Implementation of High-throughput PCIe based on FPGA and PowerPC

Kun Cheng
Department of Modern Physics and Hefei National Laboratory for Physical Sciences at Microscale
University of Science and Technology of China
Hefei, China
e-mail: chengkun@live.cn

Shengkai Liao, Qi Shen, Chengzhi Peng
The Chinese Academy of Sciences (CAS) Center for Excellence and Synergetic Innovation Center in Quantum Information and Quantum Physics
University of Science and Technology of China
Shanghai, China

***Abstract*—This paper designed and implemented a direct memory access (DMA) architecture of PCI-Express (PCIe) between Xilinx field programmable gate array (FPGA) and Freescale PowerPC. The DMA architecture based on FPGA is compatible with the Xilinx PCIe core while the DMA architecture based on POWERPC is compatible with VxBus of VxWorks. The solutions provide a high-performance and low-occupancy alternative to commercial products. In order to maximize the PCIe throughput while minimizing the FPGA resources utilization, a novel strategy for the DMA engine is adopted, where the DMA register list is stored not only inside the FPGA during initialization phase but also in the central memory of the host CPU. The FPGA design package is complemented with simple register access to control the DMA engine by a VxWorks driver. The design is compatible with Xilinx FPGA Kintex Ultrascale Family, and operates with the Xilinx PCIe endpoint Generation 1 with lane configurations x8. A data throughput of more than 666 MBytes/s (memory write with data from FPGA to PowerPC) has been achieved with the single PCIe Gen1 x8 lanes endpoint of this design.**

Keywords—PCIe, FPGA, PowerPC, DMA

I. INTRODUCTION

In lots of modern large-scale scientific experimental facilities, such as Shanghai Synchrotron Radiation Facility (SSRF) and Shanghai Deep Ultra Violation Free Electric Laser (SDUV-FEL), there is a strong demand of real-time online transmission and processing of a large amount of data [1]. These scientific systems should contain data-acquiring and data-processing subsystems with a Gigabyte-per-second data transmission rate in-between. Depending on the performance specifications of the particular application, FPGAs with unique parallel processing and good timing control characteristics are usually used to acquire data from experiments, and PowerPCs with powerful computational capacity are sometimes adopted as host embedded systems to process data online. Consequently, the bottleneck of data processing rate between the subsystems usually lies in the data transmission link between FPGA and PowerPC. In this paper, FPGA needs to send data to PowerPC at a rate of no less than 500 MBps. To achieve this goal, we have developed a PCI Express (PCIe) as a data link.

PCIe is a widely-used reliable high-speed data transmission protocol. Both FPGA and PowerPC support basic PCIe device. Xilinx provides FPGA (XCKU040) with PCIe IP core (Gen3) [2]. Freescale provides PowerPC (MPC8641D) with PCIe integrated as a peripheral device [3]. A board support package (BSP) compatible with PowerPC is supplied by WindRiver VxWorks. To achieve the highest data throughput while reducing the occupancy of PowerPC, DMA engine is required to overcome the limitations caused by the PowerPC scheduling of the operating system (OS). Previous works in Refs. [4], [5],[6] has implemented DMA architectures for PCIe core based on FPGA of Xilinx, but subjecting to a limitation of a single FPGA device with a high cost.

To overcome these drawbacks, we designed and implemented a high-performance and compact DMA engine architecture which is fully compatible with Xilinx FPGA Ultrascale family, together with a custom-designed VxWorks driver based on VxBus. Two development kit boards are adopted in this work: KCU105 is for XCKU040-FPGA, and HPCN8641D is for MPC8641D-PowerPC. This paper illustrates the DMA engine architecture in FPGA, the VxWorks driver in PowerPC, and the handshaking sequence between FPGA and PowerPC. Different payload length transmissions are discussed in the end.

II. SYSTEM OVERVIEW OF POINT-TO-POINT PCIE

PCIe is a layered protocol, containing a transaction layer, a data link layer, and a physical layer [7]. The structures of these layers for two different PCIe devices are illustrated in Fig. 1 [8].

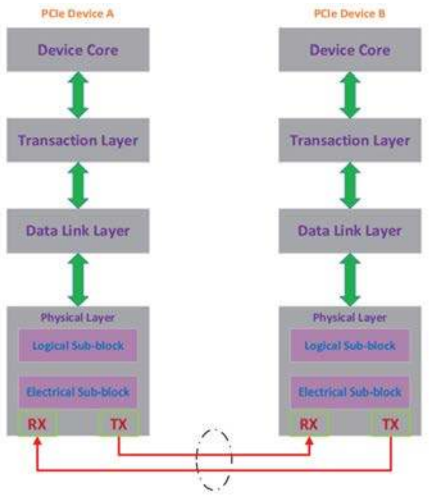

Figure 1. PCIe layer.

The data link (DL) layer is subdivided to include a media access control (MAC) sublayer. The physical layer is subdivided into logical and electrical sublayers. The logical sublayer contains a physical coding sublayer (PCS). The PCIe IP core of Xilinx comprises the DL layer and physical layer, offering an interface which complies with the bus of AXI-4 [9] of transaction layer. It is convenient for users to consider only the transaction layer protocol (TLP) logic to achieve PCIe transmission and message signaled interrupts (MSI). The PCIe is a peripheral of the PowerPC, and the BSP of the PowerPC contains a basic driver of PCIe. Users will need to develop a compatible driver complied with VxBus for PCIe [10].

There are three kinds of memory spaces in PCIe: memory space, configuration space, and I/O space. In this work, we mainly develop PCIe with memory space. The configuration space contains the PCIe configuration registers, each including 6 base address registers (BAR 0-5). In this work, only BAR0 is adopted, with the memory size configured as 2K bytes. The version of PCIe IP core is Gen3 in XCKU040 and Gen1 in MPC8641D. In order to match the FPGA and PowerPC the PCIe IP core in FPGA is configured as Gen1 (since the PCIe is downward compatible).

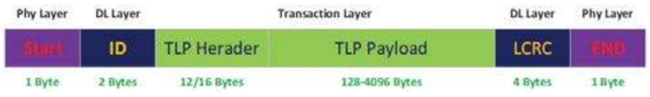

Figure 2. PCIe protocol.

PCIe Gen1 offers a data link operating at 2.5 Gbps in each lane and uses 8B/10B encoding. Therefore, the actual maximum throughput of each its lane is 2 Gbps. Additional packet overhead, except the basic payloads, are illustrated in Fig. 2. The maximum theoretical throughput V for PCIe Gen1 can be calculated as Equation (1) [8].

$$V = \frac{P_l}{P_l + H_{\text{pcie}}} N \times 250 \,\text{MBps}. \tag{1}$$

where P_l is the maximum payload size, H_{pcie} is the protocol header, and N is the number of lanes.

III. PCIe Based on FPGA

A. A High-Throughput DMA Architecture for PCIe Application

The DMA engine designed in this work adopts a stream mode in order to maximize the data throughput and minimize the FPGA resource utilization. The complete architecture of PCIe-DMA is shown in Fig. 3.

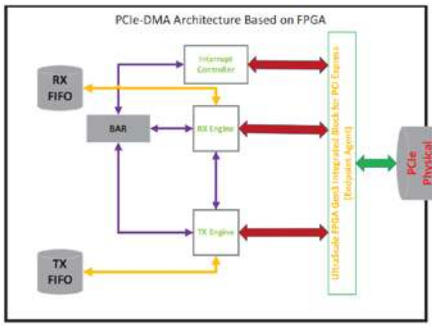
PCIe-DMA Architecture Based on FPGA
RX FIFO
BAR
Interrupt Controller
RX Engine
TX Engine
TX FIFO
UltraScale FPGA Gen3 Integrated Block for PCI Express (Endpoint Agent)
PCIe Physical Interface

Figure 3. Architecture of the DMA engine.

FPGA is configured as the PCIe bus master to start memory write (MWR) and memory read (MRD) to PowerPC. The DMA uses TX engine to transmit data to host and RX engine to receive data from host. The PCIe core of XCKU040 offers an advanced extensible interface (AXI4). The width of AXI-4 interface is mainly configured as 128 bits. The AXI-4 contains four groups interface shown in Figure 3.

1) Completer request interface (CRI): User application receives completers from host via this interface group.

2) Completer completion interface (CCI): User application replies the requester from host and delivers each TLP on this interface.

3) Requester request interface (RRI): User application delivers each TLP of requester as an AXI4-stream packet.

4) Requester completion interface (RCI): Host replies user requester and delivers the TLP to FPGA via this interface.

User logic of TX engine is complied with RRI and CCI of PCIe core in AXI4-stream slave mode, while the RX engine is complied with CRI and RCI. TX engine then sends MWR and MRD TLP via RRI, and RX engine subsequently receives requester completion information for MRD by RCI. Xilinx offers a basic PCIe communication example which implements MWR and MRD processing launched by host. It is worth noting that MWR and MRD are launched by FPGA. The user clock in FPGA is configured as 125 MHz and the IP core is supplied with a reference clock 100 MHz from standard PCIe slot which is mounted on HPCN8641D.

B. Base Address Register

BAR0 in FPGA is implemented as a list of registers which are adopted as handshaking controlling. The width of each register is 32 bits according to the RISC width of PowerPC. The registers contain the following functions.

- Initialization flag.
- MWR and MRD start flag.
- MWR and MRD interrupt processing flag: It indicates that the PCIe in FPGA generates a MSI and is waiting for the PowerPC finishing processing the interrupt.
- MWR address, payload length and MWR times.
- MRD address, payload length and MRD times.
- DMA MWR and MRD performance flag: It is used to indicate the current throughput of PCIe.

C. TX Engine

A finite state machine (FSM) diagram of the MWR DMA engine is shown in Fig. 4 and is described below, together with the handshaking sequence with the VxWorks driver.

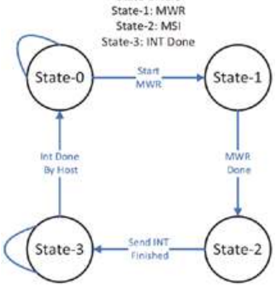

Figure 4. FSM of MWR DMA-PCIe in FPGA.

1) Initialization phase (state 0): PowerPC sets the register values including times of memory write (MWR), payload length and initial address in BAR0 space.

2) After the initialization phase, the FSM waits in idle (state 0) for starting MWR (state 1).

a) FPGA loads the payload length and address to which it will send the data.

b) FPGA checks the MWR times and updates the WR pointer with the last loaded address.

3) If FPGA has already prepared the data for once DMA-TX, then it starts moving data from FPGA to PowerPC (state 2).

4) After finishing MWR, FPGA generates an MSI (state 2).

5) If MSI is sent to IP core, then FPGA will wait for the "int processed done" signal from host (state 3).

6) CPU deals with the MSI and returns "interrupt done" to FPGA, then FPGA returns to idle (state 0).

We denote the DMA MWR payload length (DWORDs number) with PL_{mwr}, and DMA MWR times with N_{mwr}. It takes one clock for FPGA to request MWR once, and the rest of clocks are for valid data. The clock of user logic is 125 MHz. Then the theoretical efficiency F_{mwr} and speed V_{mwr} of DMA, respectively, are shown in Equation (2a) and (2b).

$$F_{\text{mwr}} = \frac{PL_{\text{mwr}}}{PL_{\text{mwr}}+1}, \tag{2a}$$

$$V_{\text{mwr}} = \frac{PL_{\text{mwr}} \times 4}{(PL_{\text{mwr}}+1) \times 8} \text{ GBps.} \tag{2b}$$

In our work, we count the real clock consumption as the following: when "mwrstartsig" is valid, the counter will start to count. When "mwrdone", indicating the termination of DMA-MWR, is valid, the counter stops counting. Each clock denotes 8 ns, so it takes 8 × $\text{counter}_{\text{value}}$ ns [[it takes ... to ..., add to ...]]. The actual MWR speed can be calculated as Equation (3).

$$V_{\text{mwr,real}} = \frac{D_{\text{mwr}}}{8 \times \text{counter}_{\text{value}}} \text{ GBps.} \tag{3}$$

where D_{mwr} (bytes) is the data size.

D. RX Engine

An FSM diagram of the memory read (MRD) DMA engine is illustrated in Fig. 5. The details, together with the handshaking sequence with the VxWorks driver, are described.

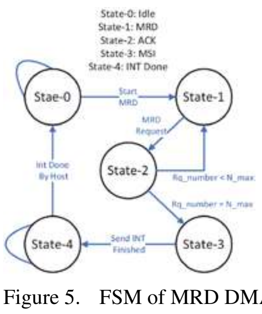

Figure 5. FSM of MRD DMA-PCIe in FPGA.

1) Initialization phase (state 0): PowerPC sets the register values including times of MRD, payload length and initial address in BAR0 space.

2) After the initialization phase, PowerPC prepares the data which will be moved from PowerPC to FPGA. Once the data is ready, PowerPC sends a MRD command (state 0).

3) If FPGA receives the MRD command, it starts the MRD (state 1).

a) FPGA loads the payload length and initial address.

b) FPGA sends an MRD TLP to PowerPC.

4) FPGA waits for the ack from host, updates the RD pointer with the last loaded address and checks whether it is the last transmission of a MRD.

5) After FPGA receives all the data from PowerPC, it generates a MSI (state 3) and waits for the MRD stop command from VxWorks (state 2).

6) If MSI is sent to IP core, then FPGA waits for the "int processed done" signal from host (state 4).

7) After FPGA receives the stop command, it returns to idle (state 0).

We denote the DMA MRD payload length (DWORDs number) with PL_{mrd} and DMA MRD times with N_{mrd}. It takes one clock for FPGA to request MRD once. Then FPGA will receive the completion packet from PowerPC, and the completion TLP contains one clock header for user logic to analyze. The rest of the closes are for valid payloads. The clock of user logic is 125 MHz. Then the theoretical efficiency F_{mrd} and speed V_{mrd} of DMA, respectively, are shown in Equation (4a) and (4b).

$$F_{\text{mrd}} = \frac{PL_{\text{mrd}}}{PL_{\text{mrd}}+1}, \tag{4a}$$

$$V_{\text{mrd}} = \frac{PL_{\text{mrd}} \times 4}{(PL_{\text{mrd}}+1) \times 8}. \tag{4b}$$

In our implementation, we count the real clock consumption as the following: when the TLP in RX is valid, the counter starts counting, until "mrddone" which indicates the termination of DMA-MRD becomes valid. Each clock denotes 8 ns, so it takes 8 × $\text{counter}_{\text{value}}$ ns [[takes ... to do what?]]. If we denote the data size with D_{mrd} bytes, the actual MRD speed can be calculated as Equation (5).

$$V_{\text{mrd,real}} = \frac{D_{\text{mrd}}}{8 \times \text{counter}_{\text{value}}} \text{ GBps.} \tag{5}$$

E. Interrupt Controller

It is convenient to generate an interrupt of MSI with Xilinx PCIe core by producing corresponding signals to the IP core.When the IP core feeds back a sent signal, it means the IP core has already successfully sent a MSI to PowerPC. Here, finishing both DMA-MWR and DMA-MRD will generate MSI, and PowerPC will poll the MSI register which illustrates the current interrupt type in BAR0 space. The actual MSI timing diagram acquired by ILA is shown in Fig. 6, which is the same as in [2].

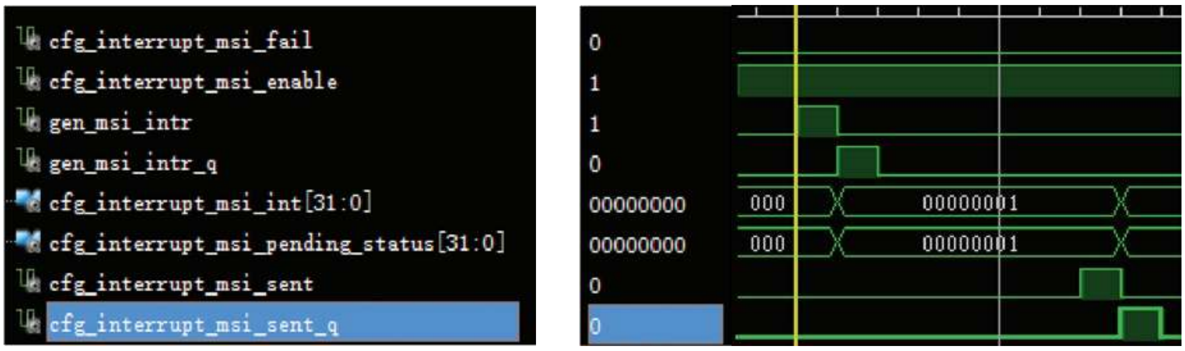

Figure 6. MSI timing diagram (AXI4-128) in FPGA.PCIe Based on Power PC

The BSP in VxWorks contains a kernel, an I/O system, a file system and a network support. WindRiver develops a brand-new architecture called VxBus for driver development since VER 6.6. The relationship between VxBus and VxWorks is shown in Fig. 7.

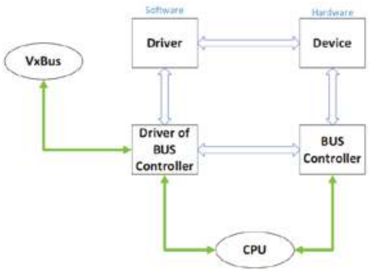

Figure 7. Diagram of VxBus in VxWorks [10].

VxWorks maintains two linked lists: driver list and devices list. After power is on, the initialization will match the drivers and devices. If the two are matched, the driver and device will combine into an instance. This instance is connected to an instance list which can be called by user application. Generally, user needs to register the custom-designed driver to VxBus via "vxbDevRegister ()" function. Then "pcieInstInit ()", "pcieInstInit2 ()", and "pcieInstConnect ()" functions will be called to initialize PCIe device. After that, "pcieDMAInt ()" is defined as the interrupt service function and connected to the list of interrupt list.

The control registers that used as handshaking with FPGA is listed as a head file in the driver. The registers are the same as registers in BAR0 of PCIe in FPGA. The buffers for TX and RX are both allocated for 2 Mb. The driver also realizes MWR and MRD with payload length of double words (DWORD) to handshake with FPGA via PCIe.

The interrupt service routine (ISR) in this work deals with interrupt as follows.

- After initialization, the ISR waits for coming of interrupt from MSI via PCIe.
- If the interrupt is generated, ISR first reads the interrupt status register in FPGA, and then checks whether it is MWR or MRD interrupt.
- If it is DMA MWR rendering the interrupt, PowerPC will move the data from the buffer in host to another address, and process these data. After that, the PowerPC will send a "MWR ISR done" signal to FPGA. If the FPGA receives this signal, it will permit the next DMA MWR.
- If it is DMA MRD rendering the interrupt, PowerPC will prepare new data to the MRD buffer for the next transmission and send an "ISR done" signal to FPGA. If the FPGA receives this signal, it will permit the next DMA MRD.

IV. Test Result

The performance of the DMA engine has been measured using a Xilinx PCIe Gen1 core with different configurations. A KCU105 Kintex Ultrascale-PCIe board mounting on a HPCN8641D was used for the measurements with the Gen1x8 lanes endpoints. The KCU105 contains a Xilinx XCKU040 device and HPCN8641D contains a Freescale MPC8641D device. The FPGA is plugged into the PCIe slot of the HPC□N8641D. BSP is an original driver supplied by VxWorks and it is packaged as VxBus. User driver application is designed according to VxBus. The system architecture is shown in Fig. 8.

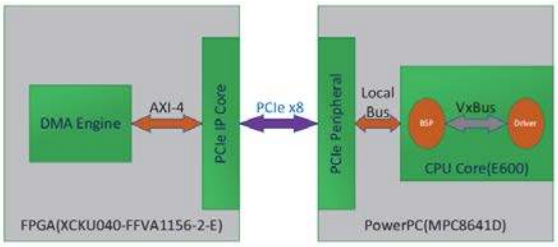

Figure 8. System overview.

The measurements don't take into account the initialization phase: the driver is loaded by the OS and registers are written into FPGA BAR0. The memory addressing is a 32-bit.

FPGA issues an MWR with a constant payload length of 16 bytes to PowerPC. There is one dummy clock beat between each MWR TLP. The measurements of this condition are issued in Fig. 9.

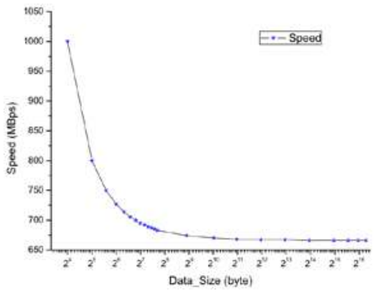

Figure 9. MWR throughput vs. Data size.

We also issued a single MRD TLP from FPGA to PowerPC with different payload lengths. The result for receiving a single MRD completion TLP from PowerPC is shown in Fig. 10.

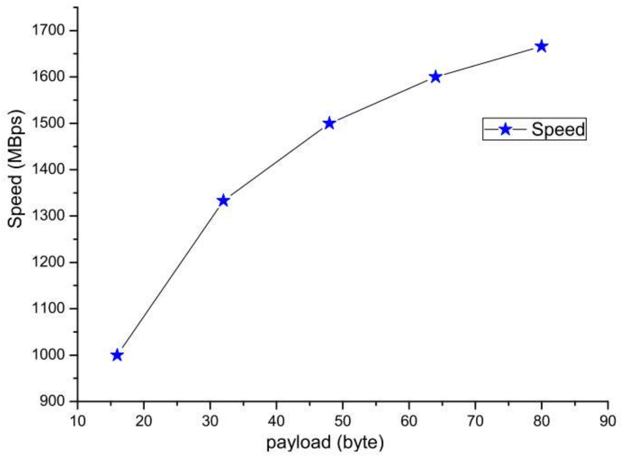

Figure 10. Single MRD completion TLP with different payload size.

The architecture of DMA-PCIe in this paper is lightweight. The resource requirements of this architecture are listed in Table I.

TABLE I. RESOURCE CONSUMPTION ON XCKU040.

Resource	Estimation	Available	Utilization
LUT	6689	242420	2.76%
LUTRAM	1174	112800	1.04%
FF	11222	484800	2.31%
BRAM	34.5	600	5.75%

V. CONCLUSION

A high-throughput PCIe with DMA engine based on FPGA and PowerPC is described in this paper, The DMA engine is made compatible with the Xilinx Kintex Ultrascale PCIe Gen1 core, and a special PCIe driver complied with VxBus is implemented in VxWorks 6.6 based on Freescale MPC8641D. We can easily apply this work in real-time data acquiring and processing system.

The DMA architecture achieves a high-throughput more than 666 MBps. This design satisfies our data transmission target. Our efficiency of DMA-MRD is limited by a long delay between the FPGA issuing MRD and PowerPC answering the MRD TLP. As a result, a MWR with payload more than 16 bytes has not been accomplished. A more efficient DMA□MWR will be designed in our future work.

REFERENCES

[1] Yang Guisen, Leng Yongbin, Lai Longwei,YU Luyang, YUAN Renxian,YAN Yingbing, Research of bunch by bunch data acquisition system in SSRF[J]. Nuclear Science, 2013, 36(12).

[2] Xilinx, “UltraScale Devices Gen3 Integrated Block for PCI Express v4.4”,2017 [Online]. Available: https://www.xilinx.com/support/documentation/ip documentation/pcie3 ultrascale/v4 4/pg156-ultrascale-pcie-gen3.pdf.

[3] Freescale Semiconductor, “MPC8641 and MPC8641D Integrated Host Processor Hardware Specifications”, 2014[Online]. https://www.nxp.com/docs/en/data-sheet/MPC8641DEC.pdf.

[4] Xillybus, “An FPGA IP core for easy DMA over PCIE with Windows and Linux”, 2013 [Online]. Available: http://xillibus.com/.

[5] PLDA, “Ezdma2 for Xilinx”, 2014 [Online]. Available: http://new.plda.com/products/fpga-ip/xilinx/fpga-ip-pcie/ezdma2-xilinx .

[6] NorthwestLogic, “Expresso DMA core”, 2014 [Online]. Available: http://nwlogic.com/packetdma/.

[7] Wikipedia, “PCI Express”, 2017 [Online]. Available: https://en.wikipedia.org/wiki/PCI Express.

[8] Rota L, Caselle M, Chilingaryan S, et al. A PCIe DMA Architecture for Multi-Gigabyte Per Second Data Transmission[J]. IEEE Transactions on Nuclear Science, 2015, 62(3):972-976.

[9] Yin L, Xie M, Li H, et al. Design and implementation of the digital radio frequency memory system based on advanced extensible interface 4.0[C]//International Congress on Image and Signal Processing, Biomedical Engineering and Informatics. IEEE, 2017.

[10] WindRiver, ”VxWorks BSP Developer’s Guide”, 2006 [Online]. Avail□able: http://www.windriver.com.