Bin Zainol, M. A., & Nunez-Yanez, J. L. (2016). CPCIe: A Compression-enabled PCIe Core for Energy and Performance Optimization. In 2016 IEEE Nordic Circuits and Systems Conference (NORCAS 2016): Proceedings of a meeting held 1-2 December 2016, Copenhagen, Denmark. Institute of Electrical and Electronics Engineers (IEEE). DOI: 10.1109/NORCHIP.2016.7792892

CPCIe: A Compression-enabled PCIe Core for Energy and Performance Optimization

Mohd Amiruddin Zainol, Jose Luis Nunez-Yanez
Department of Electrical and Electronic Engineering,
University of Bristol, United Kingdom
Email: {mb14650, j.l.nunez-yanez}@bristol.ac.uk

***Abstract*—PCIe is a hardware interface used in high-performing applications to move data from a central host and memory system to an accelerator such as a GPU or FPGA. In many memory bound applications, PCIe represents a bottleneck which limits the possible acceleration. In this paper, an open-source PCIe core is extended with a transparent layer of hardware compression/decompression with low latency and high throughput. The compressor/decompressor hardware operates on data values that match the width of the hardware interface and can be scaled up to higher parallelism. The results show an energy reduction of up to 84% in the PCIe transfers and up to 20% in the whole processing chain, thanks to the reduction in the number of bits that need to be moved over the power hungry wires that connect the main memory system to the accelerator in both directions. The overhead in terms of latency is maintained to a minimum and user selectable depending on the tolerances of the intended application.**

***Keywords*—PCIe, FPGA, data compression, energy efficiency.**

I. Introduction

Over the past few years, High-Performance Computing (HPC) platforms have been progressing from multi-core core CPUs to heavily accelerated systems using General Purpose Graphics Processing Units (GPU) or Field Programmable Gate Arrays (FPGA). In a typical configuration, these accelerators are implemented in boards that communicate with a central host via the Peripheral Component Interconnect Express (PCIe) interface for high-speed data streaming. The amount of data that needs to be moved has continued to increase as new data center and HPC applications centered in data analytics, web searches, and virtual reality are developed. The PCIe interface is seen as a potential source of a bottleneck in the system and current efforts are focused on integrating host and accelerator in the same device with a shared memory system [1]. Despite these developments, PCIe remains a popular choice and the transmission of significant amounts of data reduces performance and increases energy as well as the cost of the utility bill. Lossless data compression reduces the data size without affecting the contents and can significantly improve bandwidth by sending less data over PCIe. By reducing the transmission time, the energy efficiency can be improved as long as compression overheads are low. This work proposes a new open-source PCIe core based on streaming data compression called CPCIe (Compression-enabled PCIe) and evaluates to what extent data compression implemented on hardware can reduce energy consumption and improve performance. The contributions of this work can be summarized as follows:

- We extend the Xillybus PCIe core [2] with a tightly integrated hardware compressor/decompressor called X-MatchPRO [3]. The whole system has been released open-source at [4],
- We develop a software API for the host PC to facilitate the use of the CPCIe (Compressed PCIe) framework,
- We demonstrate the energy and performance benefits of the data compression implementation in a real prototype with hardware acceleration and accurate energy and performance measurements.

The remainder of this paper is structured as follows. Section 2 describes related work. Section 3 presents our proposed system design. Section 4 describes the evaluation of compression efficiency. Section 5 presents the demonstration system. Section 6 evaluates the energy and performance of the test system including compression/decompression overheads. Finally, section 7 concludes the paper.

II. Related Work

Modern FPGAs are being used as hardware accelerators because they contain millions of uncommitted logic elements, hardened PCIe interfaces, and can achieve peak performance over several GFLOPs thanks to dedicated floating point resources [5]. This high throughput results in hard requirements on the memory bandwidth and the corresponding interfaces. Despite these challenges, there are many examples of successful acceleration of compute intensive applications using FPGAs interfaced via PCIe to a host PC. Compression is a useful technique to reduce the demands of storage requirements in the local workstation or cloud. This sensitive data must be compressed without losses while lossy compression techniques can be applied to other data types such as images or video. An early study from Tremaine et. al. [6] uses the IBM MXT algorithm to perform lossless data compression in the FPGA and their work has contributed to other research in hardware lossless data compression. The deployment of data compression over the PCIe interface has been proposed by several researchers. SD Kim et. al. [7] proposed a memory compression mechanism for Hadoop appliances which is widely used in big data analysis with the goal of improving data storage. They reported a data reduction average 74% and 66 μs of compression time, by compressing trading log datasets using a software implementation of the LZ4 algorithm implemented on the Cortex-A9 processor present in Xilinx Zynq chips (Xilinx ZC-706 board). JY Kim et. al. at Microsoft Research presented Xpress9 [8] compressor engine to compress/decompress data in Bing web search engine [9]. For the performance evaluation, they follow a parallelization scheme and report scalability up to seven Xpress9 compressor engines fitted into an Altera Stratix V. In database management systems (DBMS), IBM developed a data warehouse appliance called Netezza [10] and created processing functions in FPGA closed to the storage devices. By implementing data compression/decompression, they have reduced the data movement between storage disks and network fabric.

Compared with previous work our solution offers lower latency limited to tens of clock cycles and is transparent to the user being tightly couple to the open-source Xillybus PCIe core. The low latency and high throughput are obtained with hardware compression that matches the interface width with the compression word width and with a fully streaming pipeline. Previous work are also proprietary, commercial and closed source while in our research we have made it open-source so that further research and optimizations are possible. CPCIe is agnostic to the acceleration function being implemented and has been tested with functions accelerated in hardware implemented using high-level synthesis tools.

III. Design and Architecture of CPCIe

We developed the overall system to perform data compression and decompression while providing streamed data for PCIe communication. As shown in Fig. 1 (a), the overall system consists of five components: host PC, CPCIe core, a hardware accelerator, power monitoring and UART display.

A. CPCIe Architecture

This section details the CPCIe hardware located between host PC and an accelerator implemented in the FPGA device. The communication between the original PCIe core and the compression/decompression system is based on AXI4-Stream interface for accelerator communication. The CPCIe core has five main components in the system: a MicroBlaze soft processor, a CPCIe custom interface, an X-MatchPRO compressor/decompressor engine, an AXI4-Stream switch and an interface to AXI4-Lite. A diagram of CPCIe architecture is illustrated in Fig. 1 (b). The MicroBlaze is used to communicate with the host PC and control the CPCIe. The capabilities of the Xillybus core are extended developing an interface to communicate the AXI4-Lite and AXI4-Stream protocols. The compression engine is based on the X-MatchPRO hardware compressor which is extended with a hardware controller. The AXI4-Stream switch is a simple switch interface that is accessible with several AXI4-Stream IP cores, and there are eight full-duplex channels. For our purpose, we pre-configured the channel to specific hardware core before synthesizing the code. Channel CH1 is connected to Xillybus FIFO stream, channel CH2 is connected to compressor channel of X-MatchPRO, channel CH3 is connected to decompressor channel of X-MatchPRO, and channel CH4 is connected to the hardware accelerator. The other four channels are left unconnected and reserved for other AXI-4 Stream based IP cores. The compressor engine supports full-duplex mode so compression and decompression can run in parallel and transactions can be made in both directions.

In the prototype system, commands are issued by the host PC and acknowledged by the MicroBlaze processor. The operational model of host PC is shown in the following order:

1) Send a command to CPCIe indicating which routing channel should be used, either to compress, decompress, or send the original file.
2) Send the datasets from host PC to input buffer on FPGA.
3) Retrieve the results from the output buffer.
4) Wait until CPCIe indicates the execution is complete.

While on the hardware side, the CPCIe will:

1) Wait for a signal from host PC.
2) MicroBlaze configure the switch based on the command from host PC.
3) Fetch input data from the input buffer, compress or decompress (based on the command) and send the compressed/decompressed results back to the output buffer.
4) Upon completion, the MicroBlaze sends a done signal to host PC.

B. Xillybus PCIe Core

To overcome the complexity of developing the controller of the transaction layer of PCIe, several third-party IP cores have been proposed and are available in open-source formats such as RIFFA [11] and Xillybus [2]. In our work, Xillybus was chosen because of its stability and the fact that no specific API is involved. Xillybus uses 32-bit data transfers and has a latency within the range of 10 to 50 μs to transmit/receive data to/from the FIFO on the FPGA. Xillybus connects the FPGA application logic using asynchronous FIFOs for read/write streaming data. The asynchronous FIFOs are essential to balance different process speeds between host PC and accelerator to ensure the data integrity. Furthermore, the Xillybus periodically checks two signals from the FIFO; the FIFO's full signal to initiates data transfers, and the FIFO's empty signal to read data from the FPGA. On the software side, Xillybus provides a device driver for Windows or Linux operating system where the file handler is opened by the host application before the user application performs the file I/O operation. The I/O files are written or read as binary files between storage disk and the FIFOs on the FPGA.

C. X-MatchPRO Compressor/Decompressor Engine

The X-MatchPRO compressor/decompressor is used in our evaluation platform since it is also available as an open-source core and it offers interesting properties regarding performance and latency. It belongs to the category of dictionary-based compressors and consists of a compressor and decompressor channel. The compression mode consists of the compressor

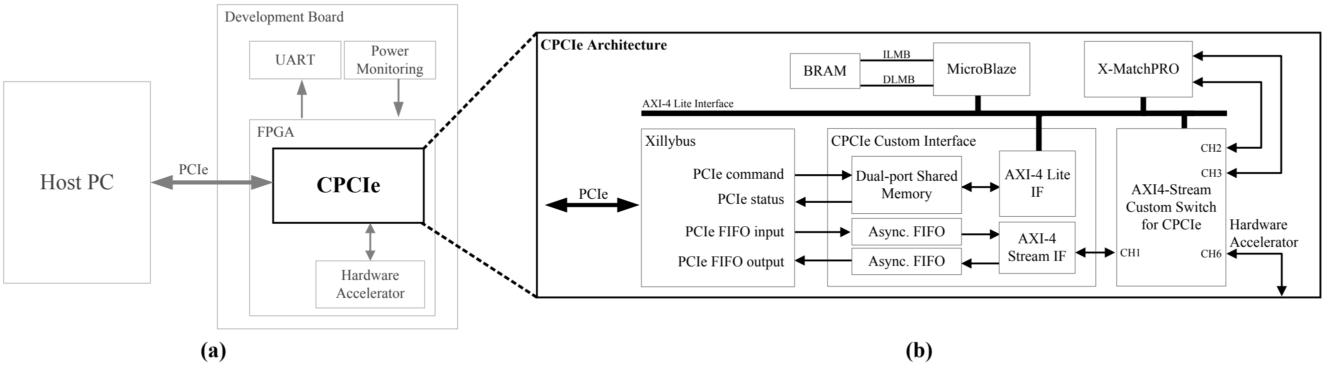

Fig. 1. Overview of the full system. In (a), the deployment of CPCIe within the evaluation system. In (b), the architecture of CPCIe.

reading uncompressed input data from the input buffer, compressing it based on the block size used, and generating 32-bit data words to the output buffer. In the decompression mode, the decompressor reads compressed input data from the input buffer, decompress the data based on the dictionary references and reconstructs the original data. The compressor and decompressor use a parallel dictionary of previously seen data to match or partially match the current data element with an entry in the dictionary. More details are available at [3].

IV. Evaluation of Compression Efficiency

In this section we evaluate the effectiveness of the compressor with typical data obtained from high-performance applications. There are four financial time series datasets acquired from public domain; sp_3m and sp_ibm, which represent the stock price of 3M and IBM market, respectively, and the other two are sp_usd and sp_prices, which represent the exchange rates between US Dollar and Japanese Yen, respectively. All datasets are coded in IEEE 754 single-precision floating-point number representation and saved as a binary file. To measure the compression efficiency, the compression ratio is used as a relative size scale. The compression ratio is defined as:

$$CompressionRatio = \frac{Original\_data\_size}{Compressed\_data\_size} \quad (1)$$

where a higher ratio indicates that a higher rate of compression can be achieved in the data. In our evaluation we compressed each dataset with different block size using X-MatchPRO compressor. Upon completion of compressing, the compressed output is decompressed to verify correctness. Fig. 2 analyzes the compression ratio for the different test datasets and compares them with block sizes ranging between 512 Bytes and 64 KB. Compression varies between factors 2X to 6X depending on the complexity of the datasets. It is possible to appreciate that a suitable block size to compress is 4 KB which is consistent with all evaluated datasets. Compressing the data in small independent blocks of 4 KB has the advantage that bit errors will only corrupt the data of the block they are located, reduces the overall latency and opens the possibility of using parallel implementations working in independent blocks in parallel.

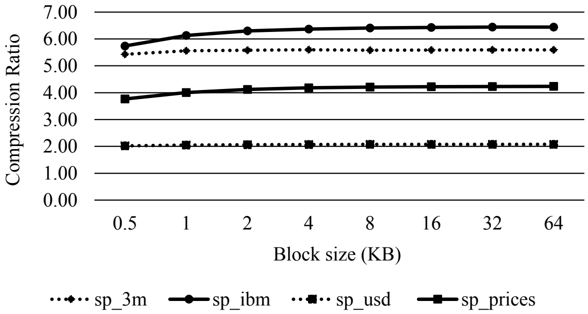

Fig. 2. The compression ratio of the datasets.

V. Demonstration System

The demonstration system used in our work is the Xilinx Virtex-7 FPGA VC707 evaluation board [12] which consists of PCIe Gen2x8 and on-board digital power monitors. To evaluate the power usage, only selected wires (or rails) on the board are measured. We only evaluate the power consumption of relevant rails which are VCCINT, MGTAVCC, and MGTAVTT; other unrelated rails are not measured. VCCINT rail provides internal power supply to FPGA resources such as flip-flop registers and LUTs. MGTAVCC and MGTAVTT rails are the analog supply to GTX transceivers in the PCIe endpoint, which are used for internal analog circuits and termination circuits, respectively. There are eight GTX transceivers which are wired to the PCIe x8 lanes, located at transceiver bank of Multi Gigabit Transceiver MGT_BANK_114 and MGT_BANK_115 on the FPGA chip. To measure the power consumption, a system is implemented to read the output power from power monitors. The power monitor chip wires the connection between voltage rails and the PMBus (power-management bus) to obtain the power reading. The system implements a MicroBlaze processor to obtain the output power from the PMBus using a specific API software. Then the output power is sent to the UART to display the power from the specified rails. Table I shows the summary of system resource utilization for each component in the CPCIe system.

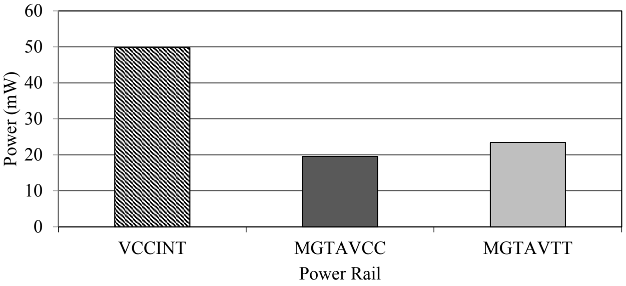

Fig. 3. The active power of three rails during PCIe transmission.

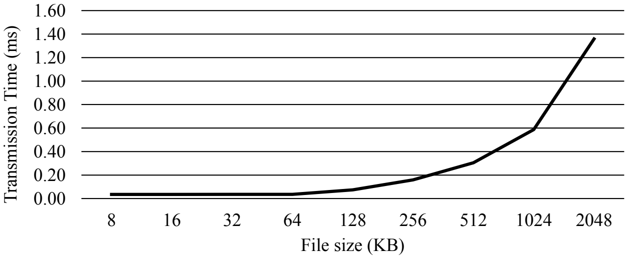

Fig. 4. The transmission time of different file size over PCIe in milliseconds.

VI. Evaluation of Energy and Performance

In this section, we evaluate the energy and performance of our proposed work. Fig. 3 shows the breakdown of active power of measured rails. In this work, we define active power is the total combination of power during the transmission of a dataset and power during computation in the accelerator. While the idle power is the power without activities and waiting for the next command. We conducted multiple measurements of the power and average all values to obtain the shown power. The active power of each rail in VCCINT, MGTAVCC, and MGTAVTT are 49.79 mW, 19.54 mW, and 23.43 mW, respectively. These power values are the combination of logic resources in the Xillybus PCIe core and the endpoint in PCIe.

The transmission time required for a file to transmit from Host PC to FPGA is plotted in Fig. 4. The relation between the transmission time and file size ranging between 8 KB to 2 MB is roughly exponential so minimizing the file size reduces the transmission time. In this work, the PCIe is configured at 250 MHz and x8 lanes.

TABLE I
Resource utilization of each component used in the CPCIe system.

	LUTs	Registers	BRAMs
Xillybus	4418	5515	9
CPCIe Custom Interface	228	315	3
AXI4-Stream Switch	720	81	0
X-MatchPRO Engine	8013	3187	14
MicroBlaze Processor	657	1002	2
AXI4-Lite Interface	364	154	0
Total Resources of CPCIe	14400	10254	28

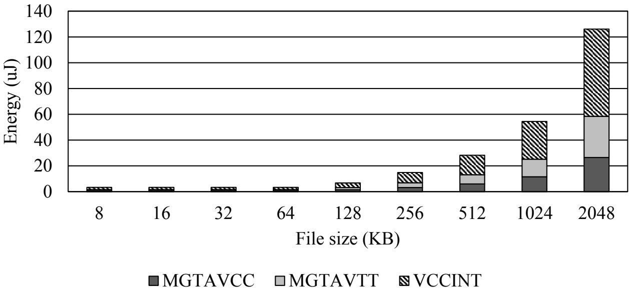

Fig. 5. The energy usage during PCIe transmission in μJ.

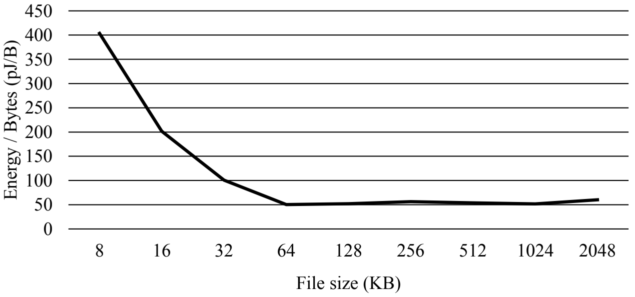

Fig. 6. The trade-off between energy and size for different file size measured in picoJoules/Byte.

Fig. 5 shows the distribution of energy consumption during file transmission as file size is increased. The energy (E) is calculated as:

$$E = Pt \tag{2}$$

where P is the active power, and t is the transmission time. The energy in rail VCCINT, MGTAVCC, and MGTAVTT are grouped into one bar to visualize the total energy of each file size. As expected, the larger the file size, the higher the energy consumed. The idea is that if we deploy a compressor/decompressor engine, we could reduce both energy and transmission time as long as the additional power required by the compression/decompression process is small.

Fig. 6 investigates the trade-off between energy and file size in unit of picoJoules/Bytes. The file size of 8 KB has the highest Energy/Bytes (403 pJ/B), followed by file size of 16 KB (201 pJ/B), and 32 KB (100 pJ/B). However, for more than 64 KB of file size, this trade-off is between 50 to 60 pJ/B. From this graph, we can conclude that for optimal energy per byte over PCIe, we need to use a basic block transfer size of more than 64 KB. The reason behind this situation is that the file size of less than 64 KB has a high amount of overhead (header, protocol, status, etc.) over the payload size.

In the next subsection, we compare two systems with and without deploying the compression technology to understand the impact of the compressor in terms of transmission speed, power, and energy. There are three components in the FPGA were evaluated which are PCIe, a hardware accelerator, and X-MatchPRO engine. We implemented a ChipScope Logic

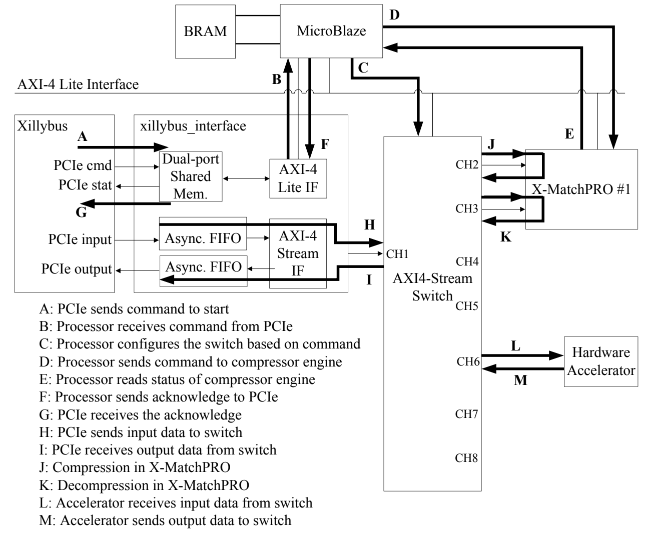
BRAM
MicroBlaze
D
B
C
AXI-4 Lite Interface
Xillybus
xillybus_interface
A
PCIe cmd
PCIe stat
G
Dual-port Shared Mem.
AXI-4 Lite IF
F
E
J
CH2
X-MatchPRO #1
CH3
K
PCIe input
Async. FIFO
AXI-4 Stream IF
H
CH1
PCIe output
Async. FIFO
I
CH4
AXI4-Stream Switch
CH5
L
Hardware Accelerator
CH6
M
CH7
CH8
A: PCIe sends command to start
B: Processor receives command from PCIe
C: Processor configures the switch based on command
D: Processor sends command to compressor engine
E: Processor reads status of compressor engine
F: Processor sends acknowledge to PCIe
G: PCIe receives the acknowledge
H: PCIe sends input data to switch
I: PCIe receives output data from switch
J: Compression in X-MatchPRO
K: Decompression in X-MatchPRO
L: Accelerator receives input data from switch
M: Accelerator sends output data to switch

Fig. 7. Transmission path of CPCIe. The latency (clock cycles) in each corresponding directions: A=6, B=5, C=5, D=5/10 (Decompression/Compression command), E=5, F=5, G=6, H=0, I=0, J=66, K=54, L=0, M=0.

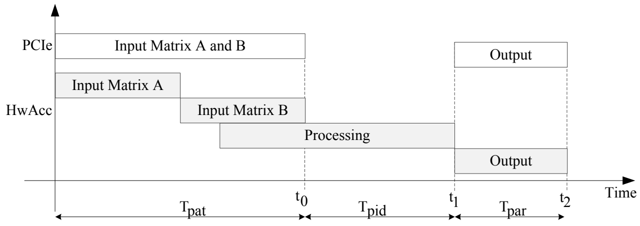

Fig. 8. The execution phase of streaming an uncompressed data.

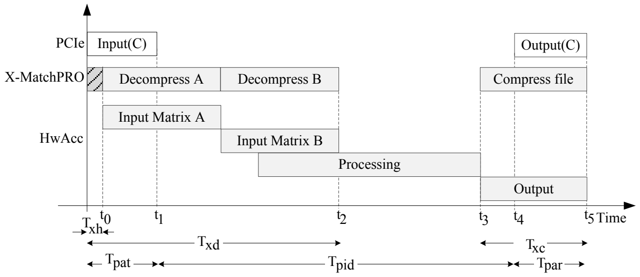

Fig. 9. The execution phase of streaming a compressed data.

Analyzer to the FPGA logic to obtain the hardware processing phases. For simplicity, the hardware accelerator is a matrix multiplication 32x32 matrices of single precision 32-bit floating point number. We developed the matrix multiplier with an AXI4-Stream interface using Vivado HLS 2013.3. The resource utilization for matrix multiplier are: 12921 LUTs, 12731 Registers, 33 BRAMs, and 160 DSP48E.

A. Streaming of uncompressed data

In this test system, we sent an uncompressed dataset from Host PC to the accelerator, and the output results of the accelerator are then sent back to the Host PC. As illustrated in Fig. 7, the transmission path is through the direction of: A → B → C → F → G → H → L → M → I. Fig. 8 illustrates the execution phase in the PCIe and the accelerator. In this figure, the Host PC transmitted both input matrices in a single file to the accelerator. As soon the accelerator received the input, it started to process the dataset. The PCIe core finished transmitting the data at t$_0$ and began to go into idle phase, while the accelerator was still processing the data. Once the output result is ready, the PCIe was activated again to receive the result from the accelerator at t$_1$. All data were completely received in host PC at t$_2$. At the bottom of this figure, there are three time periods labeled as T$_{pat}$ (the active period during Host PC transmitted the data over PCIe), T$_{pid}$ (idle period of PCIe), and T$_{par}$ (the active period during Host PC received the data from PCIe). The period of T$_{pat}$, T$_{pid}$, and T$_{par}$ were 52.05 μs, 13.63 μs, and 37.15 μs, respectively; where the period of Tpat and Tpar constitute the total activation time in PCIe of 89.20 μs, and the period of Tpat, Tpid, and Tpar constitute the total time in matrix multiplier of 102.83 μs. The measured active power of PCIe and matrix multiplier were 92.76 mW and 145.09 mW, respectively. Thus the energy in the PCIe and the accelerator was 8.27 μJ and 14.92 μJ, respectively, and the summation of both energies contributes the total energy of 23.19 μJ.

B. Streaming of compressed data

Compared to the previous subsection, in this test we transmitted a compressed dataset to the decompressor before sent it to the accelerator. The output result of the accelerator is then compressed in the compressor before sent it back to the host PC. A dataset with a compression ratio of 6:1 was chosen for this experiment. As illustrated in Fig. 7, the transmission path is through the direction of: A → B → C → D → E → F → G → H → K → L → M → J → I. Fig. 9 illustrates the execution phase of streaming the compressed data. In this figure, the execution phase of this test has an additional component of X-MatchPRO engine. The host PC transmitted the compressed input identified as Input(C) in PCIe and finished at t$_1$. When the X-MatchPRO decompressor received the compressed input, it started to decompress the header of the compressed file and finished at t$_0$ (denoted as T$_{xh}$). During the decompression of the header, the decompressed output is not ready with the original data and the latency of T$_{xh}$ is measured at 0.54 μs. After t$_0$, the decompressor started to decompress the original data and send to the accelerator. The decompressor finished the task at t$_2$ and this period is denoted as T$_{xd}$. After t$_2$, the X-MatchPRO engine became idle. Once the output result of the accelerator is ready, the accelerator sent the output to the compressor for compression at t$_3$. The X-MatchPRO compressor received the input data from the accelerator and started to compress the data. During this period, the compressed output is not ready to be fetched by host PC. At t$_4$, the host PC started to receive the compressed output identified as Output(C) and finished at t$_5$; which is denoted as T$_{par}$. The T$_{xc}$ is the period of compression time in X-MatchPRO compressor. The execution period of T$_{pat}$, T$_{pid}$, and T$_{par}$ were 8.39 μs, 88.16 μs, and 6.28 μs, respectively. The combination of T$_{pat}$ and T$_{par}$ constitute the total active time in PCIe of 14.67 μs, and thus constitute PCIe energy of 1.36 μJ. Note that the active power in PCIe and the average power in matrix multiplier during this test were about the same as previous test, and summarized in Table II. The X-MatchPRO engine has the active power of compressor and decompressor measured at 24.51 mW and 25.98 mW, respectively. In this test, the time to compress (T$_{xc}$) was 37.15 μs, and the time to decompress (T$_{xd}$) was 51.32 μs. Thus the energy cost of compressor and decompressor were 0.91 μJ and 1.33 μJ, respectively. The total energy in this case is the summation of three energies; the PCIe energy, the energy of accelerator, and the energy during activation of X-MatchPRO compressor/decompressor. Thus the total energy is 18.52 mJ.

The summary of both configurations without and with CPCIe can be seen in Table III. The PCIe energy without and with the CPCIe implementation are 8.27 mJ and 1.36 mJ, respectively, and the percentage difference is quite large at 84% of energy reduction in PCIe. However, the percentage difference of the whole energy is masked by the presence of the accelerator, and this contributes to the total energy reduction by 20%. From this observation, we found that the total energy to complete a task is greatly dependent on the energy of the accelerator. By referring to the resources used by matrix multiplier, a high number of FPGA resources has led to the high energy consumed by the accelerator.

Another important to note that, we also made other experiments with compression ratio of 2X and 4X. For the compression ratio of 2X, the PCIe energy and the total energy was 3.75 mJ and 20.91 mJ, respectively; thus the percentage difference of PCIe energy and total energy was 55% and 9%, respectively. While for the compression ratio of 4X, the PCIe energy and the total energy was 2.55 mJ and 19.71 mJ, respectively; thus the percentage difference of PCIe energy and total energy was 69% and 15%, respectively. Thus we conclude that the energy reduction of PCIe energy ranges between 55% to 84% depending on the compression ratio from 2X to 6X. For the total energy to complete a task in the system, the energy reduction ranges between 9% to 20%.

TABLE II
THE SUMMARY OF ACTIVE POWER IN EACH COMPONENT.

Component	Power (mW)
Xillybus PCIe	92.76
X-MatchPRO Compressor	24.51
X-MatchPRO Decompressor	25.98
Matrix Multiplier	145.09

TABLE III
ENERGY DISCUSSION USING DATASET WITH COMPRESSION RATIO OF 6X.

	PCIe energy	Total energy
Without CPCIe	8.27 μJ	23.19 μJ
With CPCIe	1.36 μJ	18.52 μJ
Difference	84%	20%

VII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed CPCIe (Compressed PCIe) framework which employs X-MatchPRO compressor/decompressor engines. We have demonstrated that data compression tightly implemented between PCIe and accelerator can obtain energy savings while maintaining performance for custom hardware accelerator designs. Our evaluation shows that our CPCIe design can reduce the PCIe energy between 55% to 84%, and reduce the size of datasets between factors of 2X to 6X depending on the complexity of the datasets. While the energy reduction of the whole system to complete a task is reduced between 9% to 20% which is masked by the presence of the accelerator. Notice also that these experiments have focused on the energy savings in the domain of the PCIe board and do not consider the costs of transmitting the data from host to accelerator which will be reduced as well. Based on our experiments, the energy efficiency during PCIe transmission can be reduced in two ways: the power in PCIe during the active and idle period, and the transmission period of input/output. Future work will focus on expanding the number of benchmarks used and in exploring parallel configurations in which multiple compressors and decompressors work in parallel.

VIII. ACKNOWLEDGEMENT

This work is supported by the UK EPSRC under grants ENPOWER (EP/L00321X/1) and ENEAC (EP/N002539/1) grants.

REFERENCES

[1] H. Giefers, R. Polig, & C. Hagleitner, ”Accelerating arithmetic kernels with coherent attached FPGA coprocessors”, *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, 2015.

[2] Xillybus *www.xillybus.com*.

[3] J. L. Nunez, & S. Jones, ”Gbit/s lossless data compression hardware”, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2003.

[4] CPCIe *https://github.com/mohdazainol/cpcie*, CPCIe project maintained by Mohd A. Zainol, 2016.

[5] G. Inggs, S. Fleming, D. Thomas, & W. Luk, ”Is high level synthesis ready for business? A computational finance case study”, *In Field-Programmable Technology (FPT), 2014 Intl. Conf. on*, 2014.

[6] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, & P. M. Bland, ”IBM Memory Expansion Technology (MXT)”, *IBM Journal of Research and Development*, 2001.

[7] S. D. Kim, S. M. Lee, S. M. Lee, J. H. Jang, J. G. Son, Y. H. Kim, & S. E. Lee, ”Compression Accelerator for Hadoop Appliance”, *in Internet of Vehicles Technologies and Services*, 8662, R. H. Hsu and S. Wang, Eds., ed: Springer International Publishing, 2014.

[8] J. Y. Kim, S. Hauck, & D. Burger, ”A Scalable Multi-Engine Xpress9 Compressor with Asynchronous Data Transfer”, *in Field-Programmable Custom Computing Machines (FCCM)*, 2014.

[9] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, & M. Haselman, ”A reconfigurable fabric for accelerating large-scale datacenter services”, *In 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014.

[10] P. Francisco, The Netezza data appliance architecture: A platform for high performance data warehousing and analytics (2011) *IBM Redguide*.

[11] M. Jacobsen, and R. Kastner, ”RIFFA 2.0: A reusable integration framework for FPGA accelerators”, *in Proc. International Conference on Field-Programmable Logic*, 2013.

[12] Xilinx, VC707 Evaluation Board for the Virtex-7 FPGA (UG885 v1.7).