The Challenges of Doing a PCI Design in FPGAs

Nupur Shah

What are the challenges of doing PCI in FPGAs? This paper covers the issues Xilinx has discovered in doing PCI in FPGAs, how they have been surmounted and what the designer needs to think about when doing such a design. Successful PCI in FPGAs requires more than a device that is PCI compliant; a usable core and good design methodology are equally important. Timing has been and continues to be a major issue as the industry begins moving into 64-bit/66 MHz PCI. Design methodology is critical to assuring a successful design.

1.0 Introduction

The Peripheral Component Interconnect (PCI) local bus specification is an interface protocol that is ideal for high bandwidth/high performance applications. The PCI interface also offers the advantages of processor independence, low power, low pin count and auto configuration. It is because of these advantages and its high performance that PCI has gained wide acceptance in the computer industry by computer and peripheral component manufactures, as well as communication and industrial applications.

When designing PCI applications, engineers have the responsibility of developing the application as well as understanding and developing a PCI interface. In addition to those responsibilities, there are the added restrictions of shorter design cycle times, reducing costs and reducing risks.

A designer has the option to implement the PCI interface using an external "off the shelf" device. However, manufacturers will opt for an integrated solution in order to reduce production costs. They have the option to implement the application in an Application Specific Integrated Circuit (ASIC) or in a Field Programmable Gate Array (FPGA).

For applications that require a short design cycle time, ASICs are often an unrealizable solution. Applications that are implemented in ASICs are considered a high-risk solution. Any minor changes required to the application can take several months before a new ASIC is available for testing. An ASIC can also prove to be a costly solution because of the high NRE fees. These issues, combined with possible changes to the PCI local bus specification, can make it difficult to deliver a product that is PCI compliant and has a short time-to-market. In order to meet all the above requirements, implementing a PCI interface in an FPGA is the best option.

Some FPGA vendors provide pre-defined PCI interface cores. These cores are delivered as a mix of source code and technology-dependent netlists. These pre-defined cores can be customized as per the requirements of the design and still have predictable timing.

What are the benefits of using a pre-defined PCI core instead of developing one from scratch? It is hard to assess the value of a pre-defined core without understanding the design process that is used to develop one.

2.0 PCI FPGA Challenges

The PCI Local Bus Specification outlines the electrical, timing and protocol guidelines in order to be 100% PCI compliant. Implementing PCI, without using a pre-defined core can prove to be a very difficult task. When designing from scratch, a designer must understand in-depth the limitations imposed by both the PCI Local Bus Specification and the architecture of the FPGA. The challenge is trying to design logic that implements the PCI protocol, implements the application and meets timing in an electrically compliant device. Using a pre-defined core will greatly simplify this challenge but some understanding of the requirements as described in this paper, will help toward successful implementation of a PCI compliant design.

2.1 The Component Electrical and Timing Specifications

This section will explore the component electrical and timing specifications. It will give the designer an overview of requirements presented by the PCI Special Interest Group. Please refer to

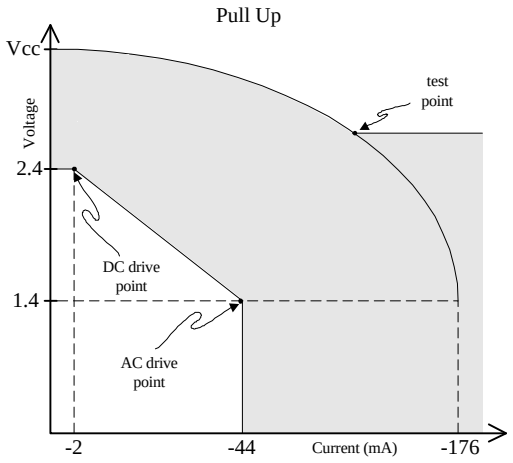


Figure 1. Pull Up V/I Curve for 5 V Signaling Environment

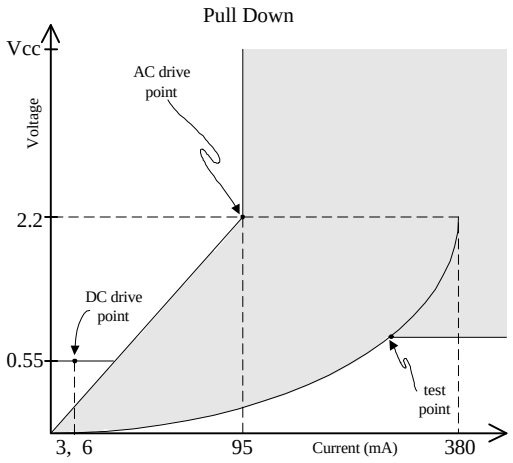


Figure 2. Pull Down V/I Curve for 5 V Signaling Environment

Chapter 4,"Electrical Specification", of the PCI Local Bus Specification Rev. 2.1 for a more detailed explanation.

2.1.1 Electrical Specification

The PCI specification outlines a set of electrical requirements that determine PCI compliance. These include specifications for signaling environment and I/O buffers. Before a designer can consider developing a PCI interface, an appropriate FPGA device must be chosen based on these specifications.

The PCI bus can operate in two signaling environments, 5 V and 3.3 V. Today, the most commonly found signaling environment is 5 V, but the 3.3 V signaling environment is gaining popularity and can be found in some embedded systems.

It is important to determine what signaling environment the device will be operating in. For 3.3 V systems, only 3.3 V devices may be used. However for a 5 V signaling environment, it is possible to use a 5 V or a 3.3 V[1] device.

One of the advantages of the PCI bus interface is that it is a low power interface. The PCI bus is synchronous and takes advantage of reflective-wave switching. Reflective wave switching allows drivers to raise the voltage on a signal only half way. The signal propagates down the bus, reflects and raises the voltage to the required level. By the PCI specification, the signal has 10 ns to reflect and propagate back. It is important that the output buffers of the device be able to switch the bus in the required time with one single reflection.

The specification outlines a set of V/I curves that define the operating regions of the PCI output buffers. There are two curves, pull up and pull down, for each signaling environment, 5 V and 3.3 V. Figure 1 and Figure 2 show the pull up and pull down curves for a 5 V signaling environment. On each curve there are three points of interest, the DC drive point, the AC drive point, and the test point. It is necessary that the output buffers of the device are able to drive the signal within the shaded region in order to switch the bus with a single reflection. A similar set of curves are available for a 3.3 V signaling environment. Please refer to Chapter 4 of the PCI Local Bus Specification Rev. 2.1 for further information.

In addition to the requirements on the component output buffers, the input buffers must use clamp diodes. For both signaling environments, buffers must be clamped to ground. However, for a 3.3 V device, buffers must also be clamped to 5 V or 3.3 V depending on the signaling environment.

The PCI Local bus specification also outlines a set of DC specifications for the I/O buffers. These DC specifications depend on the signaling environment.

The above AC and DC limitations on the I/O buffers, combined with board and pin

[1] 3.3 V devices can be used in a 5 V signaling environment provided they have I/O buffers that are 5V tolerant. Please refer to Chapter 4 of the PCI Local Bus Specification 2.1 for more details

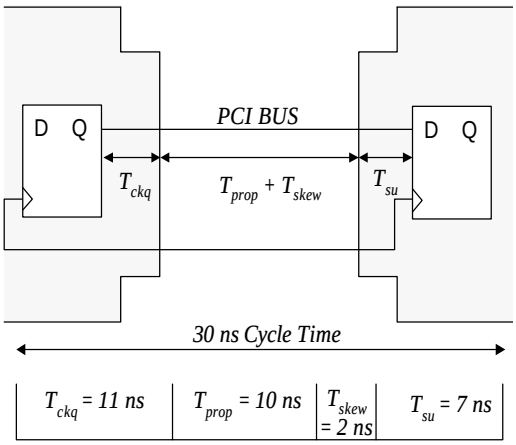


Figure 3. 33 MHz Timing Specification

specifications, will determine which FPGA vendor has devices that are PCI compliant.

2.1.2 Timing Specification

In addition to being electrically compliant, an FPGA must also meet all the timing specification for the PCI protocol. These timing specifications are measured from package pin to package pin. The PCI specification allows for PCI components to operate at any frequency up to 33 MHz (66 MHz is covered later in this paper). The PCI clock may run at any frequency from 0 to 33 MHz and may be changed at any time. Therefore, the minimum allowable clock cycle time is 30 ns. During a 30 ns clock cycle, the specification allows 10 ns for wave propagation. In addition to this, the specification also allows for 2 ns of clock skew from one PCI component to another PCI component. This means that 40% of the cycle time is lost to signal and clock distribution.

Figure 3 illustrates the elements of the 33 MHz timing specification. The remaining 18 ns are divided into two specifications, clock-to-out and setup. The value T_{ckq} represents the clock-to-out timing specification of 11 ns. Valid data must be available on the output pad of the FPGA device at a maximum of 11 ns after it has been clocked. The value T_{su} represents the setup timing specification of 7 ns. Valid data must be present at least 7 ns before the clock edge.

There are several additional timing specifications not illustrated in Figure 3. For example, the PCI specification requires the components to have a hold time, t_h, of 0 ns. The device must not require the data to remain valid for any period of time after a qualifying clock edge. In addition to this, outputs have a maximum active-to-float delay, t_{off}, of 28 ns and a minimum float-to-active delay, t_{on},

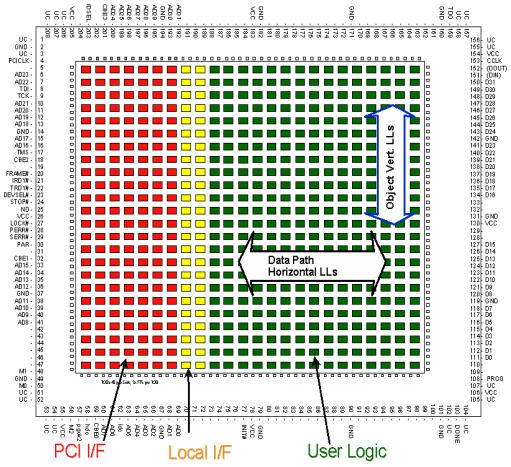


Figure 4. Recommended Layout

of 2 ns. When a signal is tristated, the device must disable the I/O buffer within 28 ns. When a signal is enabled, the device must not enable the I/O buffer for a minimum of 2 ns from a qualifying clock edge. The specification also outlines further limitations on the reset timing of the component[2].

How do these timing specifications affect the design of the application? Each specification restricts how the design is logically and physically implemented and structured.

3.0 PCI Interface and Application Design

Before designing the PCI interface and the application, it is important to determine the physical structure of the design. We have found that dividing the design into blocks can help the designer isolate timing issues. This in turn shortens the design process by reducing the number of iterations of the design cycle.

Figure 4 illustrates a recommended layout for a PCI design in an FPGA. Here the design is divided into the PCI interface, the Application Logic and the Local Interface block between the two. This allows the designer to isolate the PCI-interface-related issues from the application-design-related issues. This layout also takes advantages of the FPGA technology by utilizing such features as long lines and dedicated carry chains.

[2] These parameters are not discussed in this paper. Refer to Section 4.3.2 of the PCI Local Bus Specification Rev. 2.1 for further clarification.

After determining the structure of the design, it is important to consider the timing requirements for each part of the design. The following sections will discuss how to meet timing in the PCI interface and the application design. It will give an overview of which PCI timing specifications are the most challenging to meet and what design techniques can be utilized to meet each requirement.

3.1 Timing in the PCI Interface

Once a designer has chosen a device, which is electrically compliant with PCI, it is important to understand the implications of the PCI timing specification. Implementation of the logic, whether it is entered in HDL or schematic, is greatly influenced by the timing parameters. The designer needs to minimize the number of logic levels on the critical paths and utilize techniques such as pipelining to ensure the design runs at the required frequency.

The most critical timing parameters for the PCI interface in an FPGA are the setup, hold and clock-to-out timing. These timing requirements will constrain the allowable logic levels, and therefore must be understood before implementation can begin. In the following sections, each timing specification is explained in greater detail and appropriate design techniques to meet these specifications are suggested.

3.1.1 Setup Timing

Setup time is limited to 7 ns, specified with respect to the input clock. However, in an FPGA, an input clock signal is delayed before it arrives at a flip-flop, and therefore the clock signal delay can be subtracted from the delay a data signal requires to reach the flip-flop, thus making it easier to meet the setup time requirement at the flip-flop. Subtracting the clock delay from the data delay is equivalent to adding the clock delay to the PCI setup time requirement.

In order to determine how much extra time can be added to the 7 ns PCI setup specification, the minimum clock delay any flip-flop in the FPGA must be determined. This will give the designer the exact timing specification required for the PCI input signals to propagate from the input pad through some combinational logic and setup for the flip-flop. Paths that are immediately clocked at the input buffer will most likely be able to meet the setup timing requirements. These paths do not have any combinational logic delay. Only an FPGA where setup timing for these paths is significantly less than 7 ns should be chosen. The designer needs to be most concerned with paths that require additional combinational logic

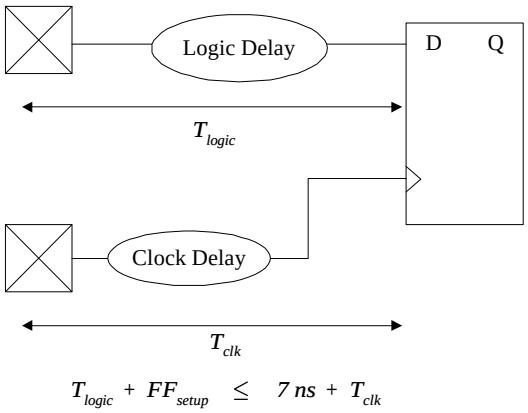


Figure 5. Setup Timing

before they arrive at an internal (outside the input pad) flip-flop.

Consider an example with the parity circuitry. In a PCI implementation, the PCI protocol requires that the PCI agent drive the parity error signal, PERR#, two clock cycles after the address/data and command/byte enable buses are valid and one clock cycle after the parity signal, PAR, is valid. The address/data bus, AD[31:0], and the command/byte enable bus, C/BE#[3:0], can be immediately clocked at the pad in this particular instance and have no difficulty meeting setup timing. However, in the following clock, the PAR signal needs to propagate through some combinational logic in order to determine if there was a parity error. This all needs to be done in 7 ns plus the minimum clock delay before it can be clocked out as the PERR# signal.

Floorplanning is the first technique that a designer can use to help meet timing. Placing the combinational logic near the input pin will reduce routing delay. Compacting the combinational logic into as few combinational blocks as possible and placing them next to each other will also reduce routing delay. The delay through combinational blocks can not be avoided. This delay is a predefined value given by most FPGA vendors. However, a majority of the delay occurs in the routing and that is where the designer has the ability to reduce the delay and meet timing.

Pin locking can also be a technique to reduce the setup time. It is beneficial to place pins that need to propagate through the same combinational logic close to each other in order to avoid routing delay.

In addition to floorplanning and pin locking, critical path placement and routing can be used to guarantee that the critical paths will always meet timing regardless of the design. Critical paths are different for different designs. Critical path placement and routing are usually determined once a designer has successfully routed a design. The design is then stripped of all non-critical path routing. The critical path placement and routing information is saved in a file. This file is then used to convey the locked placement and routing of the critical paths in the next iteration of the design. However, when using this technique, a designer must make sure that the critical logic does not change from one design iteration to the next. In order to guarantee that a place and route tool will use the locked placement and routing information, the designer must guarantee that the logic and the design hierarchy remain the same.

3.1.2 Hold Timing

The PCI specification requires 0 ns hold timing. This is applicable to both 33 MHz and 66 MHz. This specification is relative to the input clock. As with the setup timing, we need to consider the added delay to the input clock before it arrives at any flip-flop. Unfortunately, this added delay increases the difficulty in meeting a 0 ns hold time specification.

Figure 6 illustrates the restrictions for hold timing. Here the minimum data delay plus the flip-flop hold time requirement must exceed the maximum clock delay to guarantee 0 ns hold time.

We can see from Figure 5 and Figure 6 that setup and hold time create a race condition between the data and the clock, and setup and hold requirements can often conflict with each other. In order to meet both specifications, the internal clock delay must be large enough to help the slowest input paths meet timing and must be small enough to ensure the fastest input paths still don't beat the clock. It is important to choose a vendor that can guarantee a very fast clock network. This will ensure that the design will always meet the 0 ns hold time specification, while the above mentioned design techniques can be used to meet the setup timing specification.

Furthermore, if a designer is having difficulty meeting hold time requirements on some of the critical paths, one method to meet the specification is to introduce delay in those paths. Most likely, these paths have no combinational logic for data to propagate through before arriving at the flip-flop. These flip-flops can be placed away from the input pad so as to introduce some

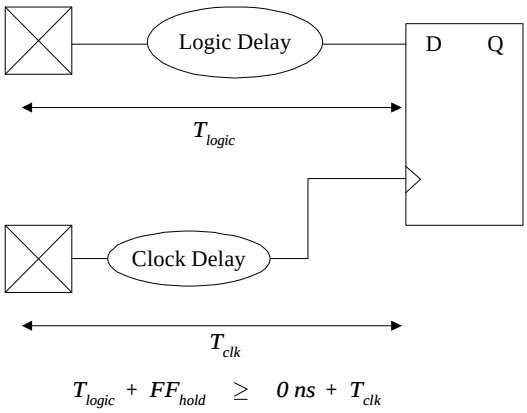


Figure 6. Hold Timing

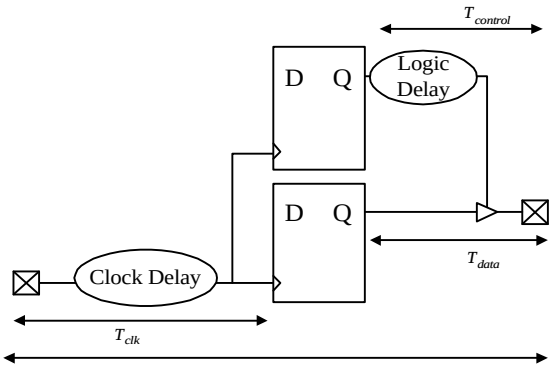
routing delay. This allows combinational blocks that are directly adjacent to the pads to be utilized for those paths that have difficulty meeting the setup timing specification. Using this design methodology along with critical path constraints can guarantee that the designer will consistently meet the setup and hold time requirements.

3.1.3 Clock-to-Out Timing

Clock-to-out timing is limited to 11 ns. As opposed to the minimum clock delay that was used in the setup and hold time, clock-to-out uses the maximum clock delay.

Figure 7 illustrates the clock-to-out timing specification. The figure demonstrates that not only does the data have to meet this timing specification but so does the control logic for any of the tristate buffers. This specification is relative to the input clock and must include the clock delay introduced by the clock network. Therefore, the clock-to-out timing is constrained by not only the data and control delay but also the maximum clock delay. Again, choosing an FPGA vendor that has a fast clock network can help the design meet this timing specification.

A majority of the delay that is found in a data or control path is the routing delay. Placing the flip-flops near the output pad can help to reduce this routing delay. Pipelining can also be used to help meet the clock-to-out timing specification, but it is important to note that pipelining can increase the clock delay because it loads the clock network with additional flip-flops. However, this additional delay to the clock network should not be greater than the logic delay that was eliminated from the clock-to-out path.



$$max(T_{clk}) + T_{data} = T_{ckq}$$

$$max(T_{clk}) + T_{control} = T_{ckq}$$

$$T_{ckq} \leq 11\ ns$$

Figure 7. Clock–to-Out Timing

Using these techniques in the PCI bus interface may not be enough to guarantee that the design meets timing. Additional consideration should be given to the application design and the local interface to that application design. The designer has to guarantee that the application can also operate at any frequency up to 33 MHz.

3.2 Interfacing Between the PCI Interface and the Application Design

Before designing the application, the specifics of how the PCI interface will deliver and receive data to and from the application design need to be determined. The designer needs to determine the primary types of transactions that the application will be performing on the PCI bus. Whether the transactions are of short burst length or long burst lengths is also important.

A well-designed local interface is one that delivers signals to the application design that can be used to quickly deliver data. Applying bus signals through combinational logic gates directly to the application is impractical and will violate the timing specifications of the PCI bus. Bus signals are heavily loaded and should only be used in the PCI interface to determine the local signals to the application. These local signals should be pipelined in the PCI interface and the application design in order to isolate the timing of the PCI interface from the application design.

There is a certain set of signals that must be handled only in the PCI interface. These signals need to respond immediately to what is presented on the PCI bus. These include but are not limited to DEVSEL#, PAR, PERR#, SERR# signals. In order to meet timing on these signals, the logic to drive these signals should be included in the PCI interface. Signals from the application design through the local interface can not be used. These signals actually require the use of other bus signals from the PCI bus in order to respond appropriately. For these signals, the techniques mentioned above will need to be utilized.

For other instances such as target termination sequences and initiating a transaction, the bus signals do not need to be used. The PCI interface can be designed to input and output intermediate signals through the local interface to and from the application design. These intermediate signals will be pipelined and will drive the appropriate values onto the bus, but without the difficulty of meeting timing. The trade-off to this is that the transaction could suffer a one to two cycle latency from when the application wants to start or stop a transfer to when it actually stops or starts a transfer on the PCI bus. However, it can easily be accommodated in the application design by allowing the data to be "backed up" or registered. This will avoid data inconsistency. In addition, for those applications that transfer large burst lengths, this latency is minimal and does not affect the overall bandwidth. Isolating the bus signals from the PCI bus in the PCI interface will also assist in meeting timing in the application design.

3.3 Designing a PCI Application

Once the PCI interface has been developed, the designer needs to consider the restrictions that are going to be imposed on the application design. How is the application design going to handle the data coming from the PCI interface and how will the application design deliver data to the PCI interface? This is where the designer needs to determine the capabilities of the application design and how it will drive the behavior of the PCI interface on the bus.

The paths that have the most difficulty meeting timing are those that must send data between the PCI interface and the application design. If the application is intending to send or receive bursts of data, these paths will need to be heavily constrained and the logic will need to be placed. If, however, the application intends to send or receive data as single transfers, these critical paths can be pipelined and can possibly avoid placement and timing constraints.

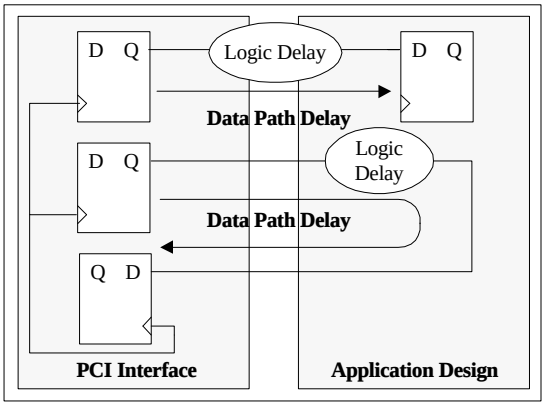


Figure 8. FFs to FFs Paths

Assuming that the application needs to be able to send and receive data in large burst sizes, we need to constraint the critical paths that traverse from one part of the design to the other. The following sections will discuss some techniques that can be utilized to meet timing on such paths.

3.3.1 Route Budget Analysis

One of the most critical paths in an application design is one that originates in the PCI interlace, propagates through combinational logic in the application design and outputs back to the PCI interface to be clocked as illustrated in Figure 8. This is a flip-flop to flip-flop timing specification. Depending on the operating frequency, this can be as little as 30 ns.

How can a designer ensure that this path will meet timing? The first step is to analyze the design and determine the maximum allowable delay for any critical path. This is called route budget analysis. The recommended design flow to perform route budget analysis is to implement the design at least once without any application design constraints and then survey the static timing analysis report to determine how much time has been allotted to these critical paths and by what margin the path is failing.

Based on the results of this analysis, the designer can then use pipelining, timing constraints or placement constraints to make the design meet timing. For example, a critical path can have a long routing delay and/or a large logic delay. If routing is where a majority of the delay is occurring, then the designer should consider using placement constraints on the application design. Placement constraints will help to reduce the routing delay by placing the critical logic adjacent to the output flip-flops of the PCI interface. If logic delay is where a majority of the delay is occurring, then the designer should consider using pipelining. Pipelining will reduce the levels of logic between flip-flops for a critical signal.

Figure 8 also illustrates a flip to flop path that originates in the PCI interface propagates through some combinational logic in the PCI interface and/or the application design and terminates at a flip-flop in the application design. Route budget analysis can also be applied to this path. If it is difficult to change the placement and the levels of logic in the PCI interface consider reducing the levels of logic and decreasing the routing delay using placement constraints and pipelining in the application design.

3.3.2 Using the Advantages of FPGAs and PCI

Use the resources that are provided by the FPGA technology. Floorplan the design such that data flows horizontally so as to make use of the available long lines. When possible, floorplan the design such that the data paths are aligned with the data registers of the PCI interface. Using long lines will introduce less routing delay and can help to improve timing. In addition to this, floorplan structured data objects such as counters, registers and adders so that they can make use of the vertical long lines and carry chain capabilities. This can assist in eliminating routing/logic delay in the application design. This is illustrated in Figure 4.

The application design will also have several data paths that interface to other (non PCI) I/O pads, and these too will have timing specifications that are determined by the application. If possible, use pipelining along the long lines to reduce the load on the data paths.

In addition to this, knowledge of the PCI protocol can help the designer increase the performance of the design. For example, when writing data, do not attempt to request the PCI bus until the data is ready to be delivered, and when another agent requests to read data, retry the transaction until the data is ready to be transmitted. Use delayed transactions. Knowledge of the protocol will allow the designer to setup the PCI interface signals appropriately and realistically.

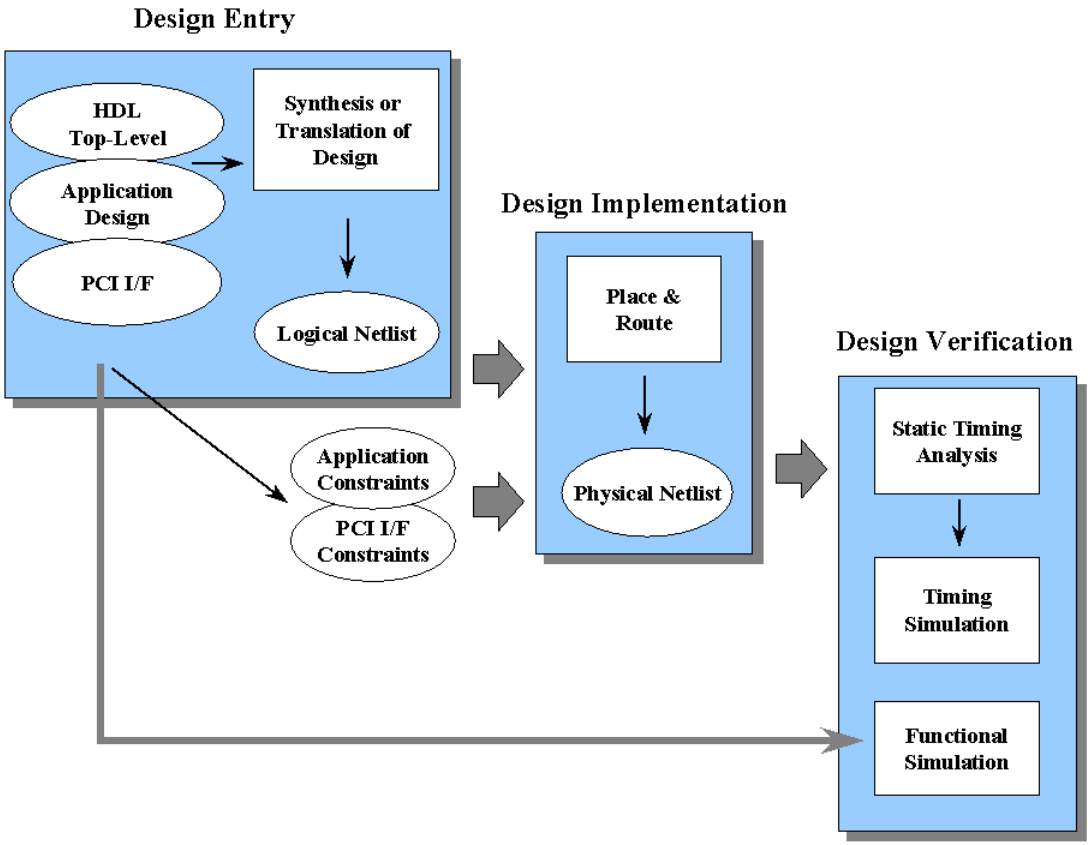


Figure 9. Design Methodology for FPGA based designs

3.4 Defining a Design Methodology

Once a designer has in-depth knowledge and understanding of the PCI timing specification and protocol, it is important to develop a design methodology that will ensure that the implemented design complies with PCI requirements. A well-defined design methodology will also assist in shortening a design cycle. In any design process, the good design methodology needs to fulfill three requirements. It must:

- Address technology/software related issues.
- Offer the designer flexibility to accommodate design variations and changes.
- Provide the designer predictability by reducing the technical issues in subsequent stages of the design cycle.

Successfully implementing PCI in an FPGA requires the designer to follow this methodology in order to develop a product that will fulfill the project objectives and have a short time to market.

In every design cycle there are three fundamental stages: design entry, design implementation, and design verification. Figure 9 illustrates the recommended design methodology for FPGA based designs. Each stage is composed of various design tasks. A good design methodology will assist the designer in overcoming the challenges discussed in Sections 3.1 through 3.3.

3.4.1 Design Entry

The first stage of a design cycle is design entry. In this stage designer develops a design that can then be synthesized or translated. This initial stage is the most critical to the design cycle because a majority of the design work focuses on reducing the technical issues that would arise later in the design cycle.

When developing a PCI design in an FPGA, the designer would develop the design using the techniques mentioned in Sections 3.1 through 3.3. The design should be pipelined and the PCI

interface and application should be isolated so as to facilitate implementation. The placement and timing constraints for the design should be determined as well. These initial timing and placement constraints can be based on the FPGA technology, desired performance, desired structure and route budget analysis. In order to perform route budget analysis, it is recommended to execute the design flow once without any application constraints. An estimate of the required application constraints can be determined after executing static timing analysis on this first iteration of the design flow. These PCI interface and application constraints will then facilitate the implementation tools to guide the design even closer to the desired performance parameters.

In addition to the development of the design and determination of the constraints, the designer should also perform functional simulation before continuing on to later stages. Although functional simulation is traditionally considered to be a part of the design verification stage, it is usually performed in this step to avoid unnecessary iterations of the design cycle. Functional verification will be discussed further in Section 3.4.3.

Once a design has been synthesized or translated, a logical netlist is produced. This logical netlist, along with the placement and timing constraints, will be used in the design implementation stage.

3.4.2 Design Implementation

The next stage in the design cycle is the design implementation stage. In this stage, the FPGA tools will translate the logical netlist into a physical netlist. During this stage, the design constraints will be applied to the design and the tools will determine if the design can meet timing or not.

After performing this step, it may be necessary to edit the design constraints or add additional constraints in order to make the design meet timing. Many FPGA vendors will have a design editor tool that will allow the designer to view the routed design. Using this tool, the designer can see if any additional floorplanning is needed. This tool may also be used to determine specific nets that may not be meeting timing specifications, so that a tighter constraint can then be applied those nets.

The tool can also be used to create files that lock critical path placement and routing. Some FPGA vendors allow the use of these files to facilitate the routing of a design. If a particular placement of a design produces a physical netlist that meets timing, this netlist contains information about the routing and placement of critical paths. The netlist can be stripped of non-critical logic, placement and routing. The implementation tools can then use the remaining file to route the critical paths exactly as they are routed in the file. This will guarantee that the design will always meet timing in the critical parts of the design. Often, this critical path placement and routing file is created from a physical netlist that does not meet all of the constraints. Using a placement and routing file can sometimes help the tool focus on routing the parts of the design that do not meet the constraints.

If design constraints can not be met, it is useful to perform static timing analysis on the design to determine where in the design the constraints are not being met. This will require the engineer to go back and rethink the design. This may include redesigning parts of the logic in order to facilitate the tools to map the design in a fewer levels of logic. However, if the design was originally developed with the capabilities of the FPGA technology in mind, this step is usually not necessary.

3.4.3 Design Verification

The final stage in the design flow is design verification. Design verification consists of two steps: functional verification and timing verification.

3.4.3.1 Functional Verification

In order to pass functional verification on a PCI design, the PCI design must be PCI complaint. To test for PCI compliance, the PCI Special Interest Group (SIG) Test Scenarios for Compliance Testing will need to be implemented. These are a set of test scenarios that test the very basic transactions between two agents on a PCI bus, one agent being the device under test (DUT) and the other being a behavioral model of a PCI Initiator. These tests verify if the DUT is PCI compliant or not. However, it is important to realize that these test scenarios do not cover every possible bus condition.

The designer has the option to purchase a testbench from a third party vendor that will allow him/her to set up conditions on the bus to test the DUT. However, learning these testbenches can prove to be just as difficult as developing one internally.

If the designer chooses to develop a testbench internally, there are two types of testing that must be performed in order to verify if the design is

functional: system level testing and PCI bus protocol testing. System level testing sets up modules in a system to transfer data back and forth. This type of testing checks to see if the data was actually sent and whether the data arrived at the destination or not. It also checks the validity of the data. However, whether the modules transferred the data using the correct operating procedure or not, is not checked system level testing.

PCI bus protocol testing is used to determine if the modules in a system operated within the rules of the protocol. Besides verifying the data, protocol testing verifies that the agents are PCI compliant. PCI protocol tests should include the very basic functional testing that is outlined in the PCI test scenarios. However, these test scenarios do not test every bus situation. Beyond them, there are several other conditions that have a high probability of occurring and are recommended for implementation.

Some of the recommended exercises that should be implemented are:

- Target termination sequences: sequences where the target issues a termination when the master inserts wait states before, during and after the termination.

- Parity checking: Sequences where the incorrect address and data parity are generated to check the ability of the DUT to report errors.

- Protocol checker: checks to see if all the operating rules are obeyed.

Developing an extensive testbench will help the designer in determining the correctness of the design before continuing on to the implementation stage of the design cycle. After implementation, the testbench can then be used for back-annotated timing simulation.

3.4.3.2 Timing Verification

Timing verification occurs once the design has been implemented. There are two stages to timing verification: static timing analysis and back annotated timing simulation.

Static timing analysis is used to determine if the design meets all the PCI timing specifications such as setup/hold time and clock-to-out timing. Static timing analyzers are provided by the FPGA vendor and will determine if 100% of the design met the timing specifications. The designer has the responsibility of specifying these timing parameters to the tool and determining what parts of the design should be attached to the each parameter. If the design does not meet timing, the static timing analyzer can be used to probe the design and determine where the design is failing. Based on this investigation, the designer has the option to go back and re-implement the design using a stricter set of timing specifications or redesign parts of the design to contain fewer levels of logic.

Once the design passes the static timing analysis, it is beneficial to run the physical netlist with the timing information through the functional testbench. This will allow the designer to determine if, in fact, the design meets timing using the actual physical delays and not the unit delays that are applied during functional simulation. Once the design has been verified it is ready to be downloaded to a part and used in a physical system.

4.0 Additional 64-bit/66 MHz PCI FPGA Challenges

What if a designer wants to implement 64-bit/66 MHz PCI in an FPGA? Are there any additional challenges? PCI 64-bit/66 MHz does introduce a new set of timing and electrical specifications that the designer must consider.

The PCI specification allows for four types of systems: 32-bit/33 MHz, 64-bit/33 MHz, and 32-bit/66 MHz and 64-bit/66 MHz systems. The difficulty of implementing these different systems in a FPGA is greater when designing a 66 MHz system. Protocol handshaking requires some additional pins beyond the 64-address/data pins of a 64-bit system. These additional pins are 4 additional C/BE# lines, a PAR64, a REQ64# and an ACK64#. With the increased width of the of address/data path, 64-bit systems are harder to implement in FPGAs. However, implementing a 64-bit system requires the same techniques mentioned in Section 3.1 to meet timing requirements.

What the designer needs to consider is the logic implementation of a 64-bit system. Since 64-bit systems operate in the same environment as 32-bit systems, there will be transfers that are not always 64-bits wide. For these situations the logic needs to be able to multiplex the data to avoid data inconsistency. The system also needs to drive the control signals based on whether a 64-bit transfer is requested or transmitted. The extra logic for accommodating 32 and 64-bit systems increases the difficulty of meeting setup and clock-to-out timing. This is the most significant challenge of doing a 64-bit system.

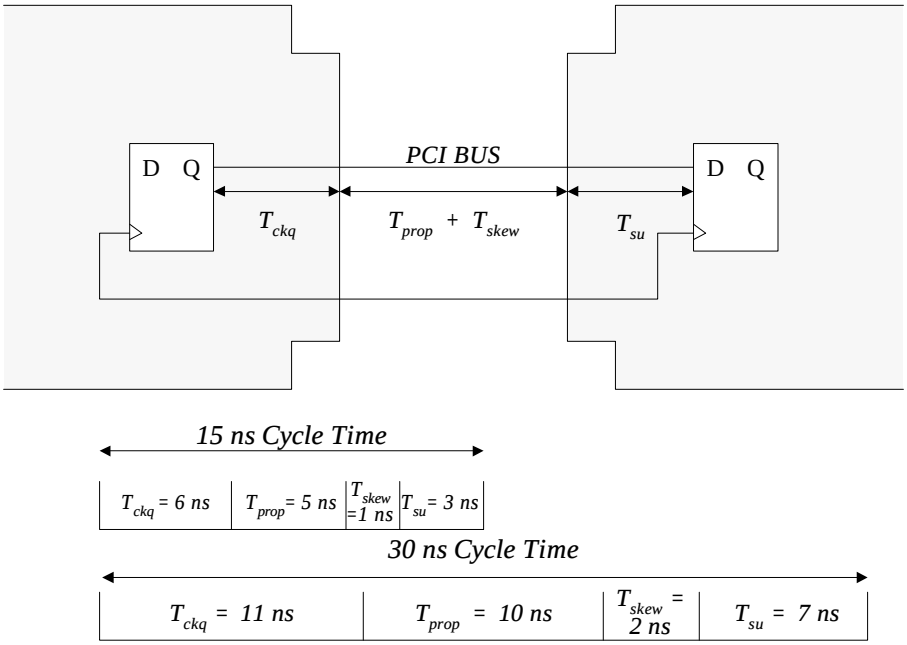


Figure 10. 66 MHz Timing vs. 33 MHz Timing

The PCI 66 MHz specification is electrically compatible to the 33 MHz system. However, the 66 MHz system can only operate in a 3.3 V signaling environment. The 66 MHz specification has the same AC and DC characteristics as the 33 MHz 3.3 V signaling environment. The 66 MHz requires the addition of one pin, M66EN that is designated as a ground pin in the 33 MHz system. Thus any FPGA device that is 33 MHz and 3.3 V electrically compliant is also 66 MHz compliant.

The timing specifications for a 66 MHz system are significantly more difficult to meet. Figure 10 illustrates the differences between 33 MHz timing and 66 MHz timing. The clock-to-out timing specification is reduced to 6 ns. The wave propagation and clock distribution only have 6 ns as well, and the setup timing is reduced to 3 ns, less than half of the setup timing for 33 MHz. However, the hold time specification still remains at 0 ns.

Meeting these timing specifications in an FPGA is very difficult. There are very few devices, if any, that can support these performance requirements. However, as technology advances and 66 MHz systems gain popularity, more and more FPGA vendors will provide a solution that will fulfill such requirements.

5.0 Being Successful Using FPGAs in PCI

Successfully implementing a PCI design in an FPGA is much easier when the designer uses a pre-defined PCI interface core. Pre-defined cores eliminate the need for in-depth knowledge of the technology and the PCI protocol specification. The PCI protocol specification outlines strict timing and electrical specifications that are difficult to implement. In a pre-defined core, the design has already been constrained to meet these specifications. As a result, a designer doesn't even have to be familiar with the FPGA architecture and tools in order complete a successful implementation. If a pre-defined core is not used the design of the PCI interface itself can take several months. For most projects, the required engineering effort is prohibitive. Many of the advantages of using an FPGA for a PCI application design, such as short time to market, are lost in the process of learning the protocol and meeting the strict electrical and timing requirements.

Due to the difficulty and increased complexity of PCI 64-bit/66 MHz, some FPGA vendors will also provide a PCI 64-bit core solution.

To always meet the PCI specification for 33 MHz or 66 MHz, the pre-defined core solution should

be in the form of a heavily constrained black box module that is configured to match the design requirements. Using pre-defined cores allows the designer to focus on other aspects of the application design.

Furthermore, constrained pre-defined cores have the added advantage of guaranteed timing. These cores are specifically designed for a particular architecture and the constraints that are applied to them guarantee that the designer will meet all of the PCI timing specifications regardless of the type of application design. In addition to this, a good FPGA vendor will only recommend those devices that are electrically compliant with the PCI specification.

The pre-testing by the vendor also eliminates many of the verification tasks. A good FPGA vendor will provide a test environment that will allow the designer to perform functional and timing verification on the design. Pre-defined cores must be supported for various design tools and design flows and often the FPGA vendor will provide the support that is necessary to integrate these cores with the application created by the designer. This support should be in the form of reference design examples, in-house expertise, hotline support, and thorough documentation that outlines the functionality of the pre-defined PCI core and the integration steps necessary to achieve a successful PCI design.

The designer must take care in selecting a core provider that will be able to fulfill all of these needs. PCI is a very complex design and requires the FPGA vendor to have to have a strong support structure to assist the designer in integrating the application with the pre-defined PCI core. Some pre-defined core providers will out-source the pre-defined PCI core design to a third party and will not provide all of the necessary documentation and support. Often, a designer will spend an equivalent amount of time integrating these types of cores, as he/she would have spent designing one. Selection of a pre-defined PCI core provider should be done based on the vendor's complete PCI design solution and past performance record. Using an FPGA for PCI is beneficial only if done correctly.

Acknowledgments

I would like to thank Wilson Yee, Eric Crabill, Per Holmberg, and Edel Young for their feedback and suggestions for this paper.

References

1. J. Case, N. Gupta, J. Mittal, D. Ridgeway, "Design Methodologies for Core-Based FPGA Designs," PCI Plus 1997 Proceedings.

2. Anne-Marie Lessertisseur, "PCI Project Design Overview, "PCI Plus Europe 1997 Proceedings

3. PCI Special Interest Group, "PCI Local Bus Specification", 1995, Rev. 2.1

4. David Ridgeway, "Programmable Logic Implementations of PCI," PCI Plus 1996 Proceedings