Design and Verification of FPGA High Speed PCIe Real-Time Data Acquisition System

Miloš Reljin, *RT-RK*, Nebojša U. Pjevalica, *Member, IEEE*, Miloš Subotić, *Student Member, IEEE*

***Abstract*—This paper describes proposed system for real-time data acquisition based on System on Chip with an FPGA and a PC. Communication between the two runs over PCI Express bus. Results show that this architecture enables high speed data acquisition for a reasonable price which makes it commercially viable.**

***Index Terms*—PCI Express, FPGA, Real-time, SoC, data acquisition**

I. Introduction

Core functionality of measurement systems is data acquisition. It represents collecting digital samples of electrical signals produced by various sensors of real-world physical properties such as temperature, force or light intensity. These systems consist of sensors, analog to digital converters, interface for connecting with a PC (USB, parallel port, ISA, PCI, PCIe, RS-232, RS-485) and a PC itself, running data acquisition software.

In this paper, the focus is on real-time data acquisition systems, that is to say, systems where data collection is executed periodically, in precisely defined intervals of time. However, the system described here does contain non real-time component, a PC, since it runs Windows or Linux operating system which does not support real-time. The main idea behind using PC in this kind of system is its already powerful and ever-increasing hardware capabilities combined with affordable price.

Some of the most notable interfaces for connecting peripheral components with a PC are: parallel port, which was used from the 1970s through the 2000s and, in its most advanced version, had a 2.5 MB/s bit rate; RS-232 serial port which was first introduced in 1960 and is still used to this day (although its usage became limited to industrial machines and scientific instruments); USB port, which was introduced in mid-1990s, superseded serial and parallel ports and became industry standard, providing transfer speeds up to 10000 Mbits/s; ISA (IBM PC/AT) bus, created in 1981 by IBM, is parallel bus with 8 or 16 bits width, later superseded by PCI (Peripheral Component Interconnect) bus in 1992 which has 32 or 64 bits width and speeds up to 533 MB/s. Today's PCs use PCI Express bus. It has width from 1 to 32 bits and provides speeds up to 31.51 GB/s (PCIe v4.0 x16).

Motivation for using PCI Express bus and such a large data transfer speeds in data acquisition and measurement systems lies in the fact that some of these systems can output enormous amounts of data in a very short time, for example complex multi-channel data acquisition systems, radar systems or video recording and processing systems.

Considering its high throughput and commercial availability, PCIe was chosen as an interface between a PC and an FPGA (Field-Programmable Gate Array) board, altogether forming a data acquisition and measurement system.

Second chapter describes conceptual advantages of PCIe bus usage in real-time data acquisition systems. Third chapter presents the proposed system. Verification process is shown in chapter four. Achieved data acquisition throughput rates and FPGA resources utilization are summarized in chapter five.

II. PCIe Interface Characteristics

As it was briefly mentioned in introduction, when selecting an interface between a PC and a high speed peripheral, PCIe comes up naturally, considering price to performance ratio.

PCI Express can be viewed as a network rather than a bus. Each card in network has its physical connection to the switch logic like in a local Ethernet network. Communication is done via packets, also including flow control, error detection and retransmissions. There are no MAC addresses but instead a physical location of the card is used to define it, before its address is allocated in the I/O space.

PCI Express uses four types of transactions [1]: memory read/write (data transfers to or from memory mapped locations), I/O read/write (data transfers to or from I/O locations), configuration read/write (discovering device capabilities) and messages (event signaling and general purpose messaging).

Two types of interrupts are supported: legacy INTx and MSI (Message Signaled Interrupts). INTx interrupts are supported to allow bridging between PCI bus and PCI Express. Since INTx had problems like interrupt sharing and the need for each interrupt handling routine to check who

Miloš Reljin is with RT-RK Institute for Computer Based Systems, Narodnog Fronta 23a, 21000 Novi Sad, Serbia (email: milos.reljin@rt-rk.com).

Nebojša U. Pjevalica is with the University of Novi Sad, Faculty of Technical Sciences, Computing and Control Engineering dept., Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (phone: 381-21-4801298; email: pjeva@uns.ac.rs)

Miloš Subotić is with the University of Novi Sad, Faculty of Technical Sciences, Computing and Control Engineering dept., Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia and with RT-RK Institute for Computer Based Systems, Narodnog Fronta 23a, 21000 Novi Sad, Serbia (phone: 381-21-4801291; e-mail: milos.subotic@rt-rk.uns.ac.rs).

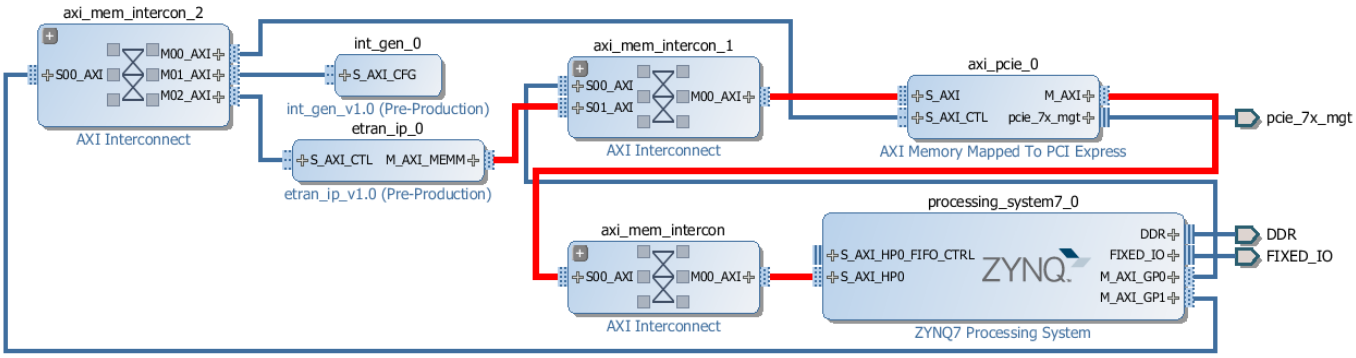

Figure 1 - Vivado Block Design

should actually handle the interrupt, MSI was introduced. Here, interrupts are signaled by simply issuing a write request to a specific address defined by the host inside peripheral's configuration space. The data value written this way indicates which interrupt vector is asserting.

III. Architecture of Proposed PCIe Acquisition System

One of the key system components is Xilinx Zynq-7000 SoC (System on Chip) which features dual-core ARM Cortex-A9 processor and Kintex-7 FPGA. It is located on PCIe card developed by RT-RK institute and it's connected to onboard DDR3 RAM module as well as peripherals like Ethernet port, UART and video inputs. Xilinx provides a variety of IP (Intellectual Property) cores for FPGA design that can be easily instantiated and combined with user defined logic. One of the provided IP cores greatly simplifies PCI Express communication by implementing complete protocol handling. User interaction is reduced to issuing addresses and providing data over AXI-Full bus [3].

As shown in Figure 1, system block design consists of *Zynq7 PS (Processing System)*, *AXI Memory Mapped To PCI Express* and *AXI Interconnect* IP cores provided by Xilinx, and user defined IP cores *int_gen* and *etran_ip*. ARM processor and DDR3 memory controller reside inside *Zynq7 PS*. *AXI Memory Mapped To PCI Express* is the IP core mentioned before which implements PCIe protocol handling. *AXI Interconnect* IPs are instantiated automatically by Xilinx Vivado tool and their purpose is routing of AXI-Full buses between components. User defined IP *int_gen* is used for triggering message signaled interrupts to a PC while *etran_ip* generates test data and passes it to *AXI Memory Mapped To PCI Express* IP.

User IP cores and *AXI Memory Mapped To PCI Express* are connected to the general purpose port 1 (*M_AXI_GP1*) of the *Zynq7 PS* using AXI-Lite bus. This connection enables ARM processor to see those components as memory-mapped devices. *AXI Memory Mapped To PCI Express* (*S_AXI* port) is additionally connected to the general purpose port 0 (*M_AXI_GP0*) of the *Zynq7 Processing System* which makes a path for processor to read and write data to the PCI Express bus. One last connection is from master bridge (*M_AXI* port) of *AXI Memory Mapped To PCI Express* to the *S_AXI_HP0* (high performance) port of *Zynq7 PS*.

Vivado VHDL synthesis report of proposed block design shows low resource usage which leaves space for implementing much more complex logic design. This fact undoubtedly shows that it's possible to include the logic for processing acquired data on FPGA before passing it to a PC via PCI Express if needed [6][7]. Resource usage report is shown in table 1.

Resource	Utilization	Available	Utilization %
FF	15000	157200	9.54
LUT	18288	78600	23.19
Memory LUT	448	26600	1.68
BRAM	19	256	7.17

Table 1 – FPGA resource usage

Table legend:

FF – Flip-Flop
LUT – Lookup Table
Memory LUT – Memory Lookup Table
BRAM – Block RAM

IV. Verification of Proposed System

Due to open source nature of GNU/Linux operating system, it was selected for the PC host OS. It's important to note that this fact does not make the proposed system less portable since it's possible to implement the PC side software relatively symmetrically on Microsoft Windows OS.

Verification process is designed with the idea to use simple counter data sequence which is generated on Zynq SoC and passed to a PC. Communication with PCI Express card on the PC side is handled from Linux kernel device driver written by the authors of this paper.

Upon loading the driver, it connects to the PCIe card by using function *pci_get_device* provided with card specific vendor and device identification numbers [4]. After enabling the card, the driver gets the physical address of PCIe BAR0 (Base Address Register) using *pci_resource_start* function [8]. Then it assigns virtual address to the BAR by calling *check_mem_region*, *request_mem_region* and *io_remap_nocache* functions. BAR space length is obtained via *pci_resource_len*. At this point, the driver enables message signaled interrupts by calling *pci_enable_msi*

function and registers its interrupt handler routine with *request_irq*. The only thing left to do in initialization is to allocate a contiguous block of physical memory by calling *pci_alloc_consistent* and fill it with zeroes.

After the driver initialization is finished, test application can issue IOCTL (I/O control) message to the driver which represents a request for transfer start. When driver receives the IOCTL message, it writes the physical address of allocated buffer at the beginning of the memory that BAR0 is pointing to. This space is actually inside RAM on the PCIe card and is used as a configuration space for passing transfer parameters. After writing the address, driver sets a flag in the next location in configuration space which signals the request for transfer start to the PCIe card and waits for interrupt from card. This way, IOCTL command for transfer is implemented in a blocking fashion.

On PCI Express card, ARM processor polls the location with the mentioned flag in on-board RAM. When it detects that a PC changed this location, it first resets the location's value and then sends a command via AXI-Lite bus to the *etran_ip* module and polls it for transfer completion. This module in turn generates test counter sequence and sends it over AXI-Full bus to *AXI Memory Mapped To PCI Express* IP core in bursts. The data is then written over PCI Express bus into the allocated buffer inside PC RAM. The *etran_ip* then signals end of transfer by setting a flag it one of its configuration registers. ARM processor detects this and sends command via AXI-Lite bus to *int_gen* IP core which triggers message signaled interrupt. This is done by sending a pulse on one of the *AXI Memory Mapped To PCI Express* IP core ports.

Back on the PC side, registered interrupt handler routine is called. Interrupt handler copies the received data from kernel buffer to the buffer in user-space application which originally issued request and unblocks IOCTL afterwards. Code execution flow returns to the application and the transfer is complete. Application is then able to test validity of the received data by comparing it to the predefined sequence.

Operation of described system is successfully verified in the sense that the data from the source reaches the destination unaltered. However, real-time validity criterion is not satisfied. This conclusion rises from the fact that software on the PC side is running on GNU/Linux which is not a RTOS (real-time operating system). Further work on proposed system would include migrating PC software to a RTOS.

Another aspect of verification process was measurement of the achieved data throughput rates. Measurement was done by simply calling *gettimeofday* function in user-space application before issuing the IOCTL to the driver and after the IOCTL completes and then calculating the difference. Results revealed that, on average, it takes 7.3 ms for transfer to complete. Given that the buffer was 4MB large, achieved throughput rate is 548MB/s. Testing was repeated without copying data from kernel-space to the user-space buffer and result was 6 ms. This means that, for each 4MB block of data, 1.3 milliseconds are reserved for copying it from kernel to user buffer. The last test used two kernel-space buffers, each 4MB in size, to implement double buffering. Now the copying from kernel-space to user-space was done in background, while the data was being transferred from the PCIe card to second kernel-space buffer. For 500MB transfer size, throughput rate raised up to 680MB/s [2]. Still, 20% of CPU time is reserved for copying the data from kernel-space to user-space.

V. Conclusion

Proposed system was successfully designed and verified, providing data throughput rate of 548MB/s or 680MB/s, depending on whether the double buffering in Linux device driver was used. VHDL synthesis showed low FPGA logic resources usage which left space for implementing much more complex logic for processing data upon acquisition. The system is not working in real-time so far, but that is possible to accomplish when using RTOS on PC side. In conclusion, it is safe to say that modern SoCs provide a great amount of processing power and speed for a reasonable price which makes them adequate for applications in complex data measurement systems.

Acknowledgment

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number: TR32029.

References

[1] Ravi Burduk, Don Anderson, Tom Shanley, *PCI Express System Architecture*, Boston, USA: Addison-Wesley, 2008.

[2] Xilinx, *Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions, v3.3*, 2015.

[3] Xilinx, *AXI Memory Mapped To PCI Express Gen2 LogiCORE IP Product Guide, v2.6, 2015.*

[4] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, *Linux Device Drivers*, 3rd Edition, Sebastopol, CA, USA: O'Reilly Media Inc., 2005.

[5] Hossein Kavianipour, Christian Bohm, "A high-reliability PCIe communication system for small FPGAs", IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Seoul, South Korea, vol. 27, pp. 1-4, Oct 2013.

[6] T. Bergmann, D. Bormann, M. A. Howe, M. Kleifges, A. Kopmann, N. Kunka, A. Menshikov, D. Tcherniakhovski, "FPGA-based multi-channel DAQ systems with external PCI Express link to GPU compute servers", IEEE-NPSS 18th Real Time Conference (RT), Berkley, CA, USA, pp. 1-5, June 2012.

[7] LiJie Chen, WeiChao Zhou, QingZhang Wu, "A design of high-speed image acquisition card based on PCI Express", IEEE International Conference on Computer Application and System Modeling, Taiyuan, China, pp. 1-4, Oct 2010.

[8] Hai-quan Cheg, Jun Hu, Shu-yan Xu, "Research into PCI Express device's configuration space on PC platform", IEEE International Conference on Computer, Mechatronics, Control and Electrical Engineering, Changchun, China, pp. 1-4, Aug 2010.