# An Open-Source Hardware/Software Architecture for Remote Control of SoC-FPGA Based Systems

Werner Florian[1,2(✉)], Bruno Valinoti[1,2,3], Luis G. García[1,2], Marcos Cervetto[3], Edgardo Marchi[3], Maria Liz Crespo[1], Sergio Carrato[2], and Andres Cicuttin[1]

[1] The Abdus Salam International Centre for Theoretical Physics—MLAB, Strada Costiera, 11, 34151 Trieste, TS, Italy
{wflorian,bvalinot,lgarcia1,mcrespo,cicuttin}@ictp.it
[2] Università degli Studi di Trieste—Dipartimento di Ingegneria e Architettura, v. Valerio 6/1, 34127 Trieste, TS, Italy
carrato@units.it
[3] Instituto Nacional de Tecnología Industrial, Av. Gral. Paz 5445, 1650 San Martín, BA, Argentina

**Abstract.** We present an open hardware/software architecture for remote control of Field Programmable Gate Array (FPGA) based Systems on Chip (SoC). These systems, which integrate embedded processors, FPGA fabric, memory blocks and other resources, usually need to be controlled from a computer. The proposed architecture comprises a set of commands, instructions for data movement, and standardized data packets. A minimal set of specifications and design guidelines will effectively separate hardware and software developments granting compatibility to the different subsystems. A simple architectural approach ensures compatibility of computer resident software, embedded processor software, and FPGA designs. The implicit structured design methodology associated with the proposed architecture facilitates remote control as well as maintenance, debugging, and portability among SoC-FPGA vendors. We describe a concrete implementation in order to show how data and instructions can be moved across the whole system.

**Keywords:** Hardware-software codesign · System-on-chip · Embedded software · FPGA design · Real-time systems · Reconfigurable virtual instrumentation · Data acquisition systems

## 1 Introduction

Modern complex electronic devices are characterized by the growing integration of different functional units such as multicore microprocessors ($\mu$P), random access memory blocks (BRAM), digital signal processors, and FPGA fabrics. The integration of these units in the same chip includes a high degree of internal


S. Saponara and A. De Gloria (Eds.): ApplePies 2021, LNEE 866, pp. 69–75, 2022.
https://doi.org/10.1007/978-3-030-95498-7_10

interconnection. Systems based on SoC-FPGA provide a great number of computational services, low latency responses and high throughput online data processing, making them very attractive, if not mandatory, for advanced instrumentation and specialized supercomputing infrastructures [1,2]. The huge amount of reconfigurable logic and computational resources allow the implementation of very complex systems, which often require remote control from a computer [3–5] through a standard communication link such as USB or Ethernet.

Despite the complexity of these heterogeneous systems, they can be described in a simple unified way by means of an abstract model [6]. It is then possible to implement a suitable hardware/software architecture along with a set of services and remote control activities.

Although there are some commercial solutions for remote control of systems based on FPGA [7,8], they have severe limitations such as closed sources, limited functionality, and no portability. Furthermore, most of the time SoC-FPGA based system developers implement their custom procedures for remote control. Both commercial and custom solutions are typically incompatible among them due to lack of standards. In order to cope with these problems we propose an open source hardware/software architecture for remote control of systems based on SoC-FPGA.

## 2 Hardware/Software Architecture

The proposed architecture follows a simple modular approach that foresees the encapsulation of certain aspects that are specific of the target devices, e.g., adopted standard communication links, operating systems, and external hardware.

This architecture requires the whole system to be abstractly represented by a set of functional blocks interconnected via ports associated with a unique address in a global memory map, divided into non-overlapping domains. The activity of such a system can be described by a set of concurrent data exchanges among its functional blocks. The movement of data from one place to another of the global system is described by a special instruction called *Universal Direct Memory Access (UDMA)*.

A generic UDMA instruction can then be expressed as follows:

$$\text{UDMA} <src\_addr> <dst\_addr> <src\_inc> <dst\_inc> <N>$$

where the associated parameters are respectively: the addresses of the source and destination, the address step increments at the source and destination, and the number of 32-bit words to be transferred.

Even though a global memory map abstracts some implementation details showing addresses contiguously, in general the corresponding functional blocks may be physically distant, belonging to different hardware components.

Special entities are in charge of executing the data movements. Each of these entities, called Local Resource Agent (LRA), has direct and exclusive access to one memory domain of the global map and can modify its content. All LRAs

are interconnected, have a unique ID, and exchange data among them by means of packets. Inside every LRA, a UDMA processor is in charge of executing the UDMA instructions.

In Fig. 1 the common packet structure is shown.

The header contains a common keyword that announces the start of packet for asynchronous communication, the protocol number, the packet type, the priority, and the source and destination IDs of the LRAs.

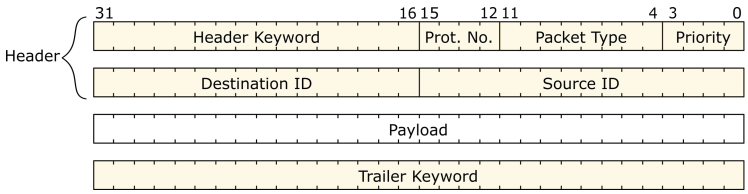


**Fig. 1.** Common packet structure.

Three essential packet types were identified for the basic operation of the system, as described below:

- *Command Packet*: It consists of a single word containing a code associated with a predefined activity (START, STOP, RESET, etc.) or error messages. It has a reduced size allowing a faster transmission and lower latency.
- *Raw data Packet*: This is the packet used for moving data among LRAs. It contains the data to be written and the destination-related part of a UDMA instruction. Given that the data may require more than one packet, indexing may be used to keep an order throughout the multiple transactions required to complete a data exchange. A data integrity check such a CRC and checksum is implemented for these packets.
- *UDMA Packet*: This packet contains a UDMA instruction which is passed to the UDMA processor inside the LRA. Depending on the source and destination, the instruction might trigger a cross-LRA exchange that will require a single or multiple raw data packets.

## 3 Single SoC-FPGA Based System

We consider a typical heterogeneous system [9] based on a single SoC-FPGA device connected on one side to a standard PC for remote control and user interface, and on the other side to external hardware, which in general will be specific to the application [2]. The FPGA, the $\mu$P and the PC offer different but complementary computational resources. Maximum performance can be achieved if the whole computational activity is distributed among these three subsystems taking into account their specific characteristics.

Figure 2 schematically shows a typical SoC-FPGA system with its control PC.

Following a modular approach, the SoC-FPGA can be seen as the combination of a $\mu$P and a FPGA, interconnected by vendor specific SoC bus.

The FPGA is subdivided into three main modules: (i) an external hardware controller, (ii) a Communication Block (ComBlock) [10], and (iii) the core FPGA design. These modules should have standardized interfaces to facilitate their interconnection and the communication among them. For the internal ports of the modules, the Wishbone (WB) standard bus interface [11] is proposed; the ComBlock is used to interact with the $\mu$P, abstracting the SoC bus complexities.

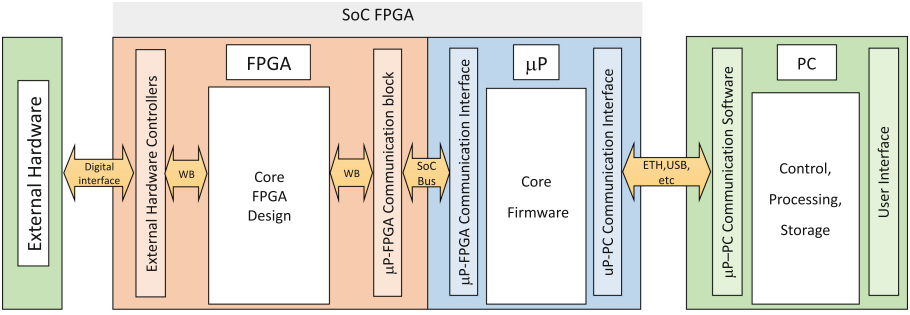


**Fig. 2.** Block diagram of a typical system based on SoC-FPGA.

Similarly, the implementation of the $\mu$P software separates the $\mu$P core program from the communication services with the PC and the FPGA. The $\mu$P relies on the UDMA firmware [12] for the communication with the PC and the FPGA. The PC resident software consists of a Python based Command Line Interpreter (CLI) to manage the communication and the interaction between the PC and the $\mu$P. The control software benefits from Python by relying on its scriptability and wide cross-platform support.

### 3.1 Implementation Example

To illustrate the proposed architecture we implemented a demonstrative system to move data among different components according to instructions generated in the PC. Figure 3 shows a simplified block diagram of this system composed by two LRAs: the PC and the SoC-FPGA.

Three type of memories were implemented in the FPGA: one BRAM, two FIFOs, and the True Dual Port Ram (TDPRAM) of the ComBlock. These memories along with the Python UDMA CLI assigned memory defined the global memory map. The PC and the $\mu$P communicated via packets over TCP-IP. Inside the FPGA, a *WB Interconnect* was used to interface the UDMA processor with the FPGA memory resources. The ComBlock registers were used by the

UDMA firmware to control the state of the UDMA processor. When a packet arrived from the CLI, the $\mu$P interpreted the header and extracted the payload. According to the type of the packet and its content, the $\mu$P performed different operations. It executed a command or a UDMA instruction, or it passed a UDMA instruction to the FPGA through the ComBlock should the affected memory domain lay on the FPGA subsystem. In this last case, the UDMA processor executed the instruction to move the data as prescribed and, once finished, it used the reserved registers to communicate to the UDMA firmware that the operation was successfully completed.

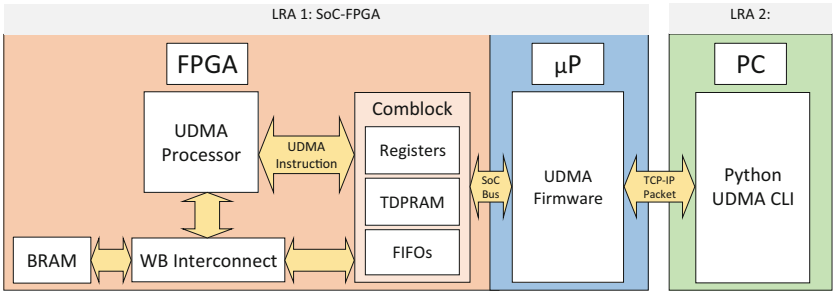


**Fig. 3.** Block diagram of the implemented system showing LRAs and inner blocks.

A test application was developed on the implemented system. The test consisted of writing data in the BRAM from the PC, and then verifying the written data. This was done by first sending a data packet from the UDMA CLI to the $\mu$P. The $\mu$P wrote the data in the ComBlock's FIFO and sent a UDMA instruction. Next, the UDMA processor interpreted the UDMA instruction and moved the data from the ComBlock's FIFO to the BRAM. To verify the success of the operation, a UDMA instruction was sent from the UDMA CLI to retrieve the data. The UDMA processor moved the data from the BRAM to the ComBlock's FIFO. Finally, the data was sent to the PC in a data packet by the UDMA firmware. With these mechanisms it was possible to arbitrarily move data between the instantiated memory resources.

The system has been successfully tested in two different FPGA based SoC development boards: the ZedBoard [13] and the CIAA-ACC [14]. The FPGA resources utilization of the *WB Interconnect*, the UDMA processor, and the ComBlock are shown in Table 1 for the CIAA.

The system has been developed to be multi-platform and communication protocol independent following the proposed architecture. Due to the UDMA processor encapsulation, the user can easily add more resources to the global memory map by just connecting them to the *WB Interconnect*.

**Table 1.** Resource utilization on a CIAA[a] (less than 1% of the total)

| | LUT | LUTRAM | Flip-flops | Slices | BRAM tiles |
|---|---|---|---|---|---|
| WB interconnect | 91 | 0 | 4 | 0 | 0 |
| UDMA processor | 193 | 0 | 506 | 87 | 1 |
| Comblock | 221 | 48 | 510 | 126 | 1 |

[a] The utilization on the Zedboard was practically identical.

## 4 Conclusions

The proposed hardware/software architecture has shown to be an effective solution for the remote control and debugging of systems based on SoC-FPGA devices. A reduced number of commands and instructions allows moving data across the entire system involving memory elements, microprocessors, reconfigurable functional blocks, and a standard computer for remote control.

The architecture modular structure also facilitates the porting of complex designs among different SoC-FPGA vendors and device families. The open source approach enables FPGA designers and embedded software programmers to benefit from the freely available IP blocks and software routines to implement their designs, saving valuable time in dealing with and debugging complex communication mechanisms such as those involving Ethernet connections and SoC-Buses.

The proposed approach seems to be suitable not only for advanced instrumentation but also for high performance computing based on multiple interconnected SoC-FPGA based platforms. The presented architecture is appropriate to efficiently exploit scalable platforms such as clusters of SoC-FPGAs.

## References

1. Cicuttin, A., Crespo, M.L., Mannatunga, K.S., Samarawickrama, J.G., Abdallah, N., Sabet, P.B.: HyperFPGA: a possible general purpose reconfigurable hardware for custom supercomputing. In: 2016 International Conference on Advances in Electrical, Electronic and Systems Engineering (2016)
2. Gazzano, J., Crespo, M., Cicuttin, A., Calle, F.: Field-Programmable Gate Array (FPGA) Technologies for High Performance Instrumentation. Advances in Computer and Electrical Engineering. IGI Global (2016). ISBN:9781522502999
3. Cicuttin, A., Crespo, M.L., Mannatunga, K.S., et al.: A programmable system-on-chip based digital pulse processing for high resolution x-ray spectroscopy. In: 2016 International Conference on Advances in Electrical, Electronic and Systems Engineering, pp. 520–525 (2016)
4. Mannatunga, K.S., Ali, S.H.M., Crespo, M.L., Cicuttin, A., Samarawikrama, J.: High performance 128-channel acquisition system for electrophysiological signals. IEEE Access **8**, 366–383 (2020)
5. Velmurugan, S., Rajasekaran, C.: A reconfigurable on-chip multichannel data acquisition and processing (DAQP) system for multichannel signal processing. In: 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering, pp. 109–114 (2013)

6. Mannatunga, K.S., et al.: Design for portability of reconfigurable virtual instrumentation. In: 2019 X Southern Conference on Programmable Logic (SPL), pp. 45–52 (2019)
7. National Instruments. CompactRIO Systems. https://www.ni.com/it-it/shop/compactrio.html (2020)
8. Opal Kelly. FrontPanel. https://opalkelly.com/products/frontpanel/
9. Crespo, M.L., Cicuttin, A., Gazzano, J., Calle, F.: Reconfigurable virtual instrumentation based on FPGA for science and high-education. In: Fagerberg, J., Mowery, D.C., Nelson, R.R. (eds.) Field-Programmable Gate Array (FPGA) Technologies for High Performance Instrumentation, chap. 5, pp. 99–123. IGI Global (2016)
10. ICTP MLAB and INTI CMNT. The Core Comblock. https://gitlab.com/rodrigomelo9/core-comblock (2021)
11. Free and Open Source Silicon Foundation. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores (2010)
12. ICTP MLAB. Universal Direct Memory Access - UDMA. https://gitlab.com/brunovali/udma (2021)
13. AVNET. ZedBoard. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/
14. INTI. CIAA-AAC A Open Hardware Card for HPC and Industrial Applications. https://github.com/ciaa/CIAA_ACC_Support