Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

HB0087
Handbook
CorePCIF v4.1

Power Matters.™

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

© 2017 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) 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. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. 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 8.0

Updated changes related to CorePCIF v4.1.

1.2 Revision 7.0

Updated changes related to CorePCIF v4.0.

1.3 Revision 6.0

Updated changes related to CorePCIF v3.2.

1.4 Revision 5.0

Updated changes related to CorePCIF v3.1.

1.5 Revision 4.0

Updated changes related to CorePCIF v3.0.

1.6 Revision 3.0

Updated changes related to CorePCIF v2.1.

1.7 Revision 2.0

Updated changes related to CorePCIF v2.0.

1.8 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CorePCIF v1.0

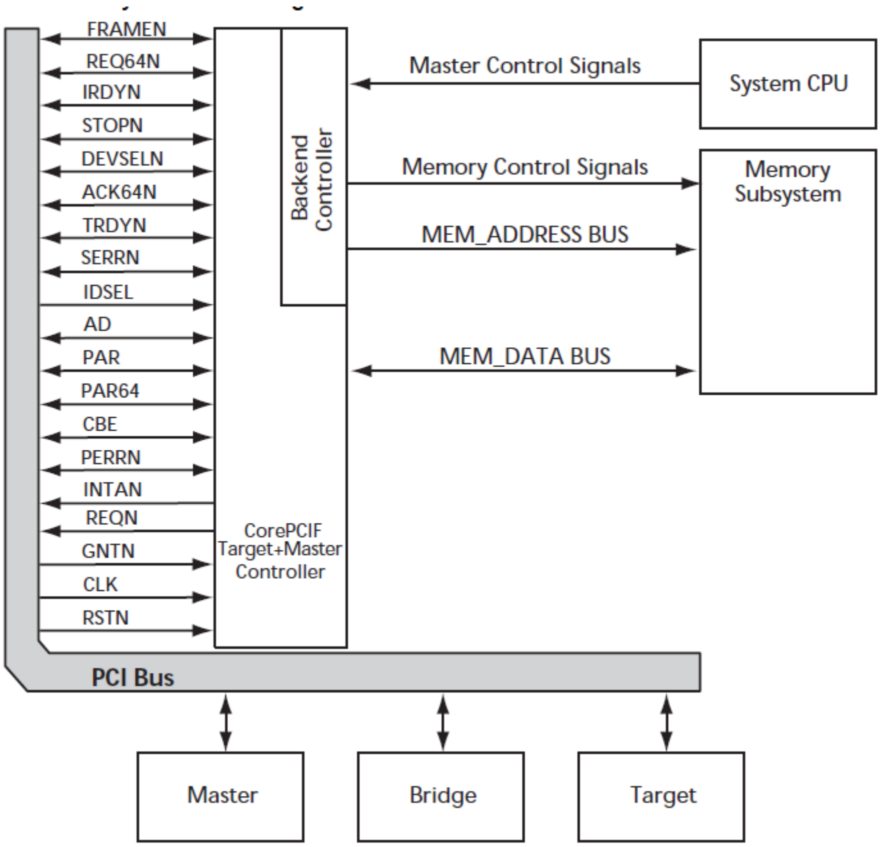
2 Introduction

CorePCIF connects memory, FIFO, and processor subsystem resources to the main system via the PCI bus. CorePCIF is intended for use with a wide variety of peripherals where high-performance data transactions are required. Figure 1 depicts typical system applications using the core. Though CorePCIF can handle any transfer rate, most applications will operate with zero wait states. When required, wait states can be inserted automatically by a slower peripheral.

CorePCIF can implement Target and/or Master functions. The Target function allows the PCI bus to access memory devices attached to the CorePCIF backend. The Master function allows CorePCIF to move data between the backend or internal registers and the PCI bus using the internal DMA engine. The DMA engine can be programmed either from the PCI bus or directly from the backend.

CorePCIF can be customized. A variety of parameters are provided to easily change features such as memory and I/O sizes along with the PCI vendor and device IDs. A single top-level core has parameters that enable and disable functions, allowing a minimal-size core to be implemented for the required functionality. The core consists of four basic units: the Target controller, the Master controller, the DMA controller, and the backend controller. The backend controller provides the necessary control for the I/O or memory subsystem, allowing external (to the core) memory and FIFOs to be directly connected to the core.

Figure 1 • **CorePCIF System Block Diagram**



2.1 Core Versions

This handbook applies to CorePCIF v4.1. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

2.2 Supported Families

- PolarFire®
- RTG4TM
- SmartFusion®2
- IGLOO®2
- SmartFusion®
- IGLOO®
- IGLOO®e
- Fusion
- ProASIC®3
- ProASIC®3E
- ProASIC®3L
- ProASICPLUS®
- Axcelerator®
- RTAX-S
- SX-A
- RTSX-S

2.3 CorePCIF Device Requirements

CorePCIF includes Target and/or Master functions. The core also has an option for a built-in DMA controller.

There are nine implementations available for the core. The SMALL32 implementation is the smallest Target core possible but does not support zero-wait-state transfers; TARG32 does support zero-wait-state transfers. MAST32 is the smallest Master-only core possible. TARGDMA32 implements a typical Target and Master function. TARGMAST32 implements a fully configured core. The remaining four implementations are 64-bit versions of the 32-bit implementations. Table 1 describes example implementations.

Table 1 • **Example Implementations**

Implementation	Description
SMALL32	32-bit Target-only core with a single base address register (BAR). The slow read function is enabled. Interrupts, BAR overflow, and hot-swap features are disabled.
TARG32	32-bit Target-only core with a single 64 kB BAR. The FIFO recovery logic is not implemented. BAR overflow logic and hot-swap features are disabled.
MAST32	32-bit Master-only core with a single 64 kB BAR. The FIFO recovery logic is not implemented. Direct DMA is enabled.
TARGDMA32	32-bit Target and Master function with a single 64 kB BAR. DMA registers are accessible from the PCI side and are memory-mapped in the second BAR. The FIFO recovery logic is not implemented. BAR overflow logic and hot-swap features are disabled. Backend access to the DMA registers is not implemented. Direct DMA is disabled.
TARGMAST32	32-bit Target and Master function with five memory BARs that have variable sizes from 64 kB to 1 GB. The DMA registers are memory-mapped to the sixth BAR. All of the memory BARs include the FIFO recovery logic. The Expansion ROM address registers are also implemented. BAR overflow logic and hot-swap features are enabled. Backend access to the DMA registers is also implemented. Direct DMA is enabled.
TARG64	64-bit Target-only core with a single 64 kB BAR. The FIFO recovery logic is not implemented. BAR overflow logic and hot-swap features are disabled. Direct DMA is disabled.
MAST64	64-bit Master-only core with a single 64 kB BAR. The FIFO recovery logic is not implemented. Direct DMA is disabled.

Table 1 • **Example Implementations**

Implementation	Description
TARGDMA64	64-bit Target and Master function with a single 64 kB memory. DMA registers are accessible from the PCI side and are memory-mapped in the second BAR. The FIFO recovery logic is not implemented. BAR overflow logic and hot-swap features are disabled. Backend access to the DMA registers is not implemented. Direct DMA is disabled.
TARGMAST64	64-bit Target and Master function with five memory BARs that have variable sizes from 64 kB to 1 GB. The DMA registers are memory-mapped to the sixth BAR. All of the memory BARs include the FIFO recovery logic. The Expansion ROM address registers are also implemented. BAR overflow logic and hot-swap features are enabled. Backend access to the DMA registers is also implemented. Direct DMA is enabled.

2.4 Utilization Statistics

Table 3 and Table 4 give the CorePCIF device utilization for both 32-bit and 64-bit implementations. The numbers in these tables are typical and will vary based on the actual core configuration and the synthesis tools used.

CorePCIF device utilization and performance varies, depending on which features are implemented. The core has approximately 50 configuration parameters.

Table 2 • **CorePCIF Utilization**

Implementation	Family	Combinational (4LUT)	Sequential (DFF)	Total	Memory Blocks	Utilization %
SMALL32	SmartFusion2 (M2S050) / IGLOO2 (M2GL050)	384	231	615	0	0.54
TARG32		497	291	788	1	0.70
MAST32		947	448	1395	1	1.24
TARGDMA32		1092	459	1551	1	1.37
TARGMAST32		1740	707	2447	1	2.18
TARG64		706	468	1174	2	1.04
MAST64		1135	635	1770	2	1.57
TARGDMA64		1169	642	1811	2	1.60
TARGMAST64		2029	932	2961	2	2.62
SMALL32	RTG4 (RTG4150)	383	231	614	0	0.20
TARG32		559	315	874	1	0.29
MAST32		1052	451	1503	1	0.49
TARGDMA32		1040	467	1507	1	0.50
TARGMAST32		1916	714	2630	1	0.86
TARG64		678	468	1146	2	0.43
MAST64		1191	634	1824	2	0.60
TARGDMA64		1251	642	1893	2	0.62
TARGMAST64		2148	920	3068	2	1.01

Table 2 • **CorePCIF Utilization**

Implementation	Family	Combinational (4LUT)	Sequential (DFF)	Total	Memory Blocks	Utilization %
SMALL32	PolarFire (MPF300T_ES)	361	231	592	0	0.10
TARG32		536	291	827	1	0.14
MAST32		893	449	1342	1	0.22
TARGDMA32		1083	459	1542	1	0.25
TARGMAST32		1722	706	2428	1	0.40
TARG64		735	468	1203	2	0.20
MAST64		1099	636	1735	2	0.29
TARGDMA64		1198	642	1840	2	0.30
TARGMAST64		1822	919	2741	2	0.46

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.

The exact parameter settings are detailed in Table 5.

Table 3 • **32-Bit CorePCIF Device Utilization**

		Cells or Tiles					
Implementation	Family	Combinatorial	Sequential	Total	Memory Blocks	Device	Utilization
SMALL32	Fusion® IGLOO®/e ProASIC®3/E ProASIC3L	544	177	721	0	AFS600	5%
TARG32		661	203	864	2	AGLE600	6%
MAST32		1,434	383	1,817	2	AGL600	13%
TARGDMA32		1,594	369	1,963	2	A3PE600	14%
TARGMAST32		2,698	658	3,356	2	A3P600	24%
SMALL32	ProASIC^{PLUS}®	658	215	873	0	APA150	14%
TARG32		716	208	924	4	APA150	15%
MAST32		1,479	422	1,901	4	APA150	31%
TARGDMA32		1,644	377	2,021	4	APA300	25%
TARGMAST32		3,020	697	3,717	4	APA300	45%
SMALL32	RTAX-S	350	178	528	0	RTAX250S	12%
TARG32		465	244	709	0	RTAX250S	17%
MAST32		799	422	1,221	0	RTAX250S	29%
TARGDMA32		867	414	1,281	0	RTAX250S	30%
TARGMAST32		2,562	2,206	4,768	0	RTAX1000S	26%
SMALL32	Axcelerator®	381	180	561	0	AX500	7%
TARG32		453	210	663	1	AX500	8%
MAST32		830	393	1,223	1	AX500	15%
TARGDMA32		874	380	1,254	1	AX500	16%
TARGMAST32		1,677	653	2,330	1	AX500	29%

Table 3 • **32-Bit CorePCIF Device Utilization *(continued)***

		Cells or Tiles			Memory		
Implementation	**Family**	**Combinatorial**	**Sequential**	**Total**	**Blocks**	**Device**	**Utilization**
SMALL32	RTSX-S	387	221	608	0	RT54SX32S	21%
TARG32		491	282	773	0	RT54SX32S	27%
MAST32		1,134	507	1,641	0	RT54SX32S	57%
TARGDMA32		966	465	1,431	0	RT54SX72S	24%
TARGMAST32		1,359	834	2,193	0	RT54SX72S	36%
SMALL32	SX-A	385	222	607	0	A54SX32A	21%
TARG32		494	285	779	0	A54SX32A	27%
MAST32		1,111	507	1,618	0	A54SX32A	56%
TARGDMA32		959	460	1,419	0	A54SX72A	24%
TARGMAST32		1,352	834	2,186	0	A54SX72A	36%

Table 4 • **64-Bit CorePCIF Device Utilization**

		Cells or Tiles			Memory		
Implementation	**Family**	**Combinatorial**	**Sequential**	**Total**	**Blocks**	**Device**	**Utilization**
TARG64	Fusion IGLOO/e ProASIC3/E ProASIC3L	930	315	1,245	4	AFS600	9%
MAST64		1,686	498	2,184	4	AGLE600 AGL600 A3PE600 A3P600	16%
TARGDMA64		1,852	484	2,336	4		17%
TARGMAST64		2,989	772	3,761	4		27%
TARG64	ProASIC^{PLUS}	961	319	1,280	8	APA150	21%
MAST64		1,770	542	2,312	8	APA150	38%
TARGDMA64		1,962	500	2,462	8	APA150	40%
TARGMAST64		3,173	814	3,987	8	APA300	49%
TARG64	RTAX-S	634	387	1,021	0	RTAX250S	24%
MAST64		1,002	565	1,567	0	RTAX250S	37%
TARGDMA64		1,087	553	1,640	0	RTAX1000S	9%
TARGMAST64		3,524	3,858	7,382	0	RTAX1000S	41%
TARG64	Axcelerator	642	317	959	2	AX500	12%
MAST64		1,021	502	1,523	2	AX500	19%
TARGDMA64		1,087	493	1,580	2	AX500	20%
TARGMAST64		1,874	765	2,639	2	AX500	33%
TARG64	SX-A	693	456	1,149	0	A54SX32A	40%
MAST64		1,095	682	1,777	0	A54SX72A	29%
TARGDMA64		1,201	645	1,846	0	A54SX72A	31%
TARGMAST64		1,711	1,200	2,911	0	A54SX72A	48%

2.5 Performance Statistics

Table 5 and Table 8 give the device speed grades required to meet either 33 MHz or 66 MHz PCI operation for the 32-bit and 64-bit cores for the three operating environments supported by Microsemi. Not all families support 64-bit or 66 MHz operation.

Table 5 • **Device Speed Grade Requirements**

	Family	Commercial	Industrial	Military
33 MHz 32-bit	Fusion	STD	STD	N/A
	IGLOO/e	STD	STD	N/A
	ProASIC3/E/L	STD	STD	N/A
	ProASIC^{PLUS}	STD	STD	STD
	RTAX-S	N/A	N/A	STD
	Axcelerator	STD	STD	STD
	RTSX-S	N/A	N/A	–1
	SX-A	STD	STD	STD
33 MHz 64-bit	Fusion	STD	STD	N/A
	IGLOO/e	STD	STD	N/A
	ProASIC3/E/L	STD	STD	N/A
	ProASIC^{PLUS}	STD	STD	STD
	RTAX-S	N/A	N/A	STD
	Axcelerator	STD	STD	STD
	RTSX-S	N/A	N/A	N/A
	SX-A	STD	STD	STD
66 MHz 32-bit	Fusion	–2	–2	N/A
	IGLOO/e	N/A	N/A	N/A
	ProASIC3/E	–2	–2	N/A
	ProASIC^{PLUS}	N/A	N/A	N/A
	RTAX-S (RTAX250S)	N/A	N/A	–1
	RTAX-S (RTAX1000S to RTAX4000S)	N/A	N/A	N/A
	Axcelerator (AX125 to AX500)	–1	–1	–1
	Axcelerator (AX1000 to AX2000)	–2	–2	–2
	RTSX-S	N/A	N/A	N/A
	SX-A	N/A	N/A	N/A

Table 5 • **Device Speed Grade Requirements *(continued)***

	Family	Commercial	Industrial	Military
66 MHz 64-bit	Fusion	–2	–2	N/A
	IGLOO/e	N/A	N/A	N/A
	ProASIC3/E	–2	–2	N/A
	ProASIC^{PLUS}	N/A	N/A	N/A
	RTAX-S (RTAX250S)	N/A	N/A	–1
	RTAX-S (RTAX1000S to RTAX4000S)	N/A	N/A	N/A
	Axcelerator (AX125 to AX500)	–1	–1	–1
	Axcelerator (AX1000 to AX2000)	–2	–2	–2
	RTSX-S	N/A	N/A	N/A
	SX-A	N/A	N/A	N/A

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

2.6 I/O Requirements

Table 7 gives the I/O requirements for CorePCIF. The number of device I/O pins required for the PCI interface varies, depending on the bus width as well as whether the core supports Target and/or Master functions. The number of backend device I/O pins that the core requires depends on the core interface. For instance, a device that implements a PCI-to-serial communication channel may only require a single device I/O pin, whereas a PCI-to-memory interface may require many I/O pins. Table 7 shows the maximum number of I/O pins, assuming all the core backend pins are connected to device I/O pins.

Table 7 • **CorePCIF I/O Requirements**

	I/O Count				
		Backend		Total	
Core	**PCI**	**Min.**	**Max.**	**Min.**	**Max.**
32-bit Target	48	1	146	49	194
64-bit Target	88	1	219	89	307
32-bit Master with backend interface	49	1	162	50	211
64-bit Master with backend interface	88	1	235	89	323
32-bit Target and Master	50	1	146	51	196
64-bit Target and Master	89	1	219	90	308
32-bit Target and Master with backend interface	50	1	162	51	212
64-bit Target and Master with backend interface	89	1	235	90	324

2.7 Electrical Requirements

CorePCIF 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. The SX-A and RTSX-S families have I/O buffers that directly support 5.0 V operation. Other families in 5.0 V PCI environments may require external voltage level translator devices, or may not be supported. See Table 8 for details.

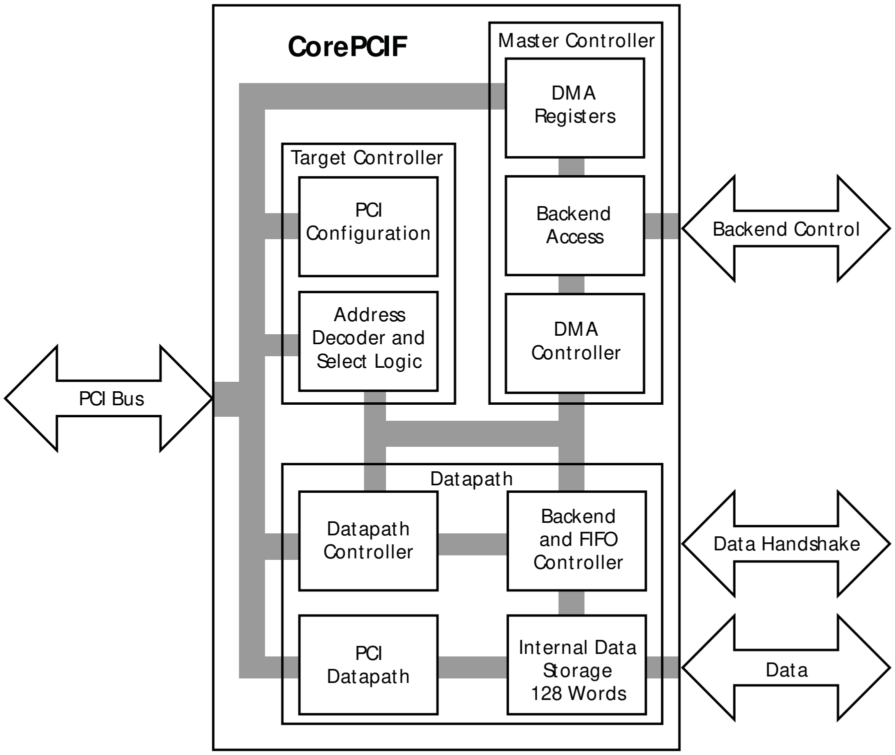
Table 8 • **Supported Electrical Environments**

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
	ProASIC<u>PLUS</u>	3.3	3.3 and 5.0
	RTAX-S	3.3	3.3 and 5.0
	Axcelerator	3.3	3.3 and 5.0
	RTSX-S	3.3 and 5.0	3.3 and 5.0
	SX-A	3.3 and 5.0	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
	ProASIC<u>PLUS</u>	Not supported	Not supported
	RTAX-S	3.3	3.3
	Axcelerator	3.3	3.3
	RTSX-S	Not supported	Not supported
	SX-A	Not supported	Not supported
	SmartFusion2	3.3	3.3 and 5.0

3 Functional Description

CorePCIF consists of three major functional blocks, shown in Figure 2. These blocks are the Target Controller, Master Controller, and Datapath. With both a Target and Master, all three blocks are required. Otherwise, only the Datapath and either the Target or Master function are required.

Figure 2 • **CorePCIF Block Diagram**



3.1 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 29.

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.

3.2 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. The controller will stop a DMA burst automatically if the backend runs out of data, and restart when data is available.

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

3.3 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.

3.4 CorePCIF – Internal Data Storage

CorePCIF includes a 64-word internal memory block for 32-bit PCI data width or a 128-word internal memory block for 64-bit PCI data width 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.

For the Axcelerator, ProASIC$^{\underline{\text{PLUS}}}$, ProASIC3, and ProASIC3E families, the data storage is implemented using FPGA memory resources. For SX-A and RTSX-S families, the storage is implemented using FPGA logic resources. For the RTAX-S family, the storage can be implemented using FPGA logic resources or memory resources. 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.

3.5 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.

3.5.1 Supported Target Commands

Table 9 lists the PCI commands supported in the CorePCIF Target implementation.

Table 9 • **Supported PCI Target Commands**

CBEN[3:0]	Command Type	Supported
0000	Interrupt Acknowledge	No
0001	Special Cycle	No
0010	I/O Read	Yes

Table 9 • **Supported PCI Target Commands**

CBEN[3:0]	Command Type	Supported
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

3.5.2 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.

3.5.3 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.

3.5.4 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.

3.5.5 Memory Write and Invalidate (1111)

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

3.5.6 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.

3.5.7 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.

3.6 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.

3.6.1 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.

3.6.2 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.

3.6.3 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 44 through Table 50.

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 10. 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 10).

Table 10 • **DMA Register Addresses**

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

3.6.4 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.

3.6.5 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.

3.7 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.

3.8 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. The insertion and extraction sequences are shown in Figure 66 and Figure 67.

3.9 CorePCIF Backend Dataflow

CorePCIF has a very flexible backend interface that supports various transfer rates as well as FIFOs. To decouple the backend data transfers from the PCI transfers, CorePCIF implements an eight-stage FIFO for each BAR. During normal operation, the FIFO stores up to four data words, the remaining four locations being used for the FIFO recovery mechanism. This is implemented using FPGA memory resources in all families except SX-A, RTSX-S, and RTAX-S.

3.9.1 Burst Transfers

CorePCIF is capable of bursting data from the PCI bus to the backend or vice versa. During transfers to the backend, the WR_BE_RDY and WR_BE_NOW signals are used to control the dataflow. When the backend asserts WR_BE_RDY, the core is allowed to write data to the backend by asserting WR_BE_NOW. A separate WR_BE_NOW signal is provided for each byte.

For transfers from the backend, RD_STB_IN and RD_STB_OUT control the dataflow. When both of these signals are active, data is transferred from the backend into the core.

3.9.2 Byte-Controlled Transfers

CorePCIF supports both write- and read-controlled byte transfers to the backend. When data is written to the backend, four (eight for 64-bit operations) write strobes (WR_BE_NOW) are provided, indicating which bytes should be written.

When data is read from the backend interface, the BYTE_ENN and BYTE_VALN signals can be used to control the byte reads. The backend should wait until BYTE_VALN is active (LOW) and then use the four (eight for 64-bit) BYTE_ENN signals (active low) to control the data read. Using the BYTE_VALN signal prevents the core from bursting data every clock cycle; in that case, data will be transferred once every four clock cycles at best.

3.9.3 Dataflow Control

CorePCIF allows the backend to stop data transfers in Master and Target mode, and to initiate transfers in Master mode. In Target mode, the BUSY signal can be used to terminate a data transfer so it will be retried. The ERROR signal can be used to simply terminate a transfer.

Likewise, in Master mode, the STOP_MASTER signal can be used to terminate a data transfer. The WR_BUSY_MASTER and RD_BUSY_MASTER signals can be used to delay a DMA transfer from starting. If STOP_MASTER and RD_BUSY_MASTER are connected to a FIFO empty signal, the DMA engine will automatically stop a DMA cycle when the FIFO becomes empty and restart it when the FIFO becomes non-empty. This allows the core to move data from a FIFO to PCI memory without any host intervention.

3.10 FIFO Recovery Logic

The CorePCIF backend interface directly supports the connection of external FIFOs using internal FPGA FIFO memories or external FIFO devices. To prevent data loss, CorePCIF includes optional FIFO recovery logic for each BAR. In normal burst operations, the core reads data from the backend at the same time as previous data is being transferred on the PCI bus. When the Master terminates the Target transfer, it is likely that data has been read from the FIFO and not transferred on the PCI bus (Figure 15). Without recovery logic, this data would be lost; however, if the FIFO recovery logic is enabled (Figure 24), the core stores this data until the next Target access to the same BAR. Data loss also potentially occurs when the core is operating in Master mode. In this case, the core also needs to recover data lost due to PCI cycles that are terminated with a disconnect without data cycle.

Figure 3 and Figure 4 show how to connect a FIFO to the backend interface, supporting Target and Master transfers. In Target mode, the FIFO empty signal is used to assert the BUSY input while the FIFO is empty and to assert RD_STB_IN when data is available.

In Master mode, the FIFO empty signal is used to assert the RD_BUSY_MASTER input while the FIFO is empty, preventing a DMA cycle from starting, and to assert RD_STB_IN when data is available. The

FIFO almost empty signal is used to assert STOP_MASTER, which will cause the current DMA cycle to be terminated as soon as possible. Additional data words may be read from the backend after STOP_MASTER has been asserted.

If both Master and Target transfers will be used, the connections in both Figure 3 and Figure 4 should be implemented.

Figure 3 • **External FIFO Connection (Target mode)**

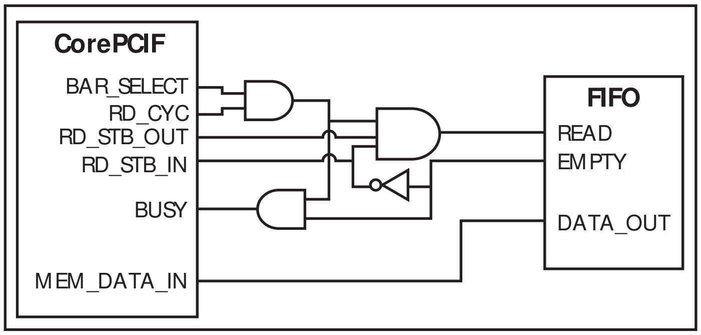
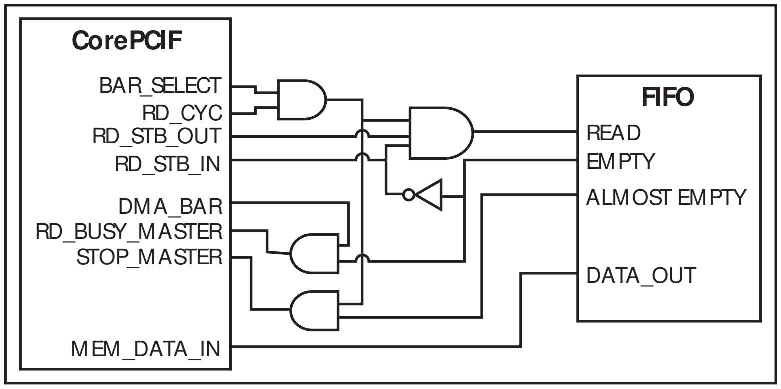


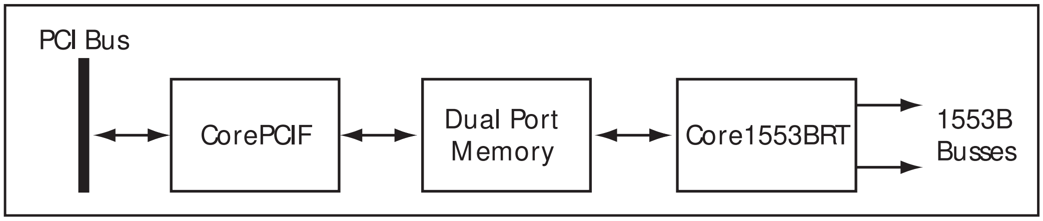
Figure 4 • **External FIFO Connection (Master mode)**



3.11 Example System Implementation

CorePCIF provides an extremely flexible PCI interface that can be configured in many ways. Figure 5 shows a PCI-to-1553 interface. In this example, CorePCIF is configured as a Target with a single memory BAR used to access the Core1553BRT memory.

Figure 5 • **Simple Target Implementation**



A more complex system is shown in Figure 6. In this case, the core supports both Target and Master operation. Core8051 is connected to the backend interface, allowing it to initiate PCI cycles. Core8051 is used to control the AES encryption core and the Core10/100 Ethernet interface. CorePCIF has two memory BARs configured. The first allows the PCI interface to access the 8051 memory space, and the second reads data from the FIFO.