PCI Express FPGA Platform for Big Data Applications

Irene González Crespo, Pedro Pérez Carballo and Antonio Núñez Ordóñez
IUMA, Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, Spain
{igcrespo,carballo,nunez}@iuma.ulpgc.es

***Abstract*— This paper presents the work done to create a platform that combines PCI Express and data classification. It is divided in two parts. The first one focus on the implementation of the PCI Express communication bus that allows a fast and efficient communication between the FPGA and the system host. By doing so the FPGA allows the use of PCI Express as a data interface. The second one focus on the development of a data sorter for Big Data applications. This allows the use of streamed data. For this reason, the combination of both parts provides the consolidation of a processing system for Big Data, but it also facilitates its re-utilization for other applications (DPI, CEP, …). The system implementation has been done on an ZC706 Xilinx development board.**

Keywords*— *PCI Express; Big Data; FPGA; sorting; classifier; RIFFA; bitonic sorting

I. INTRODUCTION

In the last decade the breakthroughs in data traffic and analysis have required new methods capable to process extensive amounts of data in real time. This implies the necessity of new systems to improve their bandwidth and latency. Big Data appears as a solution to those problems allowing the storage and processing of large quantities of data [1].

MapReduce is a parallel program model used in Big Data as basic structure [2]. A sorting of the data between the map and reduce stages is necessary to reduce the complexity of these stages. The significance of classifiers is the increasing need in easing the processing and distribution of data.

Also, the bandwidth between the FPGA and the host CPU is a key parameter that needs to be improved. PCI Express is a fast and efficient solution that exploits the bandwidth of the system [3].

II. PCI EXPRESS

PCI Express is a point-to-point serial interconnection protocol, that allows a high bandwidth and introduces scalability and versatility. It can be implemented in a huge variety of applications, like mobile devices, servers, communication platforms or embedded systems [4].

PCI Express is a communication bus that is compatible with PCI and PCI-X, its predecessors. It implements a packet based communication protocol and is part of a third generation of input/output buses of high performance. The use of a layer model with three principal layers –the physical layer, the data link layer and the transaction layer– provides integrity to the bus [5], [6]. In Fig.1 the connection between the layers is shown. It requires the use of TLPs to do the communication between the devices through PCI Express. The allowed transactions are divided into four categories: memory (read/write), input/output (read/write), configuration (read/write) and messages [7]. The messages are bursts of control information or data sent from buffer to buffer.

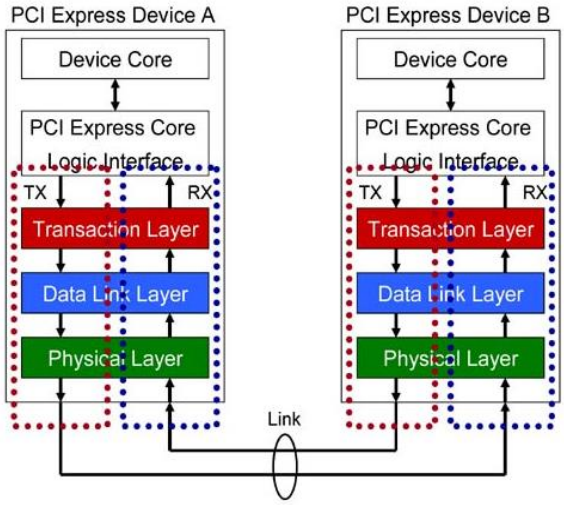

Figure 1: Layers of PCI Express

It also implements a control flow; this confirms that every receiver buffer has enough space to accept the information from the transmitter. Because of this most errors are avoided and there is no need for forwarding packets support.

ZC706 has an integrated block for PCI Express from the 7 series of Xilinx FPGAs [8]. This block allows the implementation of PCI Express over the platform, providing scalability, high bandwidth and reliability. All the characteristics of this block are based on the PCI Express specification. The implementation of Gen1 and Gen2 are available, as well as the channel configuration from 1, 2, 4 or 8 lanes. Also, it offers AMBA AXI-4 Stream interfaces as user interfaces.

Due to the use of less signals than the previous communication protocols, PCI Express is a low power bus. It allows the individual power management of each integrated device. It also supports error management and the addition of multiple data structures, including isochronous transmissions.

III. RIFFA

RIFFA (Reusable Integration Framework for FPGA Accelerators) integrates the PCI Express communication bus between the host CPU and the FPGA [9]. Its main objective is to improve the implementation of applications in FPGAs and to expand its use. RIFFA allows communication and synchronization of software and hardware through a standard interface. It includes flexibility and reusability in the designs.

RIFFA does not require specialized hardware or licensed IPs. It only requires a host with the PCI Express communication bus enabled and a FPGA with a PCI Express

peripheral. RIFFA offers support for Windows and Linux OS, and the APIs for its implementation are developed in C/C++, Python, MATLAB and Java. Nevertheless, its hardware implementation includes an interface with independent transmission and reception signaling, handshaking and first word fall through a FIFO interface [10]. Yet, the use of RIFFA provides a 97% of the allowed bandwidth of PCI Express. In Fig. 2 the architecture of RIFFA is presented.

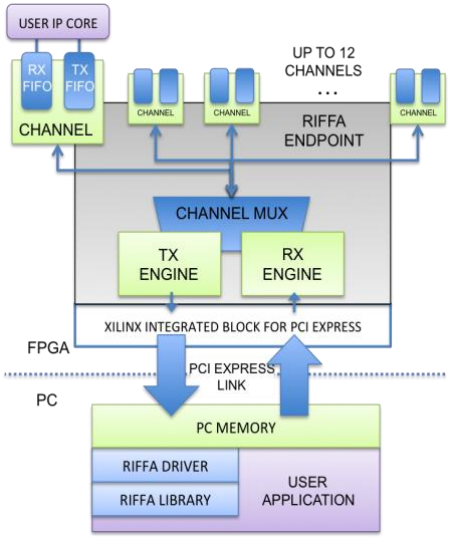

Figure 2: Architecture of RIFFA 2

However, for its implementation the FPGA Xilinx ZC706 is selected. In this project we have used RIFFA version 2.2.2. The design has been synthesized in the Vivado 2018.1 version. The original code has been adapted for this version of the tools and the prototyping board. Most of the modification are related with the Verilog code and the support for new versions of SystemVerilog.

The diagram of the RIFFA platform over ZC706 is shown in Fig. 3. In this block diagram the RIFFA block is presented, connected to the PCIeGen2x4lf128, that includes the PCIe block from Xilinx. The inputs and outputs of the system are displayed, as well as the signals for its synchronization.

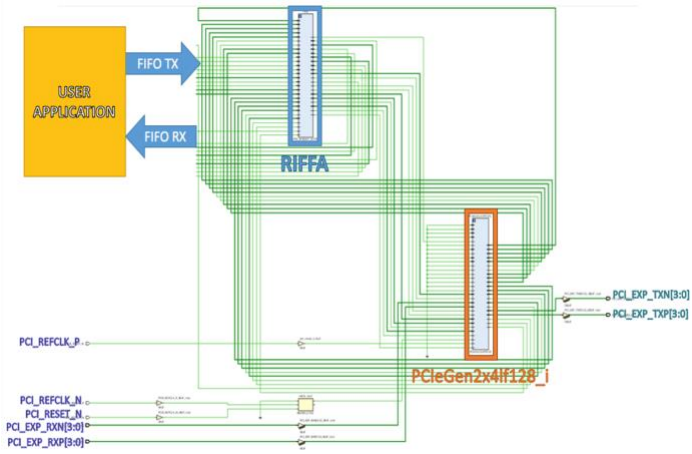

Figure 3: Diagram of the RIFFA platform over ZC706

After the completion of the synthesis and the implementation the results of its utilization resources are obtained. In this project two different implementations of RIFFA are made; the first one has an AXI interface width of 128 bit, and the second one of 64 bit. The comparison of these two projects is shown in Table 1.

TABLE I. UTILIZATION RESOURCES OF RIFFA IMPLEMENTATION OVER ZC706

Resources	RIFFA Implementation over ZC706	
	AXI 128 bit	*AXI 64 bit*
LUT	7.364 (3,37%)	5.343 (2,44%)
LUTRAM	225 (0,32%)	157 (0,22%)
FF	12.061 (2,76%)	9.186 (2,10%)
BRAM	31 (5,69%)	24 (4,40%)
IO	5 (1,38%)	5 (1,38%)
GT	4 (25,00%)	4 (25,00%)
BUFG	6 (18,75%)	4 (12,50%)
MMCM	1 (12,50%)	1 (12,50%)
PCIe	1 (100%)	1 (100%)

IV. BITONIC SORTING

Bitonic sorting is a sorting network specially used in applications with high classification rates. It is a parallel algorithm that realizes comparisons through a predefined sequence [11], [12]. It uses a bitonic sequence which first half values are ascending and its second half values are descending. The bitonic sorting algorithm starts from a bitonic sequence. Then realizes the comparison between the first element of the first half and the first element of the second half and exchanges them in case the first one is greater than the second. Subsequently, it realizes the same comparison between the second element of the first half and the second element of the second half, and so on. When it finishes these comparisons, two bitonic sequences are created. So, it realizes the comparisons for each bitonic sequence separately. These stages should be repeated, until the width of the bitonic sequence is the unit, thereby the data are sorted. In Fig. 4 an example of bitonic sorting for an 8-element array is shown.

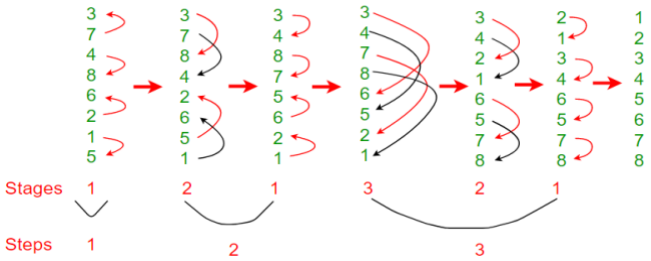

Figure 4: Performance of the bitonic sorting algorithm

Due to its parallelizable capability it is a great choice for its implementation in FPGAs [11]. In Fig. 5 the design of the bitonic sort developed in this project is presented. 8 bit AXI4-Stream in and out interfaces are selected for its further implementation in Big Data applications. The key-values are sent sequentially, because each of them has a value of 8 bit.

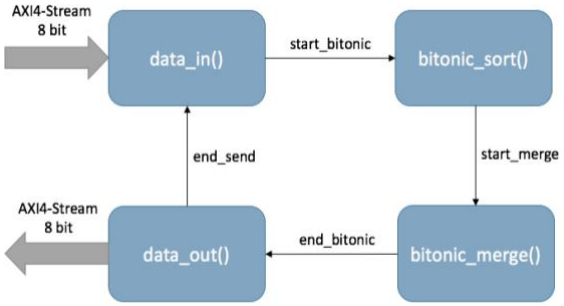

Figure 5: Design of the bitonic sort developed

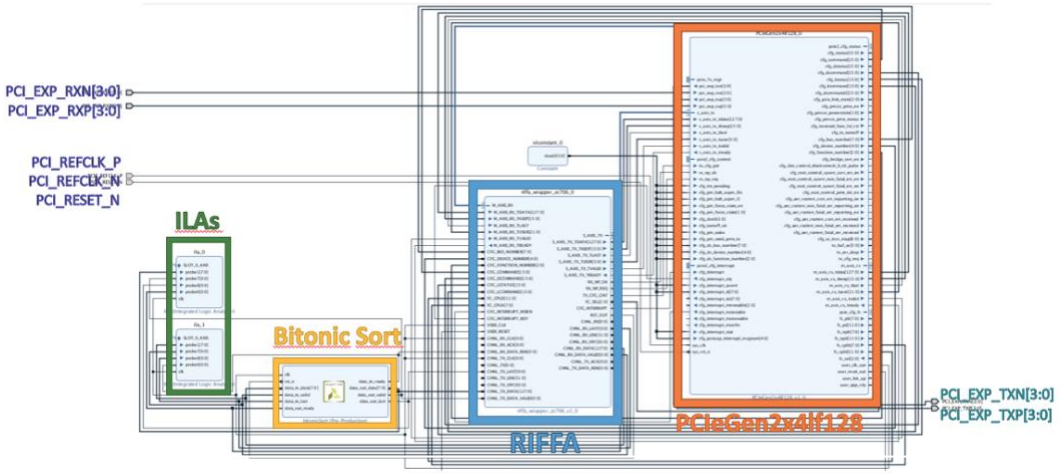

Figure 6: Block design of the final system

V. FINAL SYSTEM

Once the RIFFA and the bitonic sort are implemented in the platform, a final system including both of them is created. The block diagram of its implementation is shown in Fig. 6. The RIFFA version with the 128 bit AXI width is selected. The layout of the final system is presented in Fig. 7. As displayed, the yellow cells are the RIFFA block, the green cells are the bitonic sort and the orange cells are the PCIe Xilinx block. The utilization resources of the final system implementation are presented in Table 2. There is a comparison between the full system, the RIFFA module (without the Xilinx PCIe block), and the classifier (bitonic sort).

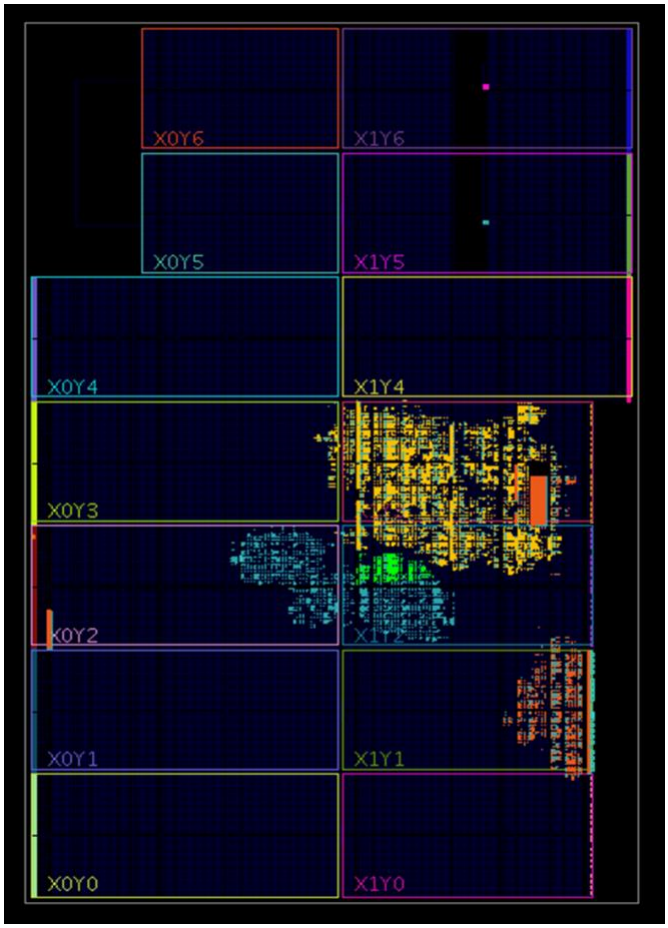

Figure 7: Block design of the final system

TABLE II. UTILIZATION RESOURCES OF RIFFA IMPLEMENTATION OVER ZC706

Resources	Final System Implementation over ZC706		
	Complete System	***RIFFA***	***Bitonic sort***
LUT	9.559 (4,37 %)	5.172 (2,37 %)	522 (0,24 %)
LUTRAM	467 (0,66 %)	187 (0,27 %)	0 (0,00 %)
FF	15.547 (3,56 %)	9.139 (2,09 %)	566 (0,13 %)
BRAM	30 (5,50 %)	24 (4,40 %)	1 (0,18 %)
IO	5 (1,38 %)	0 (0,00 %)	0 (0,00 %)
GT	4 (25,00 %)	0 (0,00 %)	0 (0,00 %)
BUFG	7 (21,88 %)	0 (0,00 %)	0 (0,00 %)
MMCM	1 (12,50 %)	0 (0,00 %)	0 (0,00 %)
PCIe	1 (100%)	0 (0,00 %)	0 (0,00%)

VI. CONCLUSIONS

In this paper, a PCI Express platform for Big Data applications is presented. The advantages of implementing a communication bus with high throughput and low power consumption on FPGA produces a system with great performance that can be implemented in a lot of applications. However, the implementation of a sorting algorithm, like bitonic sort, provides a system that easiness the processing and distribution of data. Combining these two systems an efficient platform is obtained. Hence, it can fulfill the necessities of Big Data applications.

REFERENCES

[1] K. Neshatpour, A. Sasan, and H. Homayoun, “Big data analytics on heterogeneous accelerator architectures,” *CODES+ISSS*. ACM, Pittsburg, PA (USA), 2016

[2] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for Multi-core and Multiprocessor Systems,” in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 13–24

[3] Y.-T. Yang, S.-T. Zhang, Z.-C. Li, M.-D. Zhang, and G.-C. Cao, “Design and Implementation for High Speed Data Transfer Interface of PCI Express Based

on Zynq Platform," *Dianzi Keji Daxue Xuebao/Journal Univ. Electron. Sci. Technol. China*, vol. 46, no. 3, pp. 522–528, 2017.

[4] J. Lawley, "Understanding Performance of PCI Express Systems," *Xilinx*. 2014

[5] E. Solari and B. Congdon, "Chapter 2: PCI Express Architecture Overview," in *The Complete PCI Express Reference. Design Insights for Hardware and Software Developers*, Intel Press, 2003, pp. 41–91.

[6] N. Budruk, D. Anderson, and T. Shanley, "Chapter 2: Architecture Overview," in *PCI Express System Architecture*, Ed. Addison Wesley, 2004, pp. 55–105.

[7] N. Budruk, D. Anderson, and T. Shanley, "Chapter 4: Packet-Based Transactions," in *PCI Express System Architecture*, Ed. Addison Wesley, 2004, pp. 154–209.

[8] X. Inc., *7 Series FPGAs Integrated Block for PCI Express v3.3*, PG054 ed. 2016

[9] D. Richmond and M. Jacobsen, "RIFFA 2.2.2 Documentation." 2016

[10] M. Jacobsen, D. Richmond, and M. Hogains, "RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 4, 2015

[11] M. Roozmeh, "High Level Synthesis of Bitonic Sorting Algorithm," *GitHub*, 2016. [Online]. Available: https://github.com/mediroozmeh/Bitonic-Sorting

[12] "Bitonic Sort," *GeeksForGeeks*, 2018. [Online]. Available: https://www.geeksforgeeks.org/bitonic-sort/