A framework for the integration of the development process of Linux FPGA System on Chip devices

A.Rigoni [a], G. Manduchi[a], A. Luchetta[a], C. Taliercio[a], T. Schröder[b]

[a] *Consorzio RFX (CNR, ENEA, INFN, Università di Padova, Acciaierie Venete SpA), Padova, Italy*
[b] *Max-Planck-Institut für Plasmaphysik, D-17491 Greifswald, Germany*

System on Chip is a hardware solution combining different hardware devices in the same chip. In particular, the XILINX Zynq solution, implementing an ARM processor and a configurable FPGA on the same chip, is a candidate technology for a variety of applications of interest in fusion research, where FPGA fast logic must be combined with CPU processing for high-level functions and communication. Developing Zynq based applications requires the development of the FPGA logic using the XILINX Vivado IDE, mapping information between the FPGA device and the processor address space, developing the kernel drivers for interaction with the FPGA device and developing the high level application programs in user space for the supervision and the integration of the system. The paper presents a framework that integrates all the above steps and greatly simplifies the overall process. The framework has been used for the development of a programmable timing device in Wendelstein 7-X. The development of new devices integrating data acquisition and timing functions is also foreseen for RFX-mod.

Keywords: FPGA, System on Chip, ADC, Timing Systems.

1. Introduction

The use of FPGA based solutions in control and data acquisition systems (CODAS) for nuclear fusion devices has been in the past rather limited if compared to other physics experiments such as accelerators. This fact is mainly due to the different requirements: while in accelerators it is necessary to handle a very large amount of fast events from detectors, requiring fast data reduction on the fly based on coincidences, in fusion experiments a lower number of channels is used, typically requiring the acquisition of input signals for data storage and possibly real-time control. Therefore, fusion experiment use more conventional electronic devices such as transient recorders, replaced in recent long lasting experiments by Analog to Digital (ADC) devices supporting a continuous output data stream. Moreover, the dynamics of the phenomena controlled in real-time, such as plasma stability, require in most cases a response time of the order of milliseconds, whereas the control of the fastest phenomena such as vertical stabilization in tokamaks require a response time of the order of 100 µs. These requirements can be satisfied using the current computer technology making therefore the use of general purpose computers preferable over specialized FPGA solutions. Developing FPGA solutions requires in fact skills and expertise in the Hardware Description Languages (HDL) and hardware interfaces. Considering also that the integration of custom FPGA systems in CODAS normally requires developing some kind of specialized communication protocol, the amount of required human resources to implement such solutions is often unaffordable, especially in small laboratories. For this reason, FPGA solutions have been in the past limited to specific applications in diagnostics [1,2]. A notable exception is certainly represented by the RIO FPGA architectures [3] (Compact RIO and Flex RIO) which provide an easy FPGA programming and integration via LabVIEW and have been widely adopted for plasma control [4] and other diagnostic applications [5]. This solution, proposed by National Instruments, aims at leveraging the power of FPGA by removing the main barriers in their usage, that is the expertise required in HDL programming and interfacing with the rest of the system. This solution is however quite expensive and closed to the specific choice in hardware and in the programming environment. A new modern approach for the integration of high-level software components with the power of the FPGA logic design is obtaining growing attention in the market of embedded technologies and exploits the System on Chip (SoC) solution that combines different hardware devices in the same chip. The main hardware competitors leading the SoC FPGA market are Intel/Altera and Xilinx, both proposing almost the same development solutions but with their own proprietary software. In particular the XILINX Zynq architecture [6], implementing an ARM processor and a configurable FPGA in the same chip, is a valuable candidate technology for a variety of applications of interest in fusion research, where FPGA fast logic can be combined with software functions carried out by a CPU for high level functions and communication.

A considerable number of heterogeneous hardware from many vendors have been released profiting of the high integration of SoC devices. The main advantages that these chips brings to the programmable logic are the possibility to interface and share hardware features that are typical of a complete system such as the DMA controller and external interfaces like Ethernet or SATA.

Many software solutions have been also proposed, to guide the developer through the non-trivial mechanisms of the FPGA to system interfacing, as well as covering

author's email: andrea.rigoni@igi.cnr.it

different programming approaches: from low level synthesis of Verilog and VHDL hardware description, to the higher level toolchains that compile real programming languages like SystemC OpenCL and others [7,8].

In this paper we present yet another choice named *Anacleto* (Another auto config for logic evaluation toolchains), particularly targeted to the GNU Linux embedded devices, that has been developed by RFX consortium and aims at proposing a unified standard workflow to the FPGA developer for programming both the logic and the software components in a uniform and portable way.

It is worth noting that in the development of Anacleto SoC projects the knowledge of HDL, unlike other solutions such as the National Instrument RIO LabView interface, is not hidden by the framework. The aim of this framework is indeed not to provide a new programming interface, adding another layer of logic, but to ease the development process with established well known open-source build tools. In this way Anacleto can be a way to access the low level machinery of the FPGA programming easily and uniformly, and a much cheaper solution in respect of RIO. Moreover, at this low abstraction programming level, many of the features that are usually involved are already provided free of charge by the chip vendors or with a reasonable license fee by external contributions, keeping a door open to a wide market of existing solutions.

The first candidate applications for SoC devices are timing systems, data acquisition preprocessing and fast computation. Timing systems represent a classical field of applications for FPGAs and have been implemented both in custom systems [9] and commercial products [10]. A typical timing application uses a synchronization clock signal distributed, normally via fiber optic, to all the timing devices and possibly propagating asynchronous events. The FPGA provides the generation of the required timing signals (clocks, triggers, …) based on current configuration loaded in the system using some kind of hardware interface such as PCI. A processor would introduce in this case more flexibility in the management of the configuration, letting, for example, the configuration be uploaded via the network.

Integrating configurable FPGAs in data acquisition would provide much more flexibility in data management introducing features not currently supported by ADC devices. An example is the possibility of managing deferred triggers communicated via network. Using the network to communicate triggers in data acquisition introduces delays that may compromise the precision in the reconstruction of the acquired signal. However, if a trigger message also carries the exact trigger time, and assuming that all devices have a precise knowledge of time (e.g. using IEEE 1588 timing protocol), it is possible to provide a correct reconstruction of the signal using an internal circular buffer maintaining a signal history lasting at least the delay in trigger communication [11]. The use of a configurable FPGA in data acquisition could also allow a significant reduction of the required front end when integrated signals from electromagnetic probes are acquired. In this case it would be possible to avoid analog integration before data acquisition moving integration to FPGA processing during acquisition.

Fast computation carried out by FPGA allows using more sophisticated algorithms in real-time plasma control retaining at the same time the flexibility provided by a computer system. The same approach could be used for new data processing algorithms such as feature detection from acquired video frames. In this case the processor would supervise data transfer and the FPGA would carry out intensive computing for feature detection. It is worth stressing the fact that FPGA solutions are more difficult to develop in respect of CPU based ones, and therefore the latter is preferred, provided it can satisfy the required timing constraints. As a rule of thumb, CPU based solutions should be considered when the order of magnitude of the required reaction time of the system is 100 μs or larger. Shorter times normally require FPGA implementations, however other factors may affect the choice, such as memory access issues that may reduce performance regardless the computational power, as happens also in large distributed computation carried out by General purpouse Graphical Processor Units (GGPUs) [12]

As for other FPGA solutions, SoC systems require skills and experience. For example, developing Zynq based applications requires (1) the development of the FPGA logic, (2) mapping information between the FPGA device and the processor address space, (3) developing the kernel driver for interfacing user software and the FPGA device and (4) developing the high level software applications in user space for the supervision of the system and its integration in the central CODAS. For this reason we have implemented a framework that integrates the above steps. The framework, described in the next section, makes the overall process easier, especially the integration of the FPGA components and the processor by coordinating all the required tools and by providing a set of templates that can be adapted to the specific application.

2. Framework components

Anacleto uses the Autotools [13] build infrastructure to organize the most general FPGA workflow acting like a standard toolchain compilation led by GNU make targets. The development process remains quite complex because many components in the final device board must be orchestrated (i.e. the kernel configuration, the customization of drivers to handle the newly created device, and so forth) but nevertheless the compilation is managed almost in automatic manner and, once the project is properly defined, all the steps are covered by Makefile targets that can be chained in a single make run. In order to develop a SoC application, it is necessary firstly to select the hardware system. Because we decided to make use of the Xilinx Zynq devices, as a first attempt, three low-cost solutions have been considered: RedPitaya [14], ZedBoard [15] and

Parallella [16]. RedPitaya is intended to be used as a stand-alone system for handling digital and analog I/O signals. This board hosts ready to use ADC and DAC components and therefore could result best suited for developing small self-contained applications, but for the same reason it shows a reduced flexibility in respect of the other two for the configuration of the I/O pins. The other boards are intended to be hosted in a carrier board and therefore mount no additional I/O devices. In particular, Parallella is targeted towards computing intensive applications and hosts an additional processor with 16 cores.

Several other software components, all free of charge, are required for developing a SoC application an deploying it into the target board. First of all, it is necessary to download from XILINX the Integrated Development Environment (IDE) tool VIVADO for HDL programming (Verilog and VHDL are the supported languages). In order to be used on a specific target, VIVADO requires a target-specific configuration, provided by the board developer, which specifies how the processor is configured in that particular board. Currently only Red Pitaya configuration is managed in the framework, but it is foreseen that configuration files from Zed Board and Parallella will be included, adding the choice of the target board in the configuration steps. VIVADO provides a set of configurable Intellectual Property (IP) components that carry out the connectivity between the processor (dual core ARM Cortex A9 in the Zynq chip mounted on Red Pitaya) and the FPGA application. When no DMA is involved, communication between the processor and the FPGA application is carried out by a configurable number of 32 bit registers and, optionally, one or more interrupt lines. When the developer creates a new project for a FPGA application, the IDE creates a set of IP components, carrying out the handshaking with the internal bus (AXI bus) used to exchange information between the processor and the FPGA application. The IDE provides the definition of a set of 32 bit signals that can be used by the FPGA application for communication. In a typical use case, such signals will represent the configuration to be uploaded to the FPGA application, but they can be used to exchange input and output data as well. After developing the specific application, the IDE will generate the binary code to be downloaded into the FPGA. Other configurable IP components provided by VIVADO allow the definition of up to two DMA channels for FPGA applications handling data streaming.

The configuration of the interface, i.e. number of shared registers, interrupt lines, and DMA channels must also be reflected in the device memory map of the processor. XILINX provides a github project hosting an adapted version of Linux kernel 4.4 in a Debian distribution. The project includes the toolchain for ARM processor and the kernel sources and a tool for the generation of the device-tree structure, used by Linux Kernel 4.4 for device abstraction [17]. Basically, a device-tree description provides information about the connected devices including memory addresses and size of the device registers. In this case, the device-tree description will include the registers used to communicate with the FPGA application. The specific device-tree for the Zynq chip is generated from the current VIVADO project by the Hardware Software Interface component belonging to the XILINX system development kit. The same component can generate templates for Bare Metal implementation, Linux and FreeRTOS. In particular the Linux driver template is generated within the framework based on the selected data transfer type (mapped registers and/or DMA).

Once Linux and the corresponding device-tree have been built, the final step is the development, starting from the generated template, of a Linux device driver that will allow user programs interact with the FPGA application. In the simplest configuration, a buffer in user space is mapped against the sets of registers defined in the FPGA application so that information is exchanged by reading and writing that buffer.

From the above description, it is clear that building a SoC system from scratch is not an easy task, despite the availability in the web of all required tools. The presented framework integrates all the above steps and greatly simplifies the overall process. In particular, the framework:

- Supervises he compilation of the toolchain and the Linux kernel using the components taken from the XILINX repository;
- Handles the management of the VIVADO project and the required IP components for FPGA integration;
- Supervises the construction of the device-tree required for the proper mapping of the FPGA registers into processor address space;
- Provides templates for the development of the required Linux drivers.

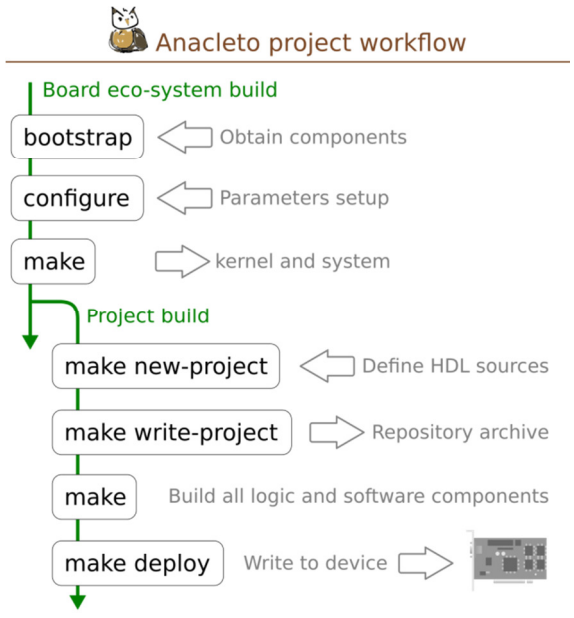

Figure 1: steps in building a new system.

Using this framework, a SoC FPGA application can be built from scratch by executing the general steps described in Figure 1:

As shown in the schema, the overall workflow can be splitted in two main stages: the building of the board system comprising the operating system kernel and software, and the specific project building with the definition of the logic and the software drivers that compile against the built kernel. The same board system can be shared among different specific projects and many different projects can be also installed in the same board. The reported steps depict a possible procedure example through the development process a developer would follow, that is:

1) Clone the framework from the github repository at: [*https://github.com/mildstone/anacleto*]
2) **boostrap** command that will set-up the environment and download all the required components and tools;
3) **configure** to set-up the system before building the toolchain and the Linux Kernel; this also compiles and shows a graphical user interface that eases the selection of the required options.
4) **make** to compile the toolchain and then the Linux Kernel;
5) **make new-project** to start a new VIVADO project for the development of the specific FPGA components in VHDL or Verilog. All the IP components required for SoC interface are created and can now be configured using VIVADO graphical interface;
6) **make write-project** once the logic has been defined with external sources and VIVADO project block designs, the whole project definition can be stored in the repository in the form of a script able to regenerate the project from scratch, even using different versions of VIVADO.
7) At this point the FPGA application can be developed. It is also necessary to write the Linux driver for communication and a skeleton Linux driver source file is generated by the framework, based on the current configuration of the interface IP components. Then **make** starts both the logic synthesis and the compilation of software components.
8) **make deploy** to generate the device-tree, compile the Kernel module, download the kernel and the bitstream into the target device.

Figure 2 shows the blocks generated by the VIVADO tool when a new SoC project is created. The top left block defines the processor; the bottom left block defines reset logic and the block in the middle defines the bus logic. The top right block hosts specific FPGA firmware (the timing device in this case) and it is connected to the bus logic block via the AXI bus. Other modules can be defined as well, all connected to the same AXI bus. These modules can then be adapted to connect the interface registers to the specific FPGA firmware.

3. Implemented and foreseen applications

The presented framework has been used to develop a general purpose timing device to be used in Wendelstein 7-X diagnostics. The timing device is implemented in a Red Pitaya board and defines two digital outputs to generate clock and gate signals, and two digital inputs to receive a synchronizing 10 MHz clock and a trigger signal. The board is configured via software to generate a pre-programmed timing sequence after the system has been armed and a trigger input signal has been received. The timing sequence is communicated via TCP/IP to the ARM processor hosted in the Zynq chip of the Red Pitaya board. In this case a set of registers have been defined as interface between the processor and the FPGA application, without using interrupt lines. All registers except one are used to specify the time sequence. The remaining register is used as command register to arm and disarm the board. Not considering the time required for developing the FPGA application written in VHDL the creation of the new project, the adaption of the driver from the template and the deploy required less than one working day.

The same github repository used to host the framework components has been used to host the timing board project and it is foreseen that all the new developed projects will be hosted there.

We are currently considering the usage of Zed Board

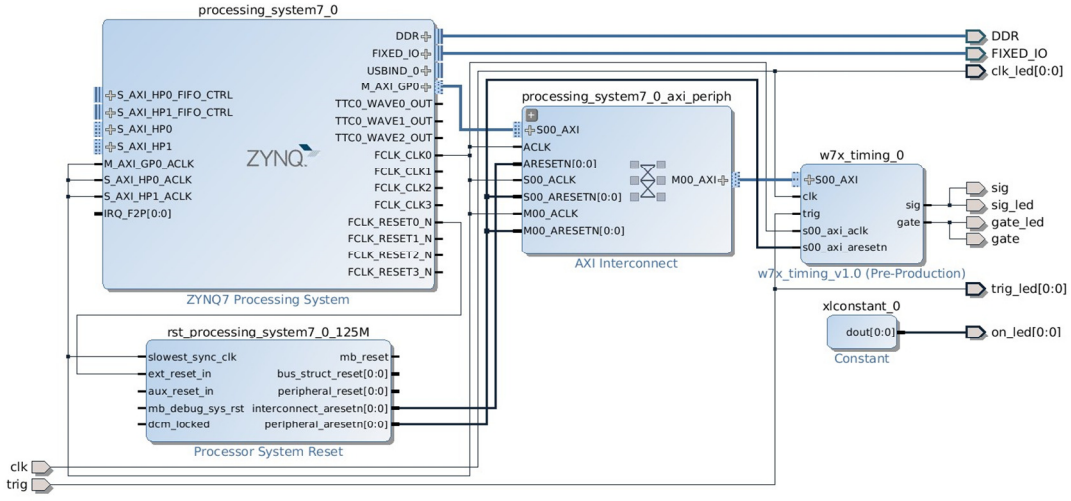

Figure 2: Blocks generated by the VIVADO tool in a SoC project.

for developing a new fast ADC to be used to acquire electromagnetic probes in the upgrade of the RFX-mod experiment currently in construction at Consorzio RFX [18]. In this experiment a large (~1000) number of electromagnetic signals is foreseen, where a configurable subset will be used for real-time control. All the acquired signals will be stored at full speed (up to 1 MHz) during the pulse in RAM memory and will be read after the discharge, using the traditional transient recorder organization (RFX-mod carries out short duration discharges), but at the same time a subsampled version of the signal will be made available for real-time control. For this purpose, 16 channels from a fast and insulated ADC, already used for vertical stabilization at JET [19], will be connected to the digital inputs of the SoC board. Each ADC channel will provide a 50 MHz signal carrying the serialized bits of every sample (ADC conversion is performed at 18 bits). The FPGA firmware will de-serialize the input channel, send the samples to RAM via a DMA channel, perform digital filtering for subsampling, and send subsampled data at 10 kHz via the second DMA link to the processor that will send samples via UDP to the real-time control system.

4. Conclusions

The SoC architecture proved to be effective in removing the "knowledge barrier" that prevents FPGA development in several fusion laboratories, especially when FPGA solutions are not strictly required to achieve requirements. The applicability of the SoC architecture has been further improved by the presented framework, hiding to the developer several intermediate steps and exposing only the necessary information for the proper system configuration. The presented framework however leaves to the developer the whole responsibility of the FPGA firmware development, a task that requires ingenuity and experience. There are however several applications of interest in fusion that don't require sophisticated FPGA firmware, as it has been the case of the presented timing board. This class of applications is likely to benefit from SoC architecture, especially when the set of required tools and configurations is transparently managed by a tool like the one presented here.

Another promising field of applications for SoC architectures is the possibility of moving critical computation into FPGA, leaving software to supervise the overall management of computation. This approach is already exploited in the RIO architecture for LabVIEW applications and may lead to extremely performing real-time systems that retain the flexibility of computer-based solutions, but allow moving time critical inner loops into FPGA firmware.

References

[1] R. C. Pereira et al. "Pulse Analysis for Gamma-Ray Diagnostics ATCA Sub-Systems of JET Tokamak" IEEE Trans. Nucl. Sci , 58, 4 (2011), pp. 1531 - 1537.

[2] S. Hernandez-Montero, J. A. Lopez, M. Sanchez, L. Esteban "Real Time FPGA-Based Crosstalk Elimination for Multichannel Interferometry Systems in Fusion Diagnostics" Real Time Conference (RT), 2012 18th IEEE-NPSS.

[3] National instruments, Compact RIO platform http://www.ni.com/compactrio/

[4] L. Giannone, M. Cerna, R. H. Cole, D. Schmidt "Data acquisition and real-time signal processing of plasma diagnostics on ASDEX Upgrade using LabVIEW RT" Fus. Eng. Des. 85, 3, pp. 303-307

[5] M. Ruiz, J. Vega, G. Ratta, E. Barrera, A. Murari, J. M. López, G. Arcas, R. Meléndez "Real Time Plasma Disruptions Detection in JET Implemented with the ITMS Platform Using FPGA Based IDAQ" IEEE Trans. Nucl. Sci , 58, 4 (2011), pp. 1576 - 1581.

[6] XILINX all Programmable SoC with Hardware and Software Programmability: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[7] Razvan Nane et al. "A Survey and Evaluation of FPGA High-Level Synthesis Tools", IEEE Trans. Comp., 55, 10 (2015), pp 1591 - 1604

[8] XILINX: Cloud Acceleration for RTL, C/C++, and OpenCL: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[9] J. Schacht, J. Skodzik, and the CoDaC Team "Multifunction-Timing Card lTTEV2 for CoDaC Systems of Wendelstein 7-X", IEEE Trans. Nucl. Sci , 62, 3 (2015), pp. 1187-1194

[10] DIO4 timing generator/digital I/O module: http://www.incaacomputers.com/products/by-function/digital-io/dio4/

[11] C. Taliercio, A. Luchetta, G.Manduchi, A. Rigoni "Distributed continuous event – based data acquisition using IEEE 1588 synchronization and FlexRIO FPGA"" IEEE Trans. Nucl. Sci , , 64, 7 (2017)

[12] T.J. Maceina, G. Manduchi "Assessment of General Purpose GPU Systems in Real-Time Control", IEEE Trans. Nucl. Sci, 64, 6 (2017)

[13] GNU AutoConf Introduction: http://www.gnu.org/software/autoconf/autoconf.html

[14] RedPitaya home page: https://redpitaya.com/

[15] ZedBoard home page: http://zedboard.org/

[16] Parallella home page: https://www.parallella.org/

[17] Device Tree specification: https://www.devicetree.org/

[18] M.E. Puiatti et al "Extended scenarios opened by the upgrades of the RFX-mod experiment" 26th IAEA Fusion Energy Conference (2016)

[19] A. Batista, A. Neto, M. Correia, A.M. Fernandes, B. Carvalho "ATCA Control System Hardware for the Plasma Vertical Stabilization in the JET Tokamak" IEEE Trans. Nucl. Sci , 57, 2 (2010), pp. 583 - 588.