Hardware Implementation of PCI Interface Using Verilog & FPGA

Iqbalur. Rahman Rokon Towsif. Sazzad, Samilat. Kaiser

Abstract - PCI (Peripheral Component Interconnect) is a computer bus for attaching hardware devices in a computer. Work on PCI first began at Intel's Architecture Development Lab, circa 1990. PCI provides direct access to system memory for connected devices, but uses a bridge to connect to the frontside bus and therefore to the CPU. PCI bus traffic is made of a series of PCI bus transactions. Each transaction is made up of an address phase followed by one or more data phases. The direction of the data phases may be from initiator to target (for write transaction) or vice-versa (for read transaction). A PCI interface will connect any generic devices to the PCI bus. This interface will create the communication between master and slave devices. The interface will read the command of the master and send corresponding response to the master. Generally, the master initiates and controls all communication that takes place over the bus. Our interface design includes read and write operations and will be able to communicate to register and RAM through the PCI.PCI has distinct interfacing pins and according to these pins we made interface with Register and RAM.

Keywords— PCI Protocol, PCI Interface, VLSI, FPGA, Verilog.

I. Introduction

The form PCI stands for Peripheral Component Interconnect, which suitably describes what it does. PCI was designed to satisfy the requirement for a standard interface for connecting peripherals to a PC, capable of sustaining the high data transfer rates needed by modern graphics controllers, storage media, network interface cards and other devices. [1]

Although many computer users think of PCI as a way of laying out electrical wires, it is actually a complete set of specifications defining how different parts of a computer should interact. PCI bus is an improved bus for PC compatible computers. It is faster than the previous buses. And it can be expandable to both 32-bit and 64-bit. The PCI specification covers most issues related to computer interface. PCI has distinct interfacing pins and according to these pins we need to interface with other devices.

The PCI bus has 4 main characteristics:

- Synchronous: bus uses one clock. PCI clock runs at 33MHz by default but can run lower to save power or higher (66MHz) if the hardware supports it.
- Transaction/Burst oriented: We start a transaction>specify the starting address>send as many data we want>end the transaction
- Bus mastering: transactions work in a master-slave relationship. A master is an agent that initiates a transaction (can be a read or a write).
- Plug-and-play: host-CPU/host-OS can: Determine the identity of each PCI board in a PCI bus, determine the abilities/requirements of each board, and Relocate each board memory space.[2]

PCI timing:

PCI specifies timing related to its clock.
With a 33MHz clock, we have:

- 7ns/0ns Tsu/Th (setup/hold) constraint on inputs
- 11ns Tco (clock-to-output) on outputs.[2]

II. Interface I/O Signals

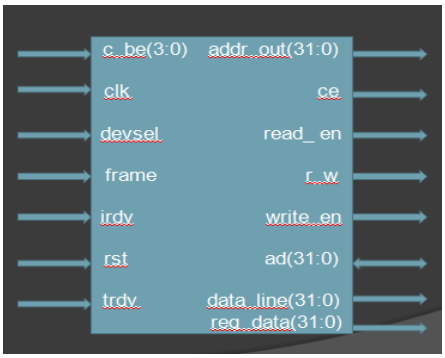

Fig. 1 Interface Block

- **Frame:** It is the activation signal. When it is low the device is ready to perform.[3]
- **Command or Byte Enable (C_be[3:0]):** It is a 4 bit bus. It sends 0010 for read command and 0011for write command to PCI.[3]

Iqbalur. Rahman Rokon is a Faculty member in North South University, Dhaka, Bangladesh. Former Sr. Engineer, VLSI Chip Research and Development (R&D), Emulex Corporation, California, USA;
(Phone: +88-01726246189 ; e-mail: irahman@northsouth.edu).

Towsif Sazzad is a graduate from North South University, Dhaka, Bangladesh. He is now working as a Senior SQA Engineer in Speedwell Softtech LTD, Dhaka, Bangladesh; (e-mail: towsifsazzad@gmail.com).

Samilat Kaiser is a graduate from North South University, Dhaka, Bangladesh. She is now working as a Pre-Sales Consultant in Tech one Global, Dhaka, Bangladesh (e-mail: samilatkaiser@gmail.com).

- **Address (Ad[31:0]):** It is a 32 bit bus. Both address bus and data bus are multiplexed in this same bus.[3]
- **Initiator Ready(irdy):** Initiator ready is low when the master device is ready to make transaction.[3]
- **Target Ready(trdy):):** Target ready is low when the slave device is ready to make transaction.[3]
- **Device Select(Devsel):** The target asserts devsel low as an acknowledgement that it has positively decoded the address. [3]
- **CE:** When it goes low, RAM is active to make the transaction.[4]
- **r_w:** when it is high, read operation occurs. When low, write operation occurs to Ram.[4]
- **Read_en:** When read enable is high data is read from the Register.
- **Write_en:** When write enable is high data is written to Register.

III. ADDRESS MAPPING

In our design, the method that we have applied to select the Slave devices is called address mapping. We have divided the address of our Master (PCI) device into two parts. One is for Ram and another is for Register. Our Master (PCI) device has a 32-bit address from which we can have 2^32 that means 4294967296 number of addresses. To map the address we have taken the most significant 4 bits. And by these 4 bits we can have 16 combinations starts from 0000 and ends up with 1111. From these 16 combinations, we have chosen the first 12 combinations for the RAM and the rest of the 4 combinations for REGISTER. So the address for RAM is selected from 0000 to 1011 and the address for REGISTER is selected from 1100 to 1111. Finally the address ranges for these two slave devices are:
Address Range for RAM: 0000_0000 to 1011_1111
Address Range for REGISTER: 1100_0000 to 1111_1111

IV. SIMULATION ENVIRONMENT

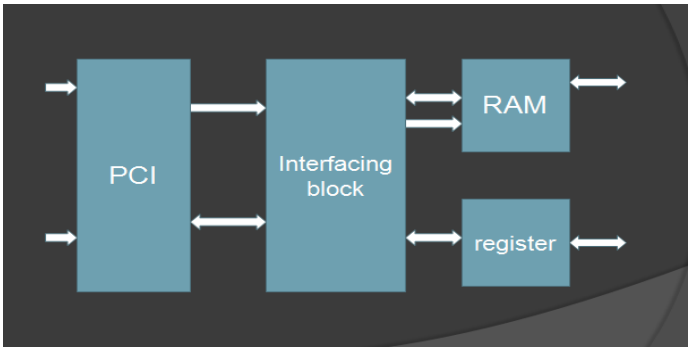

Fig. 2 Simulation Environment

PCI interface is a device, which acts as a transitional medium between Master/Initiator and Slave/Target devices. In our design, we have used PCI as the Master/Initiator and RAM and REGISTER as the Slave/Target devices. Here, we can read data from RAM and REGISTER to PCI and also can write data from PCI to RAM and REGISTER.

V. TIMING DIAGRAM

A. Read Transaction:

1) The following timing diagram illustrates a read transaction on the PCI bus:

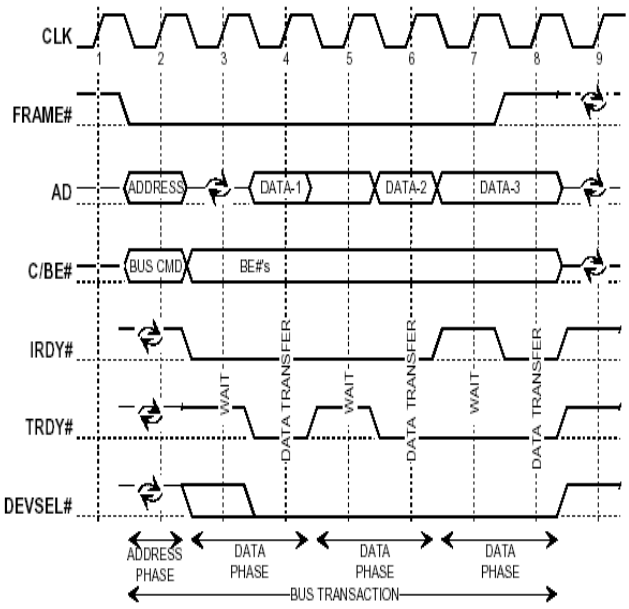

Fig. 3 Read Transfer Diagram

The following is a cycle by cycle description of the read transaction:

- Cycle 1 - The bus is idle.
- Cycle 2 - The initiator asserts a valid address and places a read command on the C/BE# signals. This is the address phase.
- Cycle 3 - The initiator tri-states the address in preparation for the target driving read data. The initiator now drives valid byte enable information on the C/BE# signals. The initiator asserts IRDY# low indicating it is ready to capture read data. The target asserts DEVSEL# low (in this cycle or the next) as an acknowledgment it has positively decoded the address. The target drives TRDY# high indicating it is not yet providing valid read data.
- Cycle 4 - The target provides valid data and asserts TRDY# low indicating to the initiator that data is valid. IRDY# and TRDY# are both low during this cycle causing a data transfer to take place. The initiator captures the data. This is the first data phase.
- Cycle 5 - The target deasserts TRDY# high indicating it needs more time to prepare the next data transfer.

- Cycle 6 - The second data phase occurs as both IRDY# and TRDY# are low. The initiator captures the data provided by the target.
- Cycle 7 - The target provides valid data for the third data phase, but the initiator indicates it is not ready by deasserting IRDY# high.
- Cycle 8 - The initiator re-asserts IRDY# low to complete the third data phase. The initiator captures the data provided by the target. The initiator drives FRAME# high indicating this is the final data phase (master termination).
- Cycle 9 - FRAME#, AD, and C/BE# are tri-stated, as IRDY#, TRDY#, and DEVSEL# are driven inactive high for one cycle prior to being tri-stated.[5]

A. Write Transaction:

The following timing diagram illustrates a write transaction on the PCI bus:

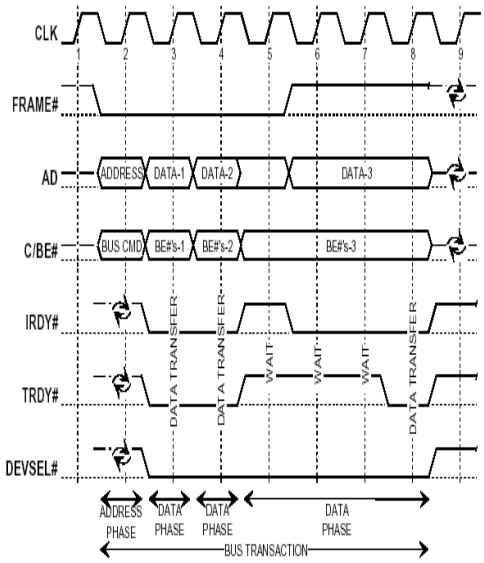

Fig. 4 Write Transfer Diagram

The following is a cycle by cycle description of the read transaction:

- Cycle 1 - The bus is idle.
- Cycle 2 - The initiator asserts a valid address and places a write command on the C/BE# signals. This is the address phase.
- Cycle 3 - The initiator drives valid write data and byte enable signals. The initiator asserts IRDY# low indicating valid write data is available. The target asserts DEVSEL# low as an acknowledgment it has positively decoded the address (the target may not assert TRDY# before DEVSEL#). The target drives TRDY# low indicating it is ready to capture data. The first data phase occurs as both IRDY# and TRDY# are low. The target captures the write data.
- Cycle 4 - The initiator provides new data and byte enables. The second data phase occurs as both IRDY# and TRDY# are low. The target captures the write data.
- Cycle 5 - The initiator deasserts IRDY# indicating it is not ready to provide the next data. The target deasserts TRDY# indicating it is not ready to capture the next data.
- Cycle 6 - The initiator provides the next valid data and asserts IRDY# low. The initiator drives FRAME# high indicating this is the final data phase (master termination). The target is still not ready and keeps TRDY# high.
- Cycle 7 - The target is still not ready and keeps TRDY# high.
- Cycle 8 - The target becomes ready and asserts TRDY# low. The third data phase occurs as both IRDY# and TRDY# are low. The target captures the write data.
- Cycle 9 - FRAME#, AD, and C/BE# are tri-stated, as IRDY#, TRDY#, and DEVSEL# are driven inactive high for one cycle prior to being tri-stated.[5]

VI. Finite State Machine

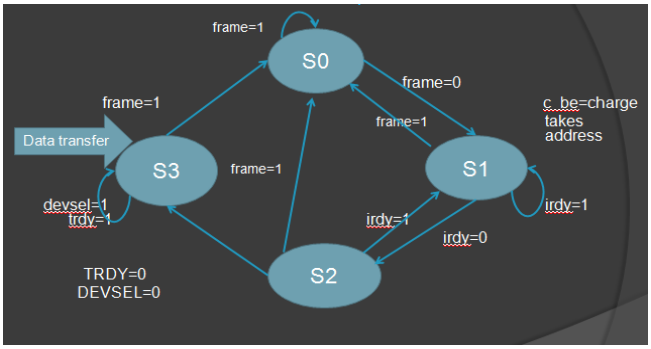

Fig. 5 State Machine

S0 = State_idle
S1 = State_take_address
S2 = State_take_data
S3 = State_semd_address

A. State_idle:

In this state all the signals are in idle mode. Or if they are not in idle state then we are reseting all the signals to idle state. That means all the signals are inactive to perform operation and wait for the master device activation signal. When the Frame signal goes low, it activates the master device that means the PCI is now active to read or write data to slave devices. And just the moment frame goes low, the ad signal works as an address bus. After this happen the device changes its state and put the address value into the address buffer (addr_buff). And if it is find that the frame is not low it

remains in the idle state. After getting the frame low it sends a read or writes command by the command bus and stays in the next state.

B. State_take_address:

The second state is the state take address. If irdy changes its state from idle state that is 1, it also changes the state and goes to the next state which is state take data. After changing the state it makes the address acknowledgement signal high and again store the ad value into a data buffer (data_buff). At that time the ad bus works as a data bus. But if the irdy signal is not low then it waits for this signal to be low. And if the frame again changes its state from low to high, it goes back to the idle state.

C. State_take_data:

The third state is the state take data and if it is found that devsel and trdy are low here and address acknowledgement is given (addr_ack = 1), it makes the data acknowledgement high and after this it changes its state to the next state. But if the signals devsel and trdy are not low, it goes back to the previous state.

D. State_send_address:

The fourth and final state is the state send ad. In this state at first it checks that weather it is a write transaction or a read transaction and which of the slave devices would be selected. If it is a write transaction and the slave device is Ram, then it makes the other internal signals as its need and makes the Ram's activation signal 'chip select' (cs) as low and also make the rd_wr (we) low. And thus the transaction occurs. Again if it is a read operation from Ram, then again it makes the other internal signals as its need and makes the Ram's activation signal 'chip select' (cs) low and r_w (we) high. Thus a read transaction performs.

Again if Register is selected as the slave device and writes command is send it makes the other internal signals as its need and write the data to Register. And if it is a read command send to Register, it also does the same thing and the data would be read from Register. This process will continue until or unless the frame signal goes high. If it goes high, then it stops the transaction and goes back to the idle state.

VII. SIMULATION AND IMPLEMENTATION

For our simulation process, we have used Xilinx 9.2i.For synthesis and implementation; we have used Xilinx9.2i and Altera DE1 development and Education board.

A. FPGA Board Used:

The purpose of the Altera DE1 Development and Education board is to provide the ideal vehicle for advanced design prototyping in the multimedia, storage, and networking. [6]

Fig. 6 Altera DE1 Board

VIII. SIMULATION RESULTS

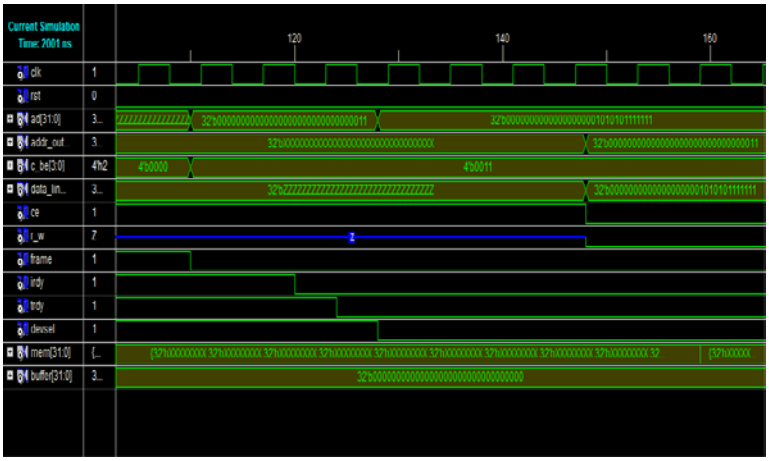

Fig. 7 Simulation waveform of write data to RAM

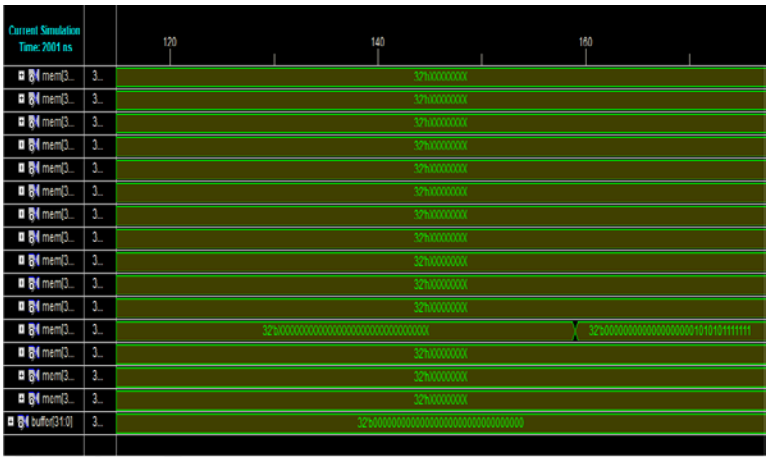

Fig. 8 Simulation waveform of write data to RAM (cont.)

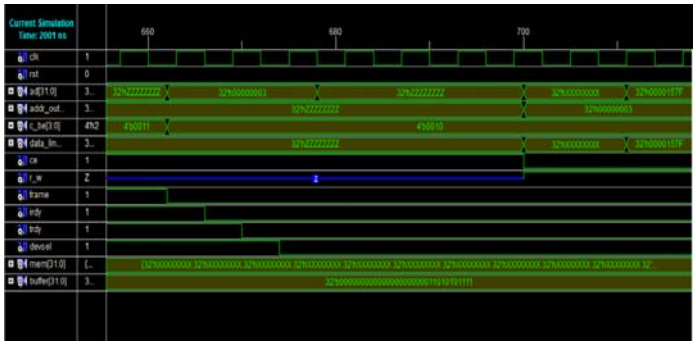

Fig. 9 Simulation waveform of Read data from RAM

IX. Synthesis Results

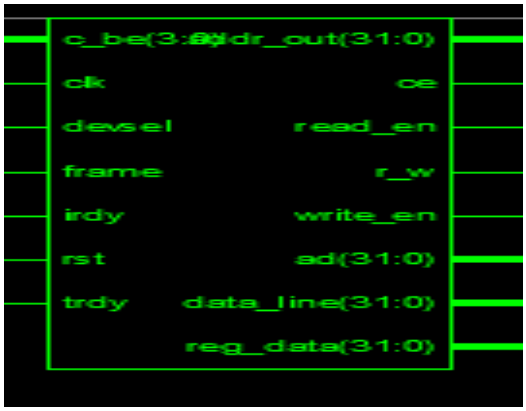

Fig. 10 Interface Block Diagram form Xilinx Schematic

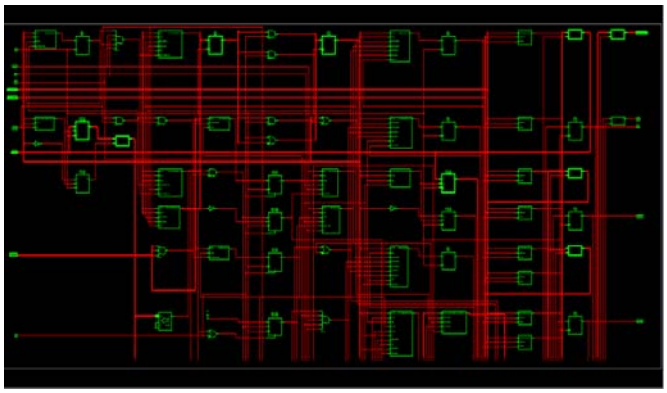

Fig. 11 Internal Logic circuit diagram of the Interface module from Xilinx schematic

X. Conclusion

Today's computer systems, with their emphasis on high resolution graphics, full motion video, high bandwidth networking, and so on, go far beyond the capabilities of the architecture that ushered in the age of the personal computer in 1982. Modern PC systems demand high performance interconnects that also allow devices to be changed or upgraded with a minimum of effort by the end user. In response to this need, PCI has emerged as the dominant mechanism for interconnecting the elements of modern, high performance computer systems. It is a well thought out standard with a number of forward looking features that should keep it relevant well into the next century. Originally conceived as a mechanism for interconnecting peripheral components on a motherboard, PCI has evolved into at least a half dozen different physical implementations directed at specific market segments yet all using the same basic bus protocol. In the form known as Compact PCI, it is having a major impact in the rapidly growing telecommunications market.

Though in our project we have made an interface with register and RAM but this will work for any generic device in the PCI network. We hope that this project will help the further PCI related invention. Technologies are becoming smarter and compact day by day, so we hope PCI protocol and PCI interface will add new dimension in that trend. This interface will play a remarkable role with its significant speed and reliability.

Acknowledgment

All praises to Allah, the Almighty, the most gracious, the most merciful, benevolent, without whose help, this work would not have been possible. We would like to express our gratitude to all those who gave us the possibility to complete this project.

We want to thank Mr. Nahid Mushfique Jawad, who supported us in our project work. He had remained with us throughout the research work. We want to thank him for all his help, support, interest and valuable hints. We also want to thank Reasat Chowdhury for his selfless concern. We want to thank our parents and other family members for their gallant concern. Finally, we want to thank the Department of Electrical Engineering and Computer Science of our prestigious North South University, Bangladesh for giving us such an opportunity to carry on our research work.

References

[1] T*ech-pro.net*. (n.d.). Retrieved August 5, 2010, from tech-pro.net: http://www.tech-pro.net/intro_pci.html

[2] *fpga4fun*. (2010, spetember 07). Retrieved september 07, 2010, from fpga4fun: http://www.fpga4fun.com/PCI1.html

[3] *PCI Local Bus Technical Summary*. (2010, october 10). Retrieved october 10, 2010, from techfest: http://www.techfest.com/hardware/bus/pci.htm

[4] *Figure Block Diagram of Static RAM Table Truth Table* . (2010, july 27). Retrieved july 27, 2010, from docstoc: http://www.docstoc.com/docs/4219029/Figure-Block-Diagram-of-Static-RAM-Table-Truth-Table

[5] *PCI Bus Timing Diagram*. (2010, august 17). Retrieved august 17, 2010, from silverhawk: http://silverhawk.net/notes/tutorials/hardware/pci-timing.html

[6] *Altera DE1 board*. (n.d.). Retrieved december 10, 2010, from terasic: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=83