An Efficient and Flexible Host-FPGA PCIe Communication Library*

Jian Gong,[1] Tao Wang,[1,3] Jiahua Chen,[1] Haoyang Wu,[1] Fan Ye,[1] Songwu Lu,[1,2,3] Jason Cong[1,2,3]
[1]Center for Energy-Efficient Computing and Applications, School of EECS, Peking University, Beijing, China
[2]UCLA Computer Science Department, Los Angeles, CA, USA
[3]PKU-UCLA Joint Research Institute in Science and Engineering
{jian.gong, wangtao, chenjiahua, wuhaoyang, yefan}@pku.edu.cn, {slu, cong}@cs.ucla.edu

***Abstract*—A high-performance interconnection between a host processor and FPGA accelerators is in much demand. Among various interconnection methods, a PCIe bus is an attractive choice for loosely coupled accelerators. Because there is no standard host-FPGA communication library, FPGA developers have to write significant amounts of PCIe related code at both the FPGA side and the host processor side. A high-performance host-FPGA PCIe communication library holds the key to broadening the use of FPGA accelerators. In this paper we target efficiency and flexibility as two important features in such a library. We discuss the challenges in providing these features, and present our solution to these challenges. We propose EPEE, an efficient and flexible host-FPGA PCIe communication library and describe its design. We implemented EPEE in various generations of Xilinx FPGAs with up to 26.24 Gbps half-duplex and 43.02 Gbps full-duplex aggregate throughput in the PCIe Gen2 X8 mode; these are at the best utilization levels that a host-FPGA PCIe library can achieve. The EPEE library has been integrated into four different FPGA applications with different data usage patterns in various institutes.**

***Keywords*—FPGA; PCIe; Efficiency; Flexibility; Communication library**

I. Introduction

Interest in using FPGAs as computing acceleration platforms has been growing rapidly. A high-performance interconnection between the host processor and FPGA accelerators is greatly needed. Among various interconnection methods, a PCIe bus is an attractive choice for loosely coupled accelerators due to its high throughput capacity.

However, because there is no standard host-FPGA communication library, FPGA developers have to write significant amounts of PCIe related HDL code (on top of the very low-level, complicated PCIe hard IP core) at the FPGA side. They also have to create special software code (e.g., drivers, communication functions) in the host computer in order to use those FPGA accelerators. Both efforts prove to be large burdens for developers and distract them from addressing the main tasks of their FPGA accelerator applications.

Therefore, an appropriate host-FPGA PCIe communication library holds the keys to broadening the use of FPGA accelerators. We believe that there should be two important features for such a library: efficiency and flexibility.

*This paper is supported by National Natural Science Foundation of China (61370056, 61103028)

High efficiency is a must for FPGA applications requiring high data throughput. The PCIe library must provide efficient performance for applications' data transfer; and it should consume a minimal amount of FPGA resources in order to leave as much resource as possible for FPGA applications to achieve the maximum efficiency of the whole system.

High flexibility is also important. A PCIe library with rich functionality would greatly simplify the development of many applications. But there is an inherent conflict between efficiency and functionality: more functionality will consume more FPGA resources and leave less for applications, which will probably reduce the efficiency of the whole system. Besides, not all FPGA developers need the same set of functionalities. Thus, a balance between efficiency and flexibility is required in the PCIe library.

In our recent work [5], we proposed a simple version of a PCIe communication library called EPEE. In this paper we present in detail a much enhanced version of EPEE, with significant progress made on both the efficiency and flexibility. Our basic idea is to build a highly efficient core layer in the library and make it extensible. Our solution also renders EPEE portable through platform dependent and independent part design. We implemented EPEE in various generations of Xilinx FPGAs based on the PCIe hard IP core [15], [14], [16]. With the 26.24 Gbps half-duplex and 43.02 Gbps full-duplex aggregate throughput in PCIe Gen2 X8 mode, EPEE has achieved the best utilization level (82% and 67% of the theoretical maximum respectively) that a host-FPGA PCIe library could achieve. EPEE has already been integrated into four practical FPGA applications with different data usage patterns in various institutes.

The contributions of this paper are as follows:

- We target efficiency and flexibility as two important features for a host-FPGA communication library. We also identify the requirements for supporting these features, and present ways to satisfy these requirements.
- We propose EPEE, an efficient and flexible host-FPGA PCIe communication library and describe its design. We implemented EPEE in multiple generations of Xilinx FPGAs. It can also be integrated with high-level synthesis tools.
- We integrated EPEE into four practical FPGA applications with different data usage patterns in various

institutes. We make EPEE open-source and share it with the community for further improvements.

The remainder of this paper is organized as follows: Background and related work are discussed in Section II. We describe the baseline functional requirement and design challenges in Section III. In Section IV we present the design of EPEE. In Section V we describe the EPEE's software and hardware interfaces. In Section VI we discuss the implementation of EPEE and evaluate our implementation. Finally, we present a conclusion in Section VII.

II. Related work

A. Background on PCIe

PCIe is a layered protocol that includes physical layer, data link layer and transaction layer. Data is packaged into packets and transferred at the transaction layer (*TLPs*, transaction layer packets). At the physical layer, the PCIe bus provides a serial high-throughput interconnection between two devices. Multiple serial lines/lanes can be used to transfer data [10]. 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. Usually 'Gen' represents the version of the PCIe protocol, and 'X' represents the number of lanes (e.g., Gen2 X8 with 32 Gbps data rate).

B. Existing Work

PCIe hard IP cores provided by FPGA vendors [16], [14], [15], [2] can be used in FPGA accelerators. However, those IP cores have complex interfaces and require developers to know much about the PCIe transaction layer protocol. Although FPGA vendors provide drivers for their simple demo applications, developers have to write their own PCIe drivers as those simple drivers can not drive user-defined PCIe hardware.

RIFFA 2.0 [6] is an accelerator framework implemented on Xilinx FPGAs. It uses FIFO and several control signals as its *DMA* (direct memory access) interface in hardware. RIFFA can run in the PCIe Gen2 X8 mode, where it can achieve a 24 Gbps DMA throughput, 75% of the theoretical maximum (32 Gbps in the Gen2 X8 mode). However, RIFFA 2.0 does not support the user controllable register (*UCR*) interface, nor does it support the user-defined interrupt. For accelerators with relatively complicated control operations, developers have to write a significant amount of code in both the FPGA and software sides to support user-defined register operations using RIFFA's DMA interface. If interrupt is needed in an accelerator, developers have to modify a significant amount of code in both RIFFA's FPGA and driver implementation.

Speedy Bus Mastering PCI Express [3] is a PCIe communication library implemented on Xilinx Virtex-5 and Virtex-6 FPGAs. It provides a solution that maps the PCIe bus to a local bus. The library provides a driver for its FPGA example design with a DDR RAM interface. It can reach a nearly 12.8 Gbps DMA write rate and 12 Gbps DMA read rate in PCIe Gen1 X8 mode (80% and 75% of the theoretical maximum). For those applications that need DDR RAM interface, the system provides good performance. However, if developers do not want to use the DDR interface, they will have to rewrite much code on both hardware and software sides.

MPRACE [9] is a platform for the communication between the host computer and FPGA's DDR RAM with driver and library support. It runs under PCIe Gen1 X4 mode with 3.04Gbps DMA read rate and 5.6Gbps DMA write rate. It has limitations similar to those of Speedy Bus Master [3].

There are also other PCIe communication libraries. Vipim et al. [12] implemented a system-level FPGA driver based on the RIFFA project combining PCIe, DRAM and Ethernet together. This system can only support the PCIe Gen2 X4 mode as it only supports RIFFA's 64bit DMA interface. Wu *et al.* [13] implemented a PCIe library for the Virtex-5 FPGA. It controls DMA through the amount of PCIe TLPs and the TLP payload size, like the mechanism used in Xilinx XAPP1052 [7]. Kavianipour *et al.* [8] implemented a PCIe library on several Xilinx FPGA platforms which can reach 748MB/s (37.4% of the theoretical maximum) throughput in DMA read and 784MB/s (39.2% of the theoretical maximum) throughput in DMA write. FPGA2 [11] is a PCIe library targeted for FPGA-GPU communication. FlexWAFE [4] provides a solution for communication between host and FPGA accelerator, but it is specifically designed and optimized for digital film processing.

III. Baseline Functional Requirement and Design Challenges

A. Baseline Functional Requirement

The PCIe communication library should provide a collection of common functions to facilitate the communication between the FPGA and host computer. These include the support of *UCR* (user controllable register), DMA and *UDI* (user-defined interrupts in PCIe bus). UCR is needed 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 computer, PCIe equipment also needs to notify the host of its internal events (from the FPGA applications). Thus, the library must support UDI as well.

B. Challenges in Efficiency

The PCIe library should provide efficient performance for applications' data transfer as well as consume a minimal amount of FPGA resources. Full-duplex DMA data transfer is essential to make our EPEE library efficient so as to fully utilize the throughput of the PCIe bus.

C. Challenges in Flexibility

High flexibility is important for the balance between efficiency and functionality. A good PCIe library with rich functionality can greatly simplify many development tasks. But more functionality will consume more FPGA resources and leave less for applications. This may reduce the efficiency of the whole system. Besides, not all FPGA developers require the same set of functionalities. For example, a WiFi baseband pipeline requires a high-level FIFO data streaming interface to PCIe, while an SoC-based system may work better with an AXI bus interface to PCIe. Thus, there is a fundamental conflict between efficiency and functionality.

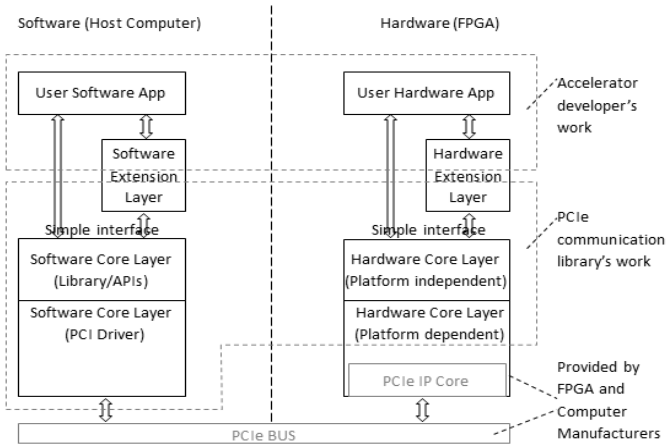

Fig. 1. EPEE system overview. EPEE consists of a software component and a hardware component; each includes a core layer and a extension layer.

IV. System Design

In this section we illustrate how our design fulfills the baseline functional requirements, as well as solving the efficiency and flexibility challenges.

A. Overall System Architecture

The EPEE library consists of a software component and an FPGA (hardware) component; each includes a core layer and an extension layer as shown in Figure 1. It also includes a PCI driver that handles the interaction between the software core layer and the PCIe bus. On the FPGA side, the PCIe hard IP core (provided by FPGA vendors) handles the interactions between the hardware core layer and the PCIe bus.

To make EPEE flexible as well as efficient, we build a small and fast EPEE core layer on the FPGA side. The EPEE core layer supports all the baseline functional requirements, i.e., UCR, full-duplex DMA, UDI, while the extension layer provides rich functionality. The extension layer consists of many plug-ins. FPGA developers can instantiate different plug-ins in different scenarios. We have already provided some common plug-ins in EPEE and developers can also share their plug-ins for reusing. As plug-ins are instantiated only when needed, FPGA developers do not have to waste resources on unneeded functionalities.

B. Design on FPGA Side (Hardware)

The structure of the FPGA library is illustrated in Figure 2. The core layer has two parts: the platform-dependent part which interfaces with the PCIe IP core from vendors, and the platform-independent part whose logic is independent from the PCIe IP core. An optional extension layer is connected to the platform-independent part for plug-ins.

The platform-independent part is modular designed for UCR (user controllable register), DMA (direct memory access) and UDI (user-defined interrupts) logics. The UCR controller handles UCR read and write requirements from the PCIe bus. The UDI controller sends interrupts required by the user application. In order to make the DMA transfer efficient, the DMA controller controls DMA read (from host to board) and DMA write (from board to host) in independent modules to support full-duplex DMA.

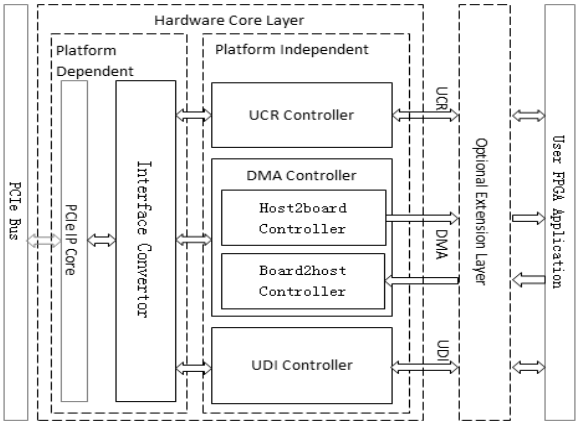

Fig. 2. Structure of FPGA Library.

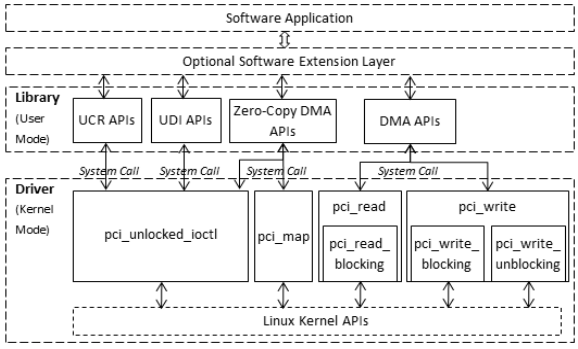

Fig. 3. Structure of Software Library

The EPEE system can be easily ported to different FPGA platforms with the platform-dependent part. To port EPEE to other FPGA platforms, the only thing required is to modify or rewrite the platform-dependent part, which is only a small portion in the hardware core library. The platform-independent part's interface is the standard PCIe transaction layer package (TLP), which is also the interface supported by the PCIe hard IP core from most vendors [16], [14], [15], [2]. Thus, modifying the platform-dependent part is relatively simple.

C. Design on Host Computer Side (Software)

The software of EPEE consists of a Linux driver, a software core layer and a software extension layer, as shown in Figure 3. The driver controls the FPGA hardware. UCR and UDI are provided in a pci_unlocked_ioctl function, which can be called by the "ioctl" system call in user mode. DMA write (DMA from FPGA to host) is controlled by pci_read_blocking, which can be called using the "read" system call. DMA read (DMA from host to FPGA) is controlled by pci_write_blocking, and pci_write_unblocking can be called by the "write" system call. The zero-copy DMA interfaces are controlled by both "ioctl" and "mmap."

The software core layer is built to provide high-level abstracted and easy-to-use APIs for software applications and software extension layers. This core layer is a wrapper for

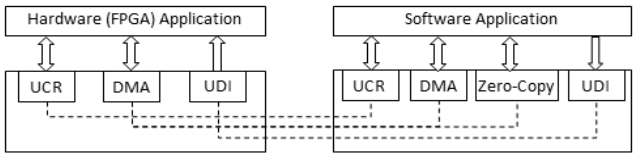

Fig. 4. EPEE Interface Overview

the driver and shields the details of data transfer. Software developers can call the core layer directly or develop some common functions in the software extension layer.

V. User Interface

A. Design Principles

High-level abstracted interfaces should be provided to developers so that they can use EPEE easily. The interface design must satisfy two requirements: 1) The interface for both the software side and hardware side should be easy to understand and easy to use. 2) The interface should shield the low-level details as much as possible. We believe exposing too much low-level detail may be counterproductive (though it gives developers more freedom). Most of the time, FPGA developers only need a limited set of common functions. The need to interact with low-level mechanisms is rare. Those who are interested in working with low-level mechanisms can use the PCIe hard IP cores provided by FPGA vendors directly.

The overview of interface functions is shown in Figure 4. UCR, DMA and UDI are provided in both hardware and software, and each software function has its counterpart in a hardware interface.

B. Hardware Interface

The main functions provided by the hardware library are UCR, DMA and UDI.

The hardware library uses a simple UCR bus as the UCR interface. The interface provides UCR read and write; both of these have request and feedback mechanisms. We consider two main reasons for the design. First, the response delay of different hardware functions differs for UCR requests. The handshake signals make the UCR operations reliable. Second, the design provides better extensibility. Using a UCR bus as interface allows the FPGA developer to allocate and configure the UCR address space freely.

We use FIFO as the DMA interface. We choose FIFO for three reasons. First, FIFO is a standard interface and its definition is relatively simple. Experienced developers know FIFO 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. Another alternative is addressable buffers such as RAM, where the developer has to wait until all the data are stored in the buffer before he can read any data. 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. Our extension layer already has a module transforming FIFO into an addressable buffer interface.

TABLE I. Hardware Interface of EPEE

	Signal	IO	Description
UCR Write	usr_ucr_wr_addr[16:0]	O	register write address
	usr_ucr_wr_data[31:0]	O	register write data
	usr_ucr_wr_req	O	register write request
	usr_ucr_wr_ack	I	register write acknowledgment
UCR Read	usr_ucr_rd_addr[16:0]	O	register read address
	usr_ucr_rd_data[31:0]	I	register read data
	usr_ucr_rd_req	O	register read request
	usr_ucr_rd_ack	I	register read acknowledgment
UDI	usr_int_req	I	interrupt request
	usr_int_vector[2:0]	I	interrupt vector
	usr_int_sw_waiting[7:0]	O	software is waiting for interrupt
	usr_int_clr	O	interrupt clear
	usr_int_enable	O	interrupt enable
DMA Read	host2board_dout[W-1:0][1]	O	DMA read data output
	host2board_empty	O	host2board FIFO empty
	host2board_rd_en	I	host2board FIFO read enable
DMA Write	board2host_din[W-1:0][1]	I	DMA write data input
	board2host_prog_full	O	board2host FIFO full
	board2host_wr_en	I	board2host FIFO write enable

[1] W is 130 in Gen2 X8 mode, 66 in other modes. The most significant two bits are data valid bits.

A group of handshake signals is used as a UDI interface because it is simple and efficient.

The main interface signals are illustrated in Table I. Some signals, such as clock and reset, are omitted in this table.

C. Software API

The principle 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 UCR operations, three APIs for DMA operations and one for UDI. The arguments and return values of the APIs are simple. Most applications' needs can be fulfilled with those three kinds of APIs. EPEE also provides a set of zero-copy DMA APIs. With zero-copy APIs, developers can access kernel buffers directly, which eliminates the memory copy operations between kernel and user buffers. Zero-copy can help to achieve better performance.

The UCR, DMA, UDI and zero-copy APIs are listed as follows. Other APIs for controlling EPEE (such as reset) are also provided.

```
int read_usr_reg(unsigned int reg, unsigned int *p_data);
int write_usr_reg(unsigned int reg, unsigned int *p_data);

int dma_host2board(unsigned int len, void *p_data);
int dma_host2board_unblocking(unsigned int len, void *p_data);
int dma_board2host(unsigned int len, void *p_data);

int block_until_interrupt(int vector_num);

void * get_zerocopy_buffer();
int release_zerocopy_buffer(void * buf);
int zerocopy_host2board(int offset, int len);
int zerocopy_board2host(int offset, int len);
```

VI. Evaluation

In this section we first present the current implementations of EPEE in various FPGA platforms; these shows its portability. Then we evaluate EPEE's performance and compare it with other PCIe communication libraries. Finally, we show EPEE's flexibility by introducing the current plug-ins in the extension layer and presenting four practical applications that employ EPEE as the underlying communication system.

TABLE II. STATISTICS ON PLATFORM-DEPENDENT CODE AMOUNT

FPGA Platform	File Num	Logic Code (line)	Wrapper Code (line)	Total Code (line)	Percentage (%)
Virtex-5	5	554	341	8799	10.2%
Virtex-6	5	554	355	8747	10.4%
Virtex-7	5	593	531	8910	12.6%

TABLE III. RESOURCE CONSUMPTION OVERVIEW

FPGA Platform	Slice Used	Slice Available
Virtex-6 (XC6VLX240T)	4243 (11%)	37680
Virtex-7 (XC7VX485T)	6652 (8%)	75900

A. Current EPEE Implementations in Various FPGAs

EPEE has been implemented on three different FPGA platforms: Xilinx Virtex-5 (XC5VLX110T-1), Virtex-6 (XC6VLX240T-2), and Virtex-7 (XC7VX485T-2). EPEE supports up to the PCIe Gen2 X8 mode (on XC7VX485T-2) now, and PCIe Gen3 mode is under development.

For Virtex-6[1] and Virtex-5, the IP cores [14], [15] provide the 64-bit TRN interface to transfer PCIe TLP packets. For Virtex-7, EPEE leverages the PCIe hard IP core for Virtex-7 FPGA [16] with 128-bit data width in the PCIe Gen2 X8 mode. The core adopts AXI4-stream as the interface for transferring PCIe TLP packets.

By dividing the EPEE core layer into platform-independent and platform-dependent parts, EPEE can be ported to different platforms with minimal effort. The amount of platform-dependent code on the three FPGA platforms is shown in Table II. We can see that they are all below 1000 lines and take less than 13% of the whole system's FPGA code.

Table III shows the resource consumption of the system (including clock domain switch plug-ins on UCR, DMA and UDI) on Virtex-6 and Virtex-7 under the PCIe Gen2 X4 and Gen2 X8 mode.[2] We find that EPEE has less resource utilization than that of Speedy Bus Master [3], which uses 14% slice resources in Virtex-6 FPGA.[3]

We are not able to port the EPEE system to Altera [2] FPGAs at this point because we did not have their developing boards or PCIe IP core. However, that IP core uses the standard PCIe transaction layer as interface. The amount of code to be modified will be equal to that of the platform-dependent part in Xilinx FPGAs, which is about 10% of the whole project.

B. Performance Evaluation

We conduct the evaluation of EPEE on a host computer with an Intel I7-3770 CPU, 16 GB memory and 64-bit Ubuntu 12.04TLS with Linux kernel 3.2.0. We measure the DMA performance on three platforms (Virtex-7, Virtex-6 and Virtex-5) by transferring 200 MB data with different DMA payload sizes. Due to space limitation, we only report performance results of the DMA zero-copy API. Table IV shows the maximum throughput that EPEE can achieve. We can see that the half-duplex DMA write throughput on Virtex-7 achieves up to 26.24 Gbps, 82% of the theoretical maximum rate under Gen2 X8 mode, and a half-duplex DMA read of 25.58 Gbps at 79.93%. The aggregate throughput of the full-duplex DMA achieves up to 43.02 Gbps (board2host and host2board in parallel).

[1]Virtex-6 also supports AXI4-stream interface, but we used the TRN interface in our implementation.

[2]The Virtex-5 development board we used only supports the PCIe Gen1 X1 mode, so its results are omitted.

[3]RIFFA did not mention the resource utilization of its whole system.

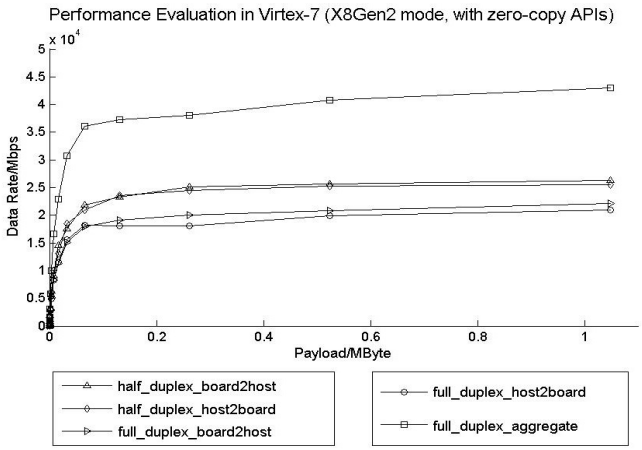

Fig. 5. Performance with Different DMA Payload Sizes Using Zero Copy (Gen2 X8 mode)

TABLE IV. PERFORMANCE EVALUATION USING ZERO COPY APIS (GBPS AND PERCENTAGE OF THE THEORETICAL MAXIMUM RATE)

Platform FPGA	half-duplex		full-duplex		
	board to host	host to board	board to host	host to board	aggre-gate
Virtex-5 (Gen1 X1)	1.72 (86%)	1.70 (85%)	1.57 (78.5%)	1.60 (80%)	3.17 (79.3%)
Virtex-6 (Gen2 X4)	13.00 (81.3%)	12.80 (80%)	12.30 (76.8%)	12.30 (76.8%)	24.6 (76.9%)
Virtex-7 (Gen2 X8)	26.24 (82%)	25.58 (79.93%)	22.09 (69.03%)	20.93 (65.41%)	43.02 (67.2%)

Figure 5 depicts in detail the performance with different DMA payload sizes in Virtex-7 (Gen2 X8). We can see that payload sizes larger than 250KB are sufficient to achieve close to maximum throughput.

There are four reasons why the system is prevented from reaching the theoretical maximum throughput: 1) Overhead of TLP header. Each TLP has a 3 or 4 DW (1 DW = 4 Bytes) header. 2) Latency when transferring DMA descriptors. EPEE cannot start data transfer before it receives the first DMA descriptor. 3) PCIe bus latency. The PCIe bus introduces some latency when it responds to EPEE's requests. 4) Latency in interrupt. Interrupt in the operating system will take some time.

Considering the overheads discussed above, common PCIe communication library implementations support about 40-75% (DMA read) and 70-80% (DMA write) of the theoretical maximum rates of the PCIe bus [9], [3]. EPEE achieves 79.93% (DMA read) and 82% (DMA write) of the theoretical maximum rates in the Gen2 X8 mode, so it is at the best utilization level that a PCIe library can achieve.

We also evaluate the latency in EPEE system using the DMA time for one DW. As the latency varies much in each test, we repeat our test 500 times and obtain the maximum, minimum and average values, as Table V shows. The gap between maximum and minimum latency is mainly introduced by interrupt, which varies from 3.54us to 26.88us in the evaluation.

TABLE VI. PERFORMANCE OF DIFFERENT PCIE PLATFORMS (GBPS)

Platform	Gen2 X2/ Gen1 X4	Gen2 X4/ Gen1 X8	Gen2 X8	Full-duplex Aggregate	UCR Support	UDI Support	Friendly Software	Supported FPGA	PCIe IP Core Interfaces[1]
EPEE	5.86(73%)	13.0(81%)	26.24(82%)	43(Gen2 X8)	YES	YES	YES	Virtex-5,6,7	TRN/AXI via portability
RIFFA [6]	Not Given	12.8(80%)	24(75%)	Not Mentioned	No	NO	YES	Spartan-6, Virtex-6,7	AXI
System-Level. [12]	Not Given	12.0(75%)	-	Not Mentioned	YES	NO	YES	Virtex-6,7	AXI
Speedy. [3]	5.84(73%)	12.8(80%)	-	Not Mentioned	YES	YES	NO	Virtex-5,6	TRN
MPRace [9]	5.6(70%)	-	-	Not Mentioned	YES	NO	YES	Virtex-4,5,6	TRN
Xapp1052 [7] [2]	7.06(88%)	14.1(88%)	-	-	NO	NO	NO	Virtex-6	TRN

[1] PCIe IP core in Virtex-4,5 uses TRN interface. Virtex-6, Spartan-6 support both TRN and AXI interface. Virtex-7 supports only AXI interface.
[2] Xapp1052 is a demo for Virtex-6 FPGA's DMA evaluation but not a PCIe library. It is without any software overhead, so its performance is almost the upper bound.

TABLE V. DMA LATENCY EVALUATION (US)

DMA Dir.	MIN	MAX	Average
DMA read	6	59	13.8
DMA write	6	58	15.9

TABLE VII. CURRENT IMPLEMENTED PLUG-INS

Plug-ins	SW	Description	App.[1]
UCR_CLK_SWITCH	N/A	Switch UCR clock domain between EPEE library and user hardware	1 2 4
UCR_BUS2REG	N/A	Do the mapping from UCR bus interface to register-based interface	2
DMA_CLK_SWITCH	N/A	Switch DMA clock domain between EPEE and user hardware	1 3 4
MULTI_CHANNEL_ DMA	Y	Provide two independent DMA channels	2
DMA_ADDRESSABLE_ BUF_INTERFACE	Y	Provide addressable buffer interface	-
INT_MANAGER	N/A	Switch UDI clock domain between EPEE library and user hardware	1 2
PARTIAL_RECONF	Y	Partially and dynamically reconfigure FPGA via PCIe link	-
RESET_GEN	N/A	Generate reset signal for user design	1 2 3 4

[1] This column lists plug-ins used by the following applications. 1: MAC prototype system, 2: WiFi baseband accelerator, 3: FPGA-based accelerating system with HLS, 4: NAND flash storage prototype

In Table VI, we summarize the performance comparison between EPEE and other systems. We can see that EPEE achieves a quite high throughput (26.24 Gbps in Gen2 X8) and PCIe utilization (82% in Gen2 X8), while supporting full-duplex DMA data transfer. It can reach 67% utilization of a PCIe Gen2 X8 link in full-duplex mode. However, other systems [6], [3], [9], [12] did not present full-duplex performance in their publications. The EPEE system achieves excellent efficiency in both half-duplex and full-duplex modes.

C. The Flexibility of EPEE and Its Practical Applications

EPEE provides flexibility with the extension layer. FPGA developers can obtain different functionality by instantiating plug-ins and combining them. Developers can also write plug-ins and share them. Some plug-ins only have hardware-side code, while others may also have corresponding software code. Table VII lists some plug-ins in our current implementation.

EPEE has been used in four practical FPGA applications in different situations. These include a MAC (media access control layer of network stack) prototype system, a WiFi baseband accelerator, an FPGA-based accelerating system supporting high-level synthesis (in particular Vivado-HLS [1]) and a NAND flash storage prototype. They have different data access requirements with different data access interfaces. Those requirements were satisfied by plug-ins in EPEE extension layers.

VII. CONCLUSION

In this paper, we discuss how we designed and built EPEE, an efficient and flexible host-FPGA PCIe communication library, by building up a highly efficient core layer for efficiency and making the library extensible. We separated IP-core dependent and independent parts in the core layer to make EPEE portable. We implemented EPEE in three generations of Xilinx FPGAs with up to 26.24 Gbps half-duplex and 43.02 Gbps full-duplex aggregate throughput in the PCIe Gen2 X8 mode, which are at the best utilization levels that a host-FPGA PCIe library can achieve. EPEE has been integrated into four different FPGA applications with different data usage patterns in multiple institutes. We plan to further optimize EPEE and use it in more applications. EPEE can be downloaded from http://cecaraw.pku.edu.cn.

REFERENCES

[1] Vivado-HLS. http://www.xilinx.com/products/design-tools/vivado/integration/index.htm.

[2] Altera. IP Compiler for PCI Express User Guide, May 2011.

[3] R. Bittner. Speedy Bus Mastering PCI Express. In *FPL 2012*.

[4] A. do Carmo Lucas, S. Heithecker, and R. Ernst. FlexWAFE - A High-end Real-Time Stream Processing Library for FPGAs. In *DAC 2007*.

[5] J. Gong, J. Chen, H. Wu, F. Ye, S. Lu, J. Cong, and T. Wang. EPEE: An Efficient PCIe Communication Library with Easy-host-integration Property for FPGA Accelerators (Abstract Only). In *FPGA 2014*.

[6] M. Jacobsen and R. Kastner. RIFFA 2.0: A reusable integration framework for FPGA accelerators. In *FPL 2013*.

[7] J. A. Jake Wiltgen. Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions, September 29 2011. Xilinx Reference Design.

[8] H. Kavianipour, S. Muschter, and C. Bohm. High performance FPGA-based DMA interface for PCIe. In *RT 2012*.

[9] G. Marcus, W. Gao, A. Kugel, and R. Manner. The MPRACE framework: An open source stack for communication with custom FPGA-based accelerators. In *SPL 2011*.

[10] T. S. Ravi Budruk, Don Anderson. *PCI Express System Architecture*. Addison Wesley, 75 Arlington St., Suite 300, Boston, MA 02116, 2003.

[11] Y. Thoma, A. Dassatti, and D. Molla. FPGA2: An open source framework for FPGA-GPU PCIe communication. In *ReConFig 2013*.

[12] K. Vipin, S. Shreejith, D. Gunasekera, S. Fahmy, and N. Kapre. System-level FPGA device driver with high-level synthesis support. In *FPT 2013*.

[13] Q. Wu, J. Xu, X. Li, and K. Jia. The research and implementation of interfacing based on PCI express. In *ICEMI 2009*.

[14] Xilinx. Virtex-6 FPGA Integrated Block for PCI Express User Guide, 2010.

[15] Xilinx. LogiCORE IP Endpoint Block Plus for PCI Express, 2011.

[16] Xilinx. 7 Series FPGAs Integrated Block for PCI Express Product Guide, 2012.