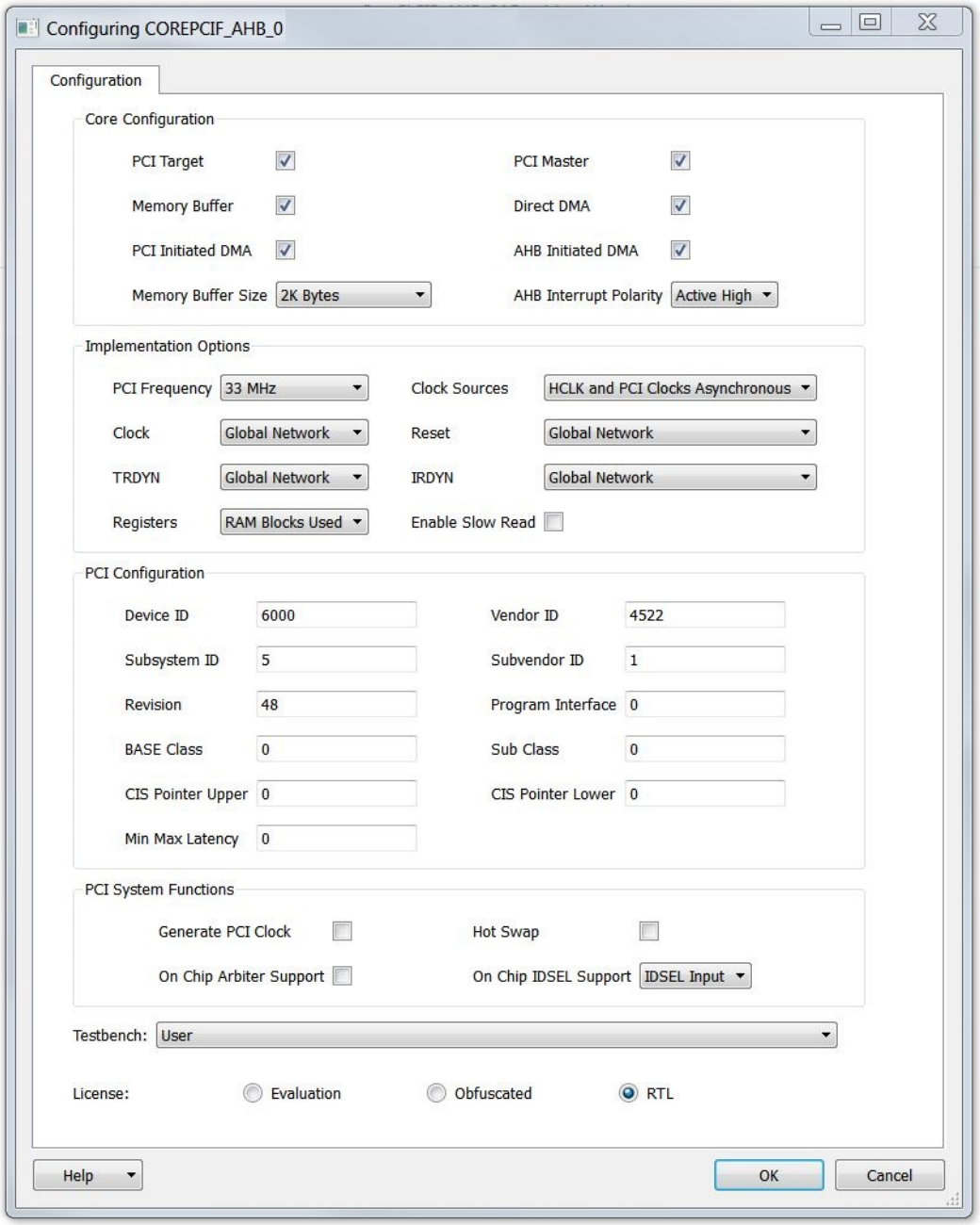
Note: CorePCIF is compatible with Libero Integrated Design Environment (IDE), Libero System-on-Chip (SoC), and Libero System-on-Chip (SoC) PolarFire. Unless specified otherwise, this document uses the name Libero to identify Libero IDE, Libero SoC.

Figure 4 • **CorePCIF_AHB Full I/O View**



The core can be configured using the configuration GUI within SmartDesign, as shown in Figure 5.

Figure 5 • **CorePCIF_AHB Configuration in SmartDesign**



General Configuration Parameters in Table 14 explains the configuration parameters and their recommended values.

PCI Target (equivalent generic PCI_TARGET)

This enables the Target function in CorePCIF.

PCI Master (equivalent generic PCI_MASTER)

This enables the Master function in CorePCIF.

Memory Buffer (equivalent generic ENABLE_MEMORY)

This enables the buffer memory.

DirectDMA (equivalent generic ENABLE_DIRECTDMA)

This enables the Direct DMA operation in CorePCIF. This allows the core to initiate PCI operations without use of the memory buffer.

PCI Initiated DMA (equivalent generic PCI_START_DMA)

This allows the PCI bus to configure and start the DMA controller inside CorePCIF.

AHB Initiated DMA (equivalent generic AHB_START_DMA)

This allows the AHB to configure and start the DMA controller inside CorePCIF. When set, the standard PCI configuration registers can also be accessed from the AHB interface.

PCI Frequency (equivalent generic PCI_FREQ)

This configures the core to support only 33 MHz or 33 and 66 MHz PCI operation.

Clock Sources (equivalent generic CLOCKS_ASYNC)

This configures the core for either synchronous or asynchronous PCI and HCLK operation.

Memory Buffer Size (equivalent generic MEMORY_SIZE)

This sets the size of the internal memory buffer.

AHB Interrupt Polarity (equivalent generic INTERRUPT_POLARITY)

This sets the polarity of the AHB interrupt output.

Enable Slow Read (equivalent generic SLOW_READ)

When set, this removes the need for two internal memory blocks within CorePCIF. The side effect is that PCI transfers to and from the buffer memory will operate at one wait state.

Reset (equivalent generic USE_GLOBAL_RESET)

This configures the core to either use a global (recommended) or a buffer tree for the reset network.

PCI Configuration

This set of parameters configures the PCI configuration space locations that set the read-only configuration values, such as Device and Vendor ID.

Generate PCI Clock (equivalent generic GENERATE_PCICLK)

This sets whether the core will generate the PCI clock.

Hot-Swap (equivalent generic ENABLE_HOT_SWAP)

This enables the PCI hot-swap facility of the core.

On-Chip Arbiter Support (equivalent generic ONCHIP_ARBITER)

This allows the CorePCIF REQn and GNTn to be directly connected to an internal PCI arbiter, as well as enabling the FRAME and IRDY core outputs required by a PCI arbiter.

On-Chip IDSEL Support (equivalent generic ONCHIP_IDSEL)

This allows the target IDSEL input to be directly connected to one of the upper AD bus outputs instead of the IDSEL I/O pad. This could be required when the FPGA is responsible for configuring the PCI bus.

TRDYN (equivalent to USE_GLOBAL_TRDY)

This allows the TRDYN net to be use a global or normal routing resources.

IRDYN (equivalent to USE_GLOBAL_IRDY)

This allows the IRDYN net to be use a global or normal routing resources.

5.3 Synthesis in Libero

To run Synthesis on the core, **set the design root to the top of the project**. This is a wrapper around the core that sets all the generics appropriately.

Make sure the required timing constraints files are associated with the synthesis tool.

Note: Use sdc constraint files with suffix "_synplicity" on Libero 9.2 or before versions, and with suffix "_designer" on Libero versions later to 9.2.

Synthesis Timing Constraints details the timing constraints that are required.

Click the **Synthesis** icon in Libero. The synthesis window appears, displaying the Synplicity® project. To run Synthesis, click the **Run** icon.

5.4 Place-and-Route in Libero

Make sure required timing and physical constraints files are associated with the place-and-route tool. There should be multiple timing and physical constraints files available, covering the PCI functions: 33/66 MHz and 32-bit operation as well as device package combinations.

For Target-only cores, the *T_* files should be used; for Master-only cores, the *M_* files should be used; and for Target and Master operation, the *TM_* files should be used.

The supplied timing constraints files assume a typical configuration of the core. Some configurations may cause the timing constraints files to cause an error when loaded by Designer. For example, if the ONCHIP_IDSEL function is enabled, the IDSEL input is not used and Synthesis will remove the IDSEL input. When this occurs, Designer will detect an error when it tries to set a timing constraint on the nonexistent IDSEL input. In this case, the timing constraints on the IDSEL input should be removed from the SDC files.

If there is not a good match for your selected device and package, you should create your own physical constraints files, following the rules in the **Implementation Hints**.

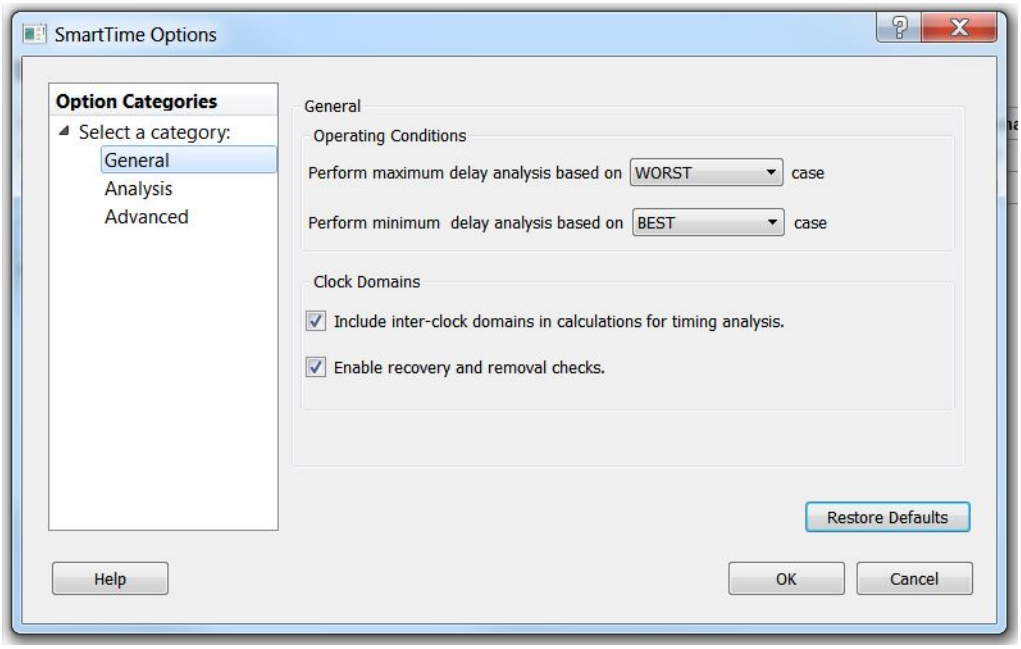
Having set the design root appropriately and run Synthesis, click the **Layout** icon in Libero to invoke Designer. CorePCIF requires no special place-and-route settings. Microsemi recommends you set the compile options given in Table 13.

Table 13 • **Designer Compile Options**

Device Family	Compile Option(s)
Fusion IGLOO/e ProASIC3/E/L	Set pdc_abort_on_error "off" Set pdc_eco_display_unmatched_objects "off" Set demote_globals "off" -promote_globals "off" Set combine_register "on" -delete_buffer_tree "off" Set report_high_fanout_nets_limit 10

When CLOCKS_ASYNC is set, the interclock domain timing can be checked. To check this timing, the **Include inter-clock domains in calculations for timing analysis** option in the timer must be set (Figure 6).

Figure 6 • **SmartTime Options**



Apart from the data path to the HC_DATA register, the core uses registers on either side of the paths that cross the clock domain. The state machines use a handshake that is independent of the relative clock frequencies. The path to the HC_DATA register is effectively a multi-cycle path in the HCLK domain and should be less than the HCLK period. The enable on the HC_DATA register is generated one clock cycle after HC_DATA is guaranteed to be stable.

When the PCI and AHB clocks are of different frequency or from different global network the data path on HC_DATA register must be analyzed as multi=cycle path.

6 Configuring Parameters

CorePCIF_AHB is a highly configurable core. The configuration is controlled by approximately 5025 top-level parameters. These are listed in Table 14 to Table 16.

6.1 General Configuration Parameters

Table 14 shows general configuration parameters for CorePCIF_AHB.

Table 14 • **General Parameters**

Name	Values	Description
FAMILY	15 to 27	Must be set to the required FPGA family: 15: ProASIC3 16: ProASIC3E 17: Fusion 18: SmartFusion 19: SmartFusion2 20: IGLOO 21: IGLOOe 22: ProASIC3L 24: IGLOO2 25: RTG4 26: PolarFire 27: PolarFire SoC
PCI_MASTER	0 or 1	When 1, the PCI Master function with DMA controller is implemented.
PCI_TARGET	0 or 1	When 1, the PCI Target function is implemented.
ENABLE_MEMORY	0 or 1	When 1, the buffer memory is implemented
ENABLE_DIRECTDMA	0 or 1	Enables core support for direct DMA operations. When 1, direct DMA mode is enabled, allowing the PCI data value to be read from and written to an internal register rather than the buffer memory.
AHB_START_DMA	0 or 1	When 1, allows the AHB master to program the PCI DMA controller and start a DMA operation.
PCI_START_DMA	0 or 1	When 1, allows another PCI master to program the PCI DMA controller and start a DMA operation.
MEMORY_SIZE	9 to 16	Sets the size of the buffer memory: • 9: Buffer memory not implemented • 11: 2 kbytes (buffer memory will use 4 memory blocks) • 12: 4 kbytes (buffer memory will use 8 memory blocks) • 13: 8 kbytes (buffer memory will use 16 memory blocks)
PCI_FREQ	33 or 66	When 66, the 66 MHz bit in the PCI configuration space is set.
SLOW_READ	0 or 1	When 1, the core inserts either one or two wait states in all read transfersone wait state in all PCI read transfers, eliminating the requirement for internal data storage within the corethe two internal memories within CorePCIF. This parameter must not be set if the FIFO recovery option is enabled.
INTERRUPT_POLARITY	0 or 1	When 1, the AHB INTERRUPT output is active high; when 0, active low.
CLOCKS_ASYNC	0 or 1	Set to 1 when the PCI clock is asynchronous to the AHB clock. In general, this should be set whenever the two clocks are not on the same global network.
GENERATE_PCICLK	0 or 1	Set to 1 when the core is required to generate the PCI clock.

Table 14 • **General Parameters *(continued)***

Name	Values	Description
USE_GLOBAL_CLK	0, 1, or 2	Controls the sort of clock buffer used. • 0: No buffer is used. The synthesis tool will insert a buffer. • 1: A standard clock buffer is used. • 2: When using AX/RTAX-S families, uses a CLKBUF instead of a HCLKBUF cell Microsemi recommends setting this parameter to 1.
USE_GLOBAL_RESET	0 or 1	When 1, a global buffer is used to drive the internal reset network in the core. When 0, normal routing resources are used, and due to the high fanout of the reset network, a buffer tree will be created for it. Microsemi recommends that this parameter be set to 1.
USE_GLOBAL_TRDY	0 or 1	When 1 and MASTER = 1, a global buffer is used to drive the internal TRDY network in the core. When 0, normal routing resources are used. Microsemi recommends that this parameter be set to 1.
USE_GLOBAL_IRDY	0 or 1	When 1 and TARGET = 1, a global buffer is used to drive the internal IRDY network in the core. When 0, normal routing resources are used. Microsemi recommends that this parameter be set to 1.
ONCHIP_ARBITER	0 or 1	In some applications, the FPGA will be the system controller as well, and include the PCI arbiter. When 1, this removes the pads from the REQN and GNTN I/Os, allowing connection inside the FPGA and enabling the FRAMEN_OUT and IRDYN_OUT outputs.
ONCHIP_IDSEL	0 to 31	In some applications, the FPGA will be the system controller as well, and include the IDSEL decoding. When *value* is non-zero, the IDSEL input is directly driven by the AD *(value*) output. When 0, IDSEL is driven from the IDSEL input.

6.2 PCI Configuration Space Parameters

Table 15 • **PCI Configuration Space Parameters**

Name	Values	Description
VENDOR_ID	0 to 65,535	Sets the user vendor ID value in the PCI configuration space. The Microsemi Vendor ID is 4522 (11AAh) and may be used with permission. Microsemi will allocate a device ID and sub-vendor ID on demand. Contact Technical Support to request this service.
DEVICE_ID	0 to 65,535	Sets the user device ID value in the PCI configuration space.
REVISION_ID	0 to 255	Sets the user revision ID value in the PCI configuration space.
BASE_CLASS	0 to 255	Sets the user base class value in the PCI configuration space.
SUB_CLASS	0 to 255	Sets the user subclass value in the PCI configuration space.
PROGRAM_IF	0 to 255	Sets the user program interface value in the PCI configuration space.
SUBVENDOR_ID	0 to 65,535	Sets the user subvendor ID value in the PCI configuration space.
SUBSYSTEM_ID	0 to 65,535	Sets the user subsystem ID value in the PCI configuration space.
CIS_UPPER	0 to 65,535	Sets the value of the upper 16 bits of the CardBus CIS pointer.
CIS_LOWER	0 to 65,535	Sets the value of the lower 16 bits of the CardBus CIS pointer.
ENABLE_HOT_SWAP	0 or 1	Enables the hot-swap register and functionality.
MINMAXLAT	0 to 65,535	Sets the minimum grant and maximum latency values at location 3Eh in the configuration space.

6.3 Default Core Parameter Settings

Table 16 details the parameter settings used to create the eight example builds in **Utilization Statistics**.

Table 16 • **Default Build Parameters**

Parameter	PTB	PTMBA	PTMBPA	PMDA	PMBDA	PTMBDPA
FAMILY	Set to the appropriate family value.					
PCI_TARGET	1	1	1	0	0	1
PCI_MASTER	0	1	1	1	1	1
ENABLE_MEMORY	1	1	1	0	1	1
ENABLE_DIRECTDMA	0	0	0	1	1	1
AHB_START_DMA	0	0	1	1	1	1
PCI_START_DMA	0	1	1	0	0	1
MEMORY_SIZE	11	11	11	9	11	11
PCI_FREQ	33	33	33	33	33	33
DEVICE_ID	300	301	302	303	304	305
VENDOR_ID	4,522	4,522	4,522	4,522	4,522	4,522
REVISION_ID	31	31	31	31	31	31
BASE_CLASS	255	255	255	255	255	255
SUB_CLASS	0	0	0	0	0	0
PROGRAM_IF	0	0	0	0	0	0
SUBSYSTEM_ID	400	401	402	403	404	405

Table 16 • **Default Build Parameters *(continued)***

Parameter	PTB	PTMBA	PTMBPA	PMDA	PMBDA	PTMBDPA
SUBVENDOR_ID	4,522	4,522	4,522	4,522	4,522	4,522
CIS_UPPER	0	0	0	0	0	0
CIS_LOWER	0	0	0	0	0	0
MINMAXLAT	0	0	0	0	0	0
SLOW_READ	1	0	0	0	0	0
ENABLE_HOT_SWAP	0	0	0	0	0	0
INTERRUPT_POLARITY	1	1	1	1	1	1
CLOCKS_ASYNC	0	0	0	0	0	1
GENERATE_PCICLK	0	0	0	0	0	1
USE_GLOBAL_CLK	1	1	1	1	1	1
USE_GLOBAL_RESET	1	1	1	1	1	1
ONCHIP_ARBITER	0	0	0	0	0	24
ONCHIP_IDSEL	0	0	0	0	0	1
USE_GLOBAL_TRDY	1	1	1	1	1	1
USE_GLOBAL_IRDY	1	1	1	1	1	1

7 Core Interfaces

7.1 PCI Bus Signals

Table 17 lists the signals used in the PCI interface. The "Used On" column indicates which core configurations use each signal. For example, REQN is only used on PCI Master cores (M), IDSEL is only used on PCI Target cores (T). Note that the "Type" column describes the characteristics of the signal in the PCI specification, not necessarily its usage in a particular core. For example, TRDYN is bidirectional for PCI Target/Master cores but only an input to PCI Target cores.

Table 17 • **PCI Bus Interface Signals**

Name	Type	Used On	Description
PCICLK	Bidirectional	All	33 MHz or 66 MHz clock input or output for the PCI core
PCIRSTN	Input	All	Active low asynchronous reset
IDSEL	Input	T	Active high Target select used during configuration read and write transactions
AD	Bidirectional	All	Multiplexed 32-bit address and data bus. Valid address is indicated by FRAMEN assertion.
CBEN	Bidirectional	All	Bus command and byte enable information. During the address phase, four bits define the bus command. During the data phase, they define the byte enables (active high).
PAR	Bidirectional	All	Parity signal. Parity is even across AD[31:0] and CBE[3:0].
FRAMEN	Bidirectional (STS)	All	Active low signal indicating the beginning and duration of an access. While FRAMEN is asserted, data transfers continue.
DEVSELN	Bidirectional (STS)	All	Active low output from the Target indicating that it is the Target of the current access
IRDYN	Bidirectional (STS)	All	Active low signal indicating that the bus Master is ready to complete the current dataphase transaction
TRDYN	Bidirectional (STS)	All	Active low signal indicating that the Target is ready to complete the current dataphase transaction
STOPN	Bidirectional (STS)	All	Active low signal from the Target requesting termination of the current transaction
PERRN	Bidirectional (STS)	All	Active low parity error signal
SERRN	Bidirectional (OD)	T	Active low system error signal. This signal reports PCI address parity errors.
REQN	Output	M	Active low output used by the PCI Master controller to request bus ownership
GNTN	Input	M	Active low input from the system arbiter indicating that the core may claim bus ownership
INTAN	Bidirectional	All	Active low interrupt input and request
INTBN	Input	All	Active low input interrupt
INTCN	Input	All	Active low input interrupt
INTDN	Input	All	Active low input interrupt

Note: Active low signals are designated with a trailing uppercase N.

Table 17 • **PCI Bus Interface Signals *(continued)***

Name	Type	Used On	Description
M66EN	Bidirectional (OD)	All	Active high signal indicating that the core supports 66 MHz operation. This output will be driven LOW if the MHZ_66 parameter is NOT set. When it is set, the output is tristated. If hot-swap is enabled, this is also used as an input to verify the PCI clock frequency during hot-swap insertion cycles. The M66EN PCI signal pin should be pulled up with a 5 kΩ resistor.

Note: Active low signals are designated with a trailing uppercase N.

7.2 Backend System-Level Signals

CorePCIF buffers the PCI clock and reset internally onto global networks. The outputs shown in Table 18 allow these global networks to be used for additional user backend logic.

Table 18 • **System-Level Signals**

Name	Type	Width	Description
CLK_OUT	Output	1	Clock output. The core uses an internal clock buffer. This is the buffered version of the clock and should be used for clocking any other logic in the FPGA that is clocked by the PCI clock (see **Clocking**).
CLK_IN	Input	1	When the GENERATE_PCICLK parameter is set, this input is used to drive the PCI clock output, and also clocks the internal core logic. CLK_OUT should be used to clock additional logic inside the FPGA (see **Clocking**).
RST_OUTN	Output	1	Reset output. This is a buffered version of the PCI reset. If USE_GLOBAL_RESET = 1, a global buffer drives the internal reset network in the core as well as this reset output. If USE_GLOBAL_RESET = 0, a buffer tree is created and a reset output from the tree is fed to the reset output. RST_OUTN should be used for resetting any other logic in the FPGA. If the hot-swap function is enabled, this reset will also be asserted during a hot-swap insertion or extraction cycle per the Hot-Swap Specification.
FRAMEN_OUT	Output	1	Buffered version of the PCI FRAMEN signal intended for connection to a PCI arbiter. Care must be exercised when using this output to avoid causing PCI setup timing issues.
IRDYN_OUT	Output	1	Buffered version of the PCI IRDYN signal intended for connection to a PCI arbiter. Care must be exercised when using this output to avoid causing PCI setup timing issues.

7.3 AHB Slave Interface Signals

Table 19 • **AHB Slave Interface Signals**

Name	Type	Description
HCLK	In	AHB clock
HRESETN	In	AHB reset (active low and asynchronous)
HWRITE	In	AHB write
HADDR	In [width–1:0]	AHB address. Width is set by the MEMORY_SIZE parameter plus one.
HREADY	Out	AHB ready
HTRANS	In [1:0]	AHB transfer type
HSIZE	In [2:0]	AHB transfer size
HBURST	In [2:0]	AHB burst size. This AHB input is unused in the core.
HPROT	In [3:0]	AHB protection. This AHB input is unused in the core.
HRESP	Out [1:0]	AHB response. This output will always be ’00’, indicating okay.
HWDATA	In [31:0]	AHB data in
HRDATA	Out [31:0]	AHB data out
INTERRUPT	Out	Interrupt output. Depending on the INTERRUPT_POLARITY parameter, it is Active low or high.

7.4 Hot-Swap Interface

The signals in Table 20 are used to implement the interface between the CorePCIF hardware connection process and the external physical connection process, as described in the PICMG 2.1 R2.0 Compact PCI Hot-Swap specification. These signals only function when the HOT_SWAP_ENABLE parameter is set.

Table 20 • **Hot-Swap Interface Signals**

Name	Type	Width	Description
HS_BDSELN	Input	1	Active low signal indicating that the board is selected and all other pins are fully connected. This input should be synchronous to the PCI clock.
HS_SWITCHN	Input	1	Active low signal from the board ejector switch. This input should be synchronous to the PCI clock and debounced outside the core.
HS_POWGOODN	Input	1	Active low signal indicating that the power supply is within specification. This input should be synchronous to the PCI clock.
HS_POWFAILN	Input	1	Active low signal indicating that the power supply is outside specification or a fault has occurred. This input should be synchronous to the PCI clock.
HS_ENUMN	Output (OC)	1	Active low signal notifying the system host that the board either has been inserted or is about to be extracted. This is an open-collector output and must be directly connected to an FPGA I/O pin.
HS_LEDN	Output	1	Active low signal to drive the external blue LED
HS_HEALTHYN	Output	1	Active low signal indicating that the board is healthy and may be released from reset and allowed onto the PCI bus

8 Timing Diagrams

8.1 PCI Bus

CorePCIF_AHB implements a PCI 2.3 compliant Master and Slave interface. Complete timing waveforms of the CorePCIF core that implements the PCI interface are provided in the CorePCIF Handbook.

When SLOW_READ is '0', all PCI accesses will operate at zero wait states. When '1', read accesses to the buffer memory will operate with one wait state.

8.2 AHB Bus

CorePCIF_AHB implements an AMBA AHB compliant Slave, allowing access to the memory buffer and CorePCIF registers. The AHB interface is compliant to the AMBA specification. Full timing diagrams are available in the AMBA AHB specification.

The AHB interface will deassert HREADY, causing wait states. The actual number of inserted wait states depends on what is being accessed, PCI activity, and whether the core is configured for asynchronous clocks.

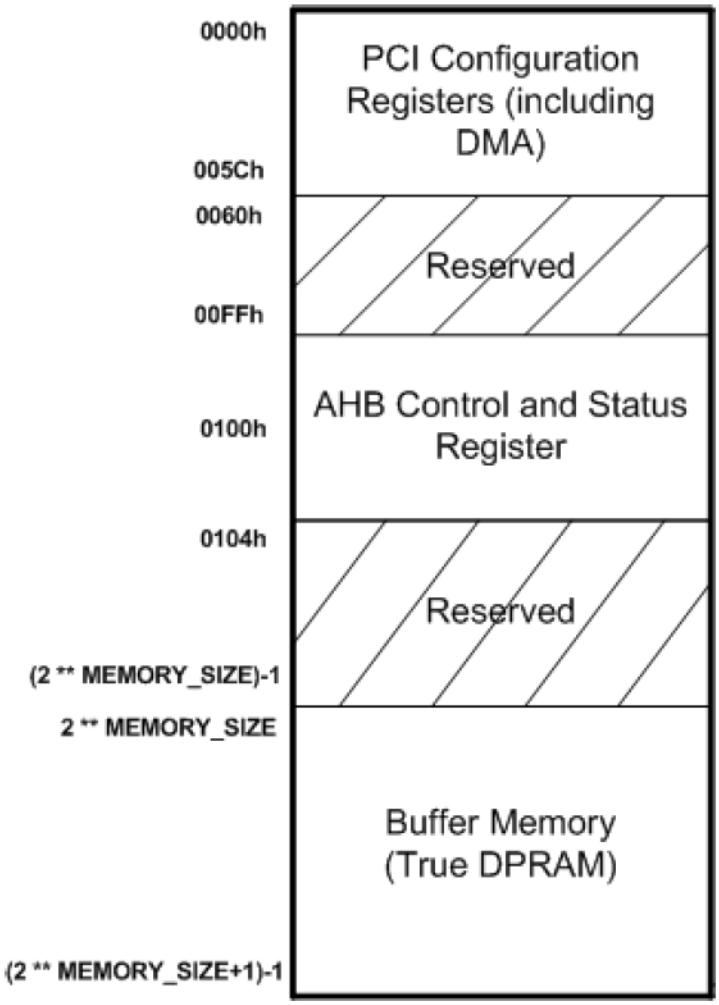
Table 21 • **AHB Bus Clocks**

Access to	Synchronous Clocks	Asynchronous Clocks
Buffer Memory	Zero wait states	Zero wait states
AHB Registers	Zero wait states	Zero wait states
PCI Registers	Wait states will be inserted if the PCI bus is accessing buffer memory or configuration space. Otherwise the first cycle will have 1 to 4 wait states, and following accesses will run at zero wait states.	Wait states will be inserted if the PCI bus is accessing buffer memory or configuration space. The first and subsequent accesses will have 1 or more wait states, depending on the relative clock speeds.

9 AHB Memory Map

The AHB memory map (Figure 7) is split into three regions: buffer memory, PCI configuration registers (including DMA Control), and AHB control and status register.

Figure 7 • **AHB Memory Map**



9.1 PCI Configuration Registers

PCI configuration registers allow the AHB interface to read and write the PCI configuration and DMA control registers in the main PCI core. The address map is identical to that described in Chapter 10 PCI Configuration Space, page 33. The DMA control registers are at addresses 50 to 5C hex.

Note: The lower 64 bytes (00-3F hex) that address the main PCI configuration space can only be modified when the CFG_UNLOCK bit is set in the AHB control register.

9.2 AHB Control and Status Registers

This is single 32-bit read write register at address 0100h. This register is used to control the AHB interrupt system and some other core features.

Table 22 • **AHB Bus Clocks**

Bits	Type	Description
0	RO	DMA_BUSY: Indicates that the PCI DMA engine is busy.
1	RW	DMAINT_EN: When set and DMAINT is active, the INTERRUPT output will be active.
2	RW	DMAINT: When set, indicates that a PCI DMA operation has completed. Cleared by writing a ’1’.
3	W	PCIINTO: When written to 1, assuming that interrupts are enabled in the PCI configuration space, will activate the PCI INTAN output on the PCI bus. This bit always reads as zero and does not need to be cleared after writing. The resulting interrupt request is cleared by the PCI master writing to the DMA control register.
4	RW	PCISERR: Indicates that the PCI system error signal (SERR) was activated; is cleared by writing a ’1’.
5	RW	PCISERREN: When set and a PCISERR is active, the AHB interrupt will be activated.
6	RO	Reserved, set to zero.
7	RO	INTERRUPT: Indicates the state of the AHB interrupt output; is ‘1’ when the interrupt is active.
11:8	RO	PCIINTS: Provides the status of the INTdcbaN interrupts. When ’1’, indicates that the PCI interrupt input is active. Bit 7 is the status for INTDN, bit 6 INTCN, and so on.
15:12	RW	PCIINTEN: Enables the PCI interrupt inputs INTdcbaN to activate the AHB interrupt output. Bit 15 enables INTDN, bit 14 INTCN, and so on.
16	RW	CFG_UNLOCK: When ’1’, allows the AHB interface to write to the PCI configuration space locations 00-03Fh.
23:17	RO	Reserved, set to zero.
31:24	RO	Core Version 31h = Core Version 31.

9.3 Buffer Memory

This is a simple memory block, whose size is set by the MEMORY_SIZE parameter. This can be used as normal AHB memory space in the system while allowing access from the PCI bus.

10 PCI Configuration Space

10.1 Target Configuration Space

The PCI specification requires a 256-byte configuration space (header) to define various attributes of the PCI Target, as shown in Table 23. All registers shown in bold are implemented. Reads of all other registers will return zero.

Table 23 • **PCI Configuration Space**

31–24	**23–16**	**15–8**	**7–0**	
Base PCI Configuration Space				**Address**
Device ID		**Vendor ID**		00h
Status		**Command**		04h
Class Code			**Revision ID**	08h
BIST	**Header Type**	**Latency Timer**	**Cache Line Size**	0Ch
Base address #0 is used to access the buffer memory				10h
Base address #1 is used to access the DMA control registers				14h
Base address #2				18h
Base address #3				1Ch
Base address #4				20h
Base address #5				24h
CardBus CIS Pointer (optional)				28h
Subsystem ID		**Subsystem Vendor ID**		2Ch
Expansion ROM base address				30h
Reserved			**Capability Pointer**	34h
Reserved				38h
Max. Latency	**Min. Grant**	**Interrupt Pin**	**Interrupt Line**	3Ch

CorePCIF uses the capability pointer to extend the configuration space. One or two capability structures are added. Vendor capability (ID = 9) is added along with optional hot-swap capability (ID = 6).

For a Target-only core, the capability pointer points to a vendor capability structure at address 44h (Table 24). The next pointer points to the optional hot-swap capability at address 40h. If hotswap capability is not implemented, address 40h will be zero, and the next pointer, at address 44h, will also be zero.

For Master cores, the capability pointer points to a vendor capability structure at 4Ch (Table 25), followed by the optional hot-swap capability. If hot-swap capability is not implemented, address 40h will be zero, and the next pointer, at address 4Ch, will also be zero.

Table 24 • **Capability Structure (Target-only Cores with Hot-swap)**

31–24	**23–16**	**15–8**	**7–0**	
Upper Configuration Space				**Address**
Reserved	Hot-Swap	Next Pointer (00h)	Capability ID (06h)	40h

Table 24 • **Capability Structure (Target-only Cores with Hot-swap)**

31–24	23–16	15–8	7–0	
Upper Configuration Space				**Address**
Microsemi Capabilities	Size (8)	Next Pointer (40h)	Capability ID (09h)	44h
Interrupt Control Register				48h
Reserved				4Ch
Reserved				50h
Reserved				54h
Reserved				58h
Reserved				5Ch

Table 25 • **Capability Structure (Master Cores with Hot-swap)**

31–24	23–16	15–8	7–0	
Upper Configuration Space				**Address**
Reserved	Hot-Swap	Next Pointer (00h)	Capability ID (06h)	40h
Reserved				44h
Reserved				48h
Microsemi Capabilities	Size (20)	Next Pointer (40h)	Capability ID (09h)	4Ch
PCI Address				50h
Backend Address/Data Value				54h
Transfer Count				58h
DMA Control Register				5Ch

10.1.1 Read-Only Configuration Registers

The following read-only registers, listed also in Table 23, have default values that are set by parameters. See the PCI specification for further information on setting these values:

- Vendor ID
- Device ID
- Revision ID
- Class Code
- Subsystem ID
- Subsystem Vendor ID
- Maximum Latency and Minimum Grant

Microsemi has an allocated Vendor ID that CorePCIF customers may use, and Microsemi will allocate a unique Device ID when the Vendor ID is used. Microsemi will allocate unique subsystem Vendor IDs on request. Contact Microsemi Technical Support (soc_tech@microsemi.com) for more information.

The capability pointer is used to point to the CorePCIF vendor capability data, and also to the hot-swap capability, if enabled. The capability list structure varies, depending on the core configuration.

10.1.2 Read/Write Configuration Registers

The following registers have at least one bit that is both read- and write-capable. For a complete description, refer to the appropriate table.

- Command Register (04h) (Table 26)
- Status Register (06h) (Table 27)
- Memory Base Address Register Bit Definition (Table 28)
- Interrupt Register (3Ch) (Table 29)
- Interrupt Control/Status Register (48h) (Table 30)
- Optional Hot-Swap Register (80h) (Table 31)

Table 26 • **Command Register 04 Hex**

Bit(s)	Type	Description
0	RW	I/O Space A value of 0 disables the device's response to I/O space addresses. Set to 0 after reset.
1	RW	Memory Space A value of 0 disables the device's response to memory space addresses. Set to 0 after reset.
2	RW	Bus Master When set to 1, this bit enables the macro to behave as a PCI bus Master. For Target-only implementation, this bit is read-only and is set to 0.
3	RO	Special Cycles Response to special cycles is not supported in the core. Set to 0.
4	RO	Memory Write and Invalidate Enable Memory Write and Invalidate Enable is not supported by the core. Set to 0.
5	RO	VGA Palette Snoop Assumes a non-VGA peripheral. Set to 0.
6	RW	Parity Error Response When 0, the device ignores parity errors. When 1, normal parity checking is performed. Set to 0 after reset.
7	RO	Wait Cycle Control No data-stepping supported. Set to 0.
8	RW	SERRN Enable When 0, the SERRN driver is disabled. Set to 0 after reset.
9	RO	Set to 0. Only fast back-to-back transactions to the same agent are allowed.
10	RW	Interrupt Disable When set, this prevents the core from asserting its INTAn output. This bit is set to 0 after reset.
15:11	RO	Reserved. Set to '00000'.

Table 27 • **Status Register 06 Hex**

Bit(s)	Type	Description
2:0	RO	Reserved. Set to '000'.
3	RO	Interrupt Status This bit reflects the status of the INTAn output.
4	RO	Capabilities List This is set to 1. CorePCIF implements a vendor capability ID and optional hot-swap capability.
5	RO	66 MHz Capable Set to 1 to indicate a 66 MHz Target, or 0 to indicate a 33 MHz Target. The value is set by the MHZ_66 parameter.
6	RO	UDF Supported Set to 0 (no user definable features).
7	RO	Fast Back-to-Back Capable Set to 0 (fast back-to-back to same agent only).

Table 27 • Status Register 06 Hex *(continued)*

Bit(s)	Type	Description
8	RW	Data Parity Error Detected If the Master controller detects a PERRn, this bit is set to 1. This bit is read-only in Target-only implementations and is set to 0. It is cleared by writing a '1'.
10:9	RO	DEVSELn timing Set to '10' (slow DEVSELn response).
11	RW	Signaled Target Abort Set to 0 at system reset. This bit is set to 1 by internal logic whenever a Target abort cycle is executed. It is cleared by writing a '1'.
12	RW	Received Target Abort If the Master controller detects a Target Abort, this bit is set to 1. This bit is read-only in Target-only implementations and is set to 0. It is cleared by writing a '1'.
13	RW	Received Master Abort If the Master controller performs a Master Abort, this bit is set to 1. This bit is read-only in Target-only implementations and is set to 0. It is cleared by writing a '1'.
14	RW	Signaled System Error Set to 0 at system reset. This bit is set to 1 by internal logic whenever the Target asserts the SERRn signal. It is cleared by writing a '1'.
15	RW	Detected Parity Error Set to 0 at system reset. This bit is set to 1 by internal logic whenever a parity error, address, or data is detected, regardless of the value of bit 6 in the command register. It is cleared by writing a '1'.

Table 28 • Base Address Registers (memory) 10 Hex to 24 Hex

Bit(s)	Type	Description
0	RO	Memory Space Indicator. Set to 0.
2:1	RO	Set to '00' to indicate anywhere in 32-bit address space.
3	RO	Prefetchable. Set by the BAR*i*_PREFETCH parameter.
31:4	RW/RO	Base Address. Depending on the BAR*i*_ADDR_WIDTH parameter, these bits may be writable or read-only. If a 128 kB address space is set (BAR*i*_ADDR_WIDTH = 17), bits 31:17 will be readable/writable, and bits 16:4 will be read-only and set to 0Sets the PCI base address of buffer memory (BAR0) or the DMA registers (BAR 1).

The base address registers (I/O), 10 hex to 24 hex, are automatically set based on the core configuration.

Table 29 • Expansion ROM Address Register 30 Hex

Bit(s)	Type	Description
0	RO	Expansion ROM enable bit. Set by the EXPR_ENABLE parameter Expansion ROM is not implemented, so bit 0 is hard coded to 0.
31:11	RW	Base Address. Depending on the EXPR_ADDR_WIDTH parameter, these bits may be writable or read-only. If a 64 kB address space is set (EXPR_ADDR_WIDTH = 16), bits 31:16 will be readable/writable, and bits 15:11 will be read-only and set to 0.

Table 30 • Capabilities Pointer 34 Hex

Bit(s)	Type	Description
7:0	R	For Target-only cores, this will be set to 44 hex. If a Master function is implemented, it will be set to 48 hex or 4C hex.

Table 30 • **Capabilities Pointer 34 Hex**

Bit(s)	Type	Description
31:8	RO	Reserved. Set to 0.

Table 31 • **Interrupt Register 3C Hex**

Bit(s)	Type	Description
7:0	RW	Required read/write register. This register has no impact on internal logic.
31:8	RO	Set to 01 hex to indicate INTAn.

Table 32 • **Hot-Swap Capability Register 40 Hex**

Bit(s)	Type	Description
7:0	RO	Hot-swap capability. Set to 06 hex.
15:8	RO	Next Capability Pointer. Set to 00 hex.
16	RO	Device Hiding Arm (DHA) Since the core only supports programming interface 0, this is '0'.
17	RW	ENUM# Signal Mask (EIM) When 1, the HS_ENUMn output is held inactive.
18	RO	Pending Insert or Extract (PIE) Set when bit 22 or 23 is set.
19	RW	LED On/Off (LOO)
21:20	RO	Programming interface (PI) Fixed at '00', programming interface 0 supports INS, EXT, LOO, and EIM.
22	RW	ENUM# Status – Extraction (EXT) When set to 1, indicates that the board is about to be extracted. Writing a '1' clears this bit.
23	RW	ENUM# Status – Insertion (INS) When set to 1, indicates that the board has just been inserted. Writing a '1' clears this bit.
31:24	RO	Reserved. Set to 0.

Table 33 • **Microsemi Capabilities Register 44, 48, or 4C Hex**

Bit(s)	Type	Description
7:0	RO	Microsemi vendor capability. Set to 09 hex.
15:8	RO	Next Capability Pointer. Set to 40 hex.
23:16	RO	Capability Size. Set to 8, 12, 20, or 24, depending on core configuration.
25:24	RO	DMA_REG_LOG Indicates the DMA register location. • 0: DMA registers are not implemented. • 1: DMA registers are only mapped in the PCI configuration space. • 2: DMA registers are mapped to memory locations 50–5F hex of the BAR, indicated by bits 28:26. • 3: DMA registers are mapped to I/O locations 50–5F hex of the BAR, indicated by bits 28:26. These two bits are set by the DMA_REG_LOC parameter.
28:26	RO	Indicates which BAR is used to access the DMA registers if mapped to memory or I/O space. These three bits are set by the DMA_REG_BAR parameter.
29	RO	Indicates that the backend interface is enabled and has access to the DMA registers. This bit is set by the BACKEND parameter.

Table 33 • Microsemi Capabilities Register 44, 48, or 4C Hex

Bit(s)	Type	Description
30	RO	When set, indicates that the BAR overflow logic in the core is disabled. Burst accesses will simply wrap within the BAR. This bit is set by the DISABLE_BAROV parameter.
31	RO	When set, indicates that the watchdog timer in the core is disabled. The core may insert more than the allowed number of wait cycles during a transfer. This bit is set by the DISABLE_WDOG parameter.

Table 34 • Interrupt Control Register 48 Hex (MASTER = 0)

Bit(s)	Type	Description
9:0	RO	Reserved. Set to 0.
10	W	Flush Internal FIFOs Only has an effect when the FIFO recovery logic is enabled. When written with a '1', all the internal FIFOs will be flushed. When the FIFOs are flushed, any data that was stored in them will be lost. Always returns 0 when read.
13:11	RO	Reserved. Set to 0.
14	RW	External Interrupt Status A '1' in this bit indicates an active external interrupt condition (assertion of EXT_INTn). It is cleared by writing a '1' to this bit. It is set to 0 after reset.
15	RW	External Interrupt Enable Writing a '1' to this bit enables support for the external interrupt signal. Writing a '0' to this bit disables external interrupt support.
31:16	RO	Reserved. Set to 0.

Table 35 • PCI Address Register 50 Hex

Bit(s)	Type	Description
1:0	RW	These two bits set the lowest two bits of the PCI address. For normal DWORD-aligned transfers, these two bits should be '00'. They may be set to non-zero values to alter the requested burst order for memory accesses or to specify a byte address for I/O accesses.
31:2	RW	This location contains the PCI start address and will increment during the DMA transfer. If using 64-bit transfers, bit 2 should also be set to 0.

Table 36 • Backend Address Register 54 Hex (ENABLE_DIRECTDMA = 0)

Bit(s)	Type	Description
1:0	RO	Set to '00'. PCI transfers must be on DWORD boundaries.
31:2	RW/ RO	This location contains the backend start address and will increment during the DMA transfer. The width of this register depends on the MADDR_WIDTH parameter. If MADDR_WIDTH is set to 20, bits 31:20 of this register are read-only and set to 0. If using 64-bit transfers, bit 2 should be set to 0.

Table 37 • Backend Address and Data Register 54 Hex (ENABLE_DIRECTDMA = 1)

Bit(s)	Type	Description
31:0	RW	When DMA_BAR = '111' (<Blue>Table 39 on page 39), this register contains the 32-bit data value that will be written to or read from the PCI bus. When DMA_BAR ≠ '111', this specifies the backend address. The core will ignore bits 1 and 0 to align the transfer count to a DWORD boundary. If using 64-bit transfers, bit 2 should also be set to 0.

Table 38 • **DMA Transfer Count 58 Hex**

Bit(s)	Type	Description
31:0	RW	Specifies the size of a DMA transfer in bytes. For 32-bit operation, this should be a multiple of four, and for 64-bit operations, a multiple of eight. Bits 1:0 are read-only and return 0. The maximum transfer size is set by the DMA_COUNT_WIDTHMEMORY_SIZE parameter. When set to zero, $2^{DMA\_COUNT\_WIDTHMEMORY\_SIZE}$ bytes will be transferred.

Table 39 • **DMA Control Register 5C Hex**

Bit(s)	Type	Description
1:0	RW	DMA Status 00: No Error 01: Master Abort 10: Parity Error 11: Target Abort
2	RW	DMA Done A '1' indicates that the DMA transfer is complete. Writing a '0' clears this bit.
3	RW	DMA Request Writing a '1' will initiate a DMA transfer, and the bit will remain set until the DMA transfer completes or an error occurs (Master abort or Target abort). This bit can only be set if the bus Master enable bit is set in the PCI Command register (<Blue>Table 26 on page 35).
7:4	RW	Cycle Type Sets the DMA transfer type and direction. These four bits directly set the PCI transfer type. Any of the sixteen PCI commands may be used, but the recommended commands are as follows: 0010 Data is moved from the PCI bus to the backend. An I/O Read command is used on the PCI bus. 0011 Data is moved from the backend to the PCI bus. An I/O Write command is used on the PCI bus. 0110 Data is moved from the PCI bus to the backend. A Memory Read command is used on the PCI bus. 0111 Data is moved from the backend to the PCI bus. A Memory Write command is used on the PCI bus. 1010 Data is moved from the PCI bus to the backend. A Configuration Read command is used on the PCI bus. 1011 Data is moved from the backend to the PCI bus. A Configuration Write command is used on the PCI bus. 1100 Data is moved from the PCI bus to the backend. A Memory Read Multiple command is used on the PCI bus.
8	RW	DMA Enable This bit must be set to 1 to enable any DMA transfers.
9	RW	Transfer Width Writing a '1' to this bit enables a 64-bit memory transaction. For 32-bit cores, this bit is read-only and is set to 0.
10	W	Flush Internal FIFOs Only has an effect when the FIFO recovery logic is enabled. When written with a '1', all internal FIFOs will be flushed. When the FIFOs are flushed, any data that was stored in them will be lost. Always returns 0 when read.
11	RO	Reserved. Returns 0.

Table 39 • **DMA Control Register 5C Hex *(continued)***

Bit(s)	Type	Description
12	RW	DMA Interrupt Status A '1' in this bit indicates that the DMA cycle has completed and the interrupt is active. It is cleared by writing a '1' to this bit. Set to 0 after reset.
13	RW	DMA Interrupt Enable Writing a '1' to this bit enables the DMA Complete interrupt. Set to 0 after reset.
14	RW	Backend Interrupt Status A '1' in this bit indicates an active backend interrupt condition (backend assertion of EXT_INTn). It is cleared by writing a '1' to this bit. Set to 0 after reset. This bit can only be set when the backend interrupt is enabled (bit 15).
15	RW	Backend Interrupt Enable Writing a '1' to this bit enables the backend interrupt. Writing a '0' to this bit disables backend interrupt support.
23:16	RW	Byte Enables These eight bits directly set the byte enable values that will be used during the DMA transfer. When bit 16 is 0, CBEN[0] will be active (LOW). Bit 17 controls CBEN[1], etc. In 32-bit cores, bits 23:20 are read-only and return 0. For normal burst DMA transfers, these bits should be set to 0.
25:24	RO	Reserved. Set to 0.
28:26	RW	DMA BAR Select Used to select which of the backend memory BARs the DMA will address. These bits are used to drive the DMA_BAR and BAR_SELECT outputs during the DMA transfer. When set to '000', BAR 0 will be selected. When set to '110', the Expansion ROM will be selected. When set to '111', a direct DMA access will be done. Data will be read from and written to the DMA data registers (54h), and no backend cycle will be carried out. When direct DMA mode is used, the transfer count register (58h) must be programmed to transfer one DWORD (0004 hex). The transfer width must be set to 32 bits.
31:29	RW	Maximum Burst Length When set to '000', the Master controller will attempt to complete the requested transfer in a single burst. When set to a non-zero value, the Master will automatically break up long bursts and limit burst transfer lengths to 2^{n-1}, where n is the decimal value of bits 31:29. Therefore, maximum transfer lengths can be limited to 1, 2, 4, 8, 16, 32, or 64 dataphases. For example, if the maximum burst length is set to '101' (16 transfers), a 1,024-DWORD transfer count would be broken up into 64 individual PCI accesses.

Note: During a DMA transfer (while bit 3, DMA Request, is set), the user should avoid writing to this register altogether, as this will interrupt the DMA transfer process. User's should poll for bit 3 cleared and then proceed with register manipulation.

11 Testbench Operation

Two testbenches are provided with CorePCIF_AHB.

- VHDL user testbench: Simple-to-use testbench written in VHDL. Additional optional tests are included, allowing verification of the AHB interface logic. This testbench is intended for customer modification.
- Verilog user testbench: Simple-to-use testbench written in Verilog. This testbench is intended for customer modification.

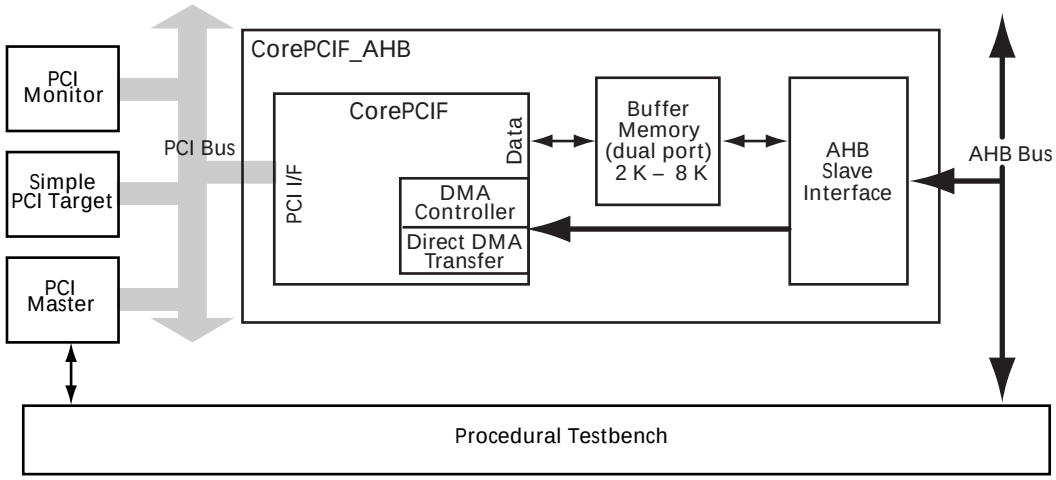
The underlying function in CorePCIF_AHB is CorePCIF. This implements all the PCI Target and Master functions. A complete verification environment is provided with CorePCIF for the PCI functionality. The VHDL user testbenches provided with CorePCIF_AHB exercise the additional AHB functions implemented here.

11.1 User Testbench

The user testbenches are intended to act as a starting point for creating a simulation environment for the end-user circuit, and are provided in both VHDL and Verilog. The testbench structure and tests carried out are identical for the VHDL and Verilog testbenches.

The testbench structure is shown in Figure 8. It instantiates a single core that is connected to the PCI bus.

Figure 8 • **User Testbench**



Also attached to the PCI bus are a PCI monitor that displays the PCI bus activity and a simple PCI Target model that is used as a Target when CorePCIF_AHB is carrying out DMA activity. The PCI Master module is used by the procedural testbench to carry out PCI cycles. The procedural testbench can also initiate AHB cycles to access the AHB Slave interface within CorePCIF_AHB.

11.1.1 Files Used in the User Testbenches

Table 40 lists all the VHDL and Verilog source files used in the user testbenches and gives a description of their functions. All source files are provided with the RTL release. With the Evaluation release, only some of the source files are provided. All others are pre-compiled into the CorePCIF_AHB simulation library.

Table 40 • **User Testbench Source Files**

File	Supplied	Function
tb_amba.vhd tb_amba.v	Yes	Top level of testbench. Creates a PCI bus and instantiates all the devices connected to the bus. It also contains the procedural testbench.

Table 40 • **User Testbench Source Files**

File	Supplied	Function
coreparameters.vhd coreparameters.v	Yes	Package or include file used to configure the core instantiated in the PCISYSTEM module. The testbench file uses this to decide which tests to run. This file is auto-generated by Libero during the core generation, and the settings will match those set in the Configuration GUI.
pcimaster.vhd pcimaster.v	RTL and Obfuscated only	PCI Master function used by the testbench to carry out PCI cycles
pcitarget.vhd pcitarget.v	RTL and Obfuscated only	Simple PCI Target function that implements a PCI Target capable of responding to memory read and write cycles
pcimonitor.vhd pcimonitor.v	RTL and Obfuscated only	PCI bus monitor that monitors PCI activity, looking for illegal activity. Also capable of tracing and displaying PCI activity.
textio.vhd	RTL only	VHDL package that provides the printf function used in the testbench. Not required for the Verilog version.
misc.vhd	RTL only	VHDL package that provides some very low-level type definitions and functions. Not required for the Verilog version.
ambasupport.vhd ambasupport.v	RTL only	Package/include file providing low-level routines that allow the testbench to carry out AHB cycles
tbtests.vhd	RTL only	VHDL package containing additional test sequences used for core verification

11.1.2 Testbench Operation

When the testbench starts, it initially reads the vendor and device IDs from the core and verifies that they are defined by the constants in the *coreconfig* file. It then sets up the PCI configuration space. This sequence is shown in Figure 9.

Figure 9 • **User Testbench Startup Sequence**

```
# CorePCIF_AHB User Testbench - Microsemi

#
#  Core Configuration
#  Configured from coreparameters
#  PCI_MASTER       1
#  PCI_TARGET       1
#  ENABLE_MEMORY    1
#  ENABLE_DIRECTDMA 1
#  AHB_START_DMA    1
#  PCI_START_DMA    1
#  MEMORY_SIZE      11
#  SLOW_READ        0
#  CLOCKS_ASYNC     0
#  GENERATE_PCICLK  0
#
##########################################
# Reading Device & Vendor IDs
# PCI CONFIG Read Slot:1 AD:00000000
# Device Vendor ID 600011AA
# PCI CONFIG Read Slot:1 AD:0000002C
# SubSystem Device Vendor ID 600011AA
# PCI CONFIG Write Slot:1 AD:00000010
# PCI CONFIG Read Slot:1 AD:00000010
# INFO: BAR0 indicates memory size is 2048 bytes
##########################################
#
##########################################
```

While these transfers are being carried out, the PCI monitor function logs all PCI bus transactions.

Once configured, the testbench will perform the sequence of tests in Table 41. If the core configuration, as set in Configuration window, does not support the required function, the test will not be performed.

Table 41 • **User Testbench Test Sequence**

Test	Required Core Parameters	Description
0	PCI_Target = 1	PCI device and vendor IDs are verified and the configuration space initialized.
1	PCI_Target = 1 ENABLE_MEMORY = 1	Single-cycle Target write and read cycle to BAR 0memory buffer.
2	PCI_Target = 1 ENABLE_MEMORY = 1	Burst Target write and read cycle to BAR 0memory buffer.
3	PCI_TARGET = 1 PCI_MASTER = 1 ENABLE_MEMORY = 1 PCI_START_DMA = 1	DMA transfer test initially from BAR 0the memory buffer to the PCI bus (the simple Target). The DMA access is initiated by the testbench using the PCI Master to write to the DMA registers. A second DMA transfer is then performed to move the data back from the PCI bus to a different location in BAR 0the memory buffer. Finally, the resultant data is verified.
4	ENABLE_MEMORY = 1	AHB read and write access to buffer memory.
5	AHB_START_DMA = 1 PCI_MASTER = 1	AHB access to PCI configuration and DMA registers.
6	None	AHB control and status register testing.

Additional verification tests can be run by typing **runall.do** at the Model*Sim* prompt. This will invoke the simulation multiple times using different core configurations (not those set in Configuration window), and will also enable additional tests.

11.1.3 Customizing the User Testbenches

VHDL User Testbench Procedures and **Verilog User Testbench Procedures** list all the procedure calls using the VHDL and Verilog testbenches. It is recommended that the *testbenchtb_amba.vhd (.v)* file be read carefully to fully understand testbench operation.

12 Implementation Hints

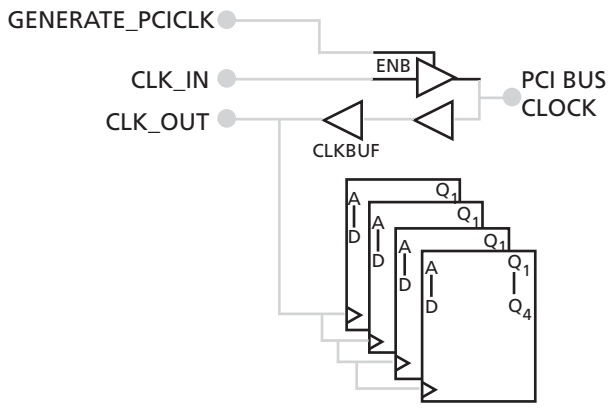
12.1 Clocking

CorePCIF_AHB supports generating the PCI clock when the FPGA is also the main bus control function. It is important that the clock networks be configured correctly to allow the core to meet the PCI setup and hold times, as well as to avoid internal clock skew.

If HCLK and the PCI clock are synchronous, Microsemi recommends using the CLK_OUT signal to directly drive the HCLK network, to minimize clock skew.

When generating the PCI clock, the clock source should be connected to the CLK_IN port. This is then routed to the PCI clock pad to drive the PCI bus, and driven back into the core using a global network. This global network on the CLK_OUT port should be used to clock the rest of the FPGA logic running off the PCI clock network (Figure 10).

Figure 10 • **Clock Generation**



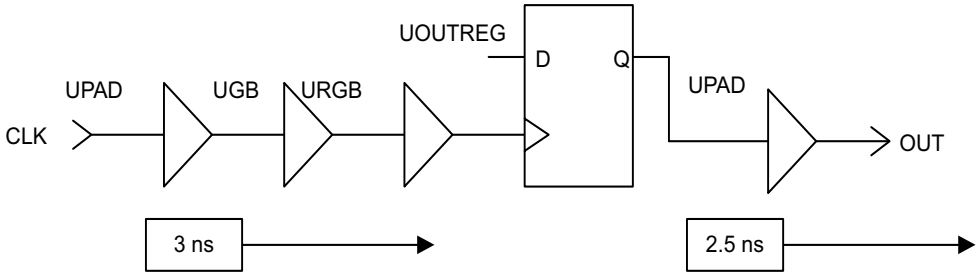
12.1.1 Example: Clocking in SmartFusion2

In the SmartFusion2 device, when using 66 MHz configuration, the CorePCIF PCI CLK input signal should be driven by a Fabric CCC (FCCC). You must instantiate the FCC in the top-level design and use the external CLK input as a reference clock. This is to ensure that external setup and clock to out timing are met in the design.

Note: The data paths (for example, AD) use registers to drive the output and you need to use IO-REG combining for these register outputs.

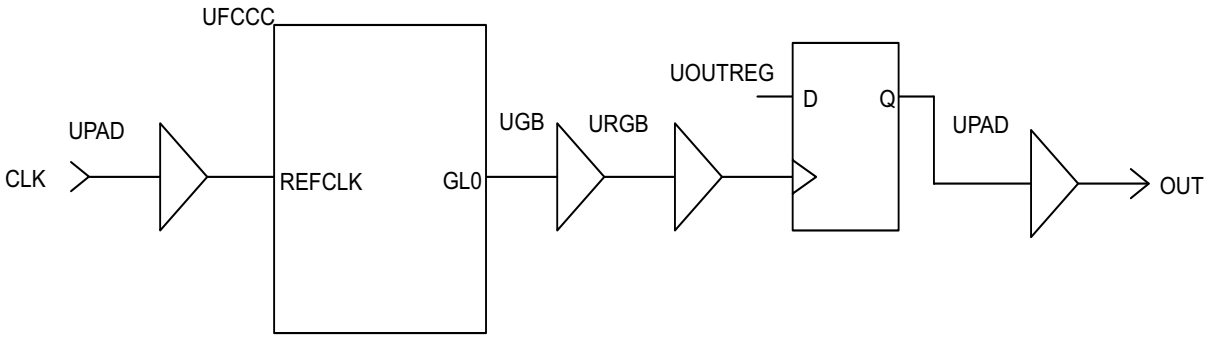
Figure 11 shows the clock-to-out path in SmartFusion2, which has a requirement of 6ns (as part of the PCIF standard. In the SmartFusion2 device, the sum of UPAD, UGB, and URGB on the clock input is around 3ns, if you use the IO bank as shown in **Pin Assignments**. The output pad delay is roughly 2.5ns. This leaves only around 500ps to route and register the output internally. This does not allow enough margin to achieve timing closure in most cases.

Figure 11 • **Clocking in SmartFusion2 with No CCC**



To increase the timing margin, it is advised to use an FCCC in the CLK data path to shift the clock backward, as shown in Figure 12.

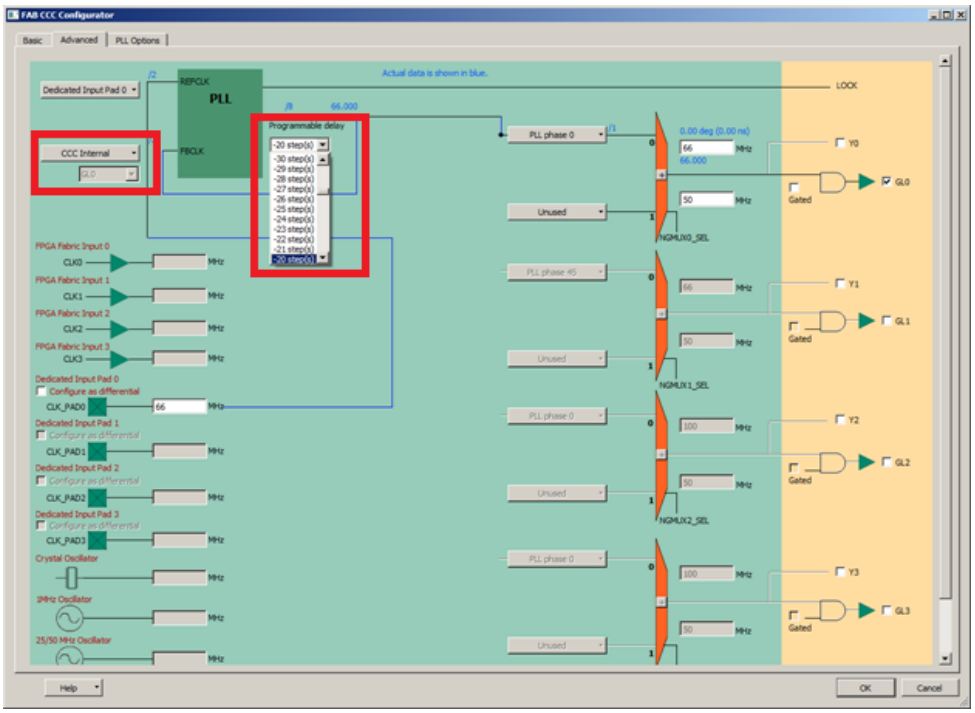
Figure 12 • **Clocking in SmartFusion2 with CCC**



Using an FCCC, the clock can be shifted forward or backward. In this particular case, the clock should be shifted backwards, which will afford a large margin on clk-out path to the user. In the Libero SoC software, this is achieved using the programmable delay element in the FCCC output. Figure 13 shows how to configure the FCCC with a negative delay.

Note: The feedback must be set to CCC Internal. Otherwise, the reference clock and the output clock GL0 will remain in phase.

Figure 13 • **FCCC Configuration in SmartFusion2**



You must simply use the FCCC output to drive the CorePCIF design at the top-level, and any other logic in the FPGA required as well (unless the CLK_OUT as described above is used).

12.2 Clock and Reset Networks

The core includes global buffers for both the PCI clock and reset inputs. The buffered versions of these signals are provided on the CLK_OUT and RST_OUTN ports. These should be used for clocking and resetting any additional logic included in the FPGA running of the PCI clock.

The core also uses two additional global resources for internal routing of the high-fanout TRDYN and IRDYN nets, if required. Target cores will require IRDYN to be routed on a global; if the Master function is implemented, the TRDYN net will be routed on a global network.

12.3 Assigning Pin Layout Constraints

You can assign pins manually with the PinEditor tool or import them directly into Designer from the corresponding pin constraint file (PDC file).

12.4 Pin Assignments

To be able to meet the critical PCI setup, hold, and clock-to-out requirements, it is critical that the PCI pin locations be assigned correctly. Two aspects need to be considered:

1. Pin assignments should minimize FPGA place-and-route issues. Pin assignment is extremely important in meeting the PCI setup, hold, and clock-to-out requirements.
2. Pin assignments should minimize PCB layout issues. The PCI specification limits the track lengths allowed on the PCB. CorePCIF_AHB parameter section details the requirements.

The PCI specification recommends that the pin order around the device align exactly with the add-in card (connector) pinout. The additional signals needed in 64-bit versions of the bus continue wrapping around the component in a counterclockwise direction in the same order they appear on the 64-bit connector extension. **PCI Pinout** provides details of the recommended pin order.

Example pin files are provided in the *layout* directory for some of the possible FPGA family, device, and package combinations. These can be adapted to support other device/package combinations.

Each supported FPGA family has different requirements to minimize FPGA layout issues; these are detailed below.

12.4.1 Fusion, IGLOO/e, ProASIC3L, and ProASIC3/E Families

The pins should be located around one side of the package in the order specified by the PCI specification. Initially, identify an I/O bank that contains the global inputs G***.

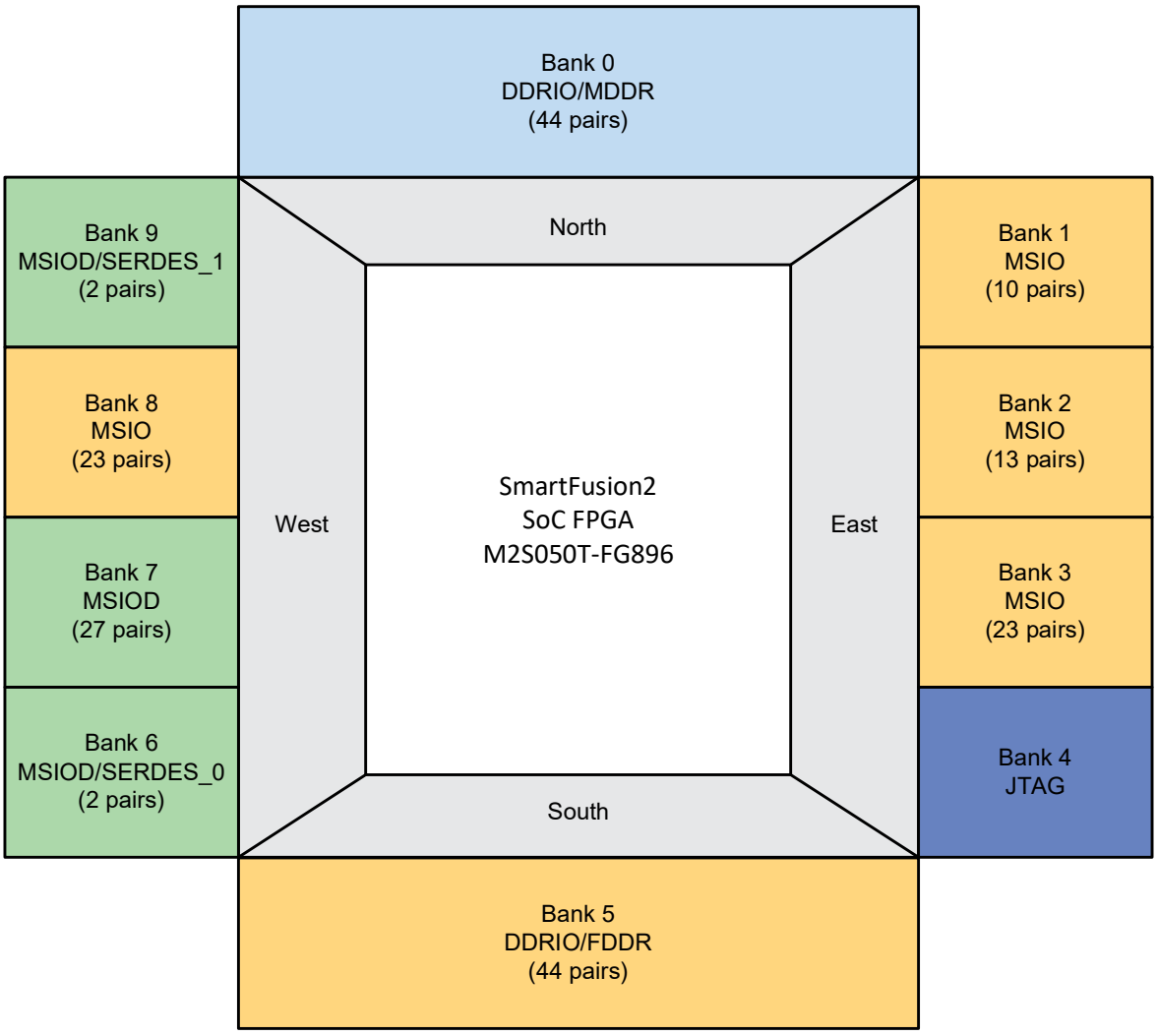
1. Assign the TRDYN and IRDYN pins to, or close to, two of these global inputs.
2. Assign the rest of the PCI pins around the package in the order that will match the add-in connector. Leave one spare normal I/O pin vacant close to the global pins. Do not use any of the special function pins.
3. For 33 MHz operation, connect the PCI CLK to a global input. For 66 MHz operation, connect the CLK to the I/O pin left vacant close to the global inputs.

Care should be taken to minimize the number of I/O banks used; the I/O banks used for PCI signals must be set to use PCI electrical levels that may be incompatible with other devices connected to the FPGA. It is recommended that the pinout chosen be verified to check that PCI timing requirements can be met before PCB layout is completed.

12.4.2 SmartFusion2

There are typically four PCI-capable banks available on most devices, that is, the MSIO banks. For example, on the M2S050T device, Banks 1, 2, 3, and 8 are PCI-capable of the PCI standard. However, to meet timing and to ease board layout, only Banks 1, 2, and 3 should be used, that is, on the east edge of the package. If using unlocked automatic pin assignment, you need to make sure that bank 8 is not used for PCI IOs.

Figure 14 • **SmartFusion2 M2S050T Device**



Note:

- Do not use bank 8 for PCI I/Os
- Use bank 1, 2, and 3 for PCI I/Os

Alternatively, this may be achieved by manually assigning I/Os to banks 1, 2, and 3. After that, achieving timing should be done either by using the provided PDC file as a reference and modifying as necessary, or by following these steps in Libero (classic constraint flow):

1. Using the following configuration and pin assignment, run an initial Place and Route **using High-Effort, Timing-driven routing**.
   - Use `-iostd` PCI to ensure that I/Os are assigned to PCI compatible pins.
   - Where possible, force the placer to use I/O registers using the `-REGISTER Yes` switch and the `-OUT_REG Yes` or `-IN_REG Yes` depending on whether the register resides in the input or output path for that particular I/O. It is possible that they are both, if both Master and Target modes are enabled.
   - Do not assign PCI I/Os to any pins.
   - Depending on the data width, TRDYN and IRDYN may be best routed through a global net. In 64-bit mode, TRDYN and IRDYN are best routed locally. In 32-bit mode, TRDYN and IRDYN are best routed globally.
2. Lock down the I/O in the **Libero I/O constraints editor** or modify the PDC file by adding the pinname the `-fixed yes` switch to each `set_io` line.
3. Run **SmartTime** to perform static timing analysis.

- Adjust the (negative) programmable delay in the FCCC (as described in **Clocking in SmartFusion2**) until External Setup violations go away in Max Delay Analysis, but not so far as to introduce Internal Setup violations.
- Adjust all input delays until hold violations on paths ending at input ports go away in Min Delay Analysis, using the -IN_DELAY x switch in the PDC.

Note: Notes on Synthesis:

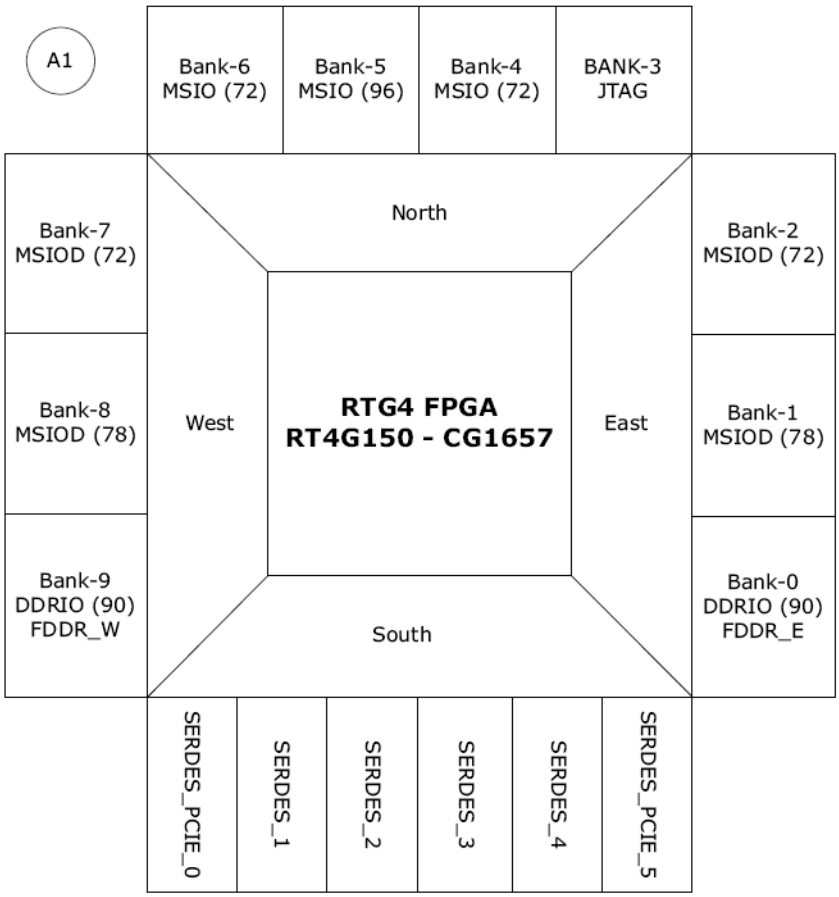
To ensure that timing is met, it is important to be able to push as many flip-flops as possible into the I/O registers. To accomplish this, keep the max fanout on output registers at 1 during synthesis. This can be accomplished using the following SDC attributes in the Synthesis constraint file (The instance name need to match with the instance name of your design):

```
define_attribute
{{i:UCORE.MAKE_TARGET.DATAPATHE64.MAKE_DATAPATH_REGISTERS.AD_REGS[31:0]}}
{syn_preserve} {1}
define_attribute {{i:UCORE.MAKE\.UDMA.CBEN_PAD[7:0]}} {syn_replicate} {1}
define_attribute {{i:UCORE.MAKE\.UDMA.CBEN_PAD[7:0]}} {syn_maxfan} {1}
define_attribute
{{i:UCORE.MAKE_TARGET.DATAPATHE.MAKE_DATAPATH_REGISTERS.AD_REGS[31:0]}}
{syn_preserve} {1}
define_attribute {{i:UCORE.MAKE_TARGET.BurstE.UA1\.MAKE_ACK64_OUT.Q_INT}}
{syn_replicate} {1}
define_attribute {{i:UCORE.MAKE\.UDMA.MAKE_REQ64N1.Q_INT}} {syn_replicate}
{1}
define_attribute {{i:UCORE.MAKE\.UDMA.MAKE_REQ64N1.Q_INT}} {syn_maxfan} {1}
define_attribute {{i:UCORE.MAKE_TARGET.BurstE.UA1\.MAKE_ACK64_OUT.Q_INT}}
{syn_maxfan} {1}
define_attribute {{i:UCORE.MAKE_TARGET.BurstE.UM1\.MAKE_IRDY_OUT.Q_INT}}
{syn_replicate} {1}
define_attribute {{i:UCORE.MAKE_TARGET.BurstE.UM1\.MAKE_IRDY_OUT.Q_INT}}
{syn_maxfan} {1}
```

12.4.3 RTG4

Three PCI capable banks are available on most of the RTG4 devices, that is, MSIO banks. For example, on RT4G150-CG1657 device, Banks 4, 5, and 6 are PCI capable. To meet timing and ease board layout consecutive MSIO banks (bank 4, bank 5, and bank 6) should be used.

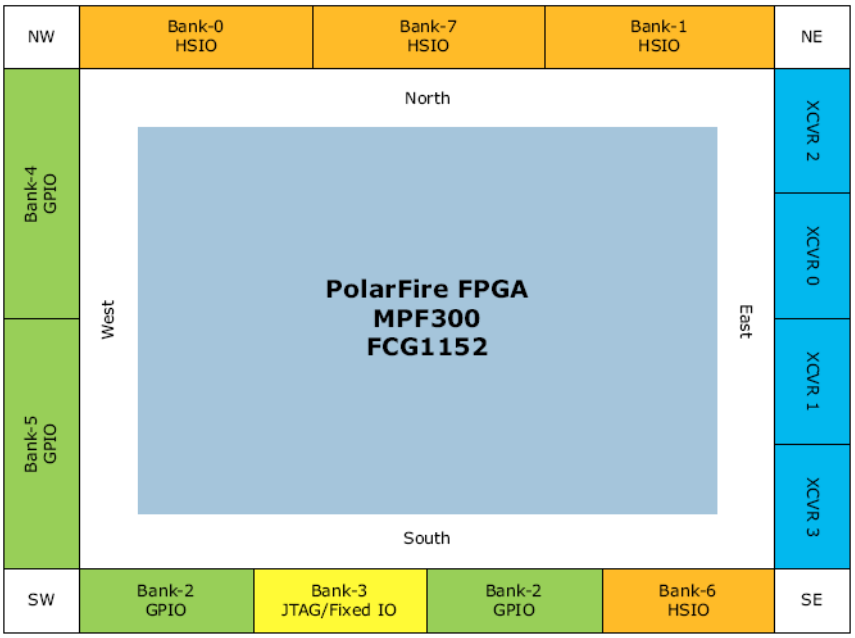
Figure 15 • **RTG4 RT4G150-CG1657 Device**



12.4.4 PolarFire

Three PCI capable banks are available on most of the PolarFire devices, that is, GPIO banks. For example, on MPF300-FCG1152 device, Bank 2, 4, and 5 are PCI capable. To meet timing and ease board layout consecutive GPIO banks (bank 4 and bank 5) should be used.

Figure 16 • **PolarFire MPF300-FCG1152 Device**



12.4.5 Meeting PCI Hold Requirements

The PCI hold time requirements should be checked post-layout. These can easily be found using the Minimum Delay Analysis View in the Timing Analyzer. All the hold times should be less than 0 ns. If any of the PCI inputs violate the hold time requirements, one of the following methods can be used to insert extra delay in the datapath to correct the hold time:

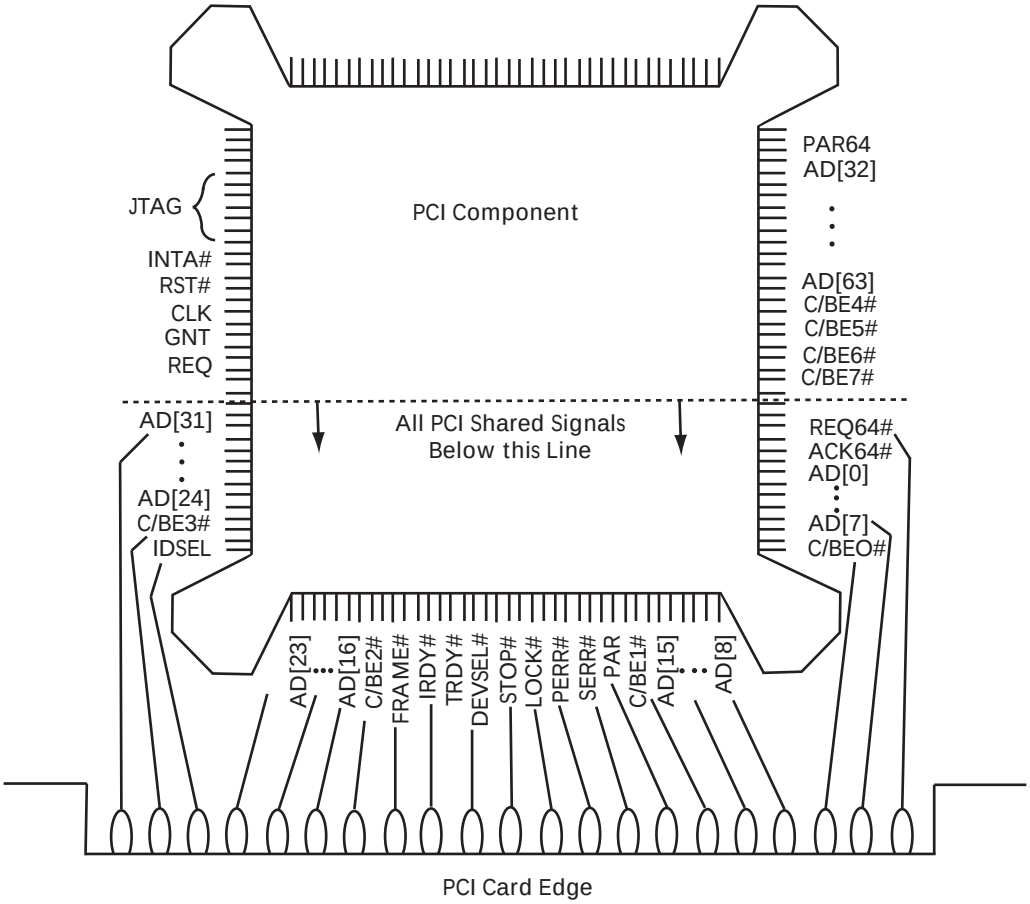
1. Modify the RTL source code, if available, to insert BUFD cells between the IOPAD and the registers violating hold time requirements. This can be done easily in the DEL_BUFF module, which allows the number of delay buffers inserted on each PCI input to be specified. Re-run synthesis and layout.
2. For families that support programmable input delays (ProASIC3E, IGLOOe, SmartFusion2, IGLOO2, RTG4, and PolarFire), the I/O pad can be configured to insert additional delay.[1] This is a good way to correct hold problems on the AD bus; however, adding additional input buffer delays on the control inputs TRDYN, IRDYN, FRAMEN, and so on, may cause other endpoints from these inputs to violate the PCI setup times.
3. Export a netlist from Designer. Modify the netlist to insert BUFD cells between the IOPAD and the registers violating hold time requirements. Re-run layout with the incremental layout feature enabled.
4. Using ChipPlanner, move registers that have a hold time violation away from the I/O pad to increase the delay and fix the hold time violation. Re-run layout with the incremental layout feature enabled.

1. Use PinEditor to select the I/O bank. Right-click the colored I/O bank in the GUI to open the Configure I/O Bank dialog box. Once you set the bank delays, you can set the input delays on all PCI pins.

13 PCI Pinout

Figure 17 shows the recommended pin ordering around the package.

Figure 17 • **Recommended PCI Pin Ordering**



14 Synthesis Timing Constraints

The required timing constraints are given in Table 42.

Table 42 • **Synthesis Timing Constraints**

Frequency (MHz)	PCI Specification (ns)			Synplicity Constraints (ns)		
	Ports	**Setup**	**Clock to Output**	**Period**	**Input Delay**	**Output Delay**
33	AD CBEN DEVSELN FRAMEN IRDYN TRDYN PAR PERRN SERRN STOPN TRDYN PAR64 ACK64N REQ64N	7	11	30	23	19
	IDSEL GNTN	10			20	
	INTAN REQN		11			19
66	AD CBEN DEVSELN FRAMEN IRDYN TRDYN PAR PERRN SERRN STOPN TRDYN PAR64 ACK64N REQ64N	3	6	15	12	9
	IDSEL GNTN	5			10	
	INTAN REQN		6			9

15 Place-and-Route Timing Constraints

The required timing constraints are given in Table 43.

Table 43 • **Place-and-Route Timing Constraints**

Frequency (MHz)	PCI Specification (ns)				Designer Constraints (ns)			
	Ports	**Setup**	**Hold**	**Clock to Output**	**Period**	**Input Delay**	**Input Hold**	**Output Delay**
33	AD CBEN DEVSELN FRAMEN IRDYN TRDYN PAR PERRN SERRN STOPN TRDYN PAR64 ACK64N REQ64N	7	0	11	30	23	0	19
	IDSEL GNTN	10	0			20	0	
	INTAN REQN			11				19
66	AD CBEN DEVSELN FRAMEN IRDYN TRDYN PAR PERRN SERRN STOPN TRDYN PAR64 ACK64N REQ64N	3	0	6	15	12	0	9
	IDSEL GNTN	5	0			10	0	
	INTAN REQN			6				9

Note: Timing closure at 66MHz is highly impacted by the other aspects of the design and resource allocation. For highly congested designs it may not be possible in some cases to achieve a 66MHz solution. Priority should be given to closing timing on the CorePCIF_AHB before moving onto other areas of the design. To close the timing of CorePCIF_AHB, the output delay value should be adjusted with respect to the PCI device (on board) timing constraints at system level.

16 VHDL User Testbench Procedures

The following is a list of the supported procedure calls in the VHDL user testbench. Microsemi recommends that you examine the *testbench.vhd* file to understand how to use these procedure calls.

```
config_write(SLOT,ADDRESS,DATA_DW ,PCICMD,PCISTAT,MSETUP);
config_write(SLOT,ADDRESS,DATA_INT ,PCICMD,PCISTAT,MSETUP);
config_write(SLOT,ADDRESS,N,DATA(0 to N-1),PCICMD,PCISTAT,MSETUP);
config_read (SLOT,ADDRESS,DATA_INT ,PCICMD,PCISTAT,MSETUP);
config_read (SLOT,ADDRESS,DATA_DW ,PCICMD,PCISTAT,MSETUP);
config_read (SLOT,ADDRESS,N,DATA(0 to N-1),PCICMD,PCISTAT,MSETUP);
memory_write(ADDRESS,DATA_DW ,PCICMD,PCISTAT,MSETUP);
memory_write(ADDRESS,DATA_INT ,PCICMD,PCISTAT,,MSETUP);
memory_write(ADDRESS,DATAH_DW,DATAL_DW,PCICMD,PCISTAT,MSETUP);
memory_write(ADDRESS,DATAH_INT,DATAL_INT,PCICMD,PCISTAT,MSETUP);
memory_write(ADDRESS,N,DATA(0 to N-1),PCICMD,PCISTAT ,MSETUP);
memory_read (ADDRESS,DATA_DW ,PCICMD,PCISTAT,MSETUP);
memory_read (ADDRESS,DATA_INT ,PCICMD,PCISTAT,MSETUP);
memory_read(ADDRESS,DATAH_DW,DATAL_DW,PCICMD,PCISTAT,MSETUP);
memory_read(ADDRESS,DATAH_INT,DATAL_INT,,PCICMD,PCISTAT,MSETUP);
memory_read (ADDRESS,N,DATA(0 to N-1),PCICMD,PCISTAT,MSETUP);
compare_data(ERRCOUNT,"Message",EXP_INT,GOT_INT);
compare_data(ERRCOUNT,"Message",EXP_DWORD,GOT_DWORD);
compare_data(ERRCOUNT,"Msg",EXP_DATA(0 to N-1),GOT_DATA(0 to N-1));
be_write(ADDRESS,DATA_DW , BYTEEN, PCICMD,PCISTAT);
be_write(ADDRESS,DATA_INT, BYTEEN, PCICMD,PCISTAT);
be_read (ADDRESS,DATA_DW , PCICMD,PCISTAT);
be_read (ADDRESS,DATA_INT, PCICMD,PCISTAT);
tb_write_ahb(ADDRESS,COUNT,DATA_DW(0 to N-1),AHB_CONTROL,AHB_STATUS)
tb_write_ahb(ADDRESS,COUNT,DATA_INT(0 to N-1),AHB_CONTROL,AHB_STATUS)
tb_write_ahb(ADDRESS,DATA_DW,AHB_CONTROL,AHB_STATUS)
tb_write_ahb(ADDRESS,DATA_INT,AHB_CONTROL,AHB_STATUS)
tb_read_ahb(ADDRESS,COUNT,DATA_DW(0 to N-1),AHB_CONTROL,AHB_STATUS)
tb_read_ahb(ADDRESS,COUNT,DATA_INT(0 to N-1),AHB_CONTROL,AHB_STATUS)
tb_read_ahb(ADDRESS,DATA_DW,AHB_CONTROL,AHB_STATUS)
tb_read_ahb(ADDRESS,DATA_INT,AHB_CONTROL,AHB_STATUS)
```

The parameters to the above procedure calls are described in Table 44. To simplify the parameters, some predefined types are used. These are defined in the *misc.vhd* package.

```
subtype NIBBLE is std_logic_vector ( 3 downto 0);
subtype DWORD is std_logic_vector (31 downto 0);
type DWORD_ARRAY is array ( INTEGER range <>) of DWORD;
```

Table 44 • **Procedure Call Parameters**

Parameter	Type	Description
SLOT	INTEGER	PCI slot number to use for configuration cycles When 0, will set the eight upper address bits to 01 hex. When 1, will set the eight upper address bits to 02 hex, etc. The testbench connects address bit 25 to the core IDSEL input; therefore, the slot number should be set to 1.
ADDRESS	INTEGER	Address for the PCI cycle
DATA_DW	DWORD	Data word
DATAH_DW	DWORD	Upper 32 bits of a 64-bit data word

Table 44 • **Procedure Call Parameters *(continued)***

Parameter	Type	Description
DATAL_DW	DWORD	Lower 32 bits of a 64-bit data word
DATA_INT	INTEGER	Data word
DATAH_INT	INTEGER	Upper 32 bits of a 64-bit data word
DATAL_INT	INTEGER	Lower 32 bits of a 64-bit data word
PCICMD	TPCICMD	Record used to communicate within the testbench. Allows the procedure to start the PCI Master cycle.
PCISTAT	TPCISTAT	Record used to communicate within the testbench. Allows the procedure to monitor the PCI Master cycle.
MSETUP	TMSETUP	Record used to set the Master transfer rates. This contains four fields that may be altered.

Name	Type	Description
IRDYRATE0	INTEGER	Initial delay from FRAME to IRDY assertion
IRDYRATEN	INTEGER	Subsequent delay between IRDY assertions
PCI64	BOOLEAN	Indicates whether to request a 64-bit transfer.
ERROR	TERROR	Allows errors conditions to be inserted. Should be set to “NONE” for normal operation. Supported error conditions are described in the VHDL source files.

AHB_CONTROL	T_AHB_C	Record structure used by testbench to control AHB interface
AHB_STATUS	T_AHB_S	Record structure used by testbench to get AHB status information

17 Verilog User Testbench Procedures

Following is a list of the supported tasks in the Verilog user testbench. Microsemi recommends that you examine the *testbench.v* file to understand how to use these tasks.

```
// PCI configuration cycles
config_write (SLOT,CADDRESS,COUNT);
config_read (SLOT,CADDRESS,COUNT);

// PCI memory cycles
memory_write (ADDRESS,COUNT,PCI64);
memory_read (ADDRESS,COUNT,PCI64);
compare_data (ERRCOUNT,COUNT);

// Writes and reads to and from the core backend interface
be_write (BADDRESS,WDATA, BYTEEN);
be_read (BADDRESS,RDATA);
// AHB read and write cycles
tb_write_ahb (ADDRESS,COUNT);
tb_read_ahb (ADDRESS,COUNT);
compare_ahb_data(ERRCOUNT,COUNT);
```

The parameters to the above tasks are described in Table 45 and Table 46. Data for the PCI configuration and PCI and AHB memory read and write cycles is passed in the pciwdata and pcirdata, or ahbwdata and ahbrdata, global arrays rather than through the task parameters.

Table 45 • **Global Descriptions**

Globals	Type	Description
pciwdata	reg [31:0] [0:31]	This is an array in which the user sets up the data that will be written before calling the memory_write or config_write tasks. For 64-bit operations, the lower DWORD is specified in the odd addresses and the upper DWORD in the even addresses.
pcirdata	reg [31:0] [0:31]	This is an array by which the memory_read and config_read functions return data. For 64-bit operations, the lower DWORD is specified in the odd addresses and the upper DWORD in the even addresses.

Table 46 • **Parameter Descriptions**

Parameters	Type	Description
SLOT	reg [2:0]	PCI slot number to use for configuration cycles. When 0, will set the eight upper address bits to 01h. When 1, will set the eight upper address bits to 02h, and so on. The testbench connects address bit 25 to the core IDSEL input; therefore, the slot number should be set to 1.
CADDRESS	reg [7:0]	Configuration space address
COUNT	reg [7:0]	Number of DWORDs to be written, read, or compared. If 64-bit operation is enabled, this must be an even number. The maximum count is thirty-two 32-bit transfers or sixteen 64-bit transfers.
ADDRESS	reg [31:0]	Memory space address
PCI64	reg	When 1, the testbench will request a 64-bit transfer.
ERRCOUNT	inout reg [31:0]	The compare routine will increment this value if it detects any errors. At the end of a test sequence, it can indicate the total number of errors.
BADDRESS	reg [7:0]	Backend address

Table 46 • **Parameter Descriptions**

Parameters	Type	Description
WDATA	reg [31:0]	Data to be written to the core backend
RDATA	output reg [31:0]	Data read from the core backend
BYTEEN	reg [3:0]	Byte enables to backend writes should be set to 4'b1111.