PCI Device Implementation for Front End Processing Module in Ultrasound Scanning System

Athul Balan Edichery Alinkeezhil, Goutham Pocklassery, Harikrishna Menon, Athul Asokan Thulasi, Tom Charly Kattakayam

athulbalanea@gmail.com, gouthamravindran@gmail.com, beinghari03@gmail.com, athulat.002@gmail.com, tomcharlyk@gmail.com

Abstract—**In this project we implement the PCI Device Interface for integration into the Front End Processing Module in a typical ultrasound scanning system. Our team focuses in implementing data manager unit with PCI communication protocol. This will be synthesized to fit into a Xilinx FPGA. It is supposed to be supporter module for communication to control and regulate data transfer over one or more units. This master/slave device will be basically used for load sharing purposes when the CPU need to communicate with memory, Ultra scanner frond end, and other device in system architecture. The design and implementation is restricted to RTL design of the PCI device in Verilog HDL following the industry PCI Standard – Peripheral Component Interface Bus Specification 2.1. The same is synthesized using Xilinx FGPA.**

***IndexTerms*—PCI, FPGA, Verilog HDL, Front End**

I. Introduction

Today, India has a population of 1.2 billion and the majority of them live in rural areas and about 60 per cent of this rural population lives on less than Rs 35 a day and nearly as many in cities live on Rs 66 a day. So the percentage of the Indian population which can afford proper healthcare is relatively small. Moreover, rural areas have very few hospitals and doctors as compared to cities. Situations might get serious in case of emergencies especially during pregnancy period. They might have to travel long distances in order to get treatment as well as the check-ups which may not be easy for pregnant women. In this project we implement the PCI Device Interface for integration into the Front End Processing Module in a typical ultrasound scanning system. Through this project we have learnt the industry PCI Standard – Peripheral Component Interface Bus Specification 2.1

II. Motivation

The majority of the population in the developing and underdeveloped nations lives in rural areas with limited access to proper health care resources. There is a wide gap between the number of people who have access to modern health care facilities and those who do not. One of the reasons is the high cost of the medical technology used to diagnose and cure. Pregnant women are a very care needing group of people. They need more medical attention than normal person to ensure proper growth of the baby. Majority of health infrastructure, medical power and other health resources are mainly located in urban areas. Most of the pregnant women in the rural areas seldom visit hospitals and might not receive any kind of medical attention. Unsafe and unhygienic birth practices, unclean water, poor nutrition and unsanitary environments in rural areas are a major risk to the health of pregnant women. This is reflected in the poor health condition of children as well as in the case of increasing infant mortality rate. With proper medical care given to the pregnant women many problems like birth defects, multiple pregnancies, baby's position in the womb, tumors of pregnancy, miscarriage etc. can be identified and taken care at early stage.

III. Related Works

Ultrasonography is widely used medical technique for visualizing internal body structures. Ultrasound images (sonograms) are made by sending a pulse of ultrasound into tissue using an ultrasound transducer (probe) with no break in the skin. A number of such devices has been developed using FPGA in recent years. In one such work this is done incorporating a high speed imaging board with analog front-end electronics and digital back-end unit, a high speed mechanical sector probe and a high frequency bipolar pulse generator. It also has a FPGA based programmable and imaging board utilizing 64-bit PCI bus for high-speed data transfer and real-time imaging [1]. In our design, we are trying to bring together front end processor, ultrasound imaging sensor, sensing modules to measure heart rate and oxygen content, electrocardiogram and BP. We implemented a 32-bit PCI Device in FPGA for managing data transfer between slave devices, memory and the CPU.

There are designs that has connection between PCI interface and FPGA using PLX interface chip PCI9054 [2]. But in our design instead of using the standardized PCI chip we design a PCI device in HDL and use FPGA to support communication with processor and other hardware devices. Realizing PCI device in FPGA makes it programmable, real-time and high resolution compactable which will enhance communication between ultrasound frond ends. The reconfigurable PCI interfaces like hardware and firmware design of the FILAR PCI interface card are examples for such works which is largely implemented in FPGA. The card features four 2 Gbit/s serial optical S-LINK channels and a 64-bit/66 MHz PCI interface [3].

IV. System Architecture

The generic system architecture of the PCI Device for Front End Processing Module in Ultrasound Scanning System is illustrated in the Figure 1. In this system is shown, a microprocessor, memory and there slave devices. The three

slave devices may be peripherals like ultra sound front end, sensing modules to measure heart rate and oxygen content, electrocardiogram and BP. In this project, we implemented a PCI Device in FPGA which will be used to manage data communications between slave devices, memory and the CPU. The device is designed with functions to support PCI protocol - Peripheral Component Interface Bus Specification 2.1. to communicate with other peripherals in the architecture and as a memory interface unit to handle data exchange with memory unit. The memory interface unit with address generator helps to read/write data into the memory from the IO ports (slave devices) without taking much clock cycles for the CPU to process the transaction. The CPU has to generate only the start address of the read or write cycles. Upon request from the slave devices which runs on a PCI peripheral bus, the PCI device will initiate a bus cycle to access memory. This will tremendously improves the speed of the system by reducing number of clock cycles required for the transaction as compared to less efficient methods like programmed IO transfers. In programmed IO transfers PCI device generates an interrupt to inform the CPU that it needs data transferred. The device interrupts service routine (ISR) causes the CPU to read from the PCI device into one of its own registers. The ISR then tells the CPU to write from its register to memory. The ISR then tells the CPU to write from its register to memory.

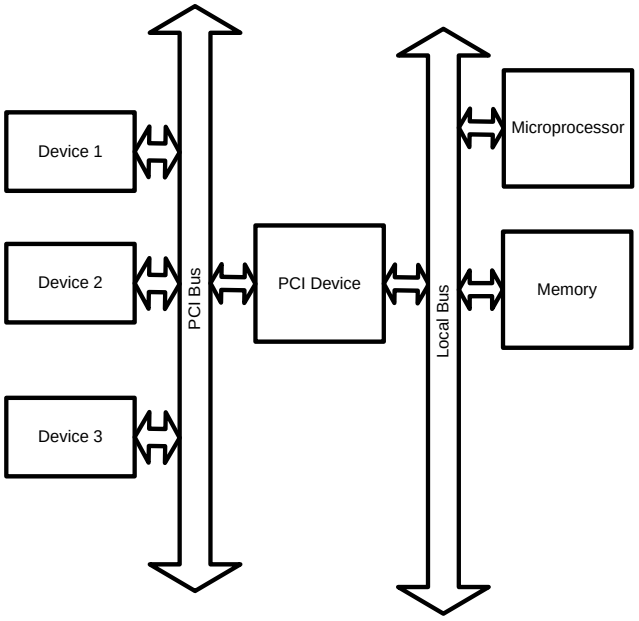

Fig. 1. System architecture diagram.

V. FPGA Device Architecture

In this project we have implemented a PCI device, a transmit FIFO, receive FIFO, and memory interface in FPGA. The PCI device has four sub-units PCI master controller, PCI read controller, PCI write controller and PCI error handling module. The FPGA device is designed with a memory interface module since PCI architecture has no central DMA controller. It also has two FIFOs for temporally holding the data during memory read and write operations. The FPGA device architecture is shown in the figure 2.

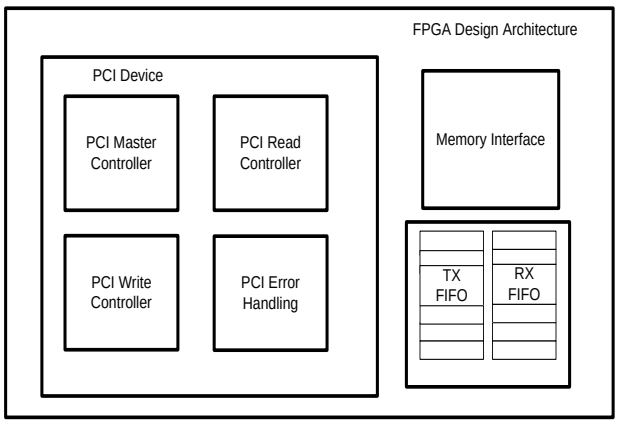

Fig. 2. FPGA device architecture.

A. Data Flow of the Designed System

The FPGA device access data from the memory starting from the address driven into the drive as input. The device has a memory interface to keep track of the address while performing read and write operation. The address generator in memory interface increments the address in each clock as long as a read or write operation is requested. The data to be written into the memory is first stored into the transmit FIFO and then upon establishing ownership over data bus to the memory it is read from the FIFO and written in the memory. Similarly the receive FIFO stores the data from the memory and then transfers to peripherals when they are ready for reception.

VI. Implementation

The register transfer level (RTL) design of the system is done using Verilog hardware description langague (HDL). ModelSim EDA tool is used for simulation of the RTL. Using Xilinx ISE, the RTL design is synthesized into an equivalent hardware file which can be programmed into a FPGA.

A. Block Diagram

1) PCI Device:

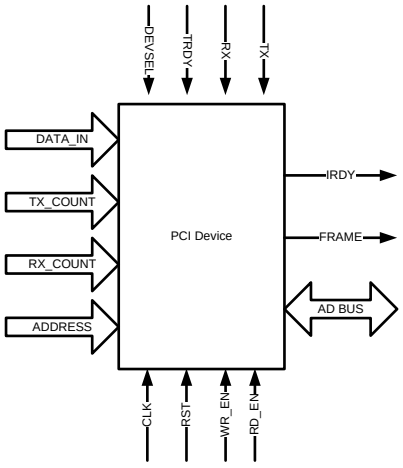

Fig. 3. PCI device module diagram.

TABLE I. PIN DESCRIPTION OF PCI DEVICE

Signal Name	I/O	Description
CLK	Input	System Clock
RST	Input	System Reset
DATA_IN	Input	Data line to write into the transmit FIFO from CPU and other devices
TX_COUNT	Input	Number of memory write transaction to be done
RX_COUNT	Input	Number of memory read transaction to be done
ADDRESS	Input	Start address for burst read/write transaction
WR_EN	Input	Write enable in Receive FIFO buffer
RD_EN	Input	Read enable in transmit FIFO buffer
TX	Input	Memory write transaction enable
RX	Input	Memory read transaction enable
DEVSEL#	Input	PCI device select signal. Asserted by the targets of the PCI transaction to claim the transaction.
TRDY#	Input	PCI target ready signal
FRAME#	Output	PCI frame signal. Used by PCI initiator for signaling beginning and end of PCI transaction
IRDY#	Output	PCI initiator ready

2) Transmit/Receive FIFO:

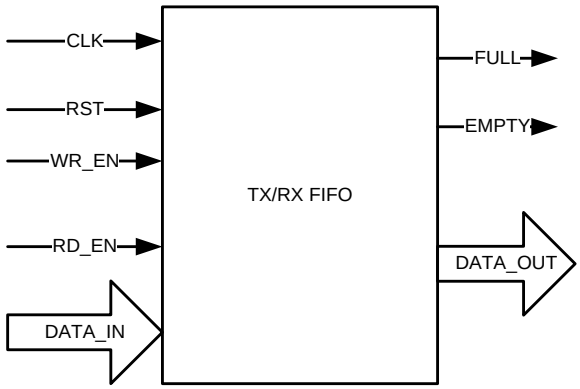

Fig. 4. Module diagram of FIFO

TABLE II. PIN DESCRIPTION OF FIFO

Signal Name	I/O	Description
CLK	Input	System Clock
RST	Input	System Reset
WR_EN	Input	Write enable for FIFP
RD_EN	Input	Read enable for FIFO
DATA_IN	Input	Input data for FIFO
FULL	Output	Signal to indicate FIFO full status
EMPTY	Output	Signal to indicate FIFO empty Status
DATA_OUT	OUTPUT	Output data bus for FIFO

B. RTL Behavioral Description

We have implemented a PCI Device with PCI protocol with burst read and write transfers, that too without any wait states. The read/write transaction is designed as per the PCI protocol. The memory read transaction starts by de-asserting the FRAME and IRDY, driving address onto AD bus and command 0110 onto C/BE bus. The target latch and decode address along with de-asserting DEVSEL. Then, there would be a turn-around cycle, to stop driving the AD bus from the design following which the targets starts driving data into AD bus. The memory write transaction starts by asserting the Frame#, driving address onto AD bus and command 0111 onto C/BE bus. The 'Byte enable' change along with each data phase as that in read transaction. The target latch and decode address along with asserting DEVSEL#. Then, the target starts driving data into AD bus, since turn around cycle is not requires in write truncation. The absence of turn-around cycle in write transaction is because both address and data is drive from same PCI bus agent. This cycle is required to avoid a collision of data when changing the ownership of the AD bus. A state machine to illustrate the read/write transaction is show in figure 5.

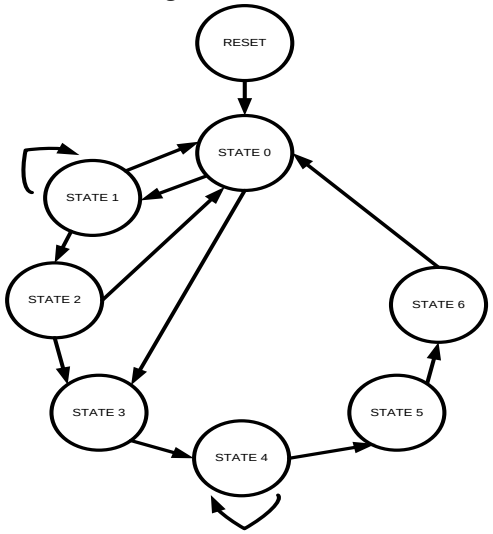

Fig. 5. State machine

There are seven states in the state machine and a reset state for the PCI device Verilog implementation. In the reset state, signals like FRAME and IRDY are asserted and state is changed to state 0. In state 0, the state machine checks whether a read or write transaction is requested. If there is request for read/write transaction, the FRAME bit is de-asserted. The state is change to state 1 for write request and state 3 for read request. Along with the request for a read/write transaction the target will also provide the start address and number of location in the memory need to be read/written as input to the PCI device. Thus these inputs are stored to local register in state 0. In state 0, C/BE bus is driven with a command '0110' for read request, whereas, C/BE is driven with command '0111' to write into the memory.

In state 1, PCI device will write data into the memory, if TRDY and DEVSEL is de-asserted, and will remain in the same state until completing number of transactions requested. When the requested transactions are completed, the state is changed to state 2 along with asserting FRAME bit. This state also checks for underrun error in the PCI device. Underrun error bit is asserted in state, when the transmit FIFO buffer is empty or the request number of read transaction is more than the data stored in transmit FIFO buffer. Along with underrun

error bit FRAME and IRDY are asserted to stop transaction. In state 2, IRDY signal is asserted and state is change to state 0, if TRDY and DEVSEL are asserted.

State 3 is a turn-around state. This state is utilized to turn-around the ownership of the AD bus from initiator to the target to read the memory. In state 4, byte enable is driven and data is read through the AD bus, if TRDY and DEVSEL are de-asserted, and will remain in the same state until number of requested transactions completed. When the requested transactions are completed, the state is changed to state 5 along with asserting FRAME bit. Receive FIFO buffer is monitor for overrun error, in this state. If the front end processor fails read the data quickly enough and the receive FIFO buffer becomes full, an Overrun Error will occur, and read data will be lost. And in case of overrun error, the state is changed to state 0. In state 5, IRDY is asserted and state is changed to state 6, when TRDY and DEVSEL is asserted. State 6 is a idle state but it moves the state machine back to state 0.

Apart from this state machine, the design has two FIFOs-transmit FIFO and receive FIFO. The transmit FIFO temporarily stores the data from CPU and devices which is to be written into the memory through PCI device. And when there is request to write more data than transmit FIFO holds, the FIFO becomes empty and under-run error occurs. The receive FIFO temporarily holds the data read from the memory to CPU or other device through PCI device. Overrun error is associated with this FIFO. The depth of both FIFOs is defined as eight locations of byte wide.

VII. VERIFICATION

The register transfer level (RTL) design is simulated and tested to meet the design specification in ModelSim EDA tool. The PCI device, design under test is simulated, to tested read/write cycles, under-run error flag and overrun error flag. The test bench architecture-with its interacting signals between the test bench and design under test is shown in the figure 6 below.

For testing memory write through PCI device, initially data was loaded into transmit FIFO by asserting signal 'WR_EN' which is internally wired to write enable in the transmit FIFO. The transmit FIFO generates address and stores these data coming through input data line until 'full' bit of the transmit FIFO is asserted. The PCI device determines whether is the request is for a memory write/read by sampling the TX and RX input. When the TX bit is triggered from the test bench the PCI device starts a memory write cycle by asserting the read enable bit of the transmit FIFO. Along with the TX bit the test bench also stimulate a start 'ADDRESS'. Then, starting from the second clock after triggering TX signal, the transmit FIFO will drive data which is to be written into the memory through AD bus. The test waveform is shown in the figure 7.

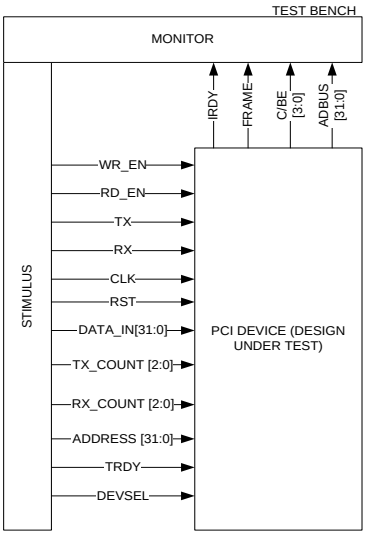

Fig. 6. Test bench architecture

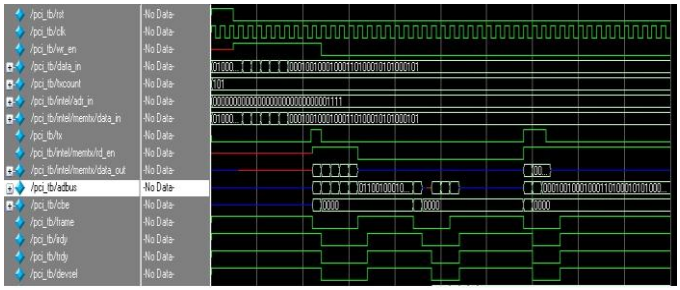

Fig. 7. Waveform for the test case: Memory Write

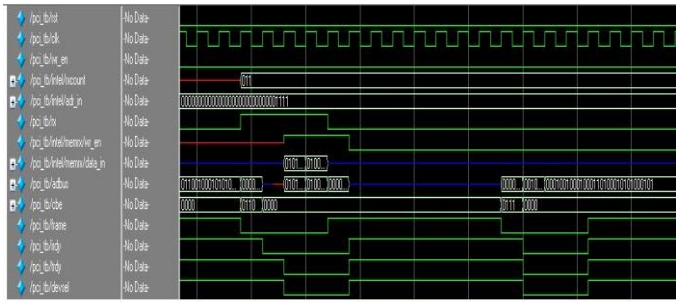

Fig. 8. Waveform for the test case: Memory Read

When the RX bit is triggered from the test bench the PCI device starts a memory read cycle by asserting the write enable bit of the receive FIFO. Along with the TX bit the test bench also stimulate a start 'ADDRESS'. Then, starting from the second clock after triggering RX signal, the transmit FIFO start drive data which is read from the memory through AD bus. The receive FIFO generates address internally and stores these data coming through input data line until 'full' bit of the transmit FIFO is asserted. When the receive FIFO is full, overrun error will occur. The test waveform is shown in the figure 8.

VIII. Synthesis Report

The design has been successfully synthesized in FPGA SPARTAN-3A XC3S700A (FGG484)-4.

A. Device utilization summary

Selected Device: 3s700anfgg484-4

Number of Slices:	107	out of 5888	1%
Number of Slice Flip Flops:	64	out of 11776	0%
Number of 4 input LUTs:	208	out of 11776	1%
Number used as logic:	144		
Number used as RAMs:	64		
Number of IOs:	116		
Number of bonded IOBs:	115	out of 372	30%
IOB Flip Flops:	32		
Number of GCLKs:	1	out of 24	4%

B. Timing Summary:

Minimum period: 5.926ns (Maximum Frequency: 168.758MHz).
Minimum input arrival time before clock: 5.970ns.
Maximum output required time after clock: 7.516ns.
Maximum combinational path delay: No path found

IX. Conclusion

In this project work we have built a FPGA based PCI Interface for integration into the Front End Processing Module in a typical ultrasound scanning system.The design, implementation and verification of the PCI device in donein Verilog HDL following the industry PCI Standard. The same is synthesized using Xilinx FGPA. This project gave us a very good opportunity to learn the industry PCI Standard – Peripheral Component Interface Bus Specification 2.1.

Acknowledgment

We are really grateful to our guide Prof. Rajesh Kannan Megalingam without whose support we would have never done this project. We are also thankful to Humanitarian Technology Labs of Electronics and Communication department of Amrita School of Engineering, Amrita Vishwa Vidyapeetham Univeristy, Amritapuri campus, Kollam, India for providing us all the necessary lab facilities and support towards the successful completion of the project. We also thank the IEEE Standard Education Committee for funding this project.

References

[1] Weibao Qiu, Yanyan Yu, Lei Sun, " A Programmable, Cost-Effective, Real-Time High Frequency Ultrasound Imaging Board Based on High-Speed FPGA, 2010 IEEE Ultrasonics Symposium (IUS), ISBN: 978-1-4577-0382-9

[2] Wang Liu, Yunfeng Liu, Liyan Qiao, "Development of Dual-channel High-speed Data Acquisition Card Based on PCI Bus", ISBN: 978-1-4673-5683-1

[3] Wieslaw Iwanski, Stefan Haas, and Markus Joos, " A PCI Interface With Four 2-Gbit/s Serial Optical Links, IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 52, NO. 6, DECEMBER 2005

[4] PCI Local Bus Specification, Revision 2.2, 1998

[5] PCI System Architecture, Fourth Edition, MindShare Inc., Tom Shanley and Dan Anderson