EMBEDDED CONTROL SYSTEM FOR PROGRAMMABLE MULTI-PURPOSE INSTRUMENTS

M. Broseta, J. Avila-Abellan, S. Blanch-Torné, G. Cuní, D. Fernández-Carreiras, O.Matilla, J. Moldes, M. Rodríguez, S. Rubio-Manrique, J. Salabert, X. Serra-Gallifa,
ALBA Syncrothron, Cerdanyola Vallès, Spain

Abstract

At the ALBA's Computing Division, we have started the development of a high-performance electrometer, the ***Em#*** project, as a versatile and full customizable equipment. It is based on a SPEC board (simple PCIe FMC carrier) with customizable FMC cards and an SBC (Single Board Computer), altogether built in a single cost-optimized instrument. The whole device is designed to provide a wide range of functionalities to fulfill unique and complex experiments by means of configuration changes instead of having specific instruments.

Within the controls software development group, we started the development of a full embedded control software, based on a Linux OS that communicates with the SPEC's FPGA using the PCIe bus. This approach enables the integration of complex operations and functions in real time to higher software layers, as well as the local control, setup and diagnostics via an integrated touch-screen display controlled by the I2C protocol. For a user-level control, the system provides an SCPI API (Standard Commands for Programmable Instruments) allowing an easy integration to any control system. This paper describes the design process, main aspects of the data acquisition and the expected benefits during the integration in the Control System.

INTRODUCTION

High accuracy low current readout is an extensively demanded technique used in 3rd generation synchrotrons. They comprise a need both for diagnostics and data acquisition in today's photon labs. In order to tackle the problem of measuring from various sources of different nature and magnitude synchronously, while remaining flexible at the same time, ALBA developed years ago a 4 independent channel electrometer, the Em, based on trans-impedance amplifiers with high resolution ADC converters integrated and an Ethernet communication port.

The new ***Em#*** is the evolution of the 4-channel electrometer Em widely used in at ALBA since 2011. This new product solves a few limitations and provides new functionalities to make it more versatile and customizable.

ELECTRIC DESIGN

The ***Em#*** project uses a Simple PCI Express FMC Carrier (SPEC [1]) board to command a FPGA Mezzanine Card (FMC), designed to work as a 4-channel ADC and transfer the processed data to a Single Board Computer (SBC), the Intel NUC DE3815 [2], using its high-speed serial computer expansion bus (PCIe).

The SPEC, it is a cost-optimized design developed by the CERN under the Open Hardware Licence (OHL) that mainly works as FMC carrier. It is powered by Xilinx Spartan 6 FPGA [3] for custom gateware designs using High-speed serial connectivity, a PCIe interface and an FMC slot.

This hardware configuration allows ***Em#*** implement its own FPGA control code, to manage a 4-channel ADC converter in the FMC card. The communication and data sharing with the main control software in the SBC are also part of the control code routines implemented in the FPGA. But apart form the SPEC and SBC boards, there are other hardware boards included in this equipment. Figure 1 shows the hardware diagram block of the ***Em#*** project.

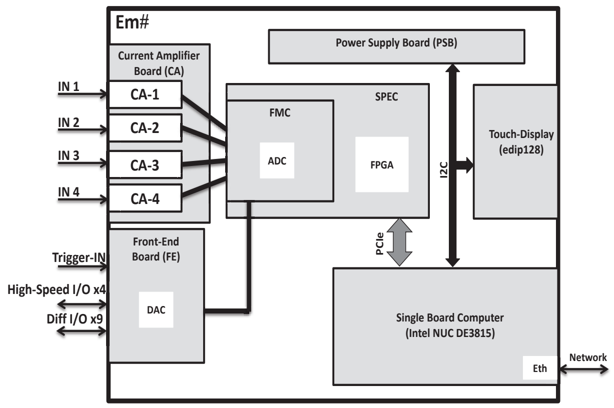


Figure 1: Em# hardware diagram block.

The Current Amplifier board (CA) contains the circuitry needed to communicate and control the 4 current amplifiers (CA-X). The Front-End board (FE) manages the trigger input and the different IO ports (4 High-Speed I/O ports and 9 Differential I/O ports). The Power Supply Board (PSB) supplies the equipment with different voltages needed by each module or board. A Touch-Screen (edip128) monitors the status and lets a general configuration of the equipment.

SOFTWARE DESIGN

The software development has been divided into three software projects, all together distributed in a single software package:

- The Linux OS
- The gateware (FPGA software)
- The main control software in the SBC (ALIN)

The ***Em#*** features a complex embedded control software based on Linux OS. For that reason, a light Embedded Linux has been customized specially for this equipment, with only the necessary drivers to control the available hardware and with enough functionality to run the required applications. This Embedded Linux has been created using Yocto [4], an open source collaboration project that provides a set of recipes, tools and methods that allow building custom Linux-based systems to be embedded, regardless of the hardware architecture. Among the different recipes or set of recipes used for this project, these are the most significant ones:

- meta-e3815: Recipes to build the Embedded Linux OS distribution for the Intel Atom Processor E3815, used for this project.
- meta-spec: Main recipe used to compile and install the Spec Linux drivers in the corresponding Linux kernel selected
- meta-python: This is the set of recipes that provides python support.
- meta-alba: Contains the set of recipes to install the main control software (ALIN) and its drivers in the SBC

The gateware software [5] runs in the SPEC FPGA. It is written in VHDL and the design focuses on fast acquisition and data sharing between the different modules inside the FPGA as well as with the software running in the SBC. The binary, is reprogrammed in the FPGA every time the system boots. Figure 2 shows the FPGA block diagram.

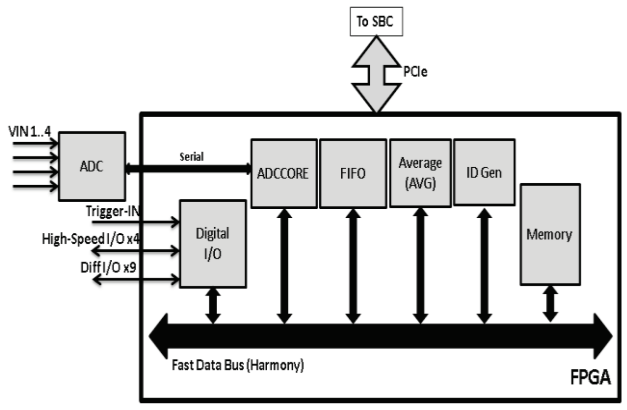


Figure 2: FPGA block diagram.

The ***Em#*** has been designed to acquire data at 400KSamples/second per channel. The main software is not fast enough to get the 4-channels data acquired at that sample rate via PCI bus. Therefore, the acquisition is carried out by the gateware software through a fast data acquisition bus, designed and implemented to share data between the different FPGA blocks. That fast data bus name is the Harmony Bus. Acquired data is stored in an FPGA memory block. The frames sent through this bus contains: the ID of the block which generates the frame, the data and the timestamp which indicates when the data was generated. The main software in the SBC configures the different acquisition types and reads the acquired data stored in the FPGA memory. Other slow and low priority data such as the information displayed in the touch-screen, are also transmitted through the PCI bus

In the SBC resides and runs the main software (**ALIN**) that has been designed aiming for high versatility in the application design and easy user control of the equipment. Versatility means that the software is easily adaptable to new features, just modifying the configuration of the FPGA, or to hardware changes. **ALIN** is a multipurpose software customized to work as an electrometer. Regarding the easy user control, it offers both remote and local control interfaces. Remote control is available via telnet (using the SCPI protocol [6]) or via web through a webserver. Local control is available through navigation menus using the touch-screen display.

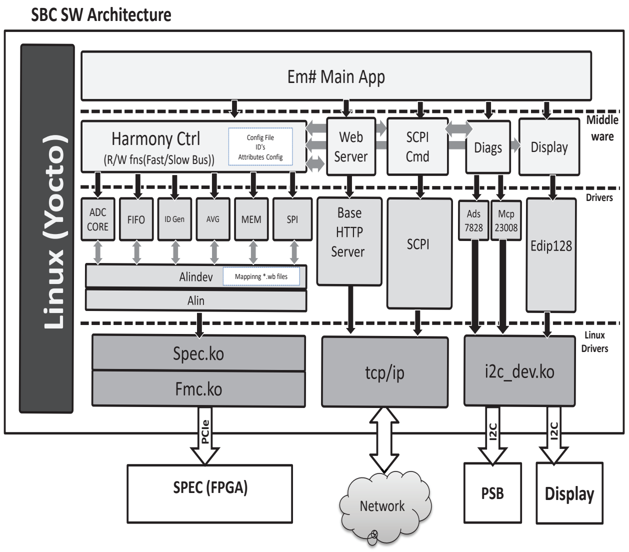


Figure 3: SBC software architecture.

Written in Python has the profits from the clean and straightforward syntax, while still performing well. The Figure 3 shows the software architecture of ALIN divided in different functional layers. The communication between modules in different layers is always from top to bottom, except for the middleware layer where a cross communication is allowed to share data between the different modules. The different layers are, from top to bottom:

- Applications: The main application is in this layer. It allows the equipment control, remote or locally. It starts when the equipment boots and it is responsible to initialize and configure the middleware modules that contain the control and diagnostics functions.
- Middleware: In this layer resides the logic that makes this equipment work as an electrometer, or as any other equipment. It also provides the functionality to interact with the equipment through the touch-screen, or through a communications port.

ISBN 978-3-95450-189-2

- Drivers: Software modules to control physical or logical devices are in this layer. There is a driver for each FPGA module, a driver for each I2C device or a driver for the SCPI that contains the protocol to remotely control the equipment.
- Linux drivers: At the bottom, there are the Linux drivers needed for the Em# project. They have been previously compiled for the Embedded Linux distribution. Examples are the SPEC driver to communicate with the SPEC board via PCI or the I2C driver to control the display and the Power Supply Board (PSB).

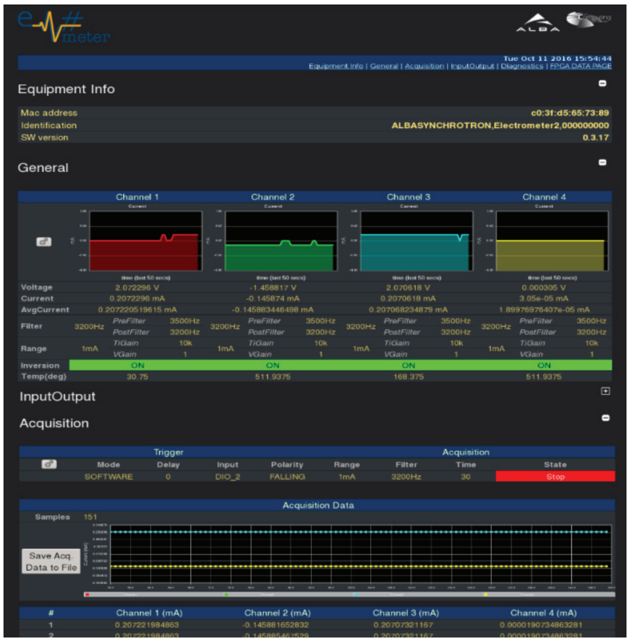
In the left-bottom side of Figure 3, the **Spec** and **FMC** Linux drivers [7, 8] are the kernel modules to control the SPEC and FMC cards via PCI bus. These Linux drivers are used by others like **alin** and **alindev** which implement read/write operations on the FPGA using the Self Describing Bus (SDB) [9]. This is a framework that helps to self-detect and manage the FPGA contents. It describes a series of structures used to provide metadata about the FPGA logic blocks, allowing the main software to automatically discover and configure them at runtime, via PCI bus. These SDB data structures are divided in records of 256 bytes size, where the first 64-byte are common between records and provide information about the FPGA blocks like the type of record, product, vendor, name, date, version and also the first-last address of the virtual memory space where the block data is located.

On top the alin drivers; there are specific drivers for each FPGA block. They provide functions to initialize write default values or read/write registers. These drivers use an external file that contains the register mapping. This file is auto generated using the same definition file which is used to define the FPGA block in the gateware software.

In the middleware layer, the **Harmony Control** middleware module uses these drivers to implement the electrometer ***Em#*** functionality. It configures the acquisition that will go through the Harmony bus in the FPGA, starts/stops the acquisition and process the acquired data in the FPGA memory. Acquired data and configuration parameters are shared to the rest of the middleware modules. External configuration of some predefined main software parameters, it is possible through a configuration file, which is loaded every time the system starts or after user request, by a command execution.

Remote control of the equipment is possible through a simple set of ASCII commands. These user control commands are implemented following a standard protocol for programmable instruments; the SCPI protocol. SCPI commands are ASCII textual strings that can contain one or more keywords, many of which take parameters. Responses to query commands are typically ASCII strings. The **SCPI middleware** module contains the list of control and configuration commands and their associated read/write call-back Em# functions. The **SCPI driver**, used by this middleware module is where the SCPI protocol is implemented.

The ***Em#*** also includes a web server for remote monitoring and overall control. The web offers a general equipment status, the current and voltages values read from the 4 channels, the configuration of the channels, the status and configuration of the acquisition, the last data acquired, the status and configuration of the 16 I/O ports, general diagnostics, etc. The **Webserver middleware** starts/stops the server using the **Base HTTP Server** [10] python library. The Webserver middleware is also responsible to gather the information to be shown in the web client, generating periodically a *Javascript Object Notation* (JSON) file [11]. Figure 4 shows the typical applications running in the browser client. It is a JavaScript application that uses jQuery library [12] to read the contents of the JSON file and keep the web contents updated. In the opposite way, when a parameter is modified in the browser client, it executes a PHP code in the server side. The webserver middleware captures the data send in the POST PHP method and then executes the corresponding call-back command in the main application.


e#meter
ALBA
Equipment Info | General | Acquisition | InputOutput | Diagnostics | FPGA DATA PAGE
Save Acq. Data to File

Tue Oct 11 2016 15:54:44

Equipment Info

Mac address	c0:3f:d5:65:73:89
Identification	ALBASYNCHROTRON,Electrometer2,000000000
SW version	0.3.17

General

	Channel 1			Channel 2			Channel 3			Channel 4		
Voltage	2.072296 V			-1.458817 V			2.070618 V			0.000305 V		
Current	0.2072296 mA			-0.145874 mA			0.2070618 mA			3.05e-05 mA		
AvgCurrent	0.20722051961 5 mA			-0.145883446498 mA			0.207068234879 mA			1.89976976407e-05 mA		
Filter	3200Hz	*PreFilter*	3500Hz	3200Hz	*PreFilter*	3500Hz	3200Hz	*PreFilter*	3500Hz	3200Hz	*PreFilter*	3500Hz
		PostFilter	3200Hz		*PostFilter*	3200Hz		*PostFilter*	3200Hz		*PostFilter*	3200Hz
Range	1mA	*TiGain*	10k	1mA	*TiGain*	10k	1mA	*TiGain*	10k	1mA	*TiGain*	10k
		VGain	1		*VGain*	1		*VGain*	1		*VGain*	1
Inversion	ON			ON			ON			ON		
Temp(deg)	30.75			511.9375			168.375			511.9375		

InputOutput

Acquisition

Trigger				Acquisition			
Mode	Delay	Input	Polarity	Range	Filter	Time	State
SOFTWARE	0	DIO_2	FALLING	1mA	3200Hz	30	Stop

Acquisition Data

Samples 151

#	Channel 1 (mA)	Channel 2 (mA)	Channel 3 (mA)	Channel 4 (mA)
1	0.207221984863	-0.145881652832	0.20707321167	0.0000190734863281
2	0.207221984863	-0.145885467529	0.20707321167	0.0000190734863281

Figure 4: Web application in browser client.

It is also possible to do a more specific remote control of the FPGA through the tools available via SSH. There is a handful set of tools that help designers to check/configure the status of the FGPA, to get the SDB structures, write binary file or read/write to the FPGA devices or to configure some general parameters of the equipment

Local equipment control is possible through to the touch-screen display. The **Display middleware** module allows the user control through navigation menus, to locally configure or check the status of the equipment via the touch-screen display. The display is programmed by

ISBN 978-3-95450-189-2

means of a protocol of high-level language graphic commands via I^{2}C [13]. The **edip128** implements the communication protocol with the display.

ALIN also includes functions to control and self-detect its own diagnostic status. That is done in the **Diagnostics middleware** module. Check the harmony bus stability, self-detect the consumption of the power supplies in the Power Supply Board (PSB) board, among other tasks are some of the diagnostics examples done by this module. To control the PSB board is done using the **Ads7828 (ADC)** and **Mcp23008 (Port-Expander)** I2C drivers.

CONCLUSIONS

There are already many customizable business solutions in the market that include an FPGA, a CPU and an FMC connector, all together on the same equipment. The ***Em#*** project tries to take the advantage of such solutions while offering a reduced cost. The software project has been designed modular to adapt to any other similar hardware approach, occasionally needing few changes in the driver modules or different configuration files. The control toolkit has been also designed to ensure an easy integration into any control system, regardless of the framework used for the control.

Direct memory access (DMA) is currently not supported in this design and therefore FPGA data is read by the NUC through virtual memory spaces via PCIe. That could be a problem due to the main software not being fast enough to meet the desired acquisition time of 400 KSamples per second. Instead, it can be considered as an advantage because the final solution applied (a FPGA memory block, the Harmony bus and the dynamic ID's) allows keeping a completely separate functionality between FPGA and NUC. The configuration and control resides in the NUC while the data acquisition is mainly in the FPGA. The result is that functionality of the ***Em#*** is not limited to work only as an electrometer but can be adapted and extended with other flavours in the control applications domain.

REFERENCES

[1] Simple PCIE FMC Carrier (SPEC). OHWR website: `http://www.ohwr.org/projects/spec/wiki`

[2] Intel NUC DE3815, `http://www.intel.eu/content/www/eu/en/nuc/nuc-kit-de3815tykhe-board-de3815tybe.html`

[3] Spartan-6 FPGA Data Sheet, `https://www.xilinx.com/support/documentation/data_sheets/ds162.pdf`

[4] Yocto Project, `https://www.yoctoproject.org`

[5] X. Serra *et al.,* "A Generic Fpga Based Solution for Flexible Feedback Systems", ALBA-CELLS Synchrotron, presented at PCaPAC16, Campinas, Brazil, Oct 2016, paper FRFMPLCO06, this conference.

[6] "Standard Commands for Programmable Instruments", European SCPI Consortium, May 1999.

[7] A. Rubini, "SPEC Software Support", CERN, February 2014.

[8] A.Rubini "FMC Bus Abstraction for Linux", CERN, February 2014

[9] A.Rubini, W. Terpstra, M. Vange, "Self Describing Bus (SDB) – Specification for Logic cores – Version 1.1", April 2013

[10] Guido van Rossum and the Python development team, "The Python Library Reference (Release 2.7.12)", Python software Foundation, September 2016.

[11] Introducing to JSON, `http://www.json.org`

[12] JQuery 1.9 documentation, `http://www.jquery.com`

[13] SMBus/I2C Protocol, `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/Documentation/i2c/smbus-protocol`