JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1031-1046 (2018)
DOI: 10.6688/JISE.201807_34(4).0014

EPEE 3.0: A Unified Host-FPGA Communication Library Framework for SDR Platforms[*]

ZHI-WEI LI, JUN LIU[+], TAO WANG AND BO-YAN DING
Center for Energy-Efficient Computing and Applications
School of Electronics Engineering and Computer Science
Peking University
Beijing, 100871 P.R. China
E-mail: {zhiwei.li; juneliu; wangtao; dboyan}@pku.edu.cn

Field programmable gate arrays (FPGA) are becoming more and more attractive as accelerator platforms. Many FPGA accelerators use frame-oriented transmission. As there is no standard frame-oriented Host-FPGA communication framework, FPGA accelerator developers have to write a significant amount of code on both the FPGA side and the host side. In this paper, we present a frame-oriented high-performance Host-FPGA communication framework EPEE 3.0 that supports PCIe and USB. It is implementted by using Xilinx Kintex-7 FPGA with PCIe Gen2 X8 and USB 3.0. Evaluation shows that EPEE 3.0 achieves a high throughput and can be easily used by FPGA developers.

Keywords: communication library, frame-oriented, SDR platform, FPGA, PCIe, USB

1. INTRODUCTION

There are many software-defined radio (SDR) platforms for research on wireless communication, such as GNU-radio [1], Warp [2] and GRT [3]. SDR platforms can be divided into two types according to the codec realization of physical layer: software-based and hardware-based. FPGA is a popular platform for hardware-based SDR systems [3, 4]. Apart from software radio functions implemented in hardware, interactions between software on the host and hardware are also needed. In order to support an efficient software-hardware co-design, a communication library between host and FPGA is necessary.

Among various interfaces between host and FPGA accelerator, peripheral component interconnect express (PCIe) and universal serial bus (USB) are two major options. The PCIe interface has high performance and low delay, thus, an ideal choice for many high-throughput accelerators. Meanwhile, the USB interface is more portable. Both PCIe and USB interfaces are widely used [5]. In order to adapt to the different application scenarios for portability and high-performance requirements, we need both PCIe/USB connection and use them in a unified framework.

In our recent work, we proposed in the *Proceedings of International Conference on*

Received August 30, 2017; revised November 7, 2017; accepted December 15, 2017.
Communicated by Xiaohong Jiang.
[+] Co-first author.
[*] The work is supported by National Key Research and Development Plan of China (2017YFB0801702) and National Natural Science Foundation of China (61531004). An earlier version of this paper was presented at the International Conference on Networking and Network Applications, October 16-19, 2017, Kathmandu City, Nepal.

Networking and Network Applications (NaNA2017) a flexible frame-oriented framework called EPEE 3.0 for Host-FPGA communication in a software-defined wireless system [6]. The paper was entitled *A Flexible Frame-oriented Host-FPGA Communication Framework for Software-defined Wireless Network*. Aside from what was mentioned in that paper, we have additional content to present. Therefore, we submit this paper for the following contributions:

(1) Design a unified communication framework for PCIe and USB protocols between Host and FPGA. Based on investigation of typical systems, we identified the requirements for frame-oriented feature and unified interface across different transmission protocols. The framework shields details of the physical bus by providing unified interfaces on both hardware and software so users don't need to care about the underlying aspects.
(2) Implement EPEE 3.0 on multiple generations of Xilinx FPGAs with both PCIe (up to Gen2 X8 mode) and USB 3.0 interfaces and evaluate the performance and resource utilization of EPEE 3.0.

2. RELATED WORKS

2.1 Background on PCIe and USB

PCIe or PCI-E [7] is a high-speed serial host expansion bus standard, which includes a physical layer, a data link layer and a transaction layer. Data is packaged into packets and transferred at the transaction layer as transaction layer packets (TLPs). At the physical layer, PCIe provides a serial high-throughput interconnection between two devices. One or more lanes can be used to transfer data. There have been three versions of the PCIe bus. For a single lane, the one-directional data transfer rates for versions 1.x, 2.x and 3.x are 2, 4 and 8 Gbps, respectively. Usually 'Gen' represents the version of the PCIe protocol, and 'X' represents the number of lanes (*e.g.*, Gen2 X8 with 32Gbps data rate) [17].

FPGA vendors provide PCIe hard IP cores [8, 9] for use in FPGA accelerators. However, the example designs that accompany these cores are too simple to drive user-defined FPGA designs. Developers have to be familiar with the cores' complex interfaces and PCIe transaction protocol to use them effectively.

USB 3.0 [10] is the third major version of the USB standard for interfacing hosts and electronic devices and has the characteristics of compatibility and portability. Among other improvements, it offers the new transfer rate referred to as SuperSpeed USB (SS), that can transfer data at up to 5Gbps (625 MB/s), which is about ten times as fast as the USB 2.0 standard.

CYUSB3KIT-003 USB 3.0 board [11] offers high-speed transfer interface for stream data with up to 3.2Gbps. It can easily achieve the ideal speed, but for wireless communication in frame format, the standard board has much programming work to do. [12] is a USB 3.0 communication library based on CYUSB3014 chip, the maximum speed of which can be up to 2.84Gbps, but the interface is for streaming data, instead of frame data.

2.2 Existing Work

The system-level FPGA driver [14] is an FPGA communication framework based on reusable integration framework for FPGA accelerators (RIFFA) 2.0 [13]. The system adds Ethernet, DDR RAM and PIO transmission support upon RIFFA. However, the system does not support hardware interrupt, and is better suited for transferring large chunks of data between host memory and FPGA's DDR RAM than frame data transfer. The system further encapsulates RIFFA's software-side code to provide more functionality, facilitating user-mode program development to control the accelerator. However, it does not provide support for kernel-mode program development.

Jetstream [15] is an open-source high-performance PCI Express 3 streaming library for FPGA-to-Host and FPGA-to-FPGA communications, achieving throughputs of up to 7.09 GB/s per link. Jetstream is primarily targeted at multi-FPGA solutions and provides high inter-board streaming data communication throughput. Like RIFFA, because of the lack of support for frame data transfer, programming I/O (PIO) and interrupt features, Jetstream has the same problem when used in SDR platforms.

ffLink [16] is a lightweight high-performance open-source PCI Express Gen3 interface for reconfigurable accelerators. ffLink achieves a throughput of up to 7 GB/s, which is 95% of the maximum available throughput of an eight-lane PCIe interface, while requiring just 11% of device area on a mid-size FPGA. Although the work above gets high performance by adopting the Gen 3 version, it still has the limitation when it comes to portability.

Our former work [17] provides an efficient and flexible Host-FPGA PCIe communication library (EPEE 2.0) by building up a highly efficient core layer and making the library extensible. We implemented the EPEE 2.0 on Xilinx FPGA with up to 18.6 Gbps throughput in the PCIe Gen2 X8 mode, which is close to the theoretical maximum throughput. But EPEE 2.0 is only for PCIe interface. It does not support the USB interface.

We conclude from the above that no existing solution provides a unified framework for PCIe and USB 3.0. They are not optimized for frame data communication.

3. DESIGN GOALS AND CHALLENGES

3.1 Design Goals

Aimed to accomplish the high-performance exchange of frame-oriented data between Host and FPGA, the design goals of the EPEE 3.0 are:

To provide the unified high-performance framework for PCIe and USB data transmission between Host and FPGA with minimal resource utilization for frame data.

To provide the unified hardware/software interfaces and interactions, which hide the details of specific communication protocol.

To accomplish the basic frame data transmission requirement between host and FPGA. To provide read, write of the registers and interrupt, so that the accelerator software can acquire FPGA hardware status.

To support flexibility of software and hardware function by providing both user-level and kernel-level interfaces, so that developers can not only utilize accelerators at the user level, but also develop drivers on their own.

A collection of common functional requirements to facilitate the communication between the FPGA and host include the support of PIO, direct memory access (DMA) and interrupt (INTR). PIO is required for the host to read/write device state/control registers, while DMA allows the FPGA board to read/write the host's main memory directly. As a peripheral of the host, INTR supports the FPGA application in notifying the host of its internal events.

3.2 Design Challenge

The frame-oriented Host-FPGA communication framework is a collaboration system of software and hardware components. The framework provides accelerator developers a unified, virtual bus so that they only need to focus on the interaction with the bus and ignore the interactive details between the host and FPGA. However, there are intrinsic differences between PCIe and USB, so how to shield these differences and implement the virtual bus are big challenges. In addition, there are also some technical challenges. For example, USB does not have built-in interrupt and PIO operation support, so effort is needed to emulate the register R/W and interrupt interface on top of USB.

4. DESIGN OF THE FRAMEWORK

This section details the design of EPEE 3.0 framework, mainly targeting the unified frame-oriented frame and interaction design.

4.1 General System Architecture

The frame-oriented Host-FPGA communication framework EPEE 3.0 is a collaboration system of software and hardware. Its general structure is shown in Fig. 1. The entire system can be regarded as a virtual bus, and the connection through PCIe and USB is chosen according to the scenario. The virtual bus provides PIO, DMA and INTR functions on both sides of the bus so developers can combine different functions in different scenarios.

The central part of the framework is the virtual bus for PCIe and USB protocols, offering developers a unified interface and interaction. Thus, developers do not need to care about Host-FPGA communication type and the interaction details.

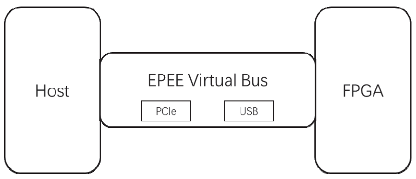
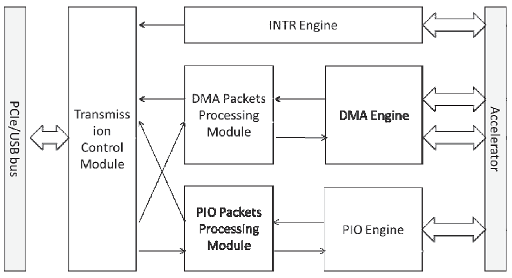


Fig. 1. General architecture for EPEE 3.0.


PCIe/USB bus
Transmission Control Module
INTR Engine
DMA Packets Processing Module
DMA Engine
PIO Packets Processing Module
PIO Engine
Accelerator
Fig. 2. Hardware structure of EPEE 3.0.

4.2 Frame-Oriented Interaction Design

We designed three transmission types, namely DMA, PIO and INTR. Among them, DMA is most closely-related to the frame-oriented design. We will discuss the DMA transmission in detail in this section.

There are three types of frames in DMA transmission: data frame, control descriptor frame and status descriptor frame. The data frame carries the data to be transmitted. The control descriptor frame is used to transmit the configuration information of the communication library and the user-defined program. Data frame processing status and user-defined information are fed back to host via the status descriptor frame.

Both the control descriptor frame and the status descriptor frame are of fixed length. This fixed-length design facilitates the hardware implementation and the length of the descriptor is sufficient to hold the control information and status information that the user needs.

The structure of the control descriptor frame of the Host-to-FPGA direction is shown in Fig. 3. The descriptor is 32DW (Double Word, 1DW = 4Bytes) in length. The first 4 DW is used by the communication framework while the last 28 DW is left for the user. The TP field is 4'b0100 (PCIe) or 4'b0101 (USB). The QN field contains the channel ID of the frame data. FRAME_ADDR and NEXT_DESC_ADDR fields are used for the chained DMA in the PCIe version. Other descriptor frames have similar structures.

The entire Host-to-FPGA interaction process is as follows: the FPGA accelerator software organizes the frame data using the frame descriptor controlled link table, and passes them to the communication framework via software API. The communication framework then puts the frame data and control descriptor into the DMA channel accordingly, and signals the accelerator hardware of the data arrival via hardware interface. The accelerator hardware gets the frame data from the DMA channel, processes the data, and sends the status descriptor back to the software. Hence, the Host-to-FPGA frame data interaction is completed.

63-56	55-48	47-40	39-32	31-24	23-16	15-8	7-0
TP QN	RSVD	FRAME_ADDR (RSVD for USB Version)					
LEN		NEXT_DESC_ADDR (RSVD for USB Version)					
User Defined Field (28DW)							

Fig. 3. Structure of Host-to-FPGA control descriptor frame.

In the process of frame data transmission from FPGA to the computer, the accelerator software prepares data buffers to store the frame data, and submits these buffers to the communication framework via software API. The communication framework then signals the hardware component of new data buffer submission. When the accelerator hardware needs to send frame data to the host, it acquires an idle buffer via the hardware interface according to certain priority sequence, and sends the data frame and status descriptor to the hardware interface of the communication framework. The communication framework then passes the data frame and the descriptor to the host, and signals the accelerator software. This concludes FPGA-to-Host frame data interaction.

In addition, we can use INTR or PIO interfaces to assist in the DMA transfer. For example, hardware can send an interrupt to inform software to process the uplink frames.

4.3 Design of Communication Library Hardware

The structure of the FPGA side is illustrated in Fig. 2. The core layer has two parts: the platform-dependent part which interfaces with the PCIe/USB bus from vendors, and the platform-independent part whose logic is independent from the PCIe/USB protocols. The platform-dependent part is the Transmission Control Module (TCM). The platform-independent part includes the DMA Engine, PIO Engine, DMA Packets Process Module (DPPM), PIO Packets Process Module (PPPM) and INTR Engine.

TCM processes the tasks related to the USB/PCIe protocol. DMA Engine, PIO Engine and INTR Engine are responsible for all hardware interactions. DPPM and PPPM deal with the packets, such as frame packing/unpacking and frame integrity check.

Unlike PCIe, USB does not have built-in interrupt and PIO support. We designed a special packet format to emulate the register R/W and interrupt on USB. The package transfer direction, type, channel number and data length are included in package header. We separate different types of package in the hardware/software part and routed them to appropriate interface. Users do not need to care about the structure of data package (Fig. 3).

4.4 Design of the Interface

(A) Hardware Interfaces

Table 1 shows the main hardware interfaces for EPEE 3.0, which are PIO, DMA and INTR. Since the signals of the DMA FPGA-to-Host (F2H) side is almost the same as those of the DMA Host-to-FPGA (H2F) side, only the H2F part is shown.

The PIO interface provides PIO read and write, or register read and write; both have request and feedback mechanisms in the hardware interaction. We have two main considerations for the design. First, the response delay of different hardware functions differs for PIO requests. The handshake signals make the PIO operations reliable. Second, the design provides better extensibility. Using a PIO bus as interface allows the FPGA developer to allocate and configure the PIO address space freely.

The DMA interface provides DMA transfer between host and FPGA, or bulk data transfer in USB protocol. We use FIFO as the DMA interface for three reasons. First, FIFO is a standard interface and its definition is relatively simple. Experienced developers know FIFO well and beginners can learn to use it quickly. Second, the latency in FIFO is small because data can be read in a few cycles after it is written to FIFO. An alternative is the use of addressable buffers such as RAM, where one has to wait until all the data are stored in the buffer before any data can be read. This will cause a much longer delay, and extra signals are needed to notify the FPGA application when it can start reading the data. Third, FIFO can be used to support other interfaces easily.

The INTR interface provides the unidirectional interrupt from FPGA to host. There are three interrupt types: Host-to-FPGA Interrupt (H2F_INT), FPGA-to-Host Interrupt (F2H_INT) and User-Defined Interrupt (UDF_INT). Host-to-FPGA Interrupt indicates the Host-to-FPGA status descriptor frame. FPGA-to-Host Interrupt is used to inform software of the FPGA-to-Host data frame and status descriptor frame. User-Defined Interrupt is customized by the user for specific needs. The INTR interface also adopts the Req-Ack interaction mechanism like PIO to ensure the proper work and to prevent the loss of continuous interrupts.

A typical Host-to-FPGA transmission used in SDR platforms with the cooperation of the three hardware interfaces is as follows:

(1) Get initialization configuration via PIO interface, such as center frequency, bandwidth and sampling rate.
(2) Wait for the h2f_ready signal to become active.
(3) Check the h2f_indication signal to confirm which channels have data frame requests, and select the highest priority channel.
(4) Give the requested channel number via h2f_qnum signal and raise h2f_req to indicate that a Host-to-FPGA frame is requested.
(5) After the h2f_ack signal is asserted, assign the h2f_req signal to 0 and check whether h2f_err signal is valid. If valid, jump to (2); if not, continue.
(6) Get the control descriptor frame from h2f control descriptor frame FIFO interface.
(7) Fetch data frame from h2f data frame FIFO interface and process the frame.
(8) After the completion of the data frame processing, feed the results back to the software via h2f status descriptor frame FIFO interface.

Table 1. Hardware interfaces.

	Signal	*IO*	*Description*
PIO Read	pio_rd_addr[16:0]	I	Register read address
	pio_rd_data[31:0]	O	Register read data
	pio_rd_req	I	Register read request
	pio_rd_ack	O	Acknowledgement of pio_rd_req
PIO Write	pio_wr_addr[16:0]	I	Register write address
	pio_wr_data[31:0]	I	Register write data
	pio_wr_req	I	Register write request
	pio_wr_ack	O	Acknowledgement of pio_wr_req
INTR	h2f_int_enable	O	Host-to-FPGA interrupt enable
	h2f_int_req	I	Host-to-FPGA interrupt request
	h2f _int_clr	O	Host-to-FPGA interrupt clear
	f2h int signals	I/O	FPGA-to-Host interrupt signals, same as h2f int interface
	udi int signals	I/O	User defined interrupt signals, same as h2f int interface
H2F DMA	clk_usr	I	User clock
	h2f_indication[15:0]	O	Indication of DMA data channel
	h2f_qnum[3:0]	I	Channel number given by user
	h2f_req	I	Data read request
	h2f_ack	O	Acknowledgement of h2f_req
	h2f_ready	O	Ready signal
	h2f_err	O	Error signal
	h2f control descriptor frame FIFO signals	I/O	FIFO signals of control descriptor frame interface
	h2f data frame FIFO signals	I/O	FIFO signals of data frame interface
	h2f status descriptor frame FIFO signals	I/O	FIFO signals of status descriptor frame interface

(9) (optional) Send Host-to-FPGA interrupt to inform software.

(B) Software Interfaces

The software core layer is built to provide high-level, abstracted and easy-to-use APIs for software applications and user drivers. As shown in Fig. 4, the software APIs of EPEE 3.0 include libepee at the operating system user level, and epee_main at the kernel level. The epee_pci and epee_usb modules are in charge of the interactions with the PCIe and USB hardwares, respectively. The epee_main is a module that connects up and down. It links the epee_usb, epee_pci modules, epee_usif and other developer-provided drivers. Because the operating system kernel-level interfaces that epee_main provides are not easy to use for simple accelerators, epee_usif module is provided to translate kernel-level interfaces to user-level interfaces. Meanwhile, EPEE 3.0 provides the libepee module to encapsulate the system calls so that developer interfaces can be provided to achieve both flexibility and high performance.

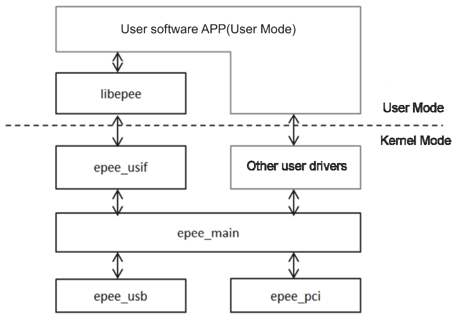


Fig. 4. Software structure of EPEE 3.0.

The design goal of the software API is to provide the functions with the simplest and least number of APIs that shield the details of low-level operations (such as interrupts in DMA operation). Only two APIs are needed for PIO operations, three APIs for DMA operations and one for INTR. The parameters and return values of the APIs are simple. Most application needs can be fulfilled with those three kinds of APIs.

The PIO, DMA, INTR user-mode APIs are listed as follows.

```
unsigned int libepee_read_reg(struct libepee_dev * dev, unsigned int reg);
void libepee_write_reg(struct libepee_dev * dev, unsigned int reg, unsigned int data);
int libepee_submit_frames(struct libepee_dev * dev, struct libepee_h2b_frame_node * frames);
int libepee_get_next_stat_desc(struct libepee_dev * dev, struct libepee_h2b_stat_desc * stat_desc);
int libepee_get_b2h_frame(struct libepee_dev * dev, struct libepee_b2h_frame * frame,
   unsigned int lib_epee_get_dev(int fd, struct libepee_dev ** dev);
int libepee_interrupt_en(struct libepee_dev * dev, unsigned int int_en);
int libepee_block_until_interrupt(struct libepee_dev * dev, enum libepee_int_type type);
```

libepee_read() and *libepee_write()* are functions for PIO. We can also start a DMA transfer to FPGA by calling *libepee_submit_frames()*, and get FPGA-to-Host frames via *libepee_get_next_stat_desc()* and *libepee_get_b2h_frame()*. *libepee_interrupt_en()* is used to enable the interrupt interface in FPGA and *libepee_block_until_interrupt()* can block the user program until the corresponding interrupt arrives.

In particular, after calling the *libepee_block_until_interrupt()*, the user thread will be blocked until the corresponding interrupt occurs. Parameter *int_type* is used to specify the source of the interrupt, which can be selected from the Host-to-FPGA Interrupt, FPGA-to-Host Interrupt, and User-Defined Interrupt. This design is different from the processing model of the interrupt handler in operating systems, which is usually a function registered to the operating system during initialization. When the interrupt is generated, this function will be called once for interrupt processing. However, this approach will force developers to use multi-threaded programming when using interrupt, because the interrupt handler and the main function is not the same thread. This solution will bring complex concepts such as synchronization and mutex, and can cause a lot of trouble for developers. Using *libepee_block_until_interrupt()* to achieve the interrupt interface, even if the developer does not use multi-process or multi-thread programming, they can still process interrupts. If they need to handle an interrupt like interrupt handler, they can use a separate thread to call the *libepee_block_until_interrupt()*, and create a new thread taking interrupt handler as the parameter when the function returns, to process the interrupt.

On the Host-to-FPGA side, the frame data will be transmitted as a unit of the linked list of frames. The benefit is to better simulate the real process of hardware and software interaction. The amount of a single frame is usually relatively small and data transmission can't achieve high efficiency so we put the frame together into the linked list. In order to support this interaction mode, we also defined the linked list of the frames as the data transmission unit in the user mode. If a simple user program needs to transfer data for a single frame, we only add one frame to the linked list.

Other APIs for controlling EPEE3.0 (such as reset) are also provided. Kernel mode APIs are similar.

EPEE 3.0 also supports multiple hardware within a single system. There may be multiple FPGA hardware within one Host. In this case, the user can choose one of them to operate via APIs. After registering the PCIe or USB drivers, each probe function will be called once for each hardware device. EPEE 3.0 constructs a unique data structure for each probe. In the EPEE system, epee_pci and epee_usb register a unique EPEE 3.0 hardware to epee_main for each probe call, and epee_main creates a unique epee_dev structure for each hardware.

5. IMPLEMENTATION OF THE FRAMEWORK

In this section, we will present the implementation details and the solutions to some problems we encountered.

5.1 Implementation of Hardware

In our implementation, our two major challenges were how to shield the difference of multi-protocols and how to support 16 different priority channels.

PCIe and USB protocols vary widely, therefore, achieving a unified interface based on a unified architecture is a big problem. On Xilinx FPGA, the PCIe IP core provides a complex interface, and requires us to process TLPs outside the IP core. In addition, there was a data boundary alignment issue in PCIe Gen2 X8 mode. The USB evaluation board provides full support for USB protocol, so we do not need the corresponding IP core to deal with USB protocol-related messages. However, the USB evaluation board shares uplink with the downstream interface; thus, additional control and processing are required for proper transmission. Therefore, we created the Transmission Control Module (TCM) to mask these differences. As there are different problems in PCIe and USB, the two versions have different TCM module designs and provide a unified interface for other modules as far as possible.

Take the detailed structure of USB version as example in Fig. 5. The TCM of USB version includes TRANS_CONTROL, TX_ENGINE and RX_ENGINE sub-modules. TRANS_CONTROL can solve the sharing interface problem. TX_ENGINE and RX_ENGINE are responsible for the uplink and downlink data forwarding. Compared with the detailed structure of PCIe version in Fig. 6, apart from the similar TX_ENGINE and RX_ENGINE, TCM also contains MWr_US, MRd_US and CplD_DS sub-modules, which are used for processing MWr, MRd and CplD packets of PCIe transaction layer. In the end, the framework supports both shared and customized functions, but provides a unified interface and interaction. Users do not need to know what mode of transmission they are using.

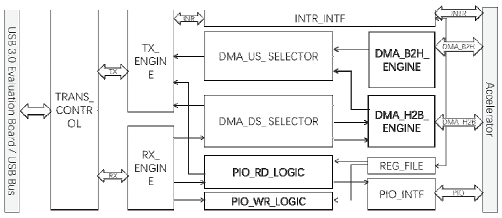


Fig. 5. Hardware structure of USB version.

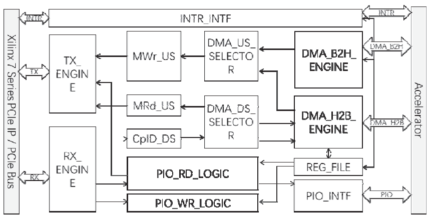


Fig. 6. Hardware structure of PCIe version.

Besides this, there are also some differences in the detailed implementation of the two versions. The unified framework supports customization of distinct functions according to the characteristics of different protocols. For example, we can use chained DMA function in the PCIe version to optimize the transmission performance of short frames.

On the Host-to-FPGA side, EPEE 3.0 supports 16 different priority channels. There are two ways to achieve the above characteristics in PCIe version. The first method is to set a buffer to store data for each channel in the hardware. When software submits a request for DMA transmission, the hardware starts the DMA operation and stores the data to the buffer. The second method is that the hardware does not immediately start the DMA operation. Instead, it waits for the accelerator to send a request to a channel, and then start the DMA operation.

The advantage of first method is low latency. Frame is pre-stored in the buffer in the FPGA, so the accelerator can get the frame immediately after sending a request to a channel. However, the method cannot support long frames because of the limited storage resources on the FPGA.

Because of this limitation, we chose the second method. The advantage of this approach is that there is no need to set a large buffer on the FPGA. Since the DMA is started after the accelerator request, the delay from the accelerator request to the accelerator getting the first data becomes larger. The delay can also result in a certain decrease in system performance. Although it has brought some performance degradation, EPEE 3.0 becomes more flexible, and can support larger frames.

5.2 Implementation of Software

During the implementation of the software part, support of both USB 3.0 and PCIe with different Linux driver frameworks is required. The epee_main module is designed to integrate PCIe and USB drivers. The hardware-dependent modules (epee_pci and epee_usb) register their epee_hw_ops structure to epee_main. The epee_hw_ops structure contains the APIs for hardware operations.

During the registration procedure, epee_main stores epee_hw_ops structure in epee_dev. When required, epee_main finds the function by looking into the epee_hw_ops in the epee_dev. As there is no difference between the USB and the PCIe registration function, epee_main is able to perform universal description and manipulation for different devices. As epee_hw_ops is the same for both PCIe and USB devices, epee_main provides the unified interface to both drivers.

6. EVALUATION

6.1 Experimental Setup

In order to test the EPEE 3.0 framework, we implemented two communication libraries, PCIe Gen2 X8 version and USB 3.0 version, on Xilinx Kintex-7 KC705 FPGA evaluation board [18]. KC705 [18] has a XC7K325T FPGA within a PCIe hard core supporting Gen2 X8 mode, two FMC interfaces used to expediently provide external I/O interfaces. CYUSB3KIT-003 by Cypress is a USB 3.0 development board carrying a CYUSB3014 chip, which can provide 3.2Gbps throughput on USB protocol layer. CYUSB3ACC-005 [19] is a pinboard used to connect CYUSB3KIT-003 and KC705 via FMC interface.

We built the EPEE project in Xilinx Vivado 2013.4 tools and conducted the evaluation on a host with an Intel I7-4770 CPU, 16 GB memory and 64-bit Ubuntu 14.04 LTS [20] with Linux kernel 3.13.0. We measured resource utilization, latency and DMA performance.

6.2 Resource Consumption

Table 2 shows the resource utilization on Kintex-7 KC705 platform under PCIe Gen2 X8 mode. We can see that the EPEE 3.0 communication library only occupied a small percentage of the resources.

Table 2. Resource consumption of EPEE 3.0.

Item	Used	Total	Util%
Slice Registers(PCIe)	18043	407600	4.42
Slice LUTs(PCIe)	11035	203800	5.41
BRAM(PCIe)	32.5	445	7.19
Slice Registers(USB)	5840	407600	1.43
Slice LUTs(USB)	3115	203800	1.52
BRAM(USB)	29	445	6.51

6.3 Latency

We tested the latency between Host and FPGA by sending short frames from host to FPGA, and looped back to host. We evaluated the transmission delay by calculating the loopback time. The loopback time of PCIe version was 32us, while the USB version was 106us. The reason for the greater latency in USB version is the USB protocol itself and the latency caused by the FPGA process of EPEE 3.0 is about 2us.

6.4 Performance

We evaluated the performance of EPEE 3.0 in this section.

(A) Maximum Read and Write Performance

Fig. 7 shows the performance of EPEE 3.0 PCIe version. 'H2B_1KB' represents the throughput of DMA write (Host-to-FPGA side) when the frame length is 1KB, and 'B2H' means FPGA-to-Host side (DMA read). We got the 15.9Gbps H2B throughput and 15.8Gbps B2H throughput. 'V2_H2B' and 'V2_B2H' indicate the performance of EPEE 2.0 PCIe version, considering of streaming transmission type in EPEE 2.0, we compared the results by transferring the same amount of data as EPEE 3.0 DMA chain.

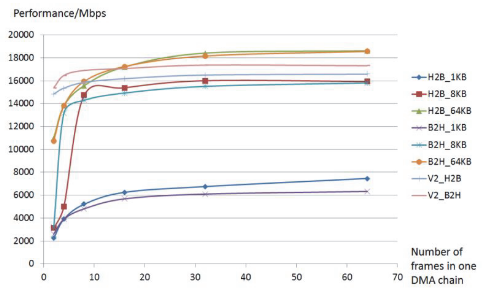


Fig. 7. Performance of PCIe version.

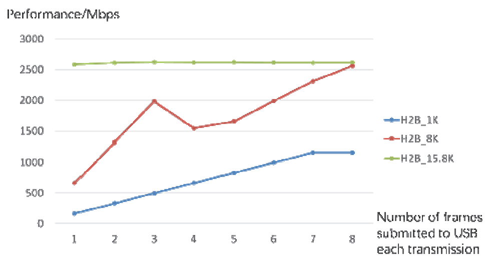


Fig. 8. Performance of USB version.

Fig. 8 shows the H2B performance of USB 3.0 version. B2H figure was not provided because there was no improvement when submitting multiple frames in one B2H transmission. In other words, the CYUSB3KIT-003 board processed the multiple frames when we submitted a long frame by fragmentation, so we do not need to do the work. We found that the USB version achieved 2.40Gbps on the FPGA-to-Host side and 2.56Gbps on Host-to-FPGA side, reaching 75% and 80% utilization of the CYUSB3-KIT-003 bandwidth, respectively.

(B) Random-Length Frame Read and Write Performance

In actual communications, the transmission length is often not fixed, but random. We put some random-length frames generated between 1KBytes to 16Kbytes in a DMA chain to evaluate the performance. The throughput was 9.52Gbps and 9.12Gbps, which was a fairly good result.

(C) Short-Frame Read and Write Performance

For wireless network, most frame data are MAC layer datagrams with KB-level size. So, an efficient short-frame-throughput of communication library is strongly needed. We tested the short frame performance of EPEE 3.0 PCIe version in Gen2X4 mode. As shown in Fig. 9, it achieved 8.84Gbps, 8.25Gbps when the frame length is 4Kbytes and 7.58Gbps, 6.51Gbps when frame length is 2Kbytes. In the latest commercial WIFI protocol IEEE 802.11ac, the highest physical layer theoretical rate with 80MHz bandwidth is 3.47Gbps. We can see that the short-frame performance is fully able to meet the network application requirement.

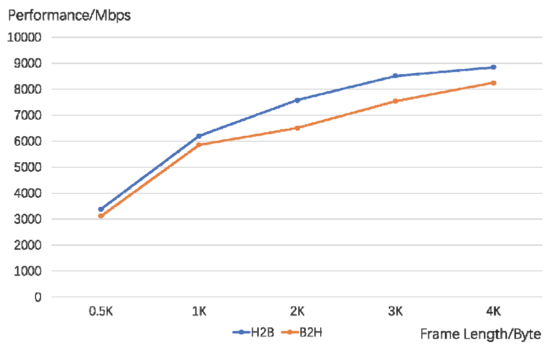


Fig. 9. Short frame performance.

6.5 Flexibility Evaluation

High flexibility requires the framework to greatly simplify the development of many applications and can be transported easily. Flexibility means no unnecessary restrictions to the developers at the software and hardware interface levels.

In EPEE 3.0, if we want to change the interface from PCIe to USB, we do not need to change the software code. As for the hardware code, there will be only 10 lines that need to be modified in DMA.

6.6 Evaluation in Real SDR Platform

EPEE 3.0 has been used long-term in an SDR platform called GRT, which provides programmability, high performance and low latency at both PHY and MAC layers. GRT has now been developed to version 3.0 with implementation of 802.11ac protocols. The EPEE 3.0 USB version meets the miniaturization requirement, and the PCIe version meets the high-throughput and low-latency requirements.

7. CONCLUSION

This paper presents EPEE 3.0, a unified frame-oriented Host-FPGA communication framework. By investigation and analysis of typical systems with frame-data transmission, the data and control interactive pattern is defined, and a unified frame-data transmission framework for PCIe and USB 3.0 is established. This framework hides the technical differences of physical buses, and provides unified software and hardware interfaces for accelerator developers. By doing this, high flexibility is achieved with frame data-based communication between the FPGA accelerator and the host. This paper described the design and implementation of the EPEE framework with PCIe and USB on a Xilinx Kintex-7 FPGA (KC705) platform. The FPGA hardware and software library were provided, as well as both the Linux kernel-level and user-level APIs for software developers. The EPEE 3.0 framework supports high-performance communication between the host and FPGA, enabling rapid accelerator integration with the host system.

For future work, we aim at successively testing more boards and providing corresponding projects. Although we have not supported the latest PCIe Gen 3 and USB 3.1 protocols, we are very confident the EPEE 3.0 can support new-generation protocols.

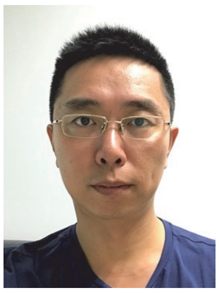
REFERENCES

1. E. Blossom, "GNU Radio: tools for exploring the radio frequency spectrum," *Linux Journal*, Vol. 2004, 2004, p. 4.
2. A. Khattab, *et al.*, "WARP: a flexible platform for clean-slate wireless medium access protocol design," *ACM Sigmobile Mobile Computing and Communications Review*, Vol. 12, 2008, pp. 56-58.
3. T. Wang and G. Y. Sun, *et al.*, "GRT: A reconfigurable SDR platform with high performance and usability," *ACM Sigarch Computer Architecture News*, Vol. 42, 2014, pp. 51-56.
4. K. Tan and H. Liu, *et al.*, "Sora: high performance software radio using general purpose multi-core processors," *Communications of the ACM*, Vol. 54, 2011, pp. 99-107.
5. K. Eguro, "SIRC: An extensible reconfigurable computing communication API," in *Proceedings of IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 135-138.
6. J. Liu, Z. W. Li, *et al.*, "A flexible frame-oriented host-FPGA communication framework for software-defined wireless network," in *Proceedings of International Conference on Networking and Network Applications*, 2017, pp. 118-124.
7. R. Bittner, "Speedy bus mastering PCI express," in *Proceedings of International Conference on Field Programmable Logic and Application*, 2012, pp. 523-526.
8. Xilinx, *Virtex-6 FPGA Integrated Block for PCI Express User Guide*, 2010.
9. Xilinx, *7 Series FPGAs Integrated Block for PCI Express Product Guide*, 2012.
10. USB-IF, SuperSpeed USB, http://www.usb.org/developers/ssusb.
11. Cypress Semiconductor Corporation, CYUSB3KIT-003 EZ-USB FX3 SuperSpeed Explorer Kit, http://www.cypress.com/?rID= 99916.

12. T. Usui, R. Kobayashi, and K. Kise, "A challenge of portable and high-speed fpga accelerator," *Applied Reconfigurable Computing*, 2015, pp. 383-392.
13. M. Jacobsen and R. Kastner, "RIFFA 2.0: A reusable integration framework for FPGA accelerators," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 2013, pp. 1-8.
14. K. Vipin and S. Shreejith, *et al.*, "System-level FPGA device driver with high-level synthesis support," in *Proceedings of International Conference on Field-Programmable Technology*, 2013, pp. 128-135.
15. M. Vesper and D. Koch, *et al.*, "JetStream: An open-source high-performance PCI express 3 streaming library for FPGA-to-Host and FPGA-to-FPGA communication," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 2016, pp. 1-9.
16. D. L. C. David, J. Korinth, *et al.*, "ffLink: A lightweight high-performance open-source PCI express gen3 interface for reconfigurable accelerators," in *Proceedings of International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2016, pp. 34-39.
17. J. Gong, T. Wang, *et al.*, "An efficient and flexible host-FPGA PCIe communication library," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 2014, pp. 1-6.
18. Xilinx, Xilinx Kintex-7 FPGA KC705 Evaluation Kit, http://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html.
19. Cypress Semiconductor Corporation, CYUSB3ACC-005 FMC Interconnect Board, http://www.cypress.com/?rID=99921.
20. Canonical Ltd., Ubuntu, http://www.ubuntu.com.

Zhi-Wei Li (李志偉) received the B.S. degree in Electronic Engineering from Peking University, China. He is currently pursuing the M.S. degree in the Department of Computer Science, Peking University, Beijing. His research interests include reconfigurable logic, wireless network and software-defined radio.

Jun Liu (劉君) received the B.S. degree in Computer Science and Ph.D. degree in Software Architecture from Northeastern University, China. She is working as a Post-Doc Researcher at Peking University. Her research interests include wireless network architecture and wireless sensing.



Tao Wang (王韜) received Ph.D. degree in Computer Science from Peking University, China. He is currently an Associate Professor in the School of Electronics Engineering and Computer Science, Peking University. His research interests include computer architecture, wireless network architecture and emotional intelligence robot.

Bo-Yan Ding (丁博岩) received the B.S. degree in Computer Science from Peking University, China. He is currently pursuing the M.S. degree in the Department of Computer Science, Peking University, China. His research interests include reconfigurable logic, wireless network and software-defined radio.