FPGA Based Parallel Communication using PCI by Developing C Code

Disha Singh[1]Mohammed Arif[2]
[1]M. Tech Student, Department of Electronics & Communication Engineering,
[2]Asst. Professor, Department of Electronics & Communication Engineering
[1, 2]Gyan Ganga Institute of Technology and Sciences, Jabalpur

Abstract— *In this paper, FPGA based parallel communication has been done using Peripheral Component Interconnect (PCI) by development of C code in Lab-Windows CVI 2012. Communication interface has been designed for six data registers to transfer data between FPGA and PC. At FPGA end, VHDL code has been composed in XILINX ISE 13.1 software environment, while at PC end, C code has been developed in Lab-Windows CVI 2012. In random, six registers have been taken and information contained in these registers are in the form of 32 bit digital data in each register. As per the analysis done, the information obtained was stable in nature. The data transfer speed achieved during the transfer of data information of each register between user FPGA and user application (i.e. PC) was 500 kbps. The information of the registers were observed through PCI on Graphical User Interface (GUI) which was developed on Lab-Windows CVI 2012 (student version).*

Keywords—PCI, FPGA, PC, API, GUI, Lab-Windows CVI 2012, Xilinx ISE 13.1.

I. Introduction

Communication interface is basically created to obtain the information over a range. Data information can be transmitted through different modes, which will make the communication interface reliable, easy and accurate.
Basically, there are two modes of communication:

- Serial Communication
- Parallel Communication

The transmission of binary data across a link can be accomplished in either of the mode that is parallel or serial mode. In parallel mode, multiple bits are sent with every clock cycle. In serial mode, 1 bit is sent with each clock cycle.

A. Parallel Communication:

Binary data, consisting of 1s and 0s strings, may be organized into groups of *n* bits each. The mechanism for parallel communication is a conceptually simple: Use *n* wires to send *n* bits at one time. That way each bit has its own wire, and all *n* bits of one group can be transmitted with each clock cycle from one device to another. The advantage of parallel transmission is speed. Parallel transmission can increase the transfer speed by a factor of *n* over serial transmission.

B. Serial Communication

In serial transmission one bit follows another, so we need only one communication channel rather than *n* to transmit data between two communicating devices. For this project, PCI interface as a parallel communication is used. PCI interface has advantage that is data transmission rate.

II. Peripheral Component Interconnect (PCI)

PCI stands for Peripheral Component Interconnect. The PCI bus can populated with the adapters requiring fast accesses to each other and/or system memory and that can be accessed by the processor at speeds approaching that of processor's full native bus speed [1].

Mostly read and write transfer over the PCI bus can be performed as burst transfers. The length of the burst is determined by the bus master. The target is given the start address and transaction type at the start of the transaction, but is not told the transfer length. As master becomes ready to transfer each data item, it informs the target whether or not it's the last one. The transaction completes when the final data item has been transferred [5].

The maximum theoretical transfer rate of the base configuration is 132Mbytes/sec. Extensions to the base specification can boost this by a factor of four to 528Mbytes/sec. The transfer protocol is optimized around transferring blocks of data. A single transfer is just a block transfer with the length of one. The configuration protocol supports up to 256 devices in the system. The electrical specification emphasize low power use including supports for both 3.3V and 5V signaling environments.

A. PCI features:

- Processor independence
- Supports for up to approximately 80 PCI functions per PCI bus
- Support for up to 256 PCI buses.
- Low- power consumption
- Fast access time
- Bus master supports
- Low pin count
- Software transparency
- Auto configuration

III. Fuse Api Functions And Bus Protocol

Fuse API function enables to receive or send data over PCI interface. These functions follow predefined protocols, i.e., following some hierarchy to perform some operations. These operations are listed below in their predefined order [2].

- Obtain a 'Locate Handle' to the detected hardware in the system using the DIME_LocateCard Function.
- Obtain a 'Card Handle' for a specific card in the system using the DIME_OpenCard function. Use the 'Locate Handle' as a parameter.
- Use the Card Handle with the functions to control the resets, clocks and configure the FPGAs.
- Obtain a handle for talking to the vidimeinterface. To do this use the 'viDIME_Open' function. This function

locks down required memory and other initialization functions for the other functions in the vidime.h library.
- Use this handle to the vidime interface in the other functions such as 'viDIME_DataWriteSingle' and 'viDIME_DataReadSingle'.
- Close the handle to the vidime interface using viDIME_Close to free the allocated memory.
- Close the card handle and locate handles using the 'DIME_CloseCard' and 'DIME_CloseLocate' functions.

A. Communication Bus Protocol:

Data is transferred between the User FPGA and the Interface FPGA using 5 control signals. Three of these signals are driven by the PCI interface. These are AS/DSL, EMPTY and BUSY [3].

B. AS/DSI:

This is an address strobe/data strobe signal. When this is HIGH, the data transferred to the User FPGA is an address. When this signal is LOW, the data transferred to the User FPGA is data from or to the last address given. Generally, addresses are only sent from the PCI FPGA to the User FPGA. This signal is always in sync with the data passed through the internal FIFOs of the PCI FPGA. The internal FIFO can have a mixture of data and addresses and this AS/DS# line will automatically indicate the true type of data.

C. EMPTY:

This signal indicates that there is data waiting to be written from the PCI FPGA interface to the User FPGA. This signal will go HIGH when there is no more data to be written to the User FPGA.

D. BUSY:

This signal indicates that the PCI FPGA interface can receive data from the User FPGA. When this signal goes HIGH, no more data should be written to the Interface FPGA. The User FPGA drives the two remaining control signals. These are RDL_WR and RENL_WENL.

E. RENI_WENI:

This signal is a read/write enable signal. When this signal is LOW, if the R#/W signal is HIGH, data is on the bus ready to be written to the PCI FPGA. If the signal is LOW and R#/W is LOW, then data will be driven onto the data bus from the PCI FPGA on the next clock edge. When this signal is HIGH, there should be no data on the bus.

F. ADIO:

This is the data bus that is used to transfer data between the PCI FPGA and the User FPGA. It is a 32-bit bidirectional bus that can be driven by both the PCI FPGA and the User FPGA. The general functionality is similar to that of FIFOs. The EMPTY and BUSY signals act similarly to FIFO_EMPTY and FIFO_FULL signals. The RDl_WR and RENl_WENl signals combine to give the RENl and WENl signals of a FIFO.

G. RDl_WR:

This signal determines the direction of the data transferred between the Interface FPGA and the User FPGA. If this signal is LOW, data is being read from the Interface FPGA and so the Interface FPGA drives the data bus. If the signal is HIGH, data is being written to the Interface FPGA and so the User FPGA drives the data bus.

The clock used for the Interface FPGA to User FPGA communications is always DSP clk.

An easy interface which allows data communication between the user FPGA (VIRTEX IV) and user application (PC) is established by PCI Interface (Spartan 2 signals). The core part has the main control state machine which controls force between the two devices. When operating a register write mode following is the sequences:
- From idle, read data from the PCI interface.
- Data should be an address (AS/DS# high)
- Determine from MSB of address whether the register access is a read or a Write (MSB= '0' – Write, MSB= '1' – Read)
- Read data from PCI interface
- Write data onto data bus along with write enable.

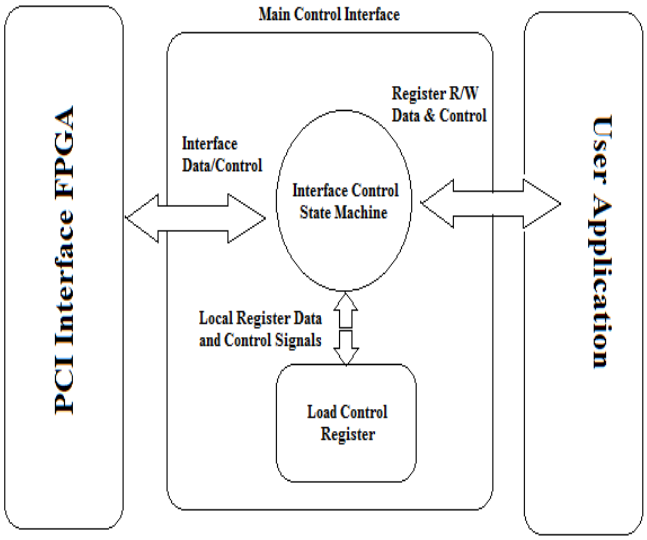

Fig.1: Functional block diagram of PCI communication Interface.

H. Register Signals:

The four register signals listed below are used during the communication mechanism. Address signal is of 31 bit address and MSB of this address signal indicates whether read or write operation is to be performed, i.e. either by indicating '0' or '1' in the MSB, while read and write strobe acts as an enable signal to perform read and write operation [4].

Table 1: Register Signals

Signal	Dir.	Description
RD_STRB	O	Read strobe.
WR_STRB	O	Write strobe.
DATA	I/O	32-bit data bus.
ADDRESS	O	31-bit address.

I. Spartan Signals:

The spartan signals listed below helps in establishing communication mechanism between the user FPGA and user application. The signal with their description along with the direction of communication are given below [4].

Table 2: Spartan Signals

Signal	Dir.	Description
EMPTY	I	Indicates if Spartan has data to transmit.
BUSY	I	Indicates if Spartan can receive data.
AS/DS#	I	Determines if data input is address or data.
RST#	I	System reset.
RD#/WR	O	Indicates if interface is writing/reading Spartan.
REN#/WEN#	O	Enables read/write operation.
INT#	O	Interrupt signal.
ADIO	I/O	32-bit data bus.

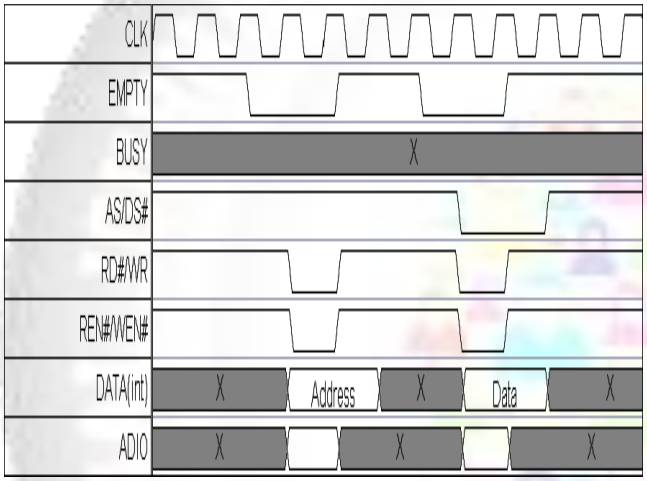

Fig. 2: Timing Diagram for Register Read

The timing diagram for register read is shown above in figure 2. Reading from the Interface FPGA is similar to reading from a FIFO. The EMPTY signal goes LOW to indicate that here is data to be read. The FIFO, whose data is read from on the Interface FPGA is a First-Word-Fall-Through with a latency of one clock cycle.

When reading from the Interface FPGA the user must ensure that the read enable is not active until at least one clock after the EMPTY signal goes LOW. The read enable should go inactive immediately after the EMPTY signal goes HIGH, although no data will be read if EMPTY is HIGH and the read enable is active [4].

IV. Proposed Algorithm And Results

Communication interface has been developed between FPGA and PC. The available FPGA board already has a PCI bus which is used to program the FPGA as well as send data information to PC. On FPGA six 32 bit registers stores the individual data information (A, B, C, D, E and F). These registers have information about the signal which is to be transmitted using PCI over interface bus. These registers are updated every time PC sends a communication start signal. Once updated these signals are hold constant till the communication has been completed for present set of data. PC sends address of individual registers and data stored at these locations are sent by the FPGA. On PC a C code has been developed in Lab-Windows CVI 2012 for this operation. Data are displayed on a GUI developed in Lab-Windows CVI 2012 (Student version).

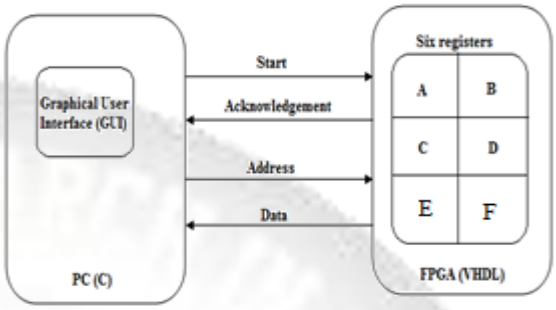

Fig. 3. Design algorithm for communication interface

Peripheral component interconnect communication interface has been developed to provide the facility to transfer information between FPGA and PC, so that information can be observed and optimization can be done to improve the results. It is designed using Fuse API C code functions on the Lab-Windows. The VHDL code on Xilinx ISE 13.1 software has been developed and is communicating with the Lab-Windows using C code at PC end with the help of Spartan control signals. The result is displayed using graphical user interface developed in Lab-Windows. The six signals stored in registers which contains data of 32 bit are displayed on the user interface window using this communication.

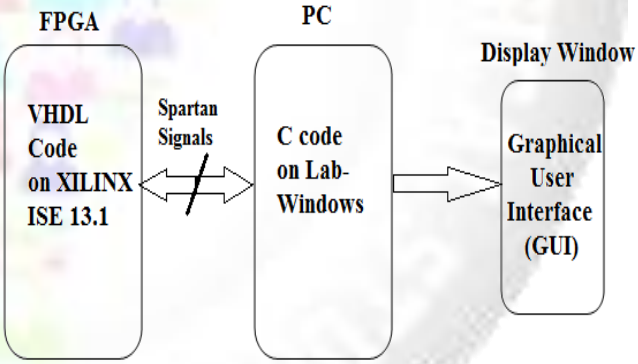

Fig. 4: Block diagram of communication interface using PCI

A. GUI Display:

The graphical user interface (GUI)display the result from the six registers, which provides the information about the data stored in these registers. As seen from the GUI window of these six registers (A, B, C, D, E and F), during the analysis, it was observed that the information of one of the register information throughout the process maintained its constant behavior to provide the correct information of the data hence maintaining stable information throughout the analysis. The GUI display of register A is shown below.

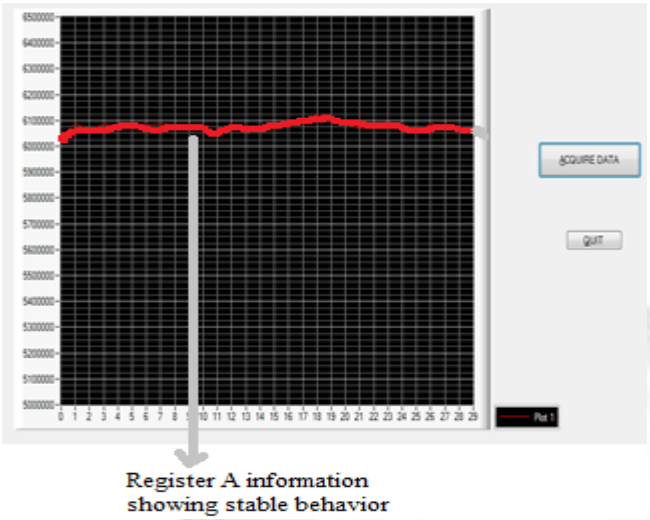

Fig. 5: GUI display of resgiter A

V. CONCLUSION

Communication interface for data transfer using parallel mode has been developed through PCI. The six register which stores data information are communicated by proposed algorithm which is to be observed at the PC end. Hence, to fulfill this condition, PCI interface is developed between the FPGA and PC. Spartan signals are used to control the transfer of information in PCI. At PC end through FUSE API functions (Lab-Windows CVI 2012) and at FPGA end through VHDL (XILINX ISE 13.1), interface has been developed. Better throughput response and speed of data transfer of about 500 kbps has been observed.

REFERENCES

[1] Application Note on "PCI Communications Core" by Nallatech, sep. 2004.
[2] Application Note on "PCI to user FPGA core" by Nallatech, Aug. 2003.
[3] Application Note on "PCI user core" by Nallatech, mar. 2002.
[4] Developer's Guide on "FUSE C/C++ API Guide" by Nallatech, June 2007.
[5] Tom Shanley, Don Anderson, "PCI system architecture", MindShare INC, fourth edition.
[6] Palanivelu , J. Shanmugam, "Design and development of PCM decommutator with PCI interface", 24th digital avionics system conference, October 30,2005.
[7] HosseinKavianipour, "High performance FPGA- based DMA interface for PCIe", IEEE transaction on nuclear science, vol.61, No.2, April 2014.