Approaching PCI Bandwidth Limits with FPGAs

Jayant Mittal, Xilinx Inc.

Introduction

Embedded systems such as network and communication equipment, printer engines and server I/O controllers need high performance. A poorly performing PCI agent can seriously degrade the overall PCI bus performance. Some of the PCI performance measures include efficiency, bus utilization, latency, wait-states and bus transfer rate. A PCI efficiency of 1 is defined as a transfer which passes 4 bytes of data on every clock cycle during which the bus is occupied. This results in a theoretical maximum of 132 Mbytes/sec for 32-bit PCI with 33 MHz clock. PCI bus utilization provides information on percentage of time the bus is occupied. When a PCI master causes additional clock cycles between FRAME# and IRDY# assertion, this is called initial latency. Latency may also occur between individual double word (Dword) transfers when a target deasserts its ready line, leading to subsequent latency. Bus transfer rate presents how much data is transferred over the bus in Mbytes/s and is limited due to initiator and target wait states.

PCI agents typically achieve a transfer rate of less than 60-70 Mbytes/sec. Since every PCI cycle requires at least one address cycle and reads require additional turn-around cycles, the maximum throughput can not be reached. Minimizing the latencies, using longer bursts, implementing advanced PCI commands, fast back-to-back transfers and following PCI rules are methods to improve PCI bus performance.

We will first investigate some factors that limit the efficiency, followed by advance PCI developments circumventing some of these issues. In the end, we will elaborate on how the Xilinx PCI core addresses some of the key performance issues such as latencies, wait states, longer bursts, PCI timing and compliance, while providing a system-on-chip solution and flexibility.

1. Limiting Factors Affecting PCI Theoretical Maximum

1.1 Address and control cycles

The idle cycles between transactions, turn-around cycle(s) in read transactions and the address cycle are some obvious bandwidth consumers. Idle cycles are defined to be those which do not have either of FRAME#, IRDY# or TRDY# asserted (See Fig. 1). Additional bandwidth is wasted while the target decodes the address before responding (Fig. 2). Likewise, the PCI bus must be tri-stated for least one cycle (turn-around) after writing the address for data reads.

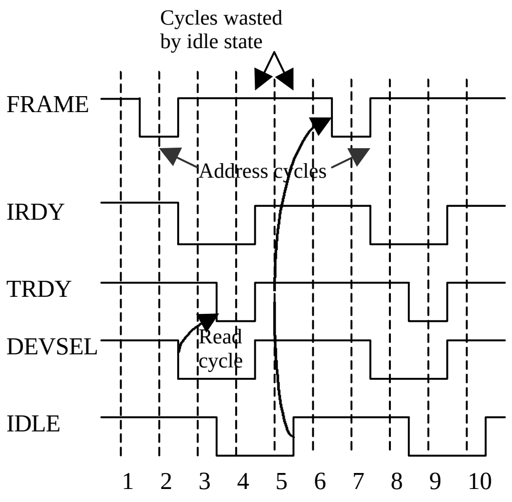


Fig. 1 Idle cycles between two transactions

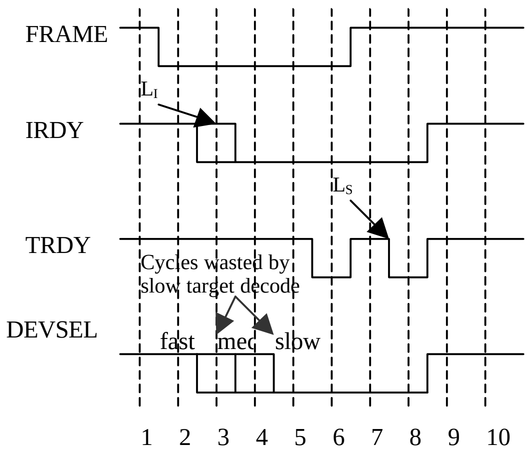


Fig. 2 Delay to first and subsequent data

1.2 Bus congestion

In a system with a large number of agents sharing the PCI bus, the arbiter limits the maximum tenure (burst size) of each agent for fairness. The limited tenure takes away the performance advantage of large bursts. Arbitration latency, which is proportional to bus congestion, results in wasted cycles when the agent waits for GNT# after issuing REQ# prior to starting the cycle by asserting FRAME#. Arbiter design is very important to reduce arbitration latency. In case there is no resource conflict, an agent must keep REQ# asserted to conserve bandwidth so that the arbiter continues issuing GNT# and long bursts are feasible. An agent receiving GNT# despite non-asserted REQ# (bus parking) must drive the bus to a known state to keep the bus at valid logic levels.

1.3 Limited PCI command Usage

Some of the more popular PCI agents implement only MR and MW (Memory Read/Write). This often limits them to short burst lengths, creating poor performance. When data is read in small blocks as may be the case with the MR command, the chipset may either need to insert wait states between successive Dwords transferred or be unable to sustain longer bursts. The MR command should be used for reading data that fits within one cache line. On Pentium Processors (Pro), a cache line is 32 bytes (8 Dwords). Excessive usage of I/O commands degrades performance by increasing CPU utilization and forcing initiator wait states. Usually, I/O commands are single data phase transactions only.

1.4 Incompatible bus width of target and initiator

When the target is shorter than the Dword width (1, 2 or 3 bytes), while initiator is 32-bit, the initiator wastes some PCI cycles in transferring less than one Dword of data in each PCI clock by selectively enabling CBE lines and hence performs sub-optimally. In 64-bit PCI, if the target cannot perform a 64-bit data transfer to the addressed location, the master is required to complete the transaction as a 32-bit master and not as a 64-bit master, losing the bandwidth advantage expected with 64-bit PCI.

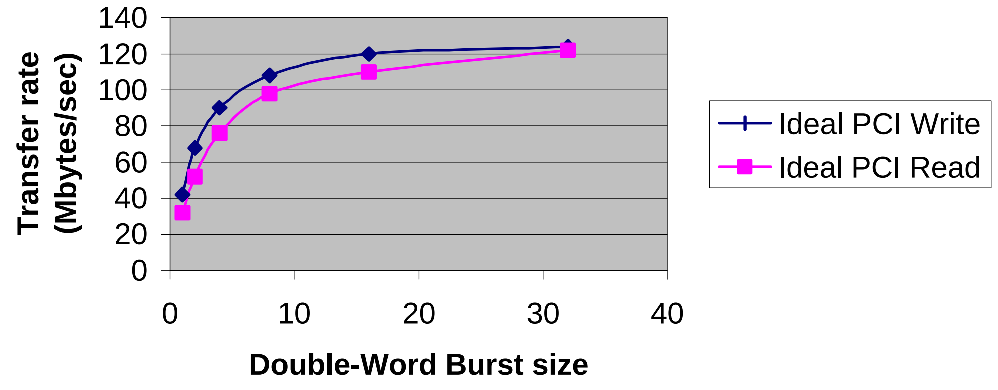
1.5 Short burst sizes

Each time an agent initiates a transfer of X Dwords, it must wait for L_I clock cycles for the transfer to begin. Once the transfer begins, it can occur at the rate of one Dword per PCI clock cycle in case of an "always ready" target. For a slow peripheral which introduces subsequent latency (L_S),

$$Efficiency = X / (X + L_I + L_S) \qquad (1)$$

A PCI agent cannot completely control L_I, L_S. If it does not maximize the burst size X, the efficiency is low. Fig. 2 shows some of the causes for delay in initial and subsequent data phases represented by L_I and L_S. Fig. 3 assumes no L_S and shows that to achieve maximum transfer rate on a PCI bus, write burst size should be at least 16 Dwords while read burst size should be at least 32 Dwords.

Fig. 3 Effect of burst size on PCI bandwidth



1.6 Initiator/ target wait states

Both initiator and target wait states lead to an increase in either L_I or L_S, as per equation (1), reducing PCI efficiency. Long initial delay (max. allowed 8 clocks) between FRAME# and IRDY# assertion increases L_I. Slower than FAST target address decode affects both read and write cycles and results in initial latency on TRDY#. A slow CPU or peripheral may not be ready to transfer data at each PCI cycle to sustain a burst and may introduce wait states to synchronize. Late target select (FRAME# -> DEVSEL#), late target response (FRAME# -> TRDY#) are other implementation overheads which lead to L_I. These latencies are described in Fig 4.

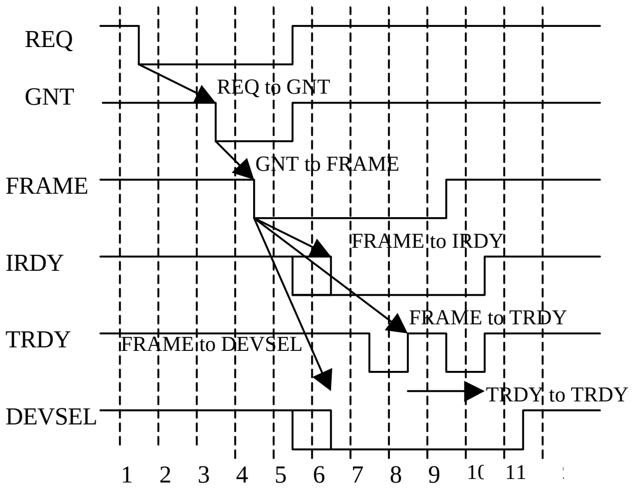


Fig. 4 Various types of latencies in a PCI system

1.7 Frequent target terminations

A slow access (> 12-14 PCI clocks) forces target to *retry* transfer starting with address of data which was next to be transferred. Normally in the time it takes to re-establish data bursts due to re-arbitration, the target should have data available, but this affects performance.

When a target does not *disconnect* when no data is available to transfer and holds the bus with wait states, it introduces latency and PCI efficiency is reduced. It is a good practice to disconnect when a burst can not be maintained. However, the master should automatically re-establish the same transfer immediately to complete burst transfer.

2.0 Recent Developments which Improve Performance

2.1 Split-transaction (Write-Only buses)

The PCI read cycles require a turn-around cycle and are therefore relatively less efficient than write cycles. Furthermore, posted writes are deterministic and easy to handle at full bandwidth using FIFOs. We could make a “request to be written back to” for reads. Using this philosophy, read could be split into two components: a request and a reply. A split-transaction protocol prevents the access time during a read, from being seen to the communication channel. With a *write-only* bus, during reads, an initiator could write a request to the target and get off the bus. When the data is ready, the original target becomes the initiator and then satisfies the request by writing to the original initiator. During writes, the initiator could write a request followed by actual data.

2.2 Fast back-to-back transactions

The bus access latency is dependent upon arbitration and initiator waiting time for the bus to be idle after receiving a bus grant as shown below.

$$\textit{Bus access latency} = \textit{Arbitration time} + \textit{Bus busy} + \textit{Idle cycle(s)} \qquad (2)$$

Fast back-to-back transactions eliminate the idle cycle following a write and thereby reduce initial latency of the next transaction. Either target or initiator could be fast back-to-back capable provided no contention occurs on TRDY# and DEVSEL#. The target must be capable of following a bus transition from final data transfer directly to an address phase while the initiator must perform the next transfer to the previous target and understand address boundaries. PCI specs recommend this feature on all new targets to utilize bandwidth more effectively.

2.3 I2O

Until now, designers have been forced to implement proprietary backplane protocols between a host CPU and intelligent I/O subsystems. This required the designer to define commands, status, mailboxes, doorbells, interrupts, messages, errors and buffers. I2O stands for intelligent I/O and is a well-defined industry standard software message passing protocol, which divides I/O software functionally between host operating system and I/O subsystem with a layered architecture and defines these things. It is useful for high performance PCI systems with one or more intelligent I/O subsystems and high speed I/O devices, and makes efficient PCI bus utilization. I2O defines efficient use of DMA, bus mastering, burst modes and interrupts. I2O also defines various command sets for LAN (WAN), IDE, SCSI, tape storage and floppy classes besides message types for initialization, configuration, private and user defined messages.

2.4 PCI-to-PCI bridges

PCI-to-PCI bridges allow addition of more PCI slots and devices to the system and isolate the bus traffic by isolating the transaction bandwidth between devices on one bus from devices on the other busses (Fig 5b). Further, bridges allow communication between two PCI busses as shown in Fig 5a. Theoretically, by using PCI-to-PCI bridges, 7 secondary PCI busses could be connected to one Primary bus, each consisting of 4 slots (Fig. 6). However, it is not recommended to cascade them because there is huge latency for single cycle transactions while no gain with burst transfers. Further, the clock skew may increase with increasing depth in bus hierarchy. Individual arbiters are required on each bus.

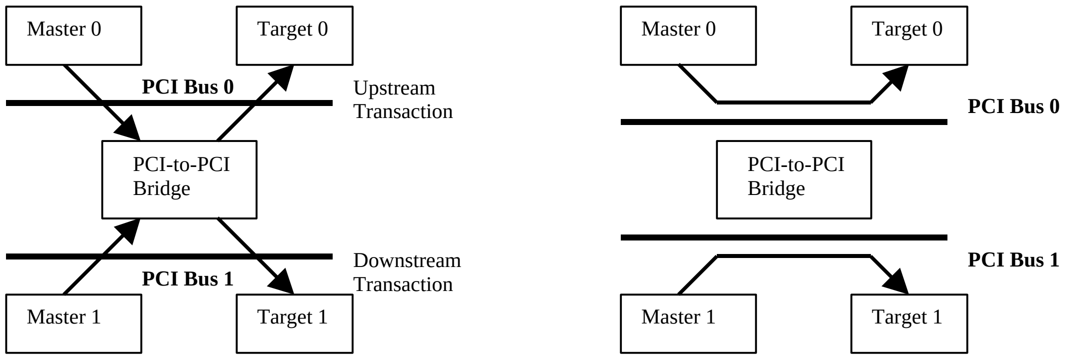


Fig. 5 PCI-to-PCI Bridge allows transaction a) between PCI busses b) concurrent operation

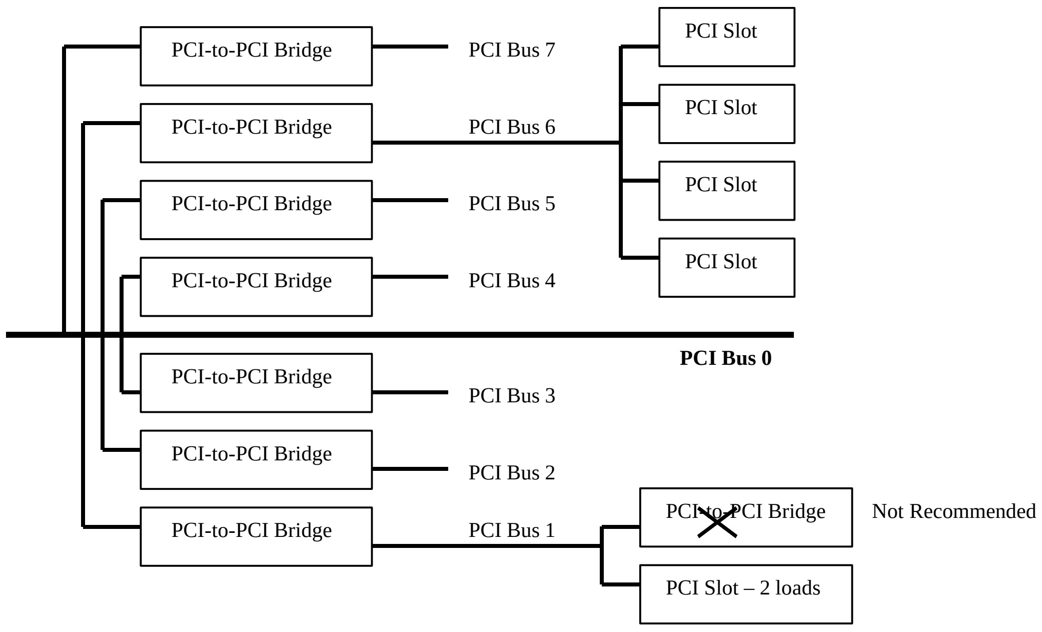


Fig. 6 Slot expansion with PCI-to-PCI bridges

2.5 Extended PCI commands

Using the right advanced command while performing PCI master memory reads may significantly increase performance. Commands control the block size of data read from target. Table 1 shows recommended command usage recognizing the fact that once a single double-word has been read, reading to the end of the cache line is extremely inexpensive.

Use of Memory Write and Invalidate (MWI) is also important for achieving high performance. MWI should be used for transferring data more than one cache line in size which begin and end on cache line boundaries (cache line aligned). A PCI device optimized for MWI should disconnect MW only on a cache line boundary, providing a convenient place for MWI to begin. MWI eliminates system cache snoop overhead and enables performing another cache line even upon losing the bus grant on the last cache line boundary as per PCI specs.

Table 1: Recommended PCI commands usage based on block size

S. No.	**PCI command**	**Read block size (in cache lines)**	**Pre-fetch**
1	Memory Read (MR)	1	No
2	Memory Read Line (MRL)	2	Yes
3	Memory Read Multiple (MRM)	> 2	Yes

2.6 Memory bandwidth improvement

Memory bandwidth is a fixed resource, often less than the system demand, shared among bus masters and arbitrated by a memory controller. Larger caches reduce processor-memory bandwidth requirements and result in more cache-snoop hits on PCI transactions. Using faster memory chips, better, increased and large FIFOs keep a full cache line ahead and handle PCI requestors individually. More memory controllers resolve PCI-CPU contention. When DMA engine, FIFOs and memory controllers are used together, they reduce delay to first data and provide 0-wait state subsequent data.

2.7 64-bit PCI

64-bit PCI provides performance improvement on memory commands. A 64-bit agent could transfer up to a double DWORD per data phase instead of a single Dword available with 32-bit PCI; however, the target must also be 64-bit. A 64-bit bus provides additional data bandwidth for agents that require it, however, wait states take away the advantage, as shown in Fig. 7.

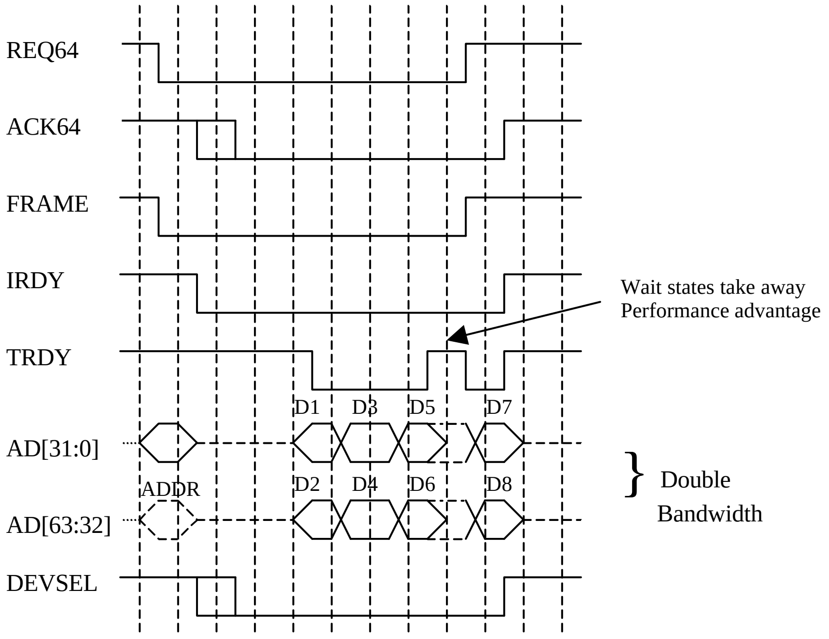


Fig. 7 Performance advantage with 64-bit PCI

3.0 Xilinx PCI Core in High Performance PCI Designs

Combining a custom user-interface with a pre-verified Xilinx 32-bit, 33 MHz PCI core produces a single-chip PCI design. The PCI LogiCORE module enables fast implementation of an FPGA-based prototype application. It is fully timing and PCI protocol compliant with PCI specs 2.1. The pin-out and CLB placement is controlled by mapping, pin-locking and LOC constraints, while routing of most critical paths is controlled by guide files and PCI timing specs. The result is predictable performance and functionality.

3.1 Core Functional Description

The LogiCORE is partitioned into six major blocks as shown in Fig. 8.

3.1.1 PCI I/O Interface

This handles physical connection to the PCI bus including all signaling, I/O synchronization, output three-state controls and request-grant handshaking for the bus master.

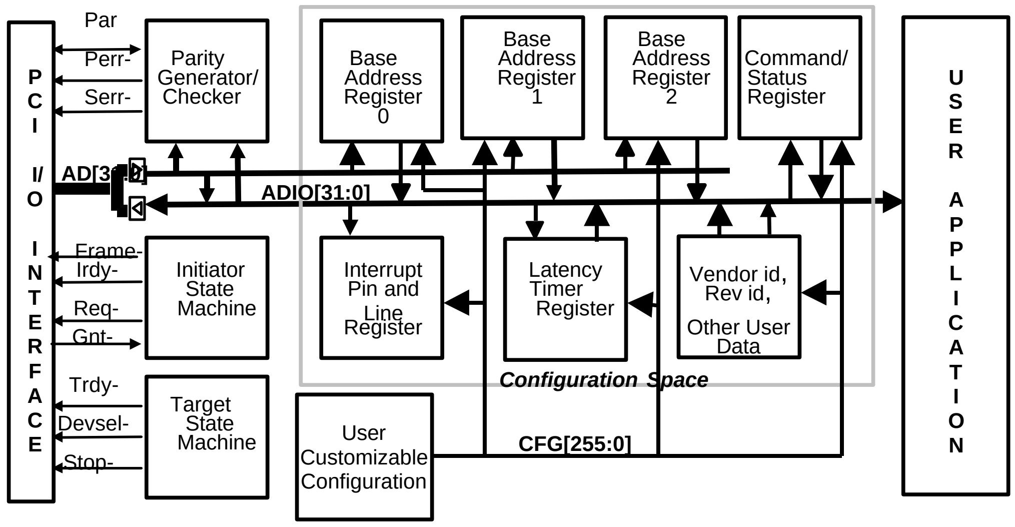


Fig. 8 LogiCORE PCI Interface block diagram

3.1.2 Parity Generator/ Checker

The Parity Generator and Checker block generates and checks even parity across the AD bus, CBE (Command/Byte Enable) lines and PAR. Data parity errors are reported on PERR- while address parity errors are reported on SERR-. As a data source, PAR generation is fully compliant with PCI specs. 2.1.

3.1.3 Target State Machine

The target state machine manages control over the PCI interface for target functions. The states implemented are a subset of equations defined in “Appendix B” of the PCI specs. The controller is a high performance state machine using one-hot encoding and is capable of continuous bursting (zero wait states).

3.1.4 Initiator State Machine

The initiator state machine manages control over the PCI interface for initiator functions. The states implemented are a subset of equations defined in “Appendix B” of the PCI specs. This also uses one-hot encoding for maximum performance and is capable of continuous bursting (zero wait states).

3.1.5 PCI Configuration Space

The PCI configuration block provides the first 64-bytes of Type 0, version 2.1 Configuration Space Header (CSH) to support software driven plug-n-play initialization and configuration using combination of CLB flip flops and look-up tables for read-write and read-only registers.

3.1.6 User Customizable configuration

The user customization configuration block allows user to customize the core for back-end application by choosing number and configuration of Base Address Registers (BARs), latency timer, device/vendor-id, external subsystem vendor-id, zero/ one-wait states, medium/slow decode and other CSH parameters.

3.2 PCI Performance features

3.2.1 Wait states and PCI transfer rates

PCI LogiCORE is switch selectable between zero and one-wait states while sourcing data to the PCI bus. Transactions where the core is the recipient (Initiator burst reads and target burst writes) are always zero wait states. No additional wait states are added in response to wait state from another agent on the bus. The ideal transaction assumes that both the initiator and the target operate with *zero wait* states and the target responds with *fast decode* speed. Initial latency on the LogiCORE target is two clock cycles more than the ideal transaction, due to *slow decode* speed (*Frame#* to *Devsel#* latency). However, LogiCore allows the user to reach the theoretical maximum on subsequent data through zero wait states. This is shown in table 2.

Table 2: LogiCORE PCI Bus Transfer Rates

S. No.	PCI transaction type	PCI transfer rate
1	Ideal PCI Write	3-1-1-1
2	Ideal PCI Read	4-1-1-1
3	Initiator Write (PCI <- Logicore)	3-1-1-2
4	Initiator Read (PCI -> Logicore)	4-1-1-2
5	Target Write (PCI -> Logicore)	5-1-1-1
6	Target Read (PCI <- Logicore)	6-1-1-1

3.2.2 Long bursting capability

The performance of any PCI application depends largely on the burst transfer capability of the interface chip. Although Logicore PCI requires additional clocks to access the PCI bus, the actual bandwidth is determined by the burst transfer size. A FIFO to support burst transfers can be efficiently implemented using the Xilinx on-chip Select RAM feature. Each CLB in 4KE/ 4KXL architecture supports a 16 bit dual ported RAM with simultaneous read-write capability, which means a synchronous 16x32 dual port FIFO which supports data transfer in excess of 33 MHz could be implemented with approximately 40 CLBs (including control logic). Implementing FIFOs with depths between 16 and 32 Dwords consumes double the RAM CLBs, and adds no delay. The user may put a FIFO deeper than 64 Dwords off-chip. The best structure for most applications is a dual-FIFO design with separate read and write FIFOs.

3.2.3 Unlimited bursting capability

For a user initiator function that requires sustained high aggregate bandwidth and is the only agent on the bus with GNT# always asserted, initiator bursting is limited only by the depth of the FIFO. Some Xilinx PCI LogiCORE customers have achieved PCI bandwidth of 124 MB/s limited only by their own system constraints, with 64Kb external FIFO and burst sizes in equivalent range.

3.2.4 Robust User Interface

3.2.4.1 De-multiplexed address

LogiCore PCI provides a simple general-purpose interface with a 32-bit data path ADIO[31:0] and a latched address for de-multiplexing the PCI address/ data.

3.2.4.2 Efficient burst transfer handling

Depending upon the largest block size of the target devices mapped onto multiple BARs, user *target address counter* size could be selected so that the target issues a Disconnect only when LogiCORE^{TM} initiator attempts to cross this boundary. Higher address bits are derived from the latched address. Likewise, depending upon the size of the largest burst transfer intended by a user initiator function, the address counter size could be fixed. User interface signals *M_DATA_VLD* and *M_SRC_EN* are used to increment enable during initiator writes and reads respectively. The LogiCORE^{TM} initiator uses the interface signal *COMPLETE* to deassert IRDY# at the end of transaction.

3.2.4.3 Modular construction

Since Target-Only and Target/Initiator options require fixed CLB resources for the PCI interface, there is flexible choice of free space available for the user design depending upon the Xilinx device selected.

3.3 Scope of performance related enhancements

3.3.1 Full medium decode speed

With faster architectures and new speed grades, the LogiCORE PCI interface can be changed to support true "medium decode", thereby decreasing the initial latency during PCI target transactions by one clock.

3.3.2 Memory Write and Invalidate

The LogiCORE target interface can receive and process a Memory Write and Invalidate (MWI) PCI command. However, the initiator interface does not support MWI because the interface does not track the cache line size.

3.3.3 Auto wait states removal during initiator last data phase

Faster FPGA technology will also permit a design change to eliminate the subsequent latency of one clock during last data phase of initiator operations.

3.3.4 Fast back-to-back transactions

The LogiCORE interface supports back-to-back transactions, as shown in Fig. 1 and can be changed to perform fast back-to-back transfers by modifying the target and initiator state machines to remove the idle state.

Conclusion

PCI bus efficiency is enhanced by following the advancements described and carefully examining the limitations in your respective PCI systems. The key to sustained high bandwidth is long bursting without terminations and wait states. The Xilinx PCI interface helps achieve this performance by using a large FIFO interface, while providing a time-to-market advantage and plug-n-play operation. The FPGA-based PCI solution allows a designer to integrate a PCI interface with value-added custom logic in a single-chip.

Acknowledgements

I wish to acknowledge the contributions of Wilson Yee and Per Holmberg for reviewing this paper and giving valuable suggestions, and to thank Xilinx PCI LogiCORE team member Eric Crabill for providing necessary feedback.

References

1. William Holland, "PCI Bus Performance A realistic Look," PCI Plus '97 proceedings
2. Frank Hady, "Efficient use of PCI", PCI Plus '97 proceedings
3. Mark Easley, "Using I2O in embedded PCI Systems", PCI Plus Europe '97 proceedings
4. Peter Forstner, "PCI to PCI Bridges", PCI Plus Europe '97 proceedings
5. Victor J. Duvanenko, "Two Writes Make a Read", Computer magazine
6. "PCI Local Bus Specification", 1995 (Revision 2.1)
7. Xilinx, "LogiCore PCI Master and Slave Interface User's guide", 1996 (Version 1.1)