XILINX

Design SUPERCON97

Designing a PCI Target/Initiator in FPGAs

Bradly Fawcett & Steven Knapp

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124

1997
On-Chip System
Design Conference

XILINX®

Design SUPERCON97

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 presentation is a case study of the design of the PCI initiator and target interfaces in an XC4013E FPGA. Emphasis is placed on the design techniques used to achieve a high-performance implementation, the tools and methodologies used to complete the design, and the use of the core in end applications.

Authors/Speakers

Current Activities

Bradly Fawcett

Brad Fawcett is currently a Manager in the Applications Engineering group at Xilinx Inc. Among his many duties, Brad is the editor of the XCell journal, the Xilinx user newsletter.

Steven Knapp

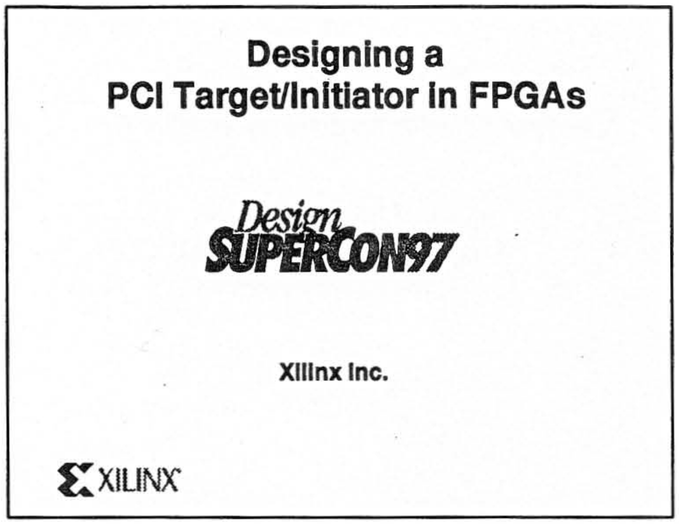
Steve Knapp holds the title of Vertical Applications Manager. Steve is one of the designers of the PCI LogiCore Interface

Authors Background

In his nine years with Xilinx Inc., Brad Fawcett has held a variety of applications engineering, technical marketing, and technical training positions. A veteran of 20 years in the industry, Brad has prior design engineering experience with Zilog Inc., Amdahl Corp., and Burroughs Corp. He holds Bachelor and Master of Science degrees in Electrical Engineering from the University of Illinois.

A veteran of ten years at Xilinx, Steve Knapp also has application engineering experience at Intel Corp., and holds a Bachelor of Science degree in Material Science from the Massachusetts Institute of Technology.

Slide #1



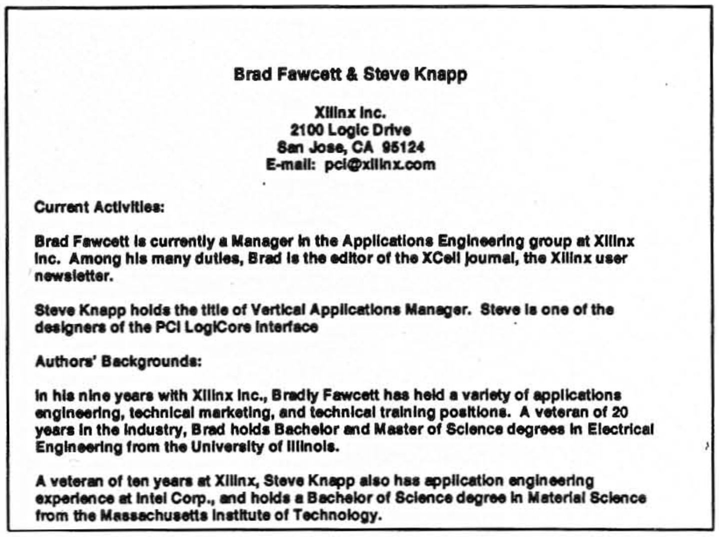
Designing a PCI Target/Initiator in FPGAs

Design SUPERCON97

Xilinx Inc.

XILINX

Slide #2



Brad Fawcett & Steve Knapp

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
E-mail: pci@xilinx.com

Current Activities:

Brad Fawcett is currently a Manager in the Applications Engineering group at Xilinx Inc. Among his many duties, Brad is the editor of the XCell journal, the Xilinx user newsletter.

Steve Knapp holds the title of Vertical Applications Manager. Steve is one of the designers of the PCI LogiCore interface

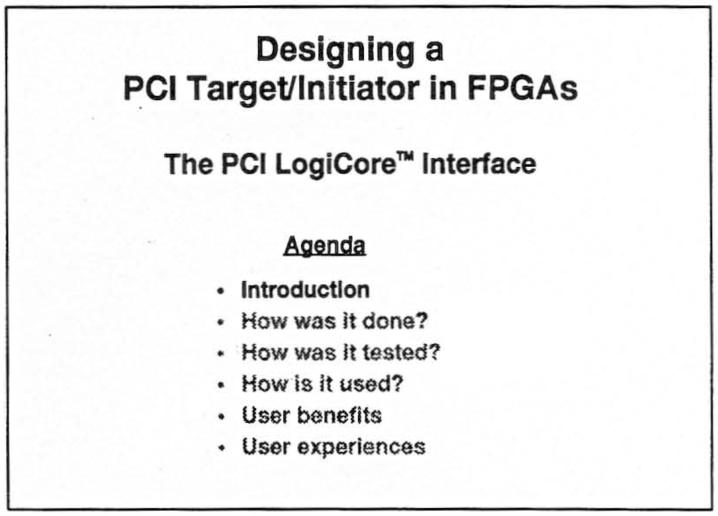
Authors' Backgrounds:

In his nine years with Xilinx Inc., Bradly Fawcett has held a variety of applications engineering, technical marketing, and technical training positions. A veteran of 20 years in the industry, Brad holds Bachelor and Master of Science degrees in Electrical Engineering from the University of Illinois.

A veteran of ten years at Xilinx, Steve Knapp also has application engineering experience at Intel Corp., and holds a Bachelor of Science degree in Material Science from the Massachusetts Institute of Technology.

With the recent emergence of PCI-compliant Field Programmable Gate Arrays (FPGAs), designers of PCI bus interfaces can now reap the flexibility and time-to-market benefits of high-density user-programmable devices, while avoiding the costs and risks of custom and semi-custom IC development. 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 the 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, many FPGA vendors offer "pre-designed" modules or similar reference designs to their users, as exemplified by the Xilinx LogiCore™ PCI Interface for the XC4000E FPGA family.

Slide #3



Designing a PCI Target/Initiator in FPGAs

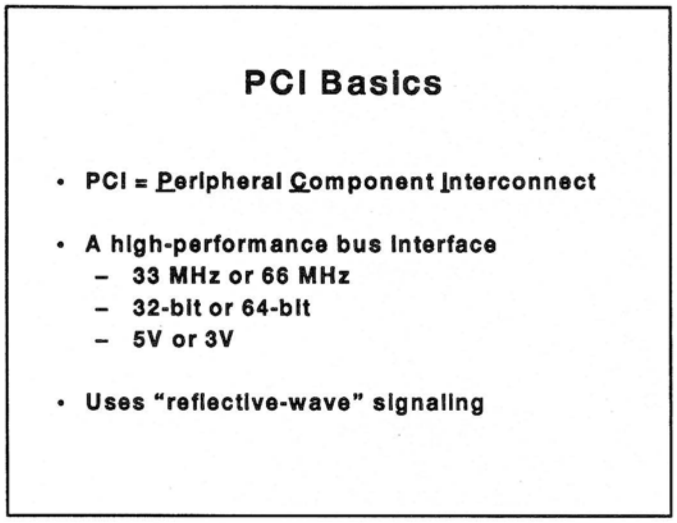
The PCI LogiCore™ Interface

Agenda

- **Introduction**
- How was it done?
- How was it tested?
- How is it used?
- User benefits
- User experiences

This presentation will overview some of the issues involved with designing FPGA-based PCI bus interfaces and then examine 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 (see slides 50 - 52).

Slide #4



PCI Basics

- PCI = Peripheral Component Interconnect
- A high-performance bus interface
  - 33 MHz or 66 MHz
  - 32-bit or 64-bit
  - 5V or 3V
- Uses "reflective-wave" signaling

The Peripheral Component Interconnect (PCI) bus definition has the admirable goal of providing high throughput on a well-defined, lightly-loaded bus while being compatible with today's IC manufacturing processes.

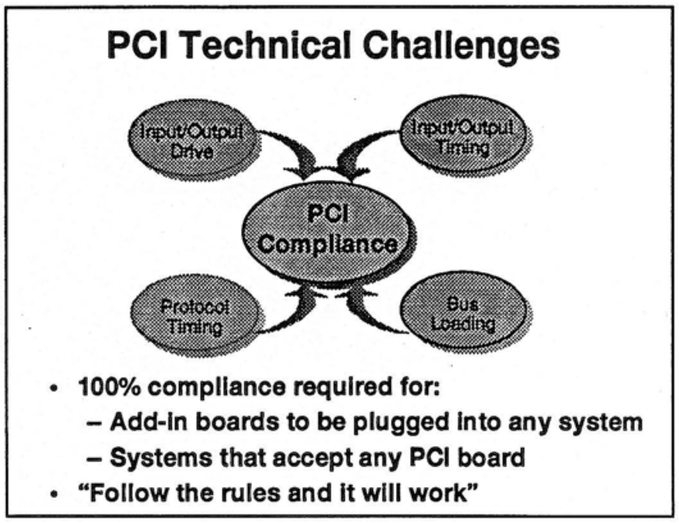
The PCI bus specification was originally defined by Intel, but is now controlled by an industry consortium called the PCI Special Interest Group (PCI SIG). The PCI bus has been adopted by nearly every major computer company, and also forms the backbone in many embedded applications.

While PCI comes in several "flavors", most implementations today employ a 32-bit, 33 MHz bus with 5V Vcc.

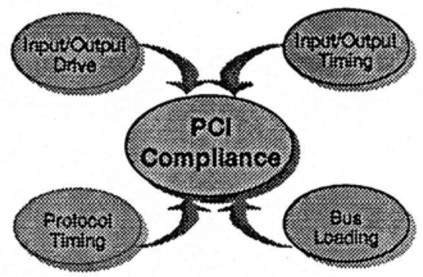
The PCI bus is unterminated, operating on the principle of reflective wave signaling. The initial output signal, therefore, has half amplitude. It travels to the end of the non-terminated bus, and gets reflected back towards the source to become a full-amplitude signal. This scheme demands strict control of device drive characteristics. In order to achieve 33 MHz operation, the round-trip delay must be limited to 10 ns, limiting the physical length and allowable capacitive loading on the bus.

At the maximum 33 MHz clock rate, the 30 ns clock period for a data transfer allocates 11 ns for the output driver, 10 ns for the round-trip bus propagation delay, 2 ns for potential clock skew, and 7 ns for input set-up time.

Slide #5



PCI Technical Challenges



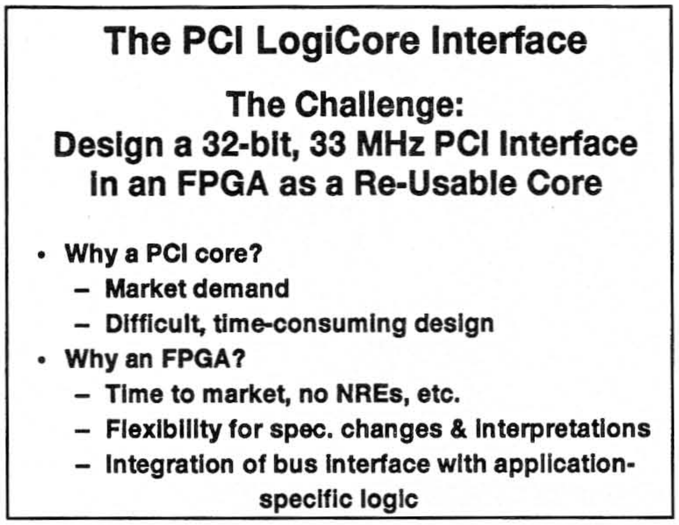
- 100% compliance required for:
  - Add-in boards to be plugged into any system
  - Systems that accept any PCI board
- "Follow the rules and it will work"

Full compliance is a requirement for any board intended to be plugged into any PCI system or any system intended to accept any PCI-compatible boards. In other words, full compliance is a must in systems that are to be sold to the general marketplace and require inter-operability among multiple vendors. However, if the system environment is "closed", such as an embedded system with no add-in capabilities, then, of course, the designer has freedom to deviate from the specification.

The PCI Special Interest Group (PCI SIG) has published the *PCI Compliance Checklist* of parameters that both system and component suppliers must adhere to in order to claim compliance. It includes a *Component Electrical Checklist* covering areas such as I/O signaling levels, timing specifications, and bus loading. It is important to review this checklist for any device that will connect directly to a PCI bus. By responding appropriately to the items in this checklist, an IC vendor demonstrates that the minimum amount of work required in order to claim compliance has been completed. Examples of PCI-compliant FPGA devices include the XC3100A, XC4000E, and XC4000EX FPGA families.

However, while meeting the checklist criteria is necessary, it does not in itself guarantee compliance. Careful design is required to meet the performance and signaling requirements of the PCI specification.

Slide #6



The PCI LogiCore Interface

The Challenge: Design a 32-bit, 33 MHz PCI Interface in an FPGA as a Re-Usable Core

- Why a PCI core?
  - Market demand
  - Difficult, time-consuming design
- Why an FPGA?
  - Time to market, no NREs, etc.
  - Flexibility for spec. changes & interpretations
  - Integration of bus interface with application-specific logic

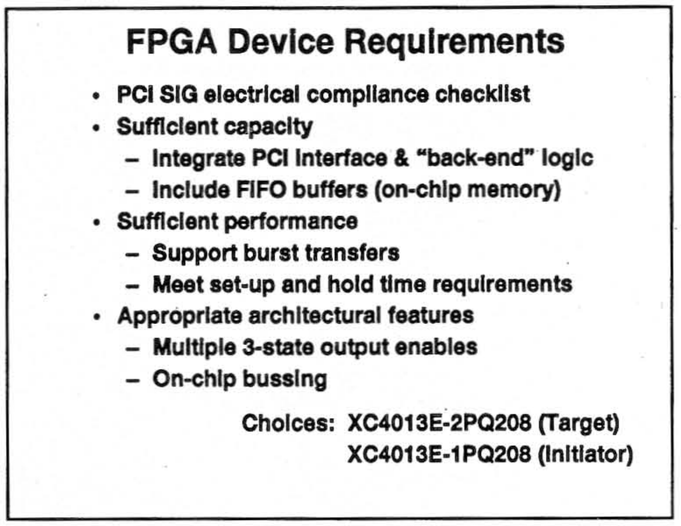
As mentioned before, the performance requirements of the PCI specification are demanding and require FPGA designs closely tailored to the device architecture. For this reason, and since the PCI bus interface portion of the design is common to many applications, many FPGA vendors offer "pre-designed" modules or similar reference designs to their users, as exemplified by the Xilinx LogiCore™ PCI Interface.

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 the design, rather than "re-inventing" a standard function.

FPGA devices are used on PCI cards for many of the same reasons that FPGAs are chosen for other electronic systems: fast time-to-market, low design risk, no non-recurring engineering charges, the convenience and flexibility of user-programmability, and so on. For PCI card designs, the flexibility of FPGAs can be key to responding to variations in the interpretation of the PCI specification, or changes to the specification itself.

Often, FPGAs are chosen over commercially-available chip sets because they provide the ability to integrate the application's unique "back-end" logic with the PCI bus interface on a single device.

Slide #7



FPGA Device Requirements

- PCI SIG electrical compliance checklist
- Sufficient capacity
  - Integrate PCI interface & "back-end" logic
  - Include FIFO buffers (on-chip memory)
- Sufficient performance
  - Support burst transfers
  - Meet set-up and hold time requirements
- Appropriate architectural features
  - Multiple 3-state output enables
  - On-chip bussing

Choices: XC4013E-2PQ208 (Target)
XC4013E-1PQ208 (Initiator)

Many programmable logic devices lack sufficient or appropriate resources for PCI bus interface design.

As mentioned earlier, an IC that connects to the PCI bus must, as a minimum, meet the requirements of the PCI SIG's PCI Compliance Checklist.

An FPGA implementation should offer sufficient capacity to hold both the PCI bus interface and the applications unique "back-end" logic. The PCI bus is a synchronous bus based on a single master clock signal; this implies that bus signals should be registered as they exit and enter bus agents. Thus, PCI interfaces demand an architecture with adequate register space for latching bus signals as well as generating state machines, pipelines, and other internal logic.

The back-end of a PCI bus interface typically connects the bus to a processor, memory subsystem, embedded controller, or peripheral device. In most designs, a data buffer, such as a FIFO buffer or dual-port memory, resides between the external device and the PCI bus interface logic, decoupling the speed of the PCI bus from the back-end controller, and allowing multiple data words to be queued to support burst transactions. Thus, on-chip memory capability is desirable in the FPGA to facilitate construction of these buffers.

Furthermore, PCI devices must implement a basic set of configuration registers, divided into a predefined, 64-byte header region and a 192-byte device-dependent region. FPGA's with on-chip

Slide #7, continued

memory capability can include some or all of the configuration bits within the FPGA.

Of course, the device must be capable of meeting the performance requirements of the PCI bus, including its stringent set-up and hold time requirements. Other, more subtle, performance requirements result from the implementation of the bus interface protocols. For example, the IRDY# and TRDY# signals sometimes must respond to a change in FRAME# within one clock cycle. These signal paths are design-dependent, involving internal logic and routing paths in the FPGA.

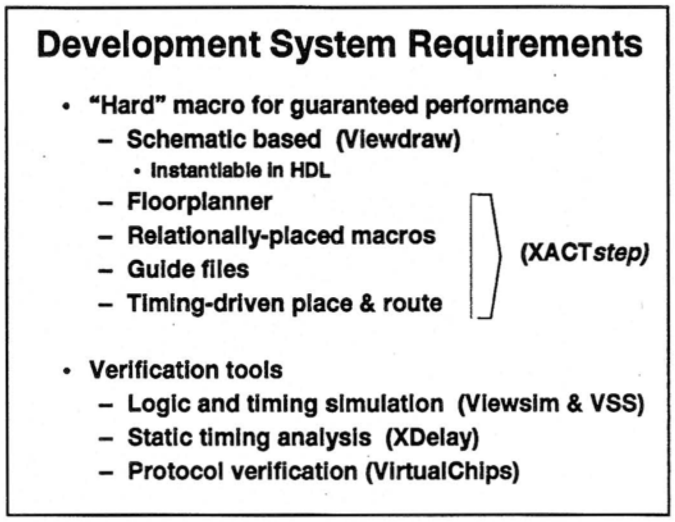
Beyond adequate register and memory capacity, other architectural features required for PCI interface design include JTAG boundary scan logic (a PCI option), the capability of efficiently implementing wide address and data busses on-chip, and plentiful, flexible output enables for three-state output buffers.

Some FPGA devices have restrictions as to the number of different signals that can be used as output enables for the device's output buffers. These devices generally are not good candidates for PCI bus interface designs. Since different bus control signals must be driven at different times and most control signals are three-state signals that must be driven by three-state buffers, PCI interface designs require significant flexibility in the generation of multiple output enable signals.

Taking all these considerations into account, the XC4013E-PQ208 FPGA device was selected as the target architecture for the LogiCore PCI Interface.

As with all Xilinx SRAM-based FPGAs, HardWire versions of the XC4000E Series FPGAs are available, providing a cost reduction path for high-volume applications that use the LogiCore PCI module.

Slide # 8



Development System Requirements

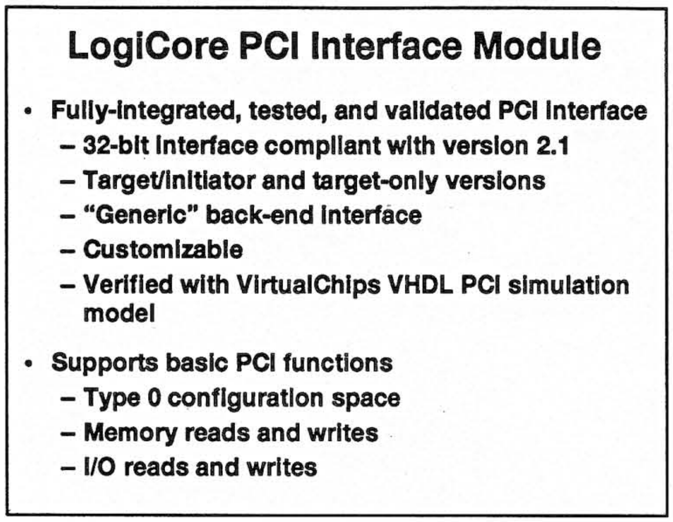
- "Hard" macro for guaranteed performance
  - Schematic based (Viewdraw)
    - Instantiable in HDL
  - Floorplanner
  - Relationally-placed macros
  - Guide files
  - Timing-driven place & route

  (XACT*step*)
- Verification tools
  - Logic and timing simulation (Viewsim & VSS)
  - Static timing analysis (XDelay)
  - Protocol verification (VirtualChips)

An often overlooked factor when evaluating FPGAs is the capabilities of their development tools. Since the performance requirements of the PCI standard tax the capabilities of most FPGA devices, development tools capable of easily producing high-performance layouts are mandatory. While automated layout tools should be robust, the intricacies of PCI design require that the designer exercise some control over the tools, especially in the placement and routing of critical paths and the placement of I/O pins. Thus, some form of placement control is needed, and floorplanning support is desirable. For FPGA-based designs, timing-driven place and route tools such as XACT-Performance™ from Xilinx can ease the design process by allowing the specification of target performance requirements for entire paths through the design. "Re-entrant" FPGA place and route tools, wherein the placement and routing of a previous version of a design can guide the implementation of a new version with minimal changes, can greatly ease the design process.

The design was entered and tested using Viewlogic Systems schematic entry and simulation tools. Both the schematics and resulting netlist files are provided. The module can be instantiated from within an HDL-based design. The design can be used as is or tuned to meet a specific requirement. Placement constraints, XACT-Performance timing constraints, and a placement guide file guarantee PCI timing requirements. (Verification tools and procedures will be discussed later.)

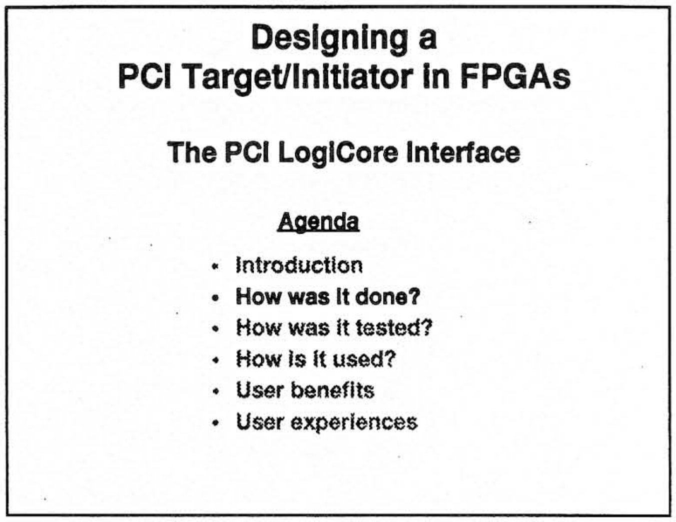
Slide #9



LogiCore PCI Interface Module

- Fully-integrated, tested, and validated PCI interface
  - 32-bit interface compliant with version 2.1
  - Target/Initiator and target-only versions
  - "Generic" back-end interface
  - Customizable
  - Verified with VirtualChips VHDL PCI simulation model
- Supports basic PCI functions
  - Type 0 configuration space
  - Memory reads and writes
  - I/O reads and writes

Slide #10



Designing a PCI Target/Initiator in FPGAs

The PCI LogiCore Interface

Agenda

- Introduction
- How was it done?
- How was it tested?
- How is it used?
- User benefits
- User experiences

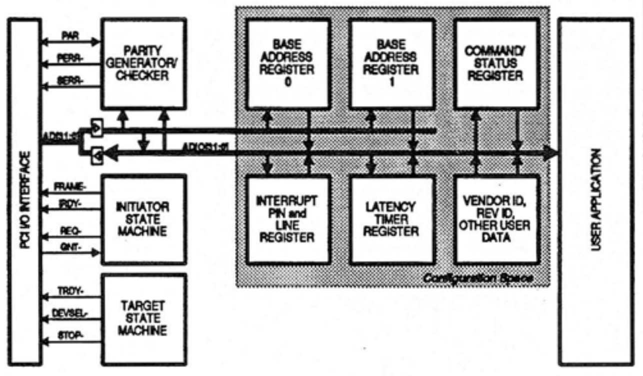
The LogiCore PCI Interface is a fully-integrated, tested, and validated schematic-based PCI Local Bus interface module design targeted for the XC4013E FPGA device. Both Initiator/Target and Target-only versions are available. This modular design can be customized by the user, and supports the quick implementation of prototype and production PCI applications.

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.

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.

Slide #11

LogiCore PCI Interface Block Diagram



The design was constructed in a hierarchical, modular manner.

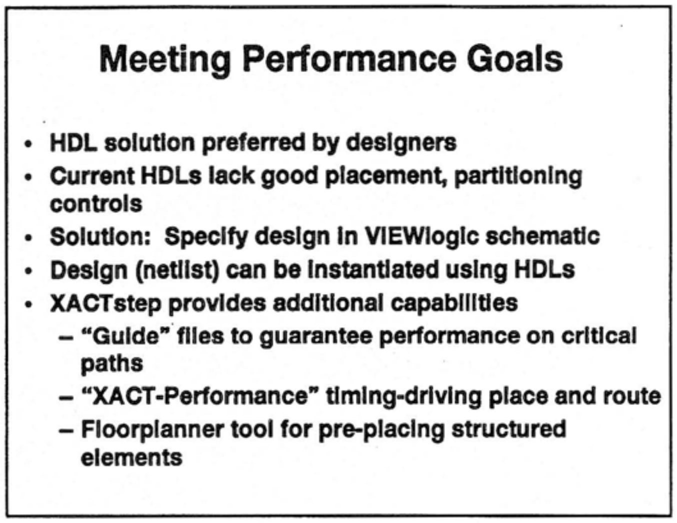
The I/O Interface block handles the physical connection to the PCI bus, including all signaling, input and output synchronization, output three-state controls, parity generation and checking, and (for initiator designs) request-grant handshaking for bus mastering. Parity errors detected on the address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

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.

Slide #12



Meeting Performance Goals

- **HDL solution preferred by designers**
- **Current HDLs lack good placement, partitioning controls**
- **Solution: Specify design in VIEWlogic schematic**
- **Design (netlist) can be instantiated using HDLs**
- **XACTstep provides additional capabilities**
  - **"Guide" files to guarantee performance on critical paths**
  - **"XACT-Performance" timing-driving place and route**
  - **Floorplanner tool for pre-placing structured elements**

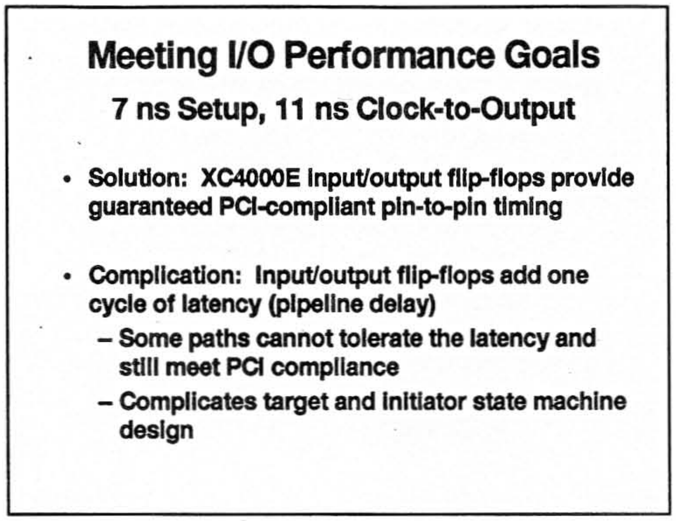
The most difficult challenge in the design of the LogiCore PCI Interface was meeting all the performance requirements while implementing the complex functions required of a PCI bus interface.

Ideally, the LogiCore PCI interface would have been specified and implemented from a hardware description language (HDL) like VHDL or Verilog. Most designers building high-density applications seem to prefer HDLs over schematics. Unfortunately, today's HDLs lack sufficient control over partitioning and pre-placement in order to effectively implement a full-speed PCI design.

Instead, the design team chose the Viewlogic schematic editor and simulator to specify the design. Using the XC4000E library, a designer can partition and pre-place logic within the device. For designers requiring HDL entry, the schematic-based netlist can be instantiated within the an HDL source file.

Careful control of the mapping, placement, and routing of the design was required to meet all the performance requirements. Fortunately, as mentioned before, the XACT*step* FPGA development system provides the appropriate controls. A "guide" file contains specific layout and routing information to guarantee some timing-critical 3-state and setup paths. Timing constraints for the timing-driven place & route tools are used to guarantee that all paths meet the necessary specifications. The floorplanner facilitated the pre-placement of structured elements such as the internal bi-directional bus and the data registers.

Slide #13



Meeting I/O Performance Goals

7 ns Setup, 11 ns Clock-to-Output

- **Solution: XC4000E input/output flip-flops provide guaranteed PCI-compliant pin-to-pin timing**
- **Complication: Input/output flip-flops add one cycle of latency (pipeline delay)**
  - **Some paths cannot tolerate the latency and still meet PCI compliance**
  - **Complicates target and initiator state machine design**

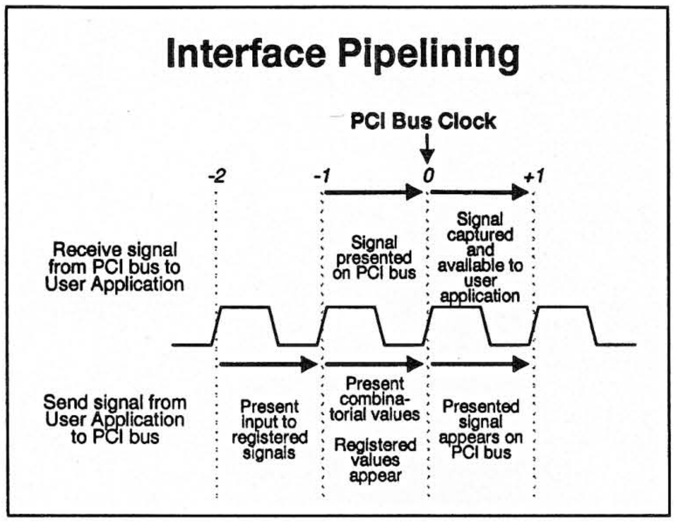
One of the most difficult challenges was meeting the 7 ns setup and 11 ns clock-to-out times. The dedicated input and output flip-flops in the XC4000E I/O blocks were used extensively.

Determining an FPGA's compliance to the I/O timing specifications can be difficult, since global buffer and interconnect delays as well as logic delays must be taken into account. For example, most FPGA data sheets specify clock-to-output timing for registered outputs relative to the clock at the output register (which may reside in an I/O block or nearby logic block). However, the PCI standard specifies pin-to-pin timing, and the clock-to-output valid delay is measured relative to the bus clock entering the FPGA device. Thus, the clock-to-output valid delay in an FPGA typically would include the delay in inputting the clock signal, bringing it to the FPGA's global clock buffer, routing the clock to the relevant I/O or logic block, the internal clock-to-output delay of the register, and the delay through the output buffer.

Similarly, set-up and hold times for registered inputs also must be examined carefully in FPGA designs. The internally-specified set-up and hold times must be adjusted to compensate for the delays associated with getting the clock signal through an input buffer and to the register. This may result in non-zero hold time requirements in some FPGAs. (The XC4000 Series devices include a deliberate delay in the data path to an input register to compensate for the clock distribution delay, and, thus, guarantee set-up and hold times relative to the clock input pin.)

Using the I/O flip-flops complicates the design in that it adds a level or two of pipelining. A majority of PCI applications can tolerate some latency. However, there are a few PCI transactions that cannot, so the control logic must be constructed accordingly.

Slide #14



Interface Pipelining

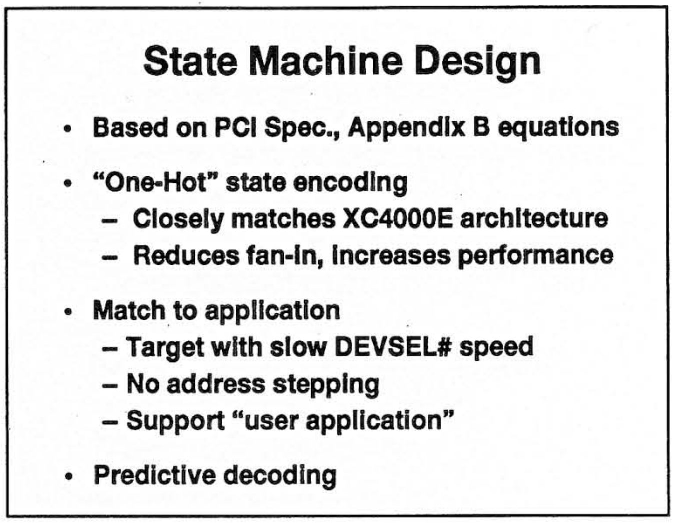
PCI Bus Clock
-2
-1
0
+1
Receive signal from PCI bus to User Application
Signal presented on PCI bus
Signal captured and available to user application
Send signal from User Application to PCI bus
Present input to registered signals
Present combinatorial values
Registered values appear
Presented signal appears on PCI bus

The LogiCore interface pipelines all of the bus control signals and the data path.

Arriving signals are captured in an input flip-flop and are available to the user application one cycle after they appear on the PCI bus. For example, data becomes available on the ADIO[31:0] bus internal to the LogiCore module one cycle after it appears on the PCI bus. Signals provided for the user interface, such as ADDR_VLD and DATA_VLD, signal the user application when valid information is available on the internal bus.

Most of the outputs connected to the PCI bus originate from an output flip-flop in order to meet clock-to-out performance requirements. Thus, when the user application is sending a combinatorial signal, the signal must be presented one cycle before it is to appear on the bus. If the signal first originates from a register in the user application, then that register's inputs must be presented two cycles before the signal is to appear on the bus.

Slide #15



State Machine Design

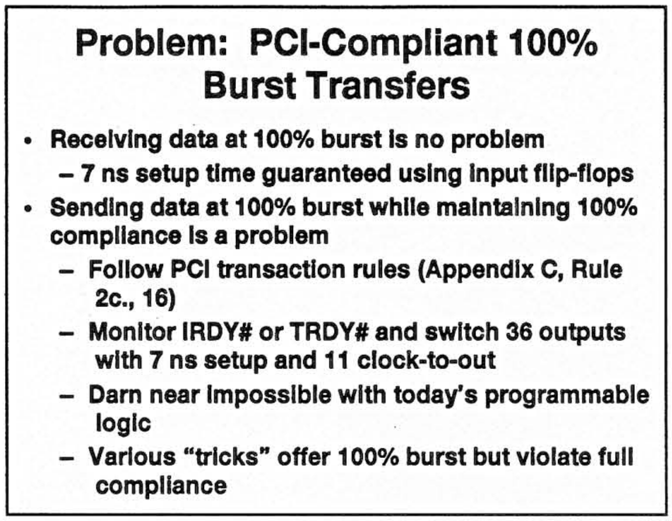
- **Based on PCI Spec., Appendix B equations**
- **"One-Hot" state encoding**
  - **Closely matches XC4000E architecture**
  - **Reduces fan-in, increases performance**
- **Match to application**
  - **Target with slow DEVSEL# speed**
  - **No address stepping**
  - **Support "user application"**
- **Predictive decoding**

The signals involved in bus transactions, such as FRAME#, IRDY#, and TRDY#, and the operation of data flow pipelines are controlled by state machines in the bus interface logic. Example state machines for controlling bus signaling are provided in Appendix B of the PCI Specification. To comply with PCI bus protocols and performance requirements, high-performance state machines are required. For FPGAs, one-hot-encoded (OHE) state machines are used (that is, state machines with one register per state and minimal decoding logic). These are well-suited for register-rich FPGA architectures.

The state machines described in Appendix B of the PCI specifications are "generic". The LogiCore PCI interface implements a subset of the full equations matched to the FPGA implementation. For example, the LogiCore interface only supports "slow" DEVSEL# decoding, so some of the equations can be significantly reduced. Likewise, some features, such as address-stepping, are not implemented. Some modifications were made to simplify the user interface, especially in the Initiator state machine.

To boost performance and decrease response time, some portions of the state machine employ predictive encoding.

Slide #16



Problem: PCI-Compliant 100% Burst Transfers

- **Receiving data at 100% burst is no problem**
  - **7 ns setup time guaranteed using input flip-flops**
- **Sending data at 100% burst while maintaining 100% compliance is a problem**
  - **Follow PCI transaction rules (Appendix C, Rule 2c., 16)**
  - **Monitor IRDY# or TRDY# and switch 36 outputs with 7 ns setup and 11 clock-to-out**
  - **Darn near impossible with today's programmable logic**
  - **Various "tricks" offer 100% burst but violate full compliance**

The PCI bus protocols encourage burst-oriented data flows between bus agents, facilitating the use of pipelined data flows within PCI bus interface logic. Pipelining techniques are often key to successfully supporting data transfers at the maximum throughput of the bus. While wait states during bus transactions are permitted and occasionally may be necessary, a high frequency of wait states is counter to the high-performance goals of the PCI standard.

While the PCI protocols are compatible with data pipelining, there are three exceptions: when an initiator inserts wait states during a burst read cycle, when a target inserts waits states during a burst write cycle, and the sequence of events that starts on the last transfer of a burst. In these situations, the IRDY#, TRDY# and FRAME# signals must be sensed directly and responded to within their 7 ns set-up time on the bus.

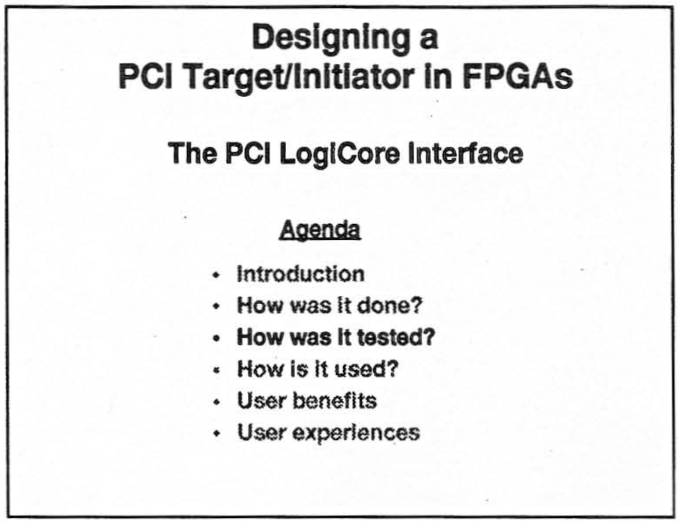
The LogiCore PCI interface can receive data at 100% burst rates. Data is captured in flip-flops and made available to the user application.

However, the LogiCore interface does not support 100% burst when it is sending data. This is because the LogiCore interface strictly adheres to the PCI transaction rules. These rules state that, once data is presented, the agent providing the data cannot change the data until the transfer is complete. To support 100% burst, an agent would need to present the data, monitor the receiving agent's ready line (IRDY# or TRDY#), and present new data once the other agent has asserted its ready line. Meeting the 7 ns setup time on the read line and then sending the next data within the 11 ns clock-to-out timing in the same cycle is nearly impossible with today's programmable logic devices. Without employing any "tricks", the LogiCore interface is limited to 50% of the maximum burst rate when it is providing data. Most importantly, it does not violate any bus transaction rules.

In some embedded applications, various tricks can be used to attain 100% burst. However, this means violating strict adherence to the PCI specification.

Some of the PCI interface designs being offered today advertise full burst support, but only by assuming that the receiving agent in a burst transaction never generates a wait state. Therefore, these designs are not fully PCI compatible.

Slide #17



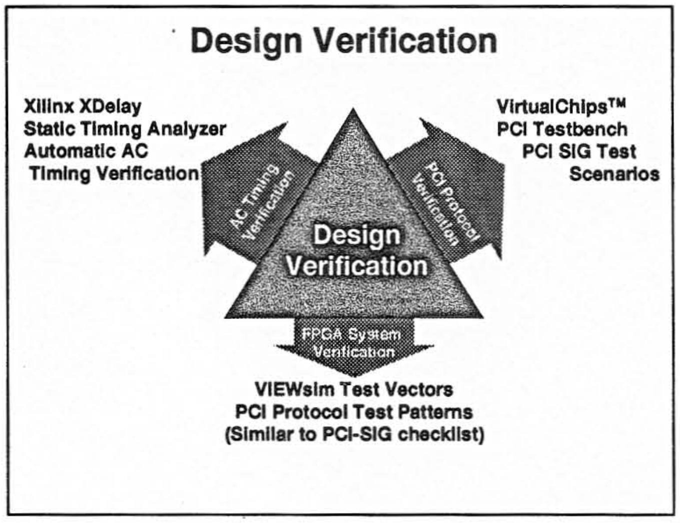
Designing a PCI Target/Initiator in FPGAs

The PCI LogiCore Interface

Agenda

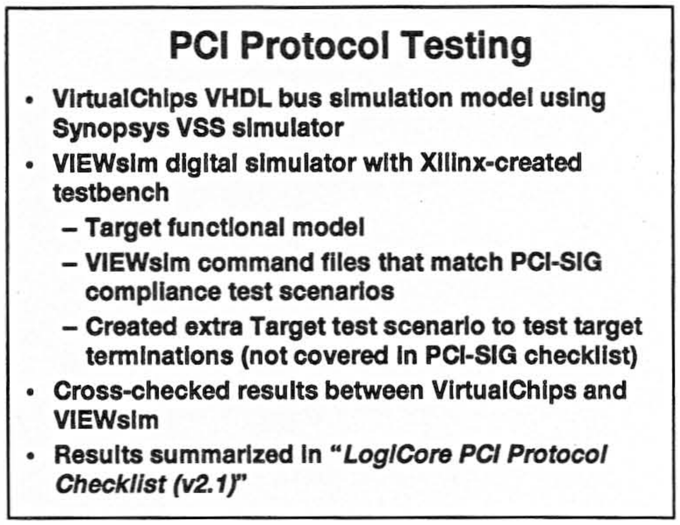
- Introduction
- How was it done?
- How was it tested?
- How is it used?
- User benefits
- User experiences

Slide #18



Tools used to verify the operation of the LogiCore PCI interface include simulators and static timing analyzers.

Slide #19



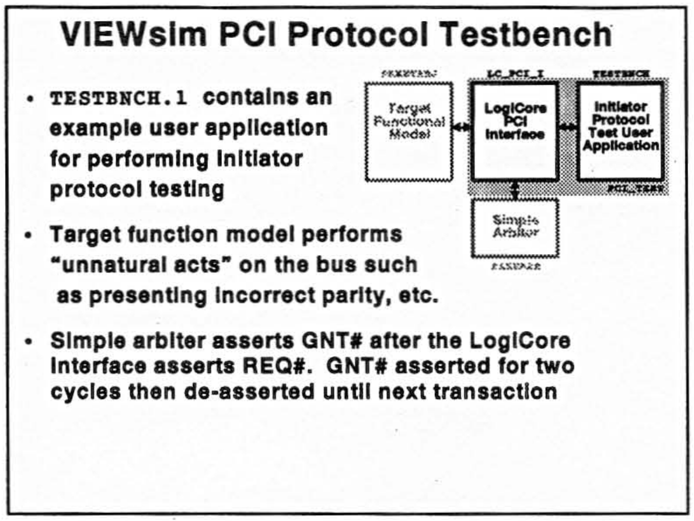
PCI Protocol Testing

- VirtualChips VHDL bus simulation model using Synopsys VSS simulator
- VIEWsim digital simulator with Xilinx-created testbench
  - Target functional model
  - VIEWsim command files that match PCI-SIG compliance test scenarios
  - Created extra Target test scenario to test target terminations (not covered in PCI-SIG checklist)
- Cross-checked results between VirtualChips and VIEWsim
- Results summarized in "*LogiCore PCI Protocol Checklist (v2.1)*"

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).

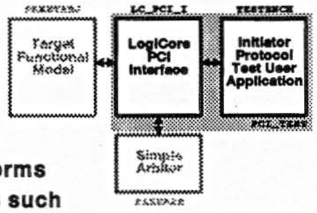
The design also was verified via actual FPGA device implementations at multiple "beta site" accounts.

Slide #20

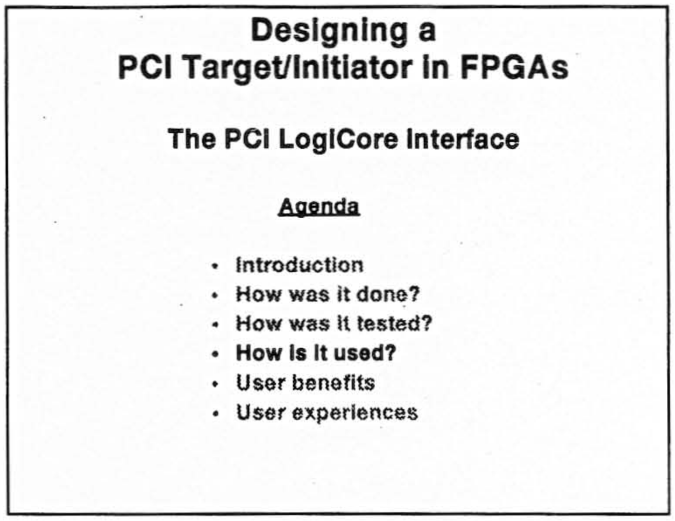


VIEWsim PCI Protocol Testbench

- TESTBNCH.1 contains an example user application for performing initiator protocol testing
- Target function model performs "unnatural acts" on the bus such as presenting incorrect parity, etc.
- Simple arbiter asserts GNT# after the LogiCore interface asserts REQ#. GNT# asserted for two cycles then de-asserted until next transaction



Slide #21



Designing a PCI Target/Initiator in FPGAs

The PCI LogiCore Interface

Agenda

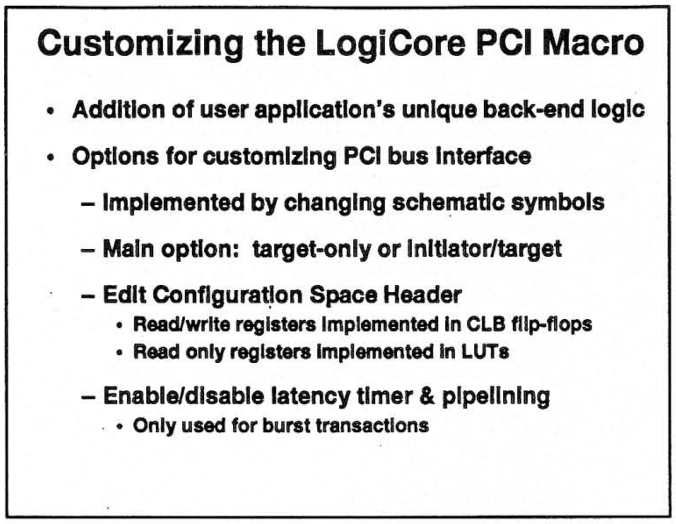
- Introduction
- How was it done?
- How was it tested?
- **How is it used?**
- User benefits
- User experiences

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. The target functional model (faketarg) has been designed to respond according to the scenarios described in the PCI SIG Compliance Checklist. During protocol testing, the target responds in ways that a "real" target would not, such as forcing invalid parity. The simple arbiter (fakearb) merely asserts GNT# after the LogiCore interface asserts it REQ# pin.

The various compliance tests can be executed individually. A full test requires about two hours to execute on a workstation platform.

Slide #22



Customizing the LogiCore PCI Macro

- Addition of user application's unique back-end logic
- Options for customizing PCI bus interface
  - Implemented by changing schematic symbols
  - Main option: target-only or initiator/target
  - Edit Configuration Space Header
    - Read/write registers implemented in CLB flip-flops
    - Read only registers implemented in LUTs
  - Enable/disable latency timer & pipelining
    - Only used for burst transactions

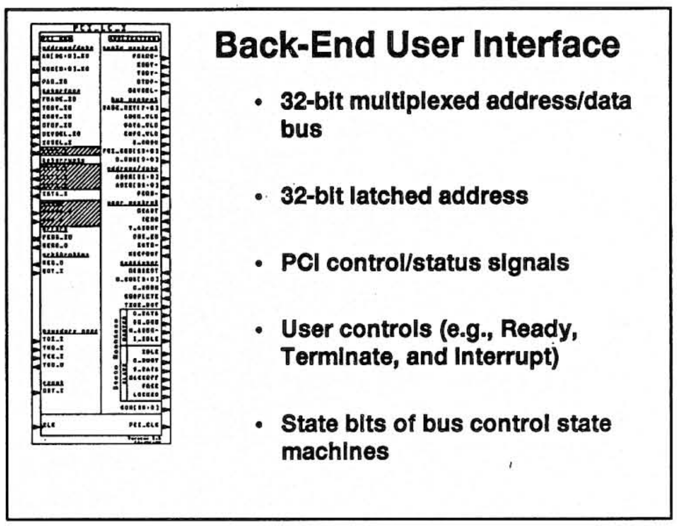
The LogiCore PCI Interface provides a foundation PCI bus interface design that can be tailored for the specific application. Besides adding the unique back-end logic for the application, users can customize the PCI bus interface itself.

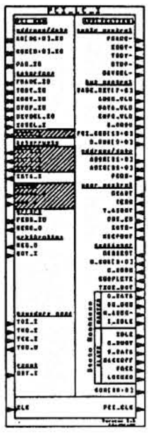
For users of the Initiator/Target version, the first step in customizing the PCI interface macro is to define whether the application is a target-only or initiator/target function. As delivered, the macro is configured as an initiator/target interface; configuring the macro to be a target-only interface 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.

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.

Slide #23



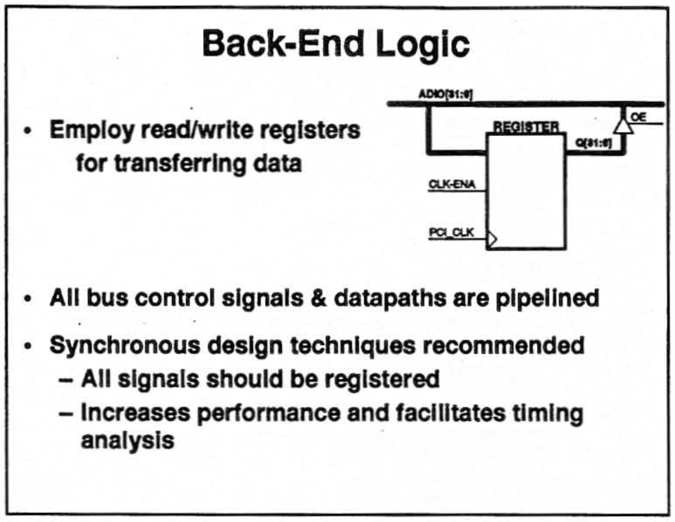


Back-End User Interface

- 32-bit multiplexed address/data bus
- 32-bit latched address
- PCI control/status signals
- User controls (e.g., Ready, Terminate, and Interrupt)
- State bits of bus control state machines

Every PCI application requires a PCI bus interface plus an interface to the user application (i.e., the "back-end" device being connected to the bus). The user connects the LogiCore module to other modules to complete the design. A general-purpose interface provided in the LogiCore design is used to connect the application's logic to the PCI bus, allowing the back-end logic to reside in the remainder of the FPGA device. Connections to the user interface include a 32-bit multiplexed address/data bus (the same address/data lines that connect to the PCI bus), a 32-bit latched address, several of the PCI bus control and status signals (including FRAME#, TRDY#, IRDY#, and CBE[0-3]), user control signals (such as READY, TERMINATE, and INTERRUPT), and several state bits from the bus control state machines. Thus, if desired, back-end logic can monitor and control all appropriate bus activity.

Slide #24



Back-End Logic

- Employ read/write registers for transferring data

ADIO[31:0]
REGISTER
OE
Q[31:0]
CLK-ENA
PCI_CLK

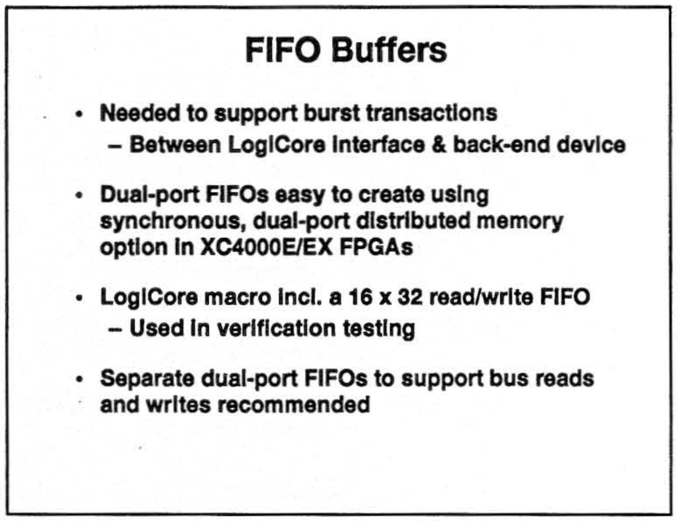
- All bus control signals & datapaths are pipelined
- Synchronous design techniques recommended
  - All signals should be registered
  - Increases performance and facilitates timing analysis

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. The figure 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.

Slide #25



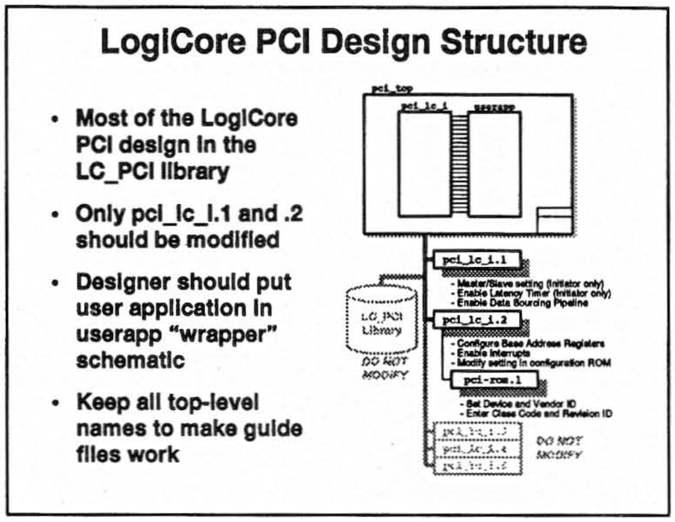
FIFO Buffers

- Needed to support burst transactions
  - Between LogiCore interface & back-end device
- Dual-port FIFOs easy to create using synchronous, dual-port distributed memory option in XC4000E/EX FPGAs
- LogiCore macro incl. a 16 x 32 read/write FIFO
  - Used in verification testing
- Separate dual-port FIFOs to support bus reads and writes recommended

Typically, 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.

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. 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.

Slide #26



LogiCore PCI Design Structure

- Most of the LogiCore PCI design in the LC_PCI library
- Only pci_lc_i.1 and .2 should be modified
- Designer should put user application in userapp "wrapper" schematic
- Keep all top-level names to make guide files work

pci_top
pci_lc_i
userapp
pci_lc_i.1
- Master/Slave setting (Initiator only)
- Enable Latency Timer (Initiator only)
- Enable Data Sourcing Pipeline
LC_PCI Library
DO NOT MODIFY
pci_lc_i.2
- Configure Base Address Registers
- Enable Interrupts
- Modify setting in configuration ROM
pci-rom.1
- Set Device and Vendor ID
- Enter Class Code and Revision ID
DO NOT MODIFY

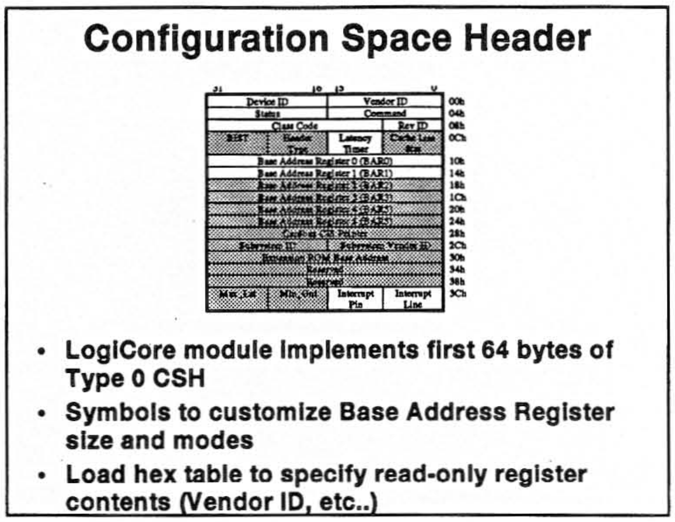
The LogiCore PCI Interface provides a foundation PCI bus interface design that can be tailored for the specific application. Besides adding the unique back-end logic for the application, users can customize the PCI bus interface itself.

The LogiCore module's design structure and modifiable schematics are shown in the figure. In most cases, customization of the interface logic simply involves replacing particular symbols in the Viewlogic schematics with other pre-prepared symbols, using the schematic editor's 'Change Component' command.

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. This is required to guarantee that critical instance names and net names match that of the "guide file". This guide file is used to direct the placement and routing of timing-critical logic, and was carefully designed to achieve maximum performance.

Where applicable, logic contained within the schematic is trimmed during the compilation process (i.e., unused logic is automatically removed from the design).

Slide #27



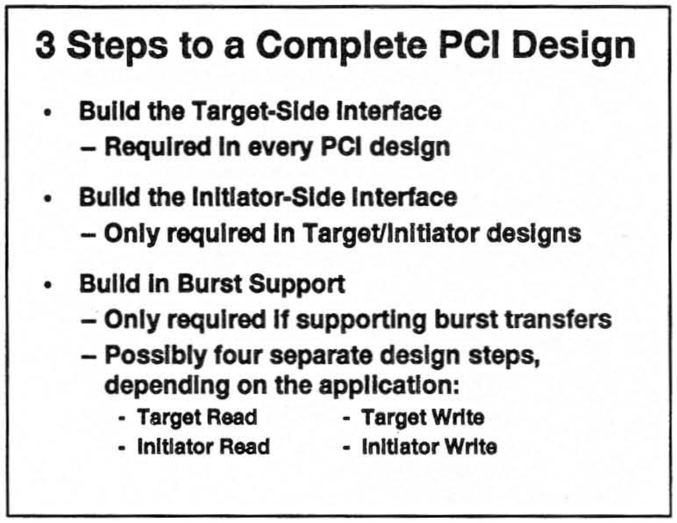
Configuration Space Header

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Register 0 (BAR0)				10h
Base Address Register 1 (BAR1)				14h
Base Address Register 2 (BAR2)				18h
Base Address Register 3 (BAR3)				1Ch
Base Address Register 4 (BAR4)				20h
Base Address Register 5 (BAR5)				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved				34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

- LogiCore module implements first 64 bytes of Type 0 CSH
- Symbols to customize Base Address Register size and modes
- Load hex table to specify read-only register contents (Vendor ID, etc..)

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.

Slide #28

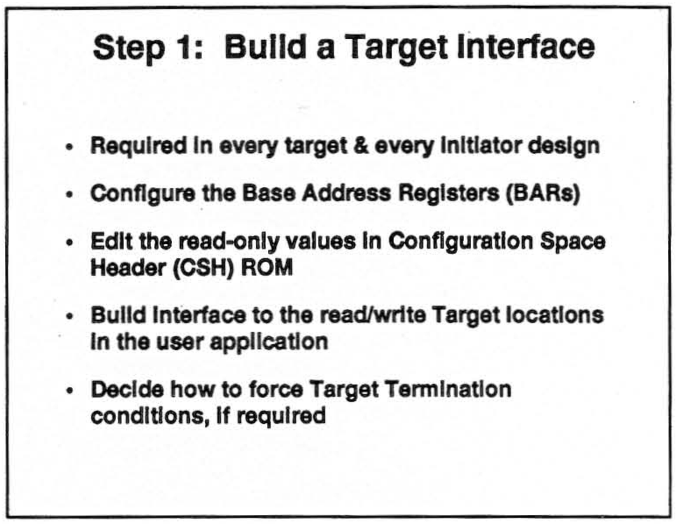


3 Steps to a Complete PCI Design

- **Build the Target-Side Interface**
  - **Required in every PCI design**
- **Build the Initiator-Side Interface**
  - **Only required in Target/Initiator designs**
- **Build in Burst Support**
  - **Only required if supporting burst transfers**
  - **Possibly four separate design steps, depending on the application:**
    - **Target Read**
    - **Target Write**
    - **Initiator Read**
    - **Initiator Write**

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.

Slide #29

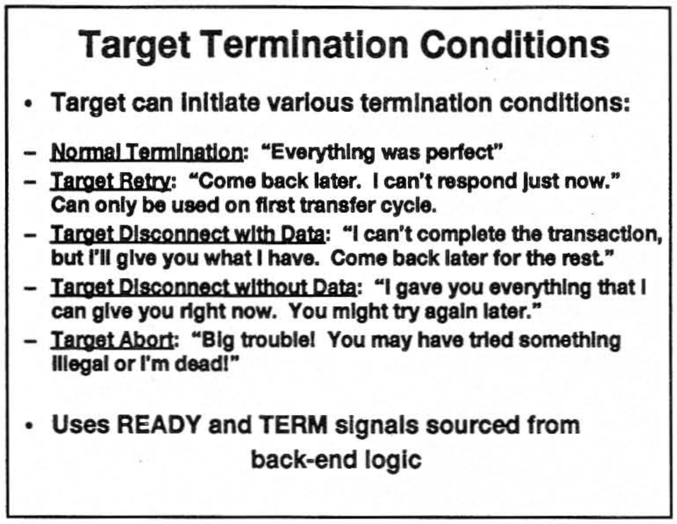


Step 1: Build a Target Interface

- **Required in every target & every initiator design**
- **Configure the Base Address Registers (BARs)**
- **Edit the read-only values in Configuration Space Header (CSH) ROM**
- **Build interface to the read/write Target locations in the user application**
- **Decide how to force Target Termination conditions, if required**

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.

Slide #30

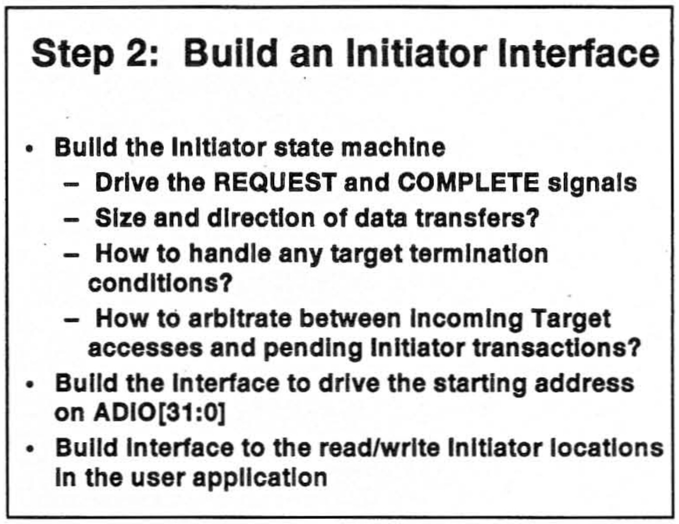


Target Termination Conditions

- Target can initiate various termination conditions:
  - Normal Termination: "Everything was perfect"
  - Target Retry: "Come back later. I can't respond just now." Can only be used on first transfer cycle.
  - Target Disconnect with Data: "I can't complete the transaction, but I'll give you what I have. Come back later for the rest."
  - Target Disconnect without Data: "I gave you everything that I can give you right now. You might try again later."
  - Target Abort: "Big trouble! You may have tried something illegal or I'm dead!"
- Uses READY and TERM signals sourced from back-end logic

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.

Slide #31

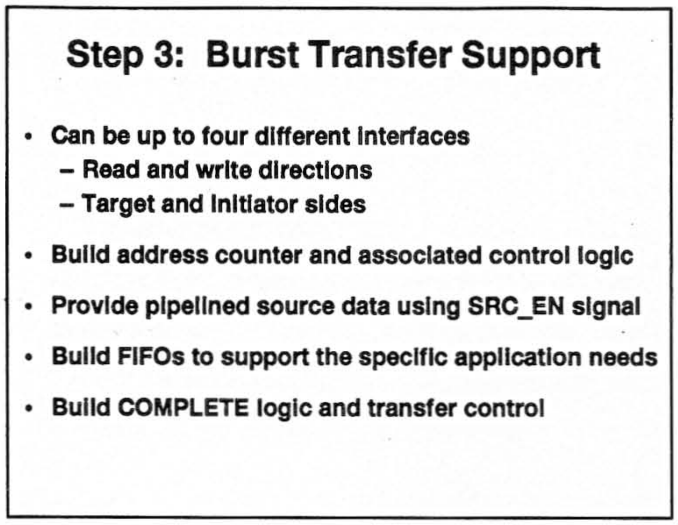


Step 2: Build an Initiator Interface

- Build the initiator state machine
  - Drive the REQUEST and COMPLETE signals
  - Size and direction of data transfers?
  - How to handle any target termination conditions?
  - How to arbitrate between incoming Target accesses and pending Initiator transactions?
- Build the interface to drive the starting address on ADIO[31:0]
- Build interface to the read/write initiator locations in the user application

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.

Slide #32

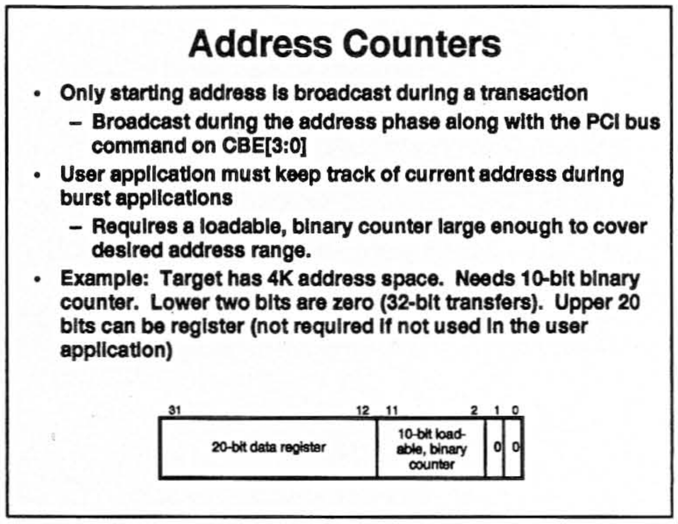


Step 3: Burst Transfer Support

- Can be up to four different interfaces
  - Read and write directions
  - Target and Initiator sides
- Build address counter and associated control logic
- Provide pipelined source data using SRC_EN signal
- Build FIFOs to support the specific application needs
- Build COMPLETE logic and transfer control

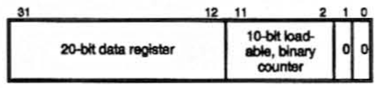
Performing a single data transfer across the PCI bus is the simplest type of transaction. However, because of the overhead of distributed address decoding, this wastes valuable bus bandwidth. Achieving high bandwidth requires the use of burst transactions, where multiple data words are transferred during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

Slide #33



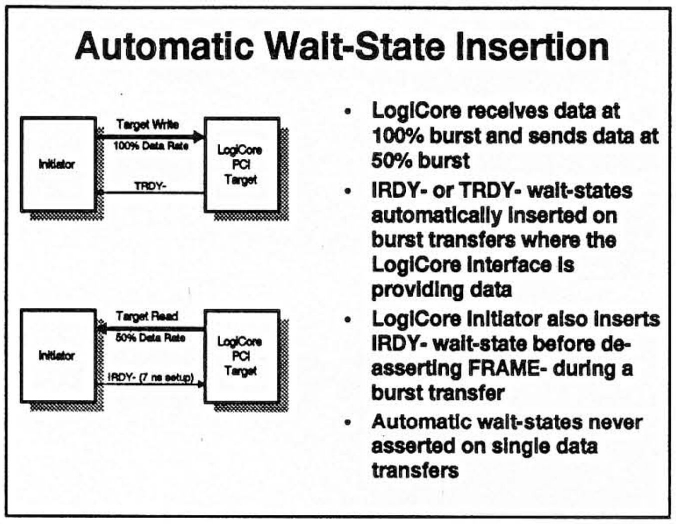
Address Counters

- Only starting address is broadcast during a transaction
  - Broadcast during the address phase along with the PCI bus command on CBE[3:0]
- User application must keep track of current address during burst applications
  - Requires a loadable, binary counter large enough to cover desired address range.
- Example: Target has 4K address space. Needs 10-bit binary counter. Lower two bits are zero (32-bit transfers). Upper 20 bits can be register (not required if not used in the user application)

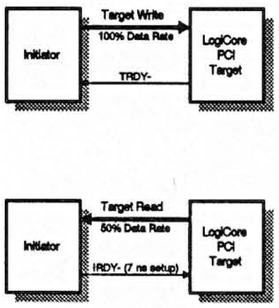


In PCI bus burst transactions, only the starting address is broadcast over the bus. Thus, during burst transfers, the back-end logic must keep track of the current address. Typically, both target and initiator control logic must keep a local copy of the current address pointer and increment it after 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 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.

Slide #34



Automatic Wait-State Insertion



- **LogiCore receives data at 100% burst and sends data at 50% burst**
- **IRDY- or TRDY- wait-states automatically inserted on burst transfers where the LogiCore interface is providing data**
- **LogiCore Initiator also inserts IRDY- wait-state before de-asserting FRAME- during a burst transfer**
- **Automatic wait-states never asserted on single data transfers**

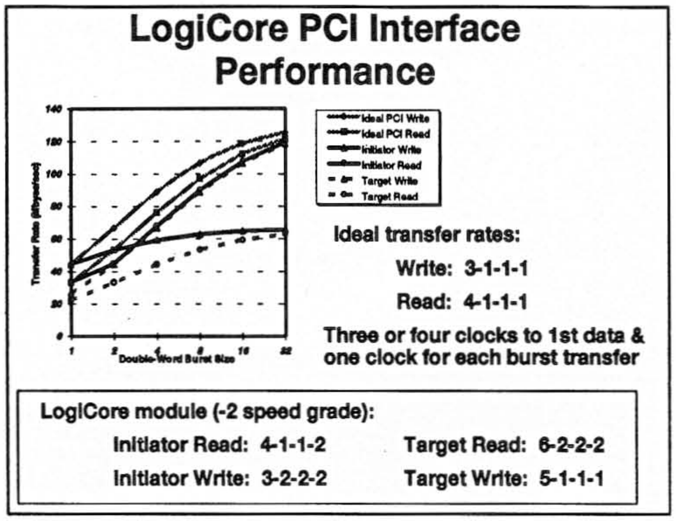
The maximum theoretical PCI bus bandwidth is 132 Mbytes/sec. However, actual system bandwidth varies tremendously from one platform to another. PCI bus performance is controlled by three key factors: aggregate bandwidth, data throughput, and access latency.

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).

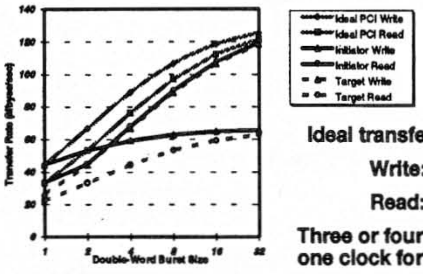
The LogiCore PCI Interface requires additional clock cycles to decode the address (referred to as SLOW decoding in the PCI Specification). Target burst write transactions do not require any wait states; thus, the default transfer rate for I/O and Memory Write transactions is 5-1-1-1. Initiators add an 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. (See slide 16.) 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.)

Slide #35



LogiCore PCI Interface Performance



Ideal transfer rates:

Write: 3-1-1-1

Read: 4-1-1-1

Three or four clocks to 1st data & one clock for each burst transfer

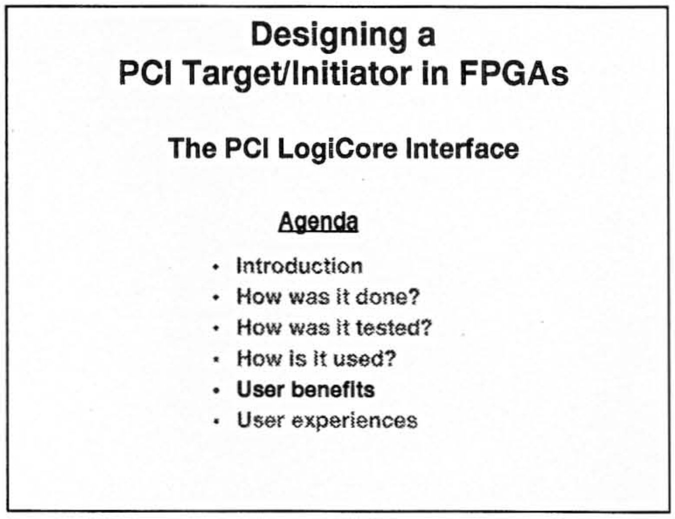
LogiCore module (-2 speed grade):

Initiator Read: 4-1-1-2	Target Read: 6-2-2-2
Initiator Write: 3-2-2-2	Target Write: 5-1-1-1

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

The bandwidth of the PCI bus is primarily determined by the size of the burst transfer.

Slide #36



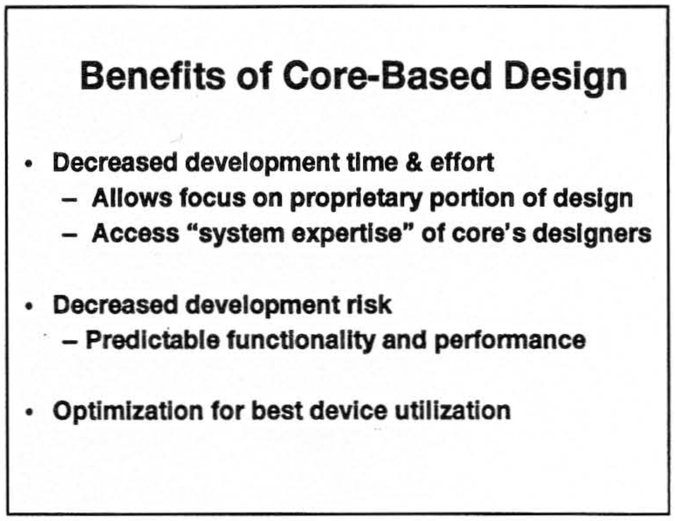
Designing a PCI Target/Initiator in FPGAs

The PCI LogiCore Interface

Agenda

- Introduction
- How was it done?
- How was it tested?
- How is it used?
- **User benefits**
- User experiences

Slide #37



Benefits of Core-Based Design

- Decreased development time & effort
  - Allows focus on proprietary portion of design
  - Access "system expertise" of core's designers
- Decreased development risk
  - Predictable functionality and performance
- Optimization for best device utilization

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.

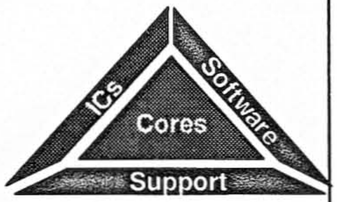
Where appropriate, cores should utilize the available architectural features of the target FPGA, such as dedicated arithmetic carry logic, internal three-state buffers, and on-chip memory.

Slide #38

Core-Based Design Requirements

Silicon, Software, Service

- **Predictable functionality and performance**
- **Optimized for best device utilization**
- **Accessible system expertise**
- **Supporting tools and collateral**



Selection criteria for choosing a core would include, of course, the functionality and performance of that core. These factors, in turn, depend on how well the design of the core is optimized for the target device architecture. However, potential users should also consider three other main factors that lead to a "complete" solution: silicon, software, and service. In other words, is the selected device an appropriate target for this function, does the development software properly support core-based design, and are technical support resources available to assist if problems arise?

Core-based designs are most appropriate for high-density FPGAs with a rich feature set. The FPGA development environment must support the appropriate design entry, synthesis, and simulation tools needed for the particular core design.

For hard cores, performance information obtained with static timing analysis and/or timing simulation should be supplied. The methodology and tools used to pre-define the implementation of the critical paths, such as placement and timing constraint files or guide designs, should be well-understood.

To insure the productivity gains that motivated the use of a core, extensive documentation and application support must be available. Of course, the degree of support required for core integration is often proportional to the complexity and flexibility of the core.

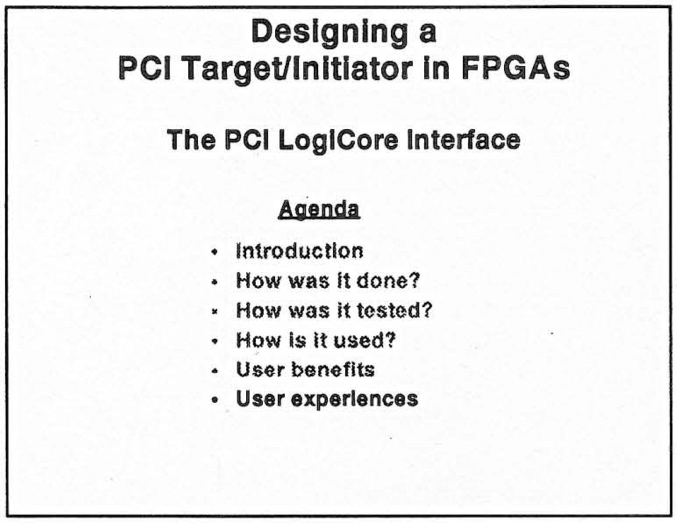
Slide #39

But It's Not a Panacea ...

- **PCI is difficult, even in ASICs**
- **Use of LogiCore PCI module requires**
  - **Understanding of PCI protocols**
  - **High-performance, pipelined design techniques**
    - **Use of guide files, TIMESPECs, floorplanner**

However, the use of pre-defined cores should not be considered a panacea for the development of complex systems. While potentially saving time and effort, cores still must be interfaced to other logic and integrated with the overall design flow. Cores can present difficulties with synthesis, simulation, layout, and system verification. Furthermore, the fact that the FPGA vendor supplies or endorses the supplier of a particular core does not preclude the designer from understanding the function of that core and ensuring that the appropriate constraints are applied during the synthesis and physical implementation of the design in the FPGA device (i.e., when running the 'place and route' tools).

Slide #40



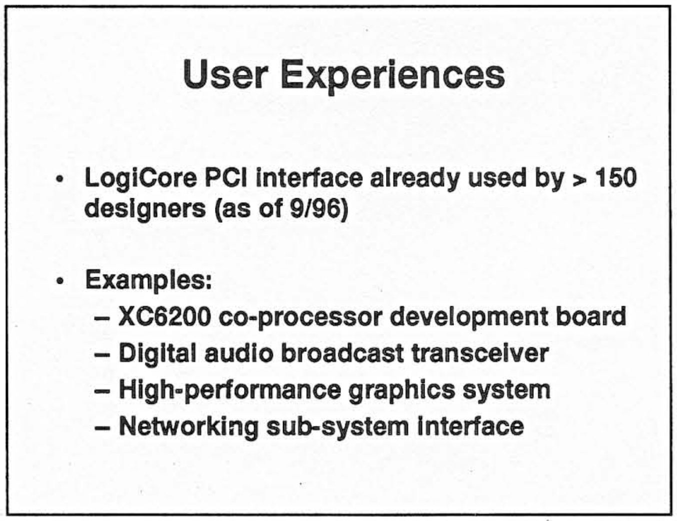
Designing a PCI Target/Initiator in FPGAs

The PCI LogiCore Interface

Agenda

- Introduction
- How was it done?
- How was it tested?
- How is it used?
- User benefits
- User experiences

Slide #41



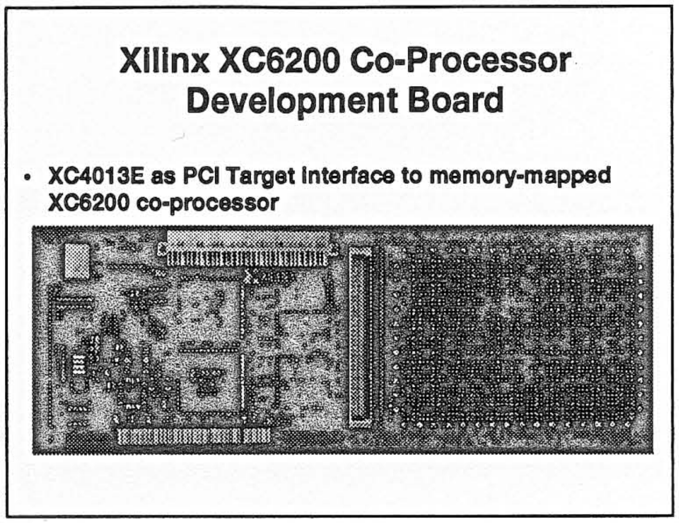
User Experiences

- LogiCore PCI interface already used by > 150 designers (as of 9/96)
- Examples:
  - XC6200 co-processor development board
  - Digital audio broadcast transceiver
  - High-performance graphics system
  - Networking sub-system interface

The LogiCore PCI interface was extensively "beta-sited" in late 1995 and early 1996. Since its introduction, over 150 users have licensed use of this module.

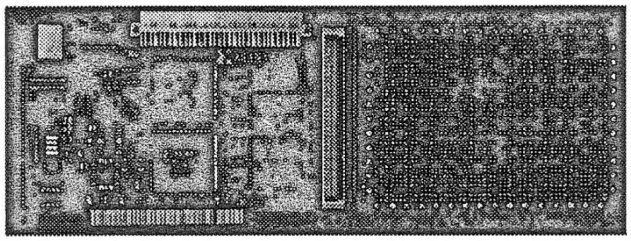
Here are four examples of applications that are using the LogiCore PCI Interface in XC4000 Series FPGAs.

Slide #42



Xilinx XC6200 Co-Processor Development Board

- XC4013E as PCI Target interface to memory-mapped XC6200 co-processor

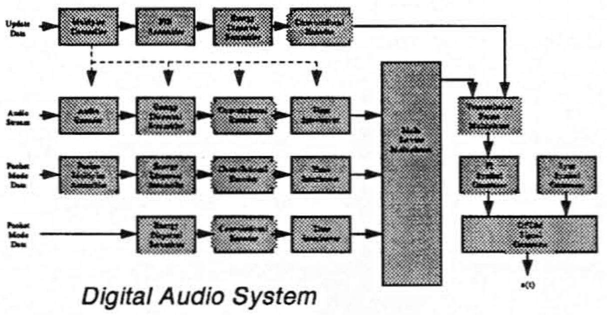


Xilinx is using the LogiCore PCI module in one if its own products, an evaluation and prototyping board for the XC6200 family FPGAs. The XC6200 devices are FPGAs whose architecture is optimized for use as a coprocessor in high-performance data processing applications. An XC4013E FPGA on the prototyping board provides the interface between a PCI slot in a PC and the XC6200 processor on this evaluation board.

Slide #43

Digital Audio Transceiver

- XC4013E as PCI Target interface



Digital Audio System

Engineers at Delco Electronics have designed a Digital Audio Broadcast (DAB) transceiver that employs an XC4013E FPGA and the LogiCore module to interface between the PCI bus and the transceiver. Much of the transceiver's logic also is integrated into the FPGA device.

Slide #44

Digital Audio Transceiver

"We chose Xilinx modules because they're fully verified. As a result, the module worked the first time, saved us over six months in development time, and allowed us to focus our resources on our system design."

Brian Warren, Senior Design Engineer, Delco Electronics

In this case, use of the LogiCore PCI module delivered on its promise of shortening the design cycle and allowing the designers to focus on the proprietary portions of their design.

Slide #45

High-Performance Graphics System

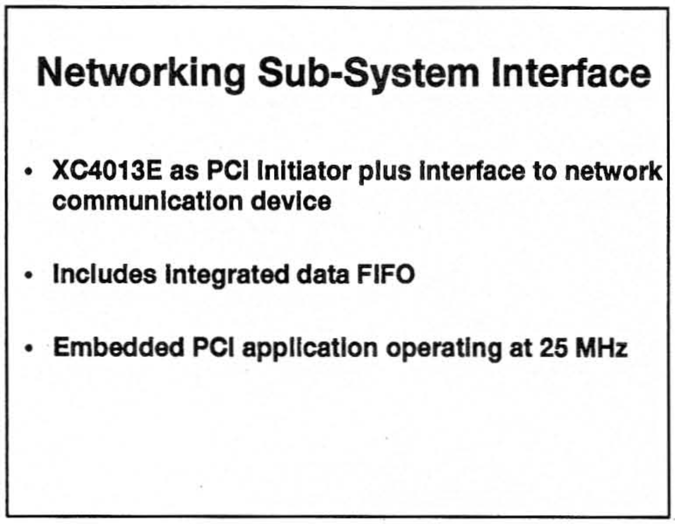
- XC4020E as PCI-PCI bridge
- Integrates bridge and support logic
- Unsupported use and unsupported device
- Implemented in Xilinx HardWire for high-volume production
- Helped to locate and repair flaw in another PCI ASIC

In this design of a high-end graphics controller by a leading electronic game manufacturer, the PCI LogiCore module served as a starting point for the development of a PCI-to-PCI bridge implemented in an XC4020E FPGA.

In this case, the flexibility of the FPGA-based approach had an unexpected benefit. When a flaw was discovered in a large ASIC device in the system, designers were able to change the logic in the FPGA to repair the problem, thereby avoiding a redesign of the ASIC device, resulting in a considerable savings both in terms of expense and development time.

This design is expected to reach significant volumes. The user plans to convert the XC4020E FPGA design to an architecturally-compatible Xilinx HardWire device as a cost savings during volume production.

Slide #46



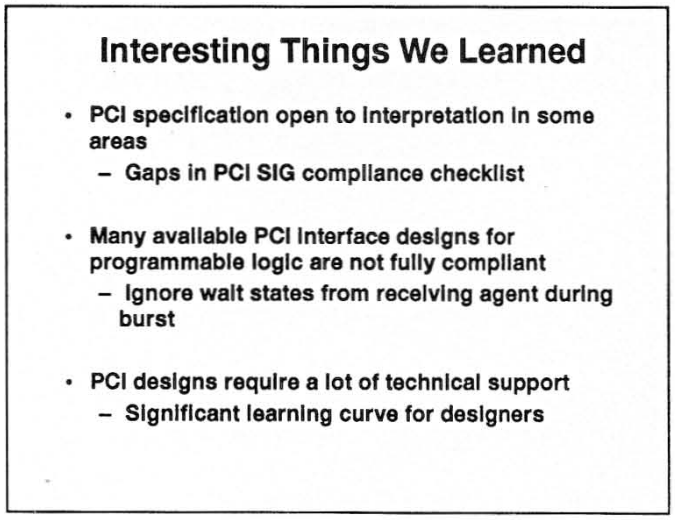
Networking Sub-System Interface

- XC4013E as PCI Initiator plus interface to network communication device
- Includes integrated data FIFO
- Embedded PCI application operating at 25 MHz

In this application, the LogiCore PCI interface replaced a dedicated PCI chip set and various glue logic. The application is a PCI interface to a network communication device. To boost overall system performance, data is burst over the bus in four-word transfers and held within FIFO buffers integrated on the FPGA with other system glue logic.

This design also has been converted to a HardWire device to reduce costs for high-volume manufacturing.

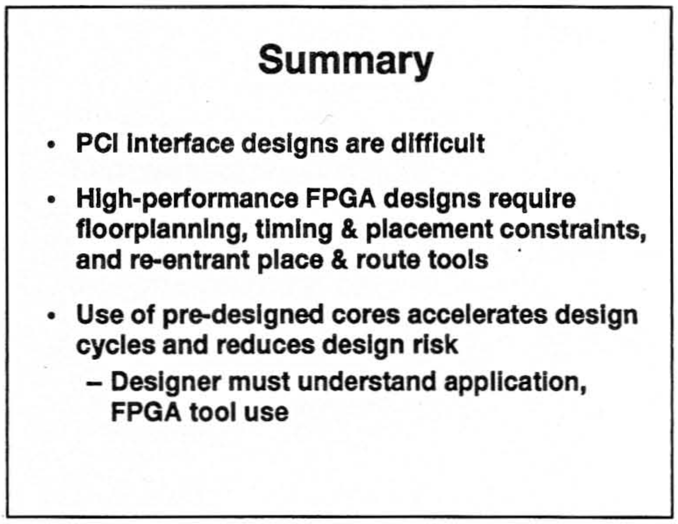
Slide #47



Interesting Things We Learned

- PCI specification open to interpretation in some areas
  - Gaps in PCI SIG compliance checklist
- Many available PCI interface designs for programmable logic are not fully compliant
  - Ignore wait states from receiving agent during burst
- PCI designs require a lot of technical support
  - Significant learning curve for designers

Slide #48



Summary

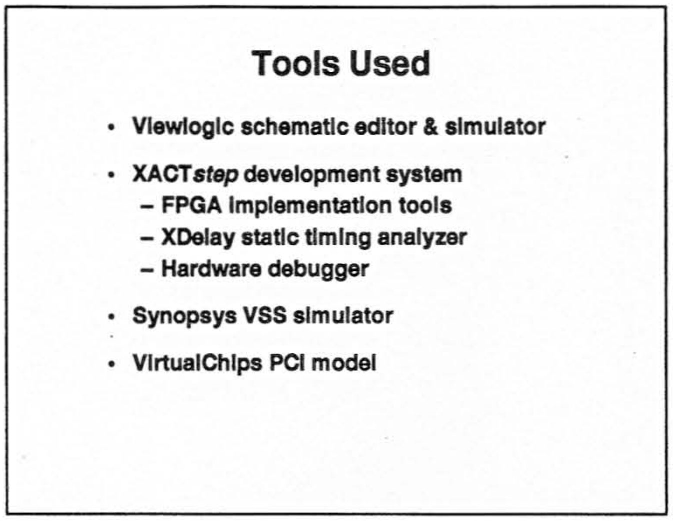
- PCI interface designs are difficult
- High-performance FPGA designs require floorplanning, timing & placement constraints, and re-entrant place & route tools
- Use of pre-designed cores accelerates design cycles and reduces design risk
  - Designer must understand application, FPGA tool use

PCI-compatible FPGA devices bring the system-integration, flexibility, and time-to-market benefits of high-density programmable logic to the PCI design community. With careful design, these devices can provide the performance, density, and routability to handle complex structures such as pipelined data paths, 32-bit parity generation, and PCI bus control.

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.

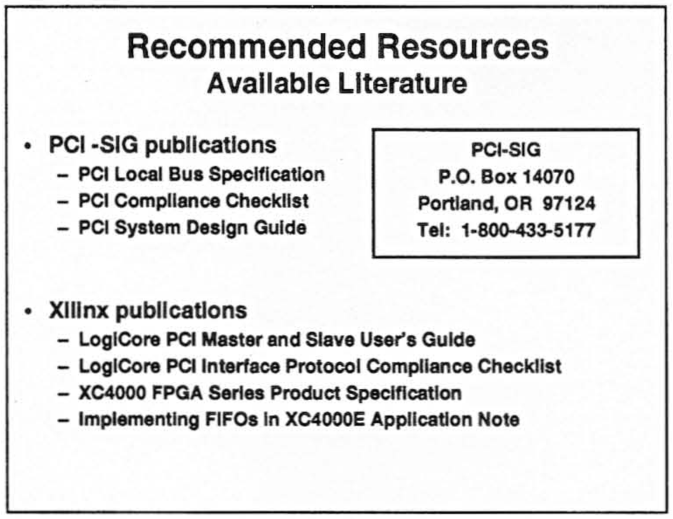
Slide #49



Tools Used

- Viewlogic schematic editor & simulator
- XACT*step* development system
  - FPGA implementation tools
  - XDelay static timing analyzer
  - Hardware debugger
- Synopsys VSS simulator
- VirtualChips PCI model

Slide #50



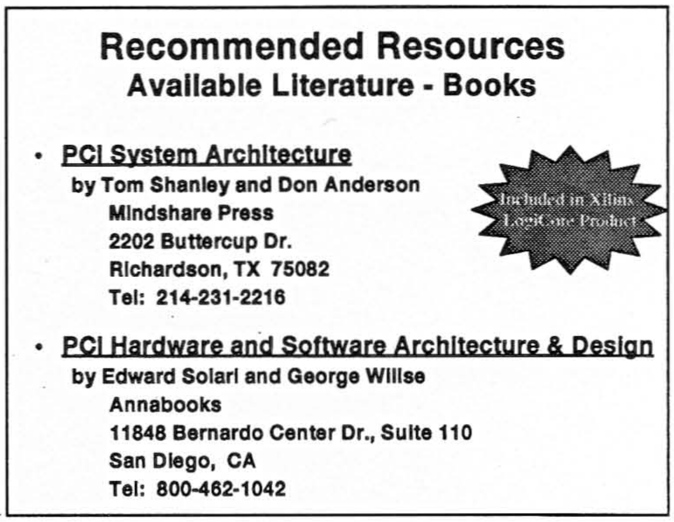
Recommended Resources
Available Literature

- PCI -SIG publications
  - PCI Local Bus Specification
  - PCI Compliance Checklist
  - PCI System Design Guide

PCI-SIG
P.O. Box 14070
Portland, OR 97124
Tel: 1-800-433-5177

- Xilinx publications
  - LogiCore PCI Master and Slave User's Guide
  - LogiCore PCI Interface Protocol Compliance Checklist
  - XC4000 FPGA Series Product Specification
  - Implementing FIFOs in XC4000E Application Note

Slide #51



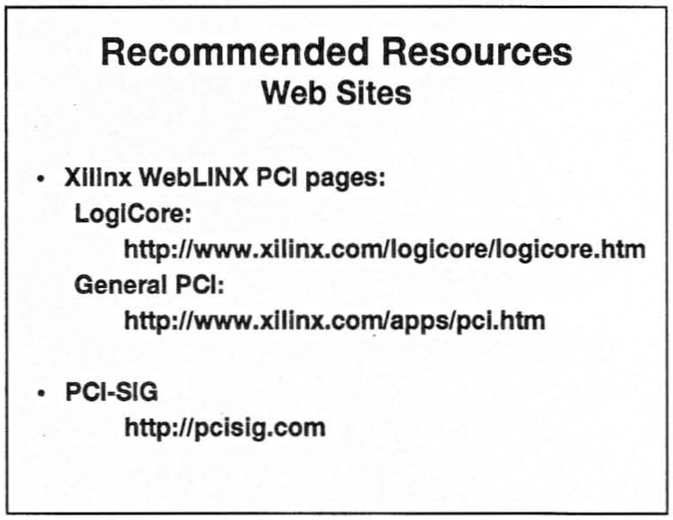
Recommended Resources
Available Literature - Books

- PCI System Architecture
  by Tom Shanley and Don Anderson
  Mindshare Press
  2202 Buttercup Dr.
  Richardson, TX 75082
  Tel: 214-231-2216

Included in Xilinx LogiCore Product

- PCI Hardware and Software Architecture & Design
  by Edward Solari and George Willse
  Annabooks
  11848 Bernardo Center Dr., Suite 110
  San Diego, CA
  Tel: 800-462-1042

Slide #52



Recommended Resources
Web Sites

- Xilinx WebLINX PCI pages:
  LogiCore:
  http://www.xilinx.com/logicore/logicore.htm
  General PCI:
  http://www.xilinx.com/apps/pci.htm
- PCI-SIG
  http://pcisig.com