Design for Portability of Reconfigurable Virtual Instrumentation

Kasun S. Mannatunga
Department of Physics
University of Sri Jayewardenepura
Nugegoda, Sri Lanka
Email: ksm@sjp.ac.lk

Luis G.G. Ordóñez, Marie B. Amador,
Maria Liz Crespo, Andres Cicuttin
Multidisciplinary Laboratory
International Centre for Theoretical Physics
Trieste, Italy

Stefano Levorato
Istituto Nazionale di Fisica Nucleare
Trieste, Italy

Rodrigo Melo, Bruno Valinoti
Instituto Nacional de Tecnología Industrial
Buenos Aires, Argentina

***Abstract*— A portable architectural design strategy is described for the implementation of reconfigurable virtual instrumentation based on programmable Systems-on-Chip integrating microprocessors and FPGA in the same physical device. The key role is played by a general purpose communication block as a means to efficiently separate the activities carried out in the microprocessor and in the FPGA. Both parts interact according to simple logic protocols by reading and writing data on the common memory resources of the communication block. The architecture of the proposed communication system can be easily implemented in practically any modern programmable System-on-Chip. With the proposed strategy, the porting of embedded software programs and associated FPGA designs among different device families and vendors is facilitated. A structured methodology is proposed for handling complex real-time systems based on these programmable Systems-on-Chip. We described a concrete communication block that has been successfully implemented and utilized for a quick implementation of a data acquisition system based on a Xilinx Zynq-7030 FPGA Mezzanine Card (FMC) and a custom FMC module with an 8-bit 500 MSPS ADC.**

Keywords*— *Hardware-Software Codesign, System-on-Chip, Embedded Software, FPGA Design, Real-Time Systems, Reconfigurable Virtual Instrumentation, Data Acquisition Systems

I. Introduction

The availability of electronic devices which integrate tightly interconnected microprocessors (uP) and Field Programmable Gate Array (FPGA) fabrics in the same chip is creating new possibilities in the area of advanced instrumentation and high-performance reconfigurable computing. These versatile hybrid devices offer many extra resources and can be considered as true fully programmable Systems-on-Chip (SoC). Notwithstanding the rather obvious advantages of this important technological opportunity, its associated complexity poses several challenges to those developers who want to benefit from these devices fully exploiting its capacities and peculiarities [1][2].

The high circuit integration allows a level of interconnectivity between the uP and the FPGA cores that cannot be normally reached in printed circuit boards with equivalent discrete devices. Increased connectivity not only means more physical lines to exchange information but also an increment of the operation frequencies with reduced dynamic power consumption due to the characteristic much lower parasitic capacitances of the interconnecting traces in an integrated circuit. The high level of interconnectivity between an FPGA fabric and an embedded microprocessor is perhaps the main distinctive aspect of these hybrid devices.

It is very important to be able to easily retarget implemented systems to profit from newer or more powerful devices for upgrading or reutilization purposes. However, the intrinsic complexity of these devices limits the portability determining an important issue if we consider that in most cases, significant designing and programming efforts are necessary to come up with a satisfactory working system. It is widely recognized nowadays that the complexity of these kinds of hybrid devices makes quite difficult to port entire designs or even recycle functional blocks as it has reported by Kevin Morris in [3] when he states that "*It is extremely challenging to develop "generic" IP for processors and FPGAs working together that is independent of the particulars of the processor, the FPGA, and the interconnect schema*". The aim of this work is that of proposing a simple and effective design strategy to speed up the implementation of new reconfigurable instruments and to facilitate the porting of the corresponding embedded software programs and associated FPGA designs among different programmable SoC families and vendors.

The uP provides a streamlined connection with standard external hardware such as DRAM and Ethernet ports freeing the FPGA designer from implementing specific hardware controllers, which typically consume precious logic resources and are difficult to implement and debug. Since these and other external hardware are directly connected to the uP, it is possible to handle these resources by software. On another side, there is also an FPGA fabric where it is possible to implement custom digital circuits to perform time-critical tasks that cannot be executed by the uP when there are stringent requirements on latencies or throughputs. Since FPGAs typically have a great capacity for interconnection with the external world by means of numerous reconfigurable input/output ports, it is usually utilized to connect to non standard or custom external hardware. The most modern FPGAs integrate specialized hardware resources beside its reconfigurable and interconnectable logic elements. Among these special circuits, there are True Dual-Port RAM (TDPRAM) and digital signal processing units allowing the implementation of multiple channels for processing large amount of data at high rates. These characteristics make this kind of devices the ideal choice for the implementation of

978-1-7281-1363-0/19/$31.00 ©2019 IEEE

modern Reconfigurable Virtual Instrumentation (RVI) [4] where the whole instrument can be naturally divided into four main concatenated subsystems: (i) Instrument Specific Hardware (ISH), (ii) FPGA, (iii) uP[1] and (iv) Personal Computer (PC). On one extreme of the RVI chain, the ISH is typically a separate hardware that handles the external electrical signals of the instrument on one hand, and on the other hand, it is connected to the FPGA by mean of digital signals. The connection with the FPGA can be either on the same PCB or by mean of especial connectors (e.g. FMC, HSMC, PMOD, etc.). On the other extreme of the chain, there is the host PC running a Graphical User Interface (GUI) for the emulation of virtual consoles from which it is possible to control and operate the instrument [5]. The PC can also offer other services such as data storage, offline data processing, networking, and in general any other non time-critical activities. To implement a reconfigurable instrument based on a programmable SoC, an efficient software program is necessary for the embedded uP and a suitable hardware design for the associated FPGA such that these two central subsystems can effectively interact and cooperate to obtain the highest possible performance [6][7].

In the following sections, we present a simple architectural design strategy for the implementation of reconfigurable virtual instrumentation based on programmable SoC to facilitate reutilization and porting among devices of different families and vendors. A central role in the proposed architecture is played by a general purpose Communication Block (CB) which efficiently separates the activities carried out in the microprocessor and in the FPGA, and facilitates the interaction and exchange of data between these two subsystems.

We describe the CB and explain how it can be used for practically any kind of cooperative interaction between the uP and FPGA. We also describe a concrete implementation of a configurable CB, and its application to a high-speed data acquisition system based on Xilinx Zynq-7000 FMC carrier hosting a custom FPGA Mezzanine Card (FMC) [8] with an 8-bit 500 MSPS ADC. Some preliminary results are also presented as well as a discussion about the impact on portability in the implementation of reconfigurable virtual instrumentation based on modern Systems on Chip.

II. FPGA AND UP INTERACTION

The central idea for implementing complex real-time systems for reconfigurable virtual instrumentation is that of efficiently split the complexity among the three main subsystems: FPGA, uP and control PC. An important aspect is that of understanding the typical time granularity of activities carried out in each subsystem. For example, we could quite approximately say that nanoseconds, microseconds and milliseconds are typical reaction times for FPGA, uP and PC, respectively. A balanced distribution of activities will definitely determine an efficient allocation of resources with maximum impact on the overall performance and efficiency of the implemented system. In many cases, fast reaction times and high throughputs are required; it comes as no surprise then that the FPGA subsystem and its tight interaction with an embedded microprocessor are crucial for the ultimate performance of reconfigurable instruments based on a programmable SoC.

Once a complex functionality is properly partitioned and decided which part will be accomplished by the uP and which by the FPGA, it remains the problem of deciding how both subsystems will interact to ensure an effective cooperation. Different programmable Systems-on-Chips have specialized buses to interconnect the uP and FPGA subsystems. These buses include AXI, Avalon or AMBA [9] depending on FPGA vendor and device family. These buses are complex and despite a high level of automation of the electronic design tools, the designer usually needs to go through extensive documentation in order to get all necessary details for its correct utilization. Since the activity of these interconnecting buses involves both the uP and the FPGA, any debugging or refining processes will inevitably imply dealing simultaneously with the embedded program of the uP and the FPGA design making these important processes lengthy and laborious. The peculiarities of each bus standard will determine in most cases that both the uP software and the FPGA design will not, in general, be portable among different device families and vendors.

To facilitate the porting of complex designs implemented in hybrid devices integrating microprocessors and FPGA fabrics, we propose a Communication Block (CB) in such a way to offer a standardized interface towards the FPGA design that abstracts the uP and its specific SoC bus. The peculiarities and complexity of the bus will be hidden inside the CB as with any functional module interacting with the external world through its ports. With this CB, it will be possible to port a complex design by essentially porting the CB only. That is, the corresponding CB in the new device will preserve the interface towards the rest of the FPGA design while manages the new bus on the uP side. Along with this interface, a simple logic protocol can be implemented to provide a generic mechanism of interaction between the FPGA and the uP.

One of the main purposes of the proposed communication block is that of providing an effective abstraction of the interacting agents. This is, at the lowest level, the uP does not deal directly with the FPGA subsystem but with the CB independently of the FPGA or whatever is on the other side. Similarly, the FPGA subsystem only deals with the CB ignoring the uP or any other interacting agent on the other side of the CB. In this way, the use of the proposed communication block renders much more independent the programming of the uP and the digital design on the FPGA.

Besides portability, the use of the CB along with a simple protocol for its utilization naturally imposes a structured design methodology which shortens developing times and

[1] In the Xilinx terminology the subsystems are normally referred as Processing System (PS) for the uP and Programmable Logic (PL) for the FPGA. In Intel-Altera terminology the uP subsystem is called Hard Processor System (HPS) and in that of Microsemi-Actel is called Microcontroller Subsystem (MSS). Unless further clarification is needed, we will simply use the terms FPGA and uP to refer, respectively, to what is implemented in the FPGA fabric and the software programmed in the uP.

facilitates debugging, maintenance and possible optimizations of the implemented reconfigurable systems.

III. THE PROPOSED COMMUNICATION BLOCK

In a typical programmable SoC, the interaction between the uP and the FPGA subsystems is essentially accomplished by exchange of data, whereby data we mean any kind of digital information including numeric data, commands, error messages, system status, etc. Since the data produced by these subsystems are in general generated with unrelated clocks it is then necessary to count on memory elements that can be independently written and read from two mutually asynchronous domains. Typical elements with this characteristic are the TDPRAMs. These specialized blocks are now present in almost every modern FPGA and can be combined in many ways in order to produce memories of different widths and lengths. TDPRAMs can also be managed with special additional logic circuits to implement asynchronous First-In First-Out memories (FIFO). Another way of communication is by mean of registers implemented in the FPGA. All these registers can be read by both the uP and the FPGA, and can be implemented in such a way that some of them can only be written by the FPGA while the others can only be written by the uP. The proposed CB contains these three essential types of memory elements:

- True Dual Port RAM
- Asynchronous FIFO
- Register

The TDPRAM can be independently read and written from two different ports. These two ports can be mapped in two different memory maps, one for the uP and the other for the FPGA. While both agents, uP and FPGA, can independently read at any time any memory position, it should be avoided possible conflicting situations in case of writing operations. To prevent potential errors it must be avoided (i) trying to read a position from one port while that position is being written from the other port, an (ii) trying to write the same position from both ports simultaneously.

The asynchronous FIFO simplifies the transmission of sequential data. The two ports of the FIFO are mapped to single memory addresses. Who writes into the FIFO must only check that a *full-flag* is not asserted to prevent false writings, and similarly, who reads must only check that the *empty-flag* is not asserted to prevent false readings.

Finally, registers can be used to statically pass information, which is not time critical. Since the outputs of the registers are updated with an independent clock, care must be taken to avoid reading corrupted data from a circuit that operates with a different unrelated clock.

Each one of the three kinds of memory elements described above has advantages and disadvantages, and can be complementary used to efficiently allow any type of complex interaction between the uP and the FPGA. For example, all data stored in registers have the advantage that can be simultaneously accessed from the FPGA side, a feature not available in RAMs, which only allow access to its data one word at a time. However, RAM structures are more efficient than registers in terms of hardware resources to achieve larger and denser data storage. Also reading and writing sequential data is easier with FIFOs than with RAMs, but its disadvantage is that when we read a word from a FIFO, it is lost if not stored somewhere else, that is, while we can read infinite times the same data from a RAM, the data from a FIFO can be read only once. These three memory elements are combined to create a CB with two independent interfaces. The Fig.1 shows a block view of the CB with its two interfaces, one for the uP and the other for the FPGA, where prefix F2M stands for "from FPGA to Microprocessor" and similarly M2F stands for "from Microprocessor to FPGA". The ports of the interface towards the uP are mapped in the general memory map of the uP and will be managed following the specifications of the corresponding SoC bus.

The interface towards the FPGA subsystem is constituted by the native ports of the memory elements. The interface towards the FPGA can be easily assimilated or adapted to be compliant with the Wishbone bus interface standard [10] benefiting in this way from the compatibility with large open repositories of IP functional modules such as those of opencores.org.

The design of the FPGA starts structurally by instantiating the CB, and the rest of the design will be done considering that the interaction with the uP will be effectuated by dealing with the corresponding interface of the CB. The communication activity is then reduced to reading and writing the memory elements of the CB ignoring how the same operations occur on the same memory elements from the uP side. Similarly, the part of the program running in the uP and dealing with the resources of the FPGA will also be reduced to writing and reading the memory elements of the CB ignoring how the same operations occur on the same memory locations from the FPGA side. The proposed CB consequently provides a concrete mean to abstractly represent the interacting agents being these uP, FPGA, or whatever is capable of reading and writing the CB.

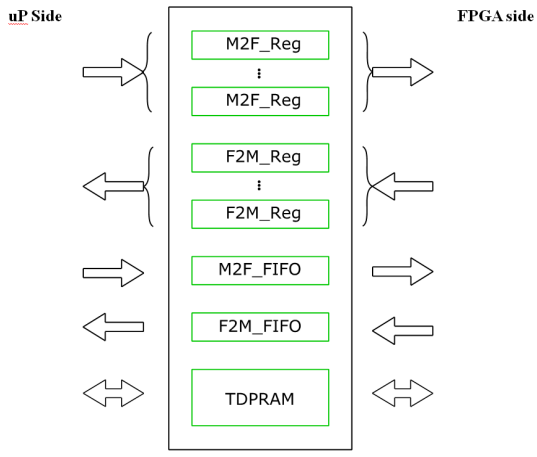

Fig. 1. A block view of the communication block.

Some memory resources of the CB can be reserved for an effective and safe communication. For example, let us suppose that we want to pass some data from the FPGA to the uP, and we want to safely use the TDPRAM for that purpose, then we can implement a simple logic protocol based on flags as described by the following steps:

1) FPGA checks if the TDPRAM is not taken by the uP by checking the flag "uP-TDPRAM-busy" (a predefined bit of a reserved register which can only be written by the uP). If it is set to '0' then:
2) FPGA sets to '1' the flag "FPGA-TDPRAM-busy" (a predefined bit of a reserved register which can only be written by the FPGA) to take the TDPRAM.
3) FPGA writes data in a non-reserved area of TDPRAM.
4) FPGA writes initial data address in a predefined reserved position of TDPRAM called F2M_DMA_ADDRESS.
5) FPGA writes the numbers of words of the transmitted data in a predefined reserved position of TDPRAM called F2M_DMA_N.
6) FPGA sets to '1' the flag "data-ready-for-uP" (a predefined bit of a reserved register which can only be written by the FPGA) to notify the uP that new data is ready in TDPRAM to be read by uP.
7) FPGA sets to '0' the bit "FPGA-TDPRAM-busy" to release the TDPRAM.

The uP simultaneously executes the following steps:

1) The uP checks the bit "data-ready-for-uP". If it is set to '1' then:
2) The uP checks the bit "FPGA-TDPRAM-busy". If it is set to '0' then:
3) The uP takes the TDPRAM by setting "uP-TDPRAM-Busy" to '1'.
4) The uP reads initial address in F2M_DMA_ADDRESS and number of words in F2M_DMA_N (both parameters in predefined positions of the reserved area of TDPRAM).
5) The uP reads F2M_DMA_N words starting from F2M_DMA_ADDRESS.
6) The uP sets "uP-TDPRAM-Busy" to '0' to release the TDPRAM concluding the transmission cycle.

A similar logic procedure is followed to pass data from uP to FPGA. With simple logic protocols, it is possible to avoid conflicting situations in dealing with the TDPRAM. The Fig. 2 shows the asynchronous timing diagrams of the flags-based protocol to transmit data through the TDPRAM of the CB.

Classical uP interrupts can be used by connecting some bits of the uP read-only registers to interrupt signals of the uP leaving intact the interface of the FPGA side. Whether the flags will be checked by the uP by polling or through interrupt mechanisms, it will not affect the behaviours of the FPGA preserving the global architecture. From the portability point of view, it is advisable to implement mechanisms based on polling instead of on interrupts. While polling mechanisms are independently decided by software, the availability of interrupts and the ways these interrupts are handled will

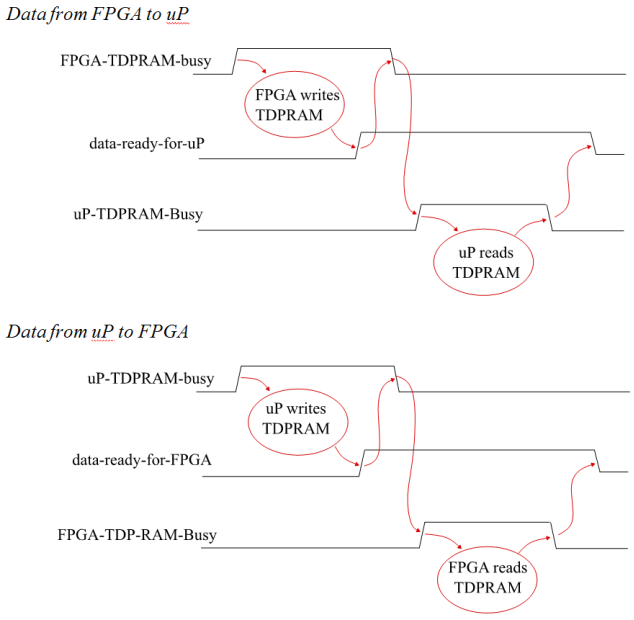

Fig. 2. The asynchronous timing diagram of the flags-based protocol to transmit data through the TDPRAM of the CB.

depend on the specific uP and eventually on the specific installed operating system.

The Fig. 3 shows a possible memory mapping of the CB ports from the FPGA and uP sides. These two different memory maps could be part of a much larger one into which it could be mapped many other resources of the whole system such as additional external memories or other external hardware resources physically connected to the FPGA. The ports on the FPGA side can be simply the corresponding native ports or could also be compliant with the Wishbone interface standard, and in both cases, the FPGA designer has total freedom in deciding how to connect and use them. In the FPGA side, it could be implemented several point-to-point connections and multiple buses for concurrent activities, or just a single simple bus for sequential access to all resources

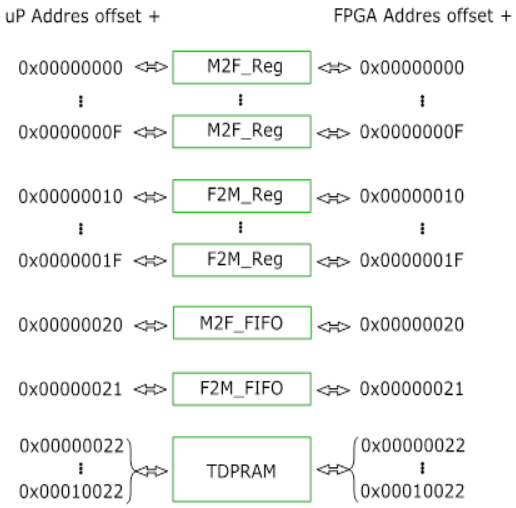

Fig. 3. A possible memory mapping of the CB ports.

of the CB.

The CB has been implemented as a configurable IP block to provide memory elements with simple interfaces for a typical FPGA designer avoiding the complexity of the SoC bus provided by the uP System, such as AXI in the Zynq-7000 family.

A first version of the CB has been implemented in Vivado Integrated Design Environment (IDE) using available IPs from Xilinx. To obtain a portable and configurable CB, a second version has been developed using VHDL 93 and memories of the FPGALIB [11] project (Asynchronous FIFO and TDPRAM, both of them tested with Xilinx, Intel/Altera, and Microsemi devices). Fig. 4 shows the wizard window to configure the Communication Block. The implemented CB provides:

- 16 input and 16 output registers (configurable up to 32 bits).
- One TDPRAM, which provides a simple RAM interface available on the FPGA side. Its inclusion, the data width, the address width and the memory depth can be configured.
- Two asynchronous FIFOs, one from uP to FPGA and another from FPGA to uP, with indication of empty/full, almost empty/full and underflow/overflow conditions. Their individual inclusion, the data width, and the memory depth can be configured.

An AXI Lite interface has been used for the registers, and AXI Full interfaces for the RAM and FIFOs to take advantage of burst operations.

TABLE I. and TABLE II. Show the resource utilization of the CM with the configuration shown in the Fig. 4, for some

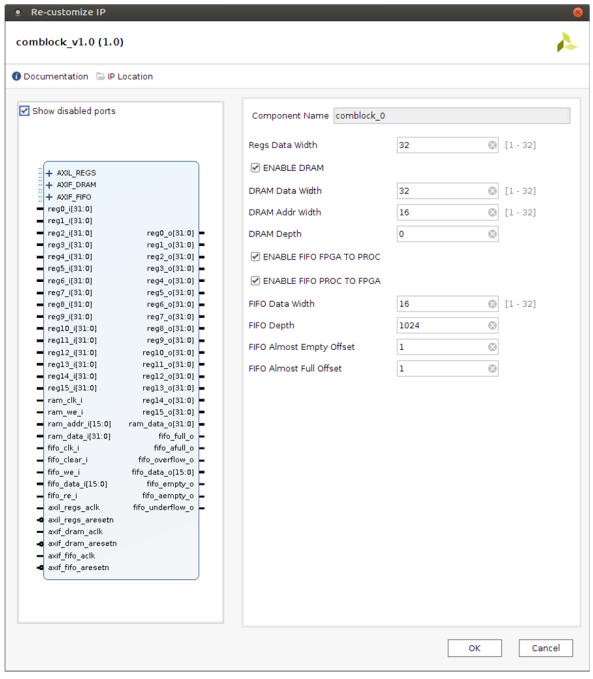

Fig. 4. Configuration wizard window for the Communication Block.

TABLE I. RESOURCE UTILIZATION OF THE CB IN XILINX FPGAS

Family	Name	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	Block RAM (kBytes)
Zynq-7000	xc7z020	726	816	128	64	290.25
Spartan-7	xc7s100	726	816	128	64	290.25
Artix-7	xc7a200	726	816	128	64	290.25
Kintex UltraScale+	xcku15p	854	816	128	64	290.25
Virtex UltraScale+	xcvu13p	854	816	128	64	290.25

TABLE II. RESOURCE UTILIZATION OF THE CB IN ALTERA/INTEL FPGAS

Family	Name	ALMs	Logic Registers	Block RAM (kBytes)
Cyclone V	5CSEMA5F31C6	764	563	290.25
Cyclone 10 LP	10CL006YU256A7G	1170	559	290.25
Cyclone IV GX	EP4CGX15BF14A7	1171	559	290.25
Cyclone IV E	EP4CE6E22A7	1170	559	290.25

of FPGAs in the Xilinx's FPGA families and the Intel FPGA families.

IV. Application to a High-Speed Data Acquisition System Based on Programmable SoC

In order to test and expose the proposed architectural design approach, a configurable communication block has been designed and used for the implementation of high-performance data acquisition instrumentation[2].

The modular hardware system is mainly composed by a Xilinx Zynq-7030 FMC carrier hosting a custom FMC ADC board with an 8-bit, 500 MSPS ADC (ADC08500, Texas Instruments).

The implemented system acquires a continuous data stream from the external ADC and performs oversampling and decimation by accumulating a variable number of input samples for every output value. In this way, it is produced a slower output data stream with a reduced effective sampling frequency but with increased amplitude resolution. This simple oversampling scheme allows gaining n_b bits of additional data amplitude resolution depending on the number R of accumulated samples used to generate one output sample according to the relation $R = 2^{2n_b}$. Thus, oversampling by a factor of R, will consequently produce an output data stream with a frequency decimated by the same factor. The decimated data will be analyzed to produce a histogram of the amplitudes during a predefined acquisition time, and at the same time, a continuous segment of the decimated data is transferred to the PC to be displayed along with the corresponding histogram.

[2] This system is being developed in the framework of R&D projects in collaboration between the International Centre for Theoretical Physics (ICTP, UNESCO-IAEA) and the Italian National Institute of Nuclear Physics (INFN).

As explained before, the whole activity of the reconfigurable data acquisition system is divided among the four main subsystems: dedicated hardware, FPGA, uP and control PC. While the FPGA subsystem is in charge of time-critical tasks, the uP subsystem is responsible for handling the communication between the PC and the FPGA, and other no time-critical tasks. The complete system comprises sensors of pressure and temperature, which are directly managed by the uP without the intervention of the FPGA. The temperature and pressure values are periodically read and processed by the uP for monitoring and slow control purposes. A GUI has also been developed in Python (PyQT) for a remote control of the system from a PC. The uP and the PC are interconnected through a dedicated point-to-point Gigabit Ethernet link.

A. Custom FMC Data Acquisition Board and SoC-based FMC Carrier

The hardware of the data acquisition systems is essentially constituted by a custom FMC ADC board coupled to an FMC carrier based on a programmable SoC.

The custom FMC ADC card is based on single channel 8-bits 500 MSPS ADC and has been designed for high time-resolution measurements. The ADC demultiplexes its digital data output to diminish the reading frequency by a factor of two. This board fully exploits all features of the ADC including self-calibration, fine adjustment of input full-scale range and offset, and multiple ADCs synchronization. The digital output data are driven by 32 physical lines implementing 16 Low-Voltage Differential Signaling (LVDS) pairs at 250 MHz. There are other three LVDS signals: an input clock, an output clock, and a fast "out-of-range" signal. There are also other 7 digital control signals that complete the digital interface of the ADC. These 45 digital signals are connected to a Low-Pin-Count FMC connector.

The adopted FMC Carrier is the CIAA-ACC[3] [12] based on a Xilinx Zynq-7030 device. This carrier has been designed for high-performance computing and advanced industrial applications and includes one FMC-HPC Connector, DDR3 (1GB) memory, and one Gigabit Ethernet connector. Fig. 5 shows the ADC card mounted on the FMC carrier.

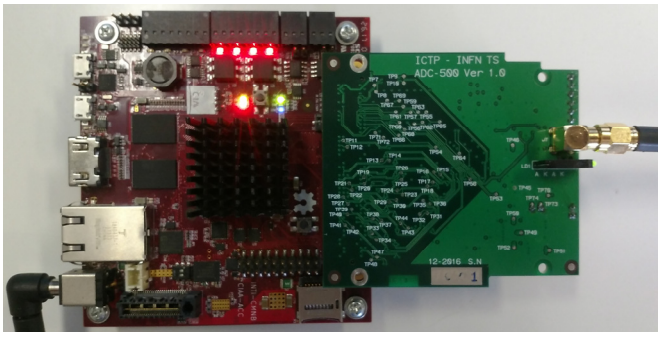

Fig. 5. ADC card mounted on the CIAA-ACC FMC carrier.

B. System Design and Implementation in the Programmable SoC

A top description of the design of the programmable SoC device is structured in three main blocks: the uP, the CB, and the specific FPGA design. The uP interacts directly with the CB by mean of AXI bus connections, while the specific FPGA directly interacts with the CB by mean of the native ports of the memory resources of the CB. Fig. 6 shows the top level schematic entry that is a block diagram of the three main components and its logical interconnections. In red are the connections of the uP (left), in blue the connections between the CB (center) and the specific FPGA design (right), and in brown the connections of the FPGA with the external hardware (the FMC ADC card).

The corresponding logical representations of these interconnections are depicted in Fig. 7 where it is shown the addresses assignment of the memory resources of the CB to be

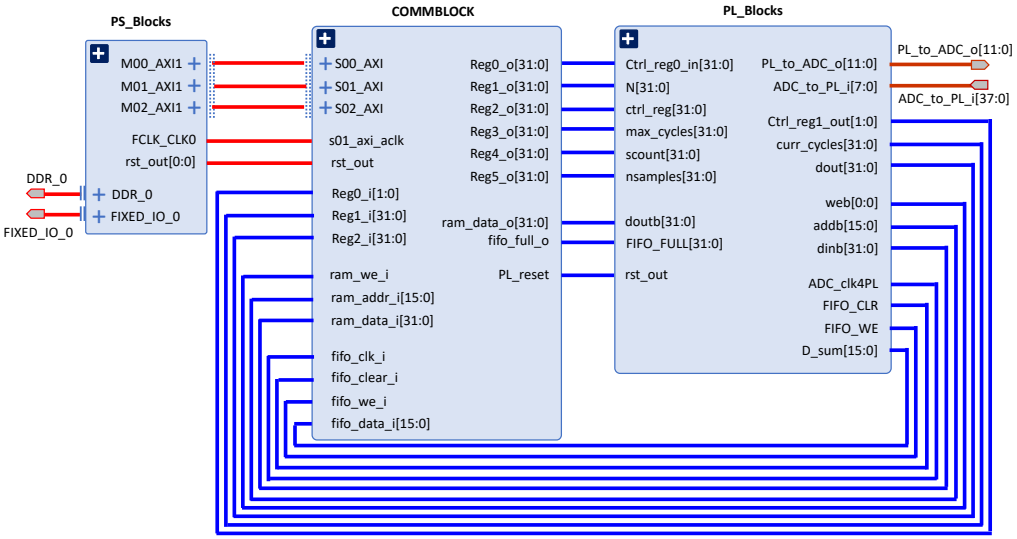

Fig. 6. Top level schematic of the Communication Block (centre) and its connections (red) with the uP-subsystem (left) through the AXI bus, and the connections (blue) with the FPGA-subsystem (right) through the native interfaces of the CB components. The connections to the dedicated external hardware are in brown.

[3] This open hardware FMC carrier has been developed by the Center of Micro and Nanoelectronics of the National Institute of Industrial Technology (INTI, Argentina)

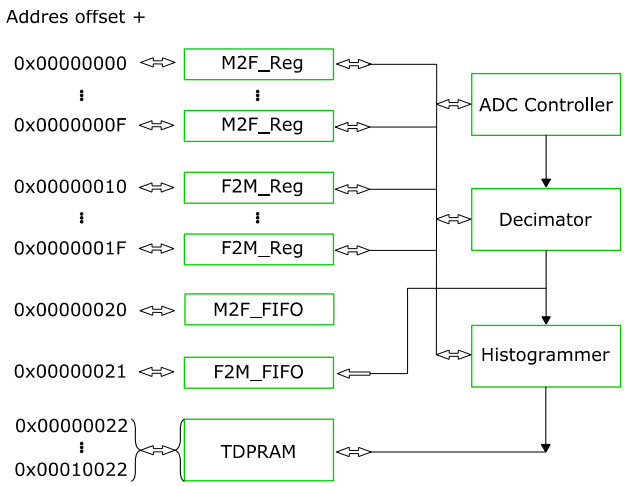

Fig. 7. The resources of the communication block and the memory mapping on the uP side, and its connections with the functional blocks implemented in the FPGA.

accessed by the uP through the AXI bus, and the direct connections with some specific functional blocks of the FPGA design through native ports of the CB on the FPGA side.

Once instantiated the CB, the rest of the FPGA design can be done practically ignoring the AXI bus and the uP can be programmed practically ignoring the implementation details of the specific FPGA design. We can see, for example, that functional blocks such as the ADC Controller or the Decimator can receive configuration parameters directly from the registers of the CB. Thus the uP can configure these modules by simply writing to the memory addresses of the corresponding registers. We can also see that the output of the decimator can be dumped into a FIFO through a native port disregarding how the uP reads that FIFO through its AXI port on the other side. The Histogrammer block also uses the TDPRAM of the CB through its native port on the FPGA side to store and build the histogram in real time.

The histogram can be read by the uP at the end of the accumulation period or at any time during its accumulation by mean of an AXI access from the uP side of the TDPRAM mapped in a certain range of the uP memory map.

The FPGA subsystem handles the external ADC and process the input data stream, and stores the results in the CB from where the uP retrieves the corresponding data. The uP prepare custom packets and send them to the control PC through the Ethernet port. The uP runs a real-time operating system (FreeRTOS) to grant predictable timing responses and to facilitate the executions of concurrent tasks including data transmission through the TCP/IP protocol. The control PC hosts resident software that receives the custom data packets, inspects the contents and displays the data in a GUI. From the GUI, it is also possible to send parameters and commands to the uP which in time may pass them to the FPGA by mean of the CB. A special functional block implemented in the FPGA takes the decimated data stream and generates a histogram of amplitudes in real time by accumulating the amplitudes falling in bins of predefined size during a variable acquisition time. A segment of the decimated data stream is also captured by the FPGA and transmitted to the microprocessor for transmission to a personal computer for visualization and eventual further elaboration.

The uP retrieves the trace from the CB and temporary stores it in the external RAM of the uP. A different task of the uP retrieves that trace from the RAM and sends it to the PC as a special data packet through an Ethernet port.

For a comparison, the same system implemented with IPs in the Vivado IP integrator instead of the communication block. Since the Vivado IP integrator does not have registers with AXI bus, AXI GPIOs were configured to use as registers and, an AXI BRAM controller had to use with the Block Memory Generator in order to use FPGA true dual port RAM. Obviously, there are few places that can be improved which need quite extensive knowledge of the Vivado tools and Xilinx methodology. FPGA resource utilization of the high-speed data acquisition system is shown in TABLE III. along with the resource usage without the communication block.

It has been identified that percentage usage of slice LUTs and slice registers in the system without CB is considerably larger than the values of the system with CM.

V. Discussion and Conclusions

An FPGA configurable communication block has been proposed and described to facilitate design portability of reconfigurable instruments based on programmable SoC integrating microprocessors and FPGAs. The main purpose of the CB is that of offering a simple standardized logic interface in such a way that the FPGA and uP subsystems can smoothly interact independently of their respective implementation details. This by itself not only increases the portability of the system across multiple device families and vendors, but at the same time, it imposes a structured design methodology by mean of an explicit separation of the work in the uP and

TABLE III. SUMMERY OF RESOURCE USAGE OF THE HIGH-SPEED DATA ACQUISITION SYSTEM WITHOUT AND WITH THE COMMUNICATION BLOCK

Name	Without Communication Block				With Communication Block			
	FPGA_ Subsystem	uP_ Subsystem	Others	Total	FPGA_ Subsystem	uP_ Subsystem	Communication _Block	Total
Slice LUTs	187	960	6691	7838 (14.73%)	186	1466	834	2486 (4.67%)
Slice Registers	264	1244	12696	14204 (13.35%)	264	1957	884	3105 (2.92%)
F7 Muxes	0	62	0	62 (0.23%)	0	62	128	190 (0.71%)
F8 Muxes	0	0	0	0 (0%)	0	0	64	64 (0.48%)
Block RAM Tile	0	0	66.5	66.5 (47.5%)	0	0	65	65 (46.43%)

FPGA domains. Most of the complexity related to the specificities of the SoC bus is hidden inside a general purpose and reusable communication block. The definitions of the interfaces are simple and independent of the internal structure of the block which can be implemented in a variety of functionally equivalent versions.

By maintaining the interfaces of the CB it is possible to reuse or port the rest of the FPGA design retargeting a different SoC device. In this case, it will be necessary to port the CB taking into account the specific characteristics of the new processor, and its corresponding SoC interconnect bus. The porting effort is then mainly limited to work inside the communication block granting the compatibility with the software program of the uP, and the logical design implemented in the FPGA.

The communication between the FPGA and the uP subsystems can be implemented following the main ideas of the OSI reference model [13] and can be done at three different hierarchical levels, where each level corresponds to a communication layer. Each layer relays on the services offered by the immediate layer below, and provides services that can be used by the layer immediately above. These three layers are:

- **Physical**: Allows a simple utilization of the resources of the CB as storage elements.
- **Logical**: Includes a basic asynchronous logic protocol for a safe utilization of the CB. It requires some reserved areas in the TDPRAM and some reserved registers.
- **Systemic**: Implements a high-level protocol based on a set of complex instructions for Direct Memory Access (DMA). It provides transparent access from any domain to all resources mapped in a global memory mapping, including those that are not immediately accessible but that are directly accessible from the other domain. It requires a DMA machine in the FPGA, and a corresponding software routine in the uP.

Since each subsystem deals with the other subsystem by reading and writing in the memory locations of the communication block, this block provides an abstract view of one subsystem to the other interacting subsystem.

Typical buses of programmable SoCs are mainly conceived to grant communication to the uP, and hence it may be a bottleneck when a great amount of data must be simultaneously moved among interacting functional blocks implemented in the FPGA independently of the uP. By restricting the use of SoC bus to handle the CB only, it is avoided any suboptimal design that utilizes the SoC bus to interconnect cooperative functional blocks implemented in the FPGA. The FPGA designer will then have plenty of freedom to interconnect all functional blocks with any interconnection topology and strategy according to what is needed independently of the SoC bus. For strict communication purposes between uP and FPGA it makes no sense to provide more resources than what is necessary for the maximum data exchange rate that the processor can handle since, in general, even a small FPGA can host designs, which can easily produce huge data rates compared to what a normal processor can handle. The proposed communication block and related utilization mechanisms should not constitute a restriction on the maximum achievable data throughput between FPGA and uP, and should neither increase latency, which in most cases is determined by the uP and its SoC bus. The implicit design approach associated with the use of a CB makes quite independent the programming of the uP and the design works on the FPGA, since these two subsystems only need to agree on the logical utilization of the CB.

VI. ACKNOWLEDGMENTS

Support from the OFID Postgraduate Fellowship and TRIL Programmes at ICTP and from the ICTP/IAEA Sandwich Training Educational Programme is gratefully acknowledged.

REFERENCES

[1] Dondo Gazzano J., Crespo M.L., Cicuttin A., Rincon Calle F. (eds). 2016. Field-Programmable Gate Array (FPGA) *Technologies for High Performance Instrumentation*. A volume in the Advances in Computer and Electrical Engineering (ACEE) book series, ISBN: 9781522502999, ISSN: 2327-039X, IGI Global.

[2] Crespo M.L., Cicuttin A., Dondo Gazzano J., Rincon Calle F. 2016. *Reconfigurable Virtual Instrumentation based on FPGA for Science and High-Education*. Field-Programmable Gate Array (FPGA) Technologies for High Performance Instrumentation, Section 3, Chapter 5, pp. 99-123, DOI: 10.4018/978-1-5225-0299-9.ch005.

[3] Kevin Morris, *Intel Delivers Xeon Scalable Processor 6138P with Arria 10 GX 1150 FPG. Ratchets Up FPGAs in Data Center*. Electronic Engineering Journal, May 29, 2018. [Online]. Available: http://www.eejournal.com/article/intel-delivers-xeon-scalable-processor-6138p-with-arria-10-gx-1150-fpga

[4] Jeevitha L., Sangeetha S., Arun pandiyan S., *Design and Implementation of Reconfigurable Virtual Instruments with User Defined Functionality*, International Journal of Computer Sciences and Engineering, Vol.2, Issue.5, pp.57-60, 2014.

[5] A.Cicuttin, M.L.Crespo, A.Shapiro, N.Abdallah, *Building an Evolvable LowCost HW/SW Educational Platform--Application to Virtual Instrumentation*, Proceedings of International Conference on Microelectronic Systems Education, San Diego, USA, pp.77-78, 2007. *IEEE Xplore* Digital Library 10.1109/MSE.2007.26.

[6] Cicuttin A., Crespo M.L., Abdallah N., Bazargan P., Mannatunga K., Samarawickrama J. (2016). *HyperFPGA: A possible general purpose reconfigurable hardware for custom supercomputing*. Proceedings of ICAEESE 2016, 14-16 November 2016, Putrajaya, Malaysia; *IEEE Xplore* Digital Library.

[7] Crespo M.L., Cicuttin A., Mannatunga K et al. (2016). *A Programmable System-on-Chip Based Digital Pulse Processing for High Resolution X-Ray Spectroscopy*. Proceedings of ICAEESE 2016, 14-16 November 2016, Putrajaya, Malaysia; *IEEE Xplore* Digital Library.

[8] Xilinx Zynq-7000 All Programmable SoC. Technical Reference Manual UG585 (v1.12.1), 2017.

[9] AXI Reference Guide UG1037 (v4.0), 2017 - AVALON Interface Specifications, Intel Quartus Prime Design Suite: 17.1, 2018 - AMBA Specification, Rev. 2.0, 1999.

[10] Specification for WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision: B.3, 2002.

[11] INTI CMNB. FPGA Lib. [Online]. Available: https://github.com/INTI-CMNB-FPGA/fpga_lib.

[12] CIAA-ACC, Open Hardware Card for HPC and Industrial Applications. INTI-CMNB and others. [Online]. Available: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:ciaa_acc:ciaa_acc_inicio.

[13] Paul Simoneau, *The OSI Model: Understanding the Seven Layers of Computer Networks*, 2006 Global Knowledge Training LLC.