USING LOGICORE™ MODULES FOR PCI CARD DESIGN

Bradly Fawcett, Steven Knapp, and Gary Lawman
Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
E-mail: pci@xilinx.com (PCP-139)

ABSTRACT

Intellectual property in the form of reusable cores can be used during high-density FPGA design to decrease development times and risks. The use of one such core, the LogiCore™ PCI Interface, accelerates the development of FPGA-based PCI card designs. This paper examines the issues and considerations surrounding the FPGA implementation of a PCI interface when using the LogiCore module.

CORE-BASED DESIGN

Continuing improvements to both fabrication processes and device architectures have led to dramatic increases in Field Programmable Gate Array (FPGA) device capacities and performance. Taking advantage of these increasing capabilities while maintaining time-to-market goals can pose formidable design challenges - challenges that cannot always be met by traditional gate-level design techniques, or even HDLs and synthesis-based design. For a growing legion of FPGA users, reusable intellectual property in the form of design "cores" (also referred to as "megafunctions" or "drop-in modules") have become a key factor in meeting the twin challenges of increasing design complexity and shorter development cycles.

Simply put, cores are complex, pre-designed and reusable functional blocks, typically hundreds to thousands of gates in size, that can be included as part of a larger FPGA design. The goal is to allow FPGA designers to act as system integrators, combining proven functional blocks with the proprietary logic of the particular application. Cores can be developed internally (for example, re-used portions of previous-generation designs) or purchased as intellectual property from the FPGA vendor or a third-party provider. In order to address a broad enough market to justify their development, cores available for purchase typically are "standard" functions that provide little opportunity for product differentiation. Examples include bus interfaces (such as PCI, PCMCIA, and USB) and common DSP functions (such as FIR and IIR filters). The selection of available cores is growing rapidly.

The main benefit of cores is the decreased development time and effort associated with using a pre-designed, proven function. Designers can focus their efforts on the proprietary portions of their designs, rather than "re-inventing" a standard function. Often, for complex functions such as PCI interfaces, utilizing a core allows the user to access the "system expertise" of the core's designers; the user does not have to acquire similar levels of expertise on that aspect of the design. Thus, the "make or buy" decision often falls on the side of purchasing the core, particularly for standard functions that are needed for market acceptance but contribute little to product differentiation.

For example, with the increased logic capacity of the latest generations of FPGAs, a typical application's unique control logic for the "back-end" device being connected to a PCI bus can be integrated with the PCI bus interface on the same FPGA device. However, the performance requirements of the PCI specification are demanding even for advanced ASIC technologies, and require FPGA designs closely tailored to the device architecture. For this reason, and since the PCI bus interface portion of the design is common to many applications, several FPGA vendors offer "pre-designed" PCI interface modules, as exemplified by the Xilinx LogiCore PCI Interface for the XC4000E FPGA family.

This paper overviews the issues involved with designing FPGA-based PCI bus interfaces by examining the use of the LogiCore PCI Interface. Readers are assumed to have some knowledge of the PCI bus protocols; complete information about the PCI bus is available in the *PCI Local Bus Specification* and several reference books.[1-3]

THE LOGICORE PCI INTERFACE

The LogiCore PCI Interface is a fully-integrated, tested, and validated schematic-based PCI Local Bus interface module design targeted for the XC4013E FPGA device.[4] Both Initiator/Target and Target-only versions are available. This modular design can be customized by the user, and supports the quick implementation of prototype and production PCI applications. By minimizing the engineering effort required to develop a PCI host interface, use of the LogiCore PCI module can save months of development time.

The LogiCore PCI Interface is a complete 32-bit PCI interface compliant with version 2.1 of the PCI Local Bus Specification. It supports all basic PCI bus functions, including Type 0 configuration space support, I/O reads and writes, and burst-mode memory reads and writes. Detailed schematics form the core PCI interface design; the design was implemented using Viewlogic Systems schematic entry and simulation tools. Where applicable, logic contained within the schematic is trimmed during the compilation process (i.e., unused logic is automatically removed from the design). Placement constraints, timing constraints, and a placement guide file are used to ensure that PCI timing requirements are met. Both complete schematics and a 'drop-in' XNF netlist are provided. The module can be instantiated from within an HDL source code file.

The LogiCore PCI Interface can be easily customized to meet specific board-level design requirements. The user can integrate the PCI Interface with other modules to complete an interface design; combining custom "back-end" logic to the fully-tested PCI bus interface produces a single-chip PCI I/O adapter.

The LogiCore PCI Interface has been optimized with a floorplanned layout for the XC4013E-2PQ208 device, but can be ported to other XC4000E and XC4000EX family FPGAs. (However, all compliance testing was performed using the XC4013E FPGA.) The full Initiator/Target design occupies less than one-half of the Configurable Logic Blocks (CLBs) in the XC4013E device, leaving ample space for the implementation of the custom back-end logic. Fully-compliant 33 MHz PCI applications require the -2 speed grade for the XC4013E device. Embedded PCI applications, terminated busses, and systems operating below 33 MHz may be able to use a slower speed grade.

The major functional blocks of the LogiCore PCI Interface are shown in Figure 1.

The I/O Interface block handles the physical connection to the PCI bus, including all signaling, input and output synchronization, output three-state controls, parity generation and checking, and (for initiator designs) request-grant handshaking for bus

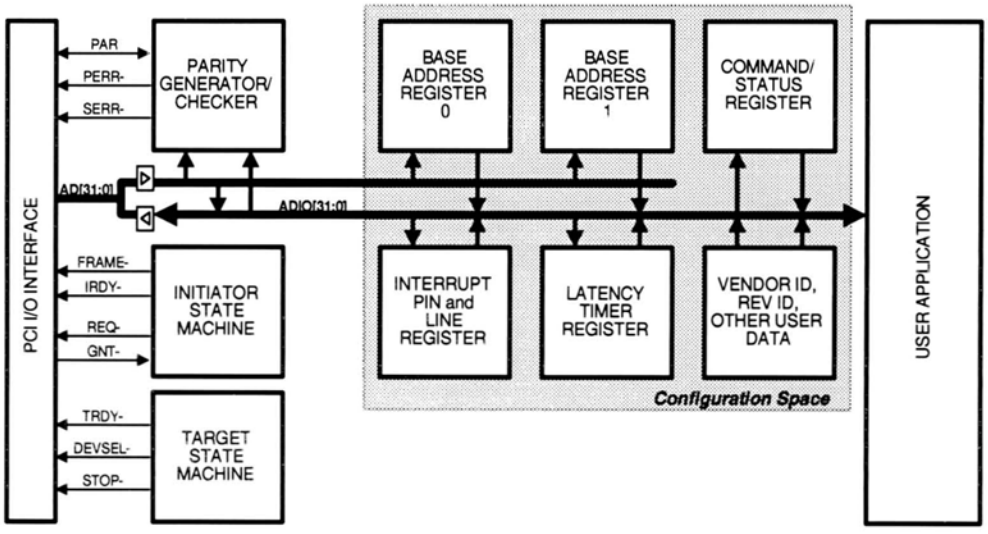

Figure 1: LogiCore PCI Interface block diagram

mastering. Parity errors detected on the address lines are flagged by asserting the SERR- signal, and data parity errors are reported using PERR-.

The Target and Initiator State Machines contain all the control logic for handling bus transactions for target and initiator agents, respectively. These controllers are high-performance state machines that use one-hot encoding for maximum performance. The states implemented are a subset of the equations defined in Appendix B of the PCI Local Bus Specification. Table 1 lists the PCI bus commands currently supported by the LogiCore PCI interface.

The Configuration Logic block holds the Configuration Space Header, the command and status registers used in operations such as "plug-and-play" initialization. The first 64 bytes of a Type 0, version 2.1 Configuration Space Header are provided. Read/write registers are implemented using flip-flops in the FPGA's configurable logic blocks (CLBs), while read-only registers are implemented as ROM memory using the CLB's lookup tables, resulting in optimized packing density and layout.

A simple, general-purpose interface is provided for connecting to the application's back-end logic, including a 32-bit data path and latched address bus. Most of the module's internal data paths and state machine control signals are included in the user interface, providing ample flexibility for customized user applications. Typically, the back-end logic would include FIFO buffers to support burst transactions on the PCI bus.

CUSTOMIZING THE LOGICORE PCI INTERFACE

The LogiCore PCI Interface provides a foundation PCI bus interface design that can be tailored for the specific application. Besides adding the unique back-end logic for the application, users can customize the PCI bus interface itself. In most cases, this customization simply involves replacing particular symbols in the Viewlogic schematics with other pre-prepared symbols, using the schematic editor's 'Change Component' command. The LogiCore module's design structure and modifiable schematics are shown in Figure 2. The back-end application logic that is to be integrated onto the FPGA with the PCI bus interface should be placed under the "userapp" hierarchy.

For users of the Initiator/Target version, the first step in customizing the PCI interface macro is to define whether the application is a target-only or

Table 1: PCI Bus Commands

CBE[3:0]	Command	PCI Master	PCI Slave
0000	Interrupt Acknowledge	No*	Ignore
0001	Special Cycle	No*	Ignore
0010	I/O Read	Yes	Yes
0011	I/O Write	Yes	Yes
0100	Reserved	Ignore	Ignore
0101	Reserved	Ignore	Ignore
0110	Memory Read	Yes	Yes
0111	Memory Write	Yes	Yes
1000	Reserved	Ignore	Ignore
1001	Reserved	Ignore	Ignore
1010	Configuration Read	Yes	Yes
1011	Configuration Write	Yes	Yes
1100	Memory Read Multiple	Yes	Yes
1101	Dual Address Cycle	No*	Yes
1110	Memory Read Line	Yes	Yes
1111	Memory Write Invalidate	No*	Yes

*The Initiator can present these commands, but they require additional user-application logic.

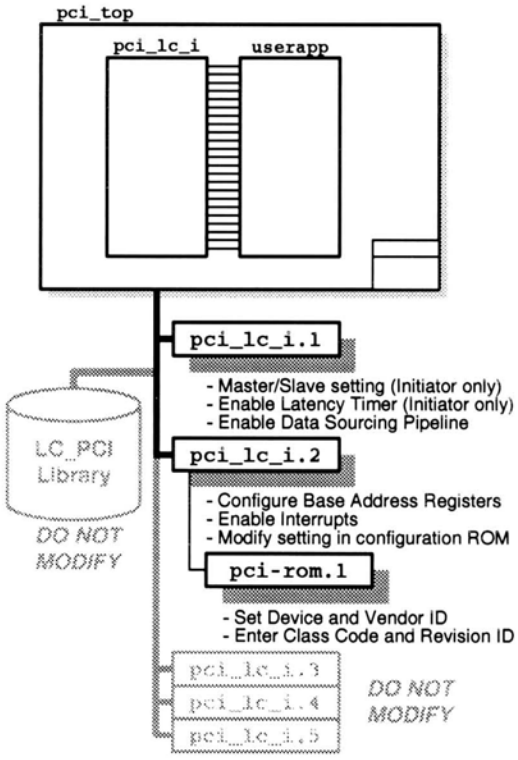

Figure 2: Design Structure of LogiCore PCI Interface

initiator/target function. As delivered, the macro is configured as an initiator/target interface; configuring the macro to be a target-only interface requires changing one symbol on the top level schematic.

Many of the customization options involve the PCI configuration register space. The default configuration includes two Base Address Registers, a Command Register, a Status Register, an Interrupt Pin and Line Register, and a read-only memory space for the Device ID, Vendor ID, Class Code, and Rev ID registers. All remaining locations in the CSH return a value of zero during configuration reads, and no operation occurs during configuration writes. If desired, additional locations in the CSH can be added by the user. Base Address Register sizes and modes can be altered by changing symbols on the schematic. A modifiable table included on the schematic for the read-only registers, such as Device ID, allows users to easily define their contents.

If the application is an initiator/target, and it will never burst more than two data cycles, than a Latency Timer can be disabled, conserving space and simplifying the control logic. Similarly, if the application supports only single data transfers, then the pipelining logic can be disabled.

Completing a PCI bus interface design using the LogiCore PCI Interface module can be broken down into three main sub-tasks: building the target-side interface, building the initiator interface, and, optionally, providing for the support of burst transfers.

BUILDING TARGET AND INITIATOR INTERFACES

A target interface is part of every PCI bus interface design, regardless of whether the bus agent is an initiator/target or target-only interface. Creating a target interface using the LogiCore module involves the following steps: configuring the Base Address Registers, defining the read-only values for the Configuration Space Header ROM, building the interface between the PCI module and the application's unique back-end logic, and (if required) deciding how to force Target Termination conditions.

The user application can force various target-initiated termination conditions. The state of two signals, TERM and READY, that have their source in the user application logic, determine the termination condition. A Target Retry condition informs the initiator that it must try the transaction again later. A Target Disconnect indicates that the target is no longer able to continue the transaction (for example, due to a full FIFO buffer during a burst write operation); data may or may not be transferred on the disconnect cycle. A Target Abort signals a serious error at the target that prevents the requested transaction.

Creating the initiator portion of a PCI interface also involves building the appropriate interface between the LogiCore module and the back-end application. Before designing the initiator's control logic, several aspects of its operation must be defined, including the type and speed of data transfers initiated by this agent, the desired response to the target-generated termination conditions described above, and the method of arbitrating between a pending initiator request and an incoming target transaction.

In the majority of applications, data is transferred to and from read/write registers in the

back-end logic. These registers often are part of a FIFO buffer, but also may be connected to I/O pins or other logic. Figure 3 illustrates a typical data connection. A clock enable signal (CLK-ENA) is used to control the capturing of data from the PCI bus (for example, when writing data to a target), and three-state buffers control the routing of data back onto the bus (for example, when reading from a target).

In order to meet the PCI bus' stringent performance requirements, the LogiCore interface pipelines all the bus control signals and the data path. Consequently, some signals must be presented up to two clock cycles before they appear on the PCI bus. Likewise, arriving signals are captured and available to the back-end logic one clock cycle after they appear on the PCI bus. Appropriate signals are included in the LogiCore module's back-end interface to signal the appearance of valid address and data information.

It is highly recommended that all signals to or from the back-end interface should be registered. Good synchronous design techniques can increase system performance and simplify timing analysis.

SUPPORTING BURST TRANSACTIONS

Performing a single data transfer across the PCI bus is the simplest type of transaction. However, because of the overhead of distributed address decoding, this wastes valuable bus bandwidth. Achieving high bandwidth requires the use of burst transactions, where multiple data words are transferred during each transaction.

In PCI bus burst transactions, only the starting address is broadcast over the bus. Thus, during burst transfers, the back-end logic must keep track of the current address. Typically, both target and initiator control logic must keep a local copy of the current address pointer and increment it after each transaction. This counter usually is small; its size depends on the target's address block size, as determined by the contents of the Base Address Registers. If a target agent detects an overflow of its address space, it should issue a Target Disconnect, indicating that it is unable to perform the requested operation. The initiator's response to a Target Termination depends on the type of termination. If the initiator receives a Target Retry, it should simply restart the transaction from the

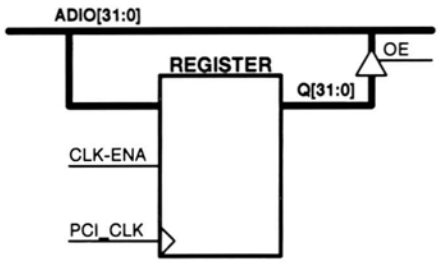

Figure 3: Example read/write register

starting address. However, if a Target Disconnect is detected, the initiator will have to use the current contents of its address pointer to know where to re-start the transaction. To complicate manners, the target can disconnect with or without data, and the initiator must increment or freeze the address pointer accordingly.

PERFORMANCE

The maximum theoretical PCI bus bandwidth is 132 Mbytes/sec. However, actual system bandwidth varies tremendously from one platform to another. PCI bus performance is controlled by three key factors: aggregate bandwidth, data throughput, and access latency.

The ideal PCI burst write transaction requires three clock cycles for the first transfer (Idle, Address, Data) and one clock cycle for each subsequent transfer, referred to as a 3-1-1-1 sequence. The ideal write transfer needs four clock cycles for the first transfer (Idle, Address, Turn-around, Data) and one clock cycle for subsequent transfers (4-1-1-1).

The LogiCore PCI Interface requires additional clock cycles to decode the address (referred to as SLOW decoding in the PCI Specification). Target burst write transactions do not require any wait states; thus, the default transfer rate for I/O and Memory Write transactions is 5-1-1-1. Initiators add an extra clock cycle at the end of a transaction to provide reliable disconnection from the bus; the default rate for initiator reads is 4-1-1-2.

A wait state is added to each transaction when the LogiCore PCI module is the source of data for a burst transaction (i.e., initiator writes and target reads). These wait states are needed to guarantee compliance with all the PCI bus operating rules.

For example, suppose the LogiCore PCI Interface is incorporated in a target agent. When an initiator reads from that target, the initiator can de-assert IRDY- before a clock edge to indicate that it is not ready for the next data transfer (that is, an initiator-generated wait state). The current implementation of the LogiCore macro cannot respond to this within the available 7 ns. Therefore, when the LogiCore PCI target macro is the source of data in a burst read transaction, it always adds a wait state to each read transaction to allow ample time to check the initiator's IRDY- signal. Hence, the default rate for target I/O and Memory Read transactions is 6-2-2-2. In systems where an initiator never generates wait states, the design could be modified to support 6-1-1-1 read transfers. Similarly, the default rate for LogiCore initiator burst writes is 3-2-2-2. (Wait states are never added to single data transfers.)

Figure 4 compares the maximum data transfer rates for ideal PCI read and write operations and LogiCore PCI initiator and target transactions.

As can be seen in Figure 4, the bandwidth of the PCI bus is primarily determined by the size of the burst transfer. Thus, most PCI designs include a FIFO buffer between the LogiCore module and the back-end logic. This buffer isolates the speed of the PCI bus from the operation of the back-end logic. Data to be transferred from the back-end device can be queued in the FIFO in preparation for a burst transfer, and data to be transferred to the back-end device can be placed into the FIFO by a burst transaction and subsequently processed by the back-end logic at its own rate.

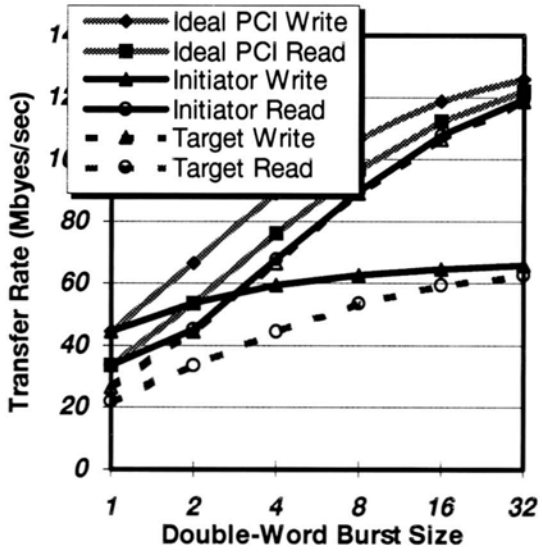

Figure 4: Effects of the burst read size and latency on overall PCI bandwidth.

The synchronous, dual-port distributed memory mode of the XC4000E and XC4000EX FPGA architectures is ideal for implementing FIFO buffers of any desired length and width.[5] As shipped, the LogiCore PCI Interface includes an example user interface - a 16 x 32 read/write burst FIFO buffer. (This application helped simplify the PCI compliance testing using the VirtualChips PCI bus simulation model.) However, the recommended structure for most designs is a dual FIFO design, with separate buffers to support bus read and write operations. FIFO buffers that are 16 double-words deep fit best in the XC4000E/EX architecture; no performance or density is gained by making the buffers shallower. FIFOs deeper than 16 double-words but less than 33 consume more logic blocks but do not add delay. FIFOs deeper than 32 double-words would consume more logic and delay.

VERIFICATION AND TESTING

The LogiCore PCI Interface has been fully verified using the Synopsys VSS VHDL and Viewlogic ViewSim logic simulators. Protocol compliance was tested according to the PCI Compliance Checklist, Revision 2.0b, published by the PCI-SIG. Both the VirtualChips VHDL PCI bus simulation model and an internally developed test suite were employed. Several tests were added to test functions not covered by the PCI-SIG checklist, such as target agent responses to various termination conditions. The test results are summarized in the Xilinx LogiCore PCI Interface Protocol Checklist (v2.1). The design also was verified via actual FPGA device implementations at multiple "beta site" accounts.

Users are encouraged to perform a functional simulation after selecting the custom options in the PCI LogiCore module and integrating the back-end logic with the LogiCore module. After running the "place and route" software that determines the physical implementation of the design in the FPGA device, the Xilinx XDelay static timing analyzer and/or a timing simulator should be used to verify performance along all critical paths. A PCI

Protocol Testbench for the Viewlogic ViewSim simulator is provided with the LogiCore module to facilitate user testing of the completed design.

SUMMARY

The LogiCore PCI Interface provides a high-quality foundation design for the development of FPGA-based PCI card solutions. Use of this module minimizes the engineering effort required to develop a PCI interface, reduces design risk, and allows designers to focus on the important system-level aspects of the design.

However, the availability of such modules should not be viewed as a panacea. PCI bus interfaces are challenging in any technology, and especially so in FPGA devices. The degree of difficulty is influenced by the maximum system clock frequency, whether a target-only or initiator/target is needed, and whether the application supports burst data transfers. A 33 MHz, fully-compliant initiator/target design should only be attempted by experienced users willing to invest the extra effort to obtain maximum bandwidth.

REFERENCES

1. *PCI Local Bus Specification rev. 2.1*, published by the PCI Special Interest Group

2. T. Shanly and D. Anderson, *PCI System Architecture*, ISBN 1-881609-08-1, Mindshare Press, Richardson, TX

3. E. Solari and G. Willse, *PCI Hardware and Software Architecture and Design*, ISBN 0-929392-19-1, Annabooks, San Diego, CA

4. *LogiCore PCI Master and Slave Interface User's Guide*, Xilinx Inc. P/N 0401561-01, Aug. 1996

5. *Implementing FIFOs in the XC4000E*, Xilinx Inc. application note, P/N 0010273-01