This document is published in:

Fusion Engineering and Design (2012), 87 (11), pp. 1872- 1879.
DOI: http://dx.doi.org/10.1016/j.fusengdes.2012.09.013

EPICS based low-level radio frequency control system in LIPAc

Julio Calvo [a,*], Mark L. Rivers [b], Miguel A. Patricio [c], Angel Ibarra [a]

[a] Centro de Investigaciones Energéticas Medioambientales y Tecnológicas, Ciemat, Spain
[b] Department of Geophysical Sciences and Center for Advanced Radiation Sources, The University of Chicago, USA
[c] Departamento de Informática, Universidad Carlos III de Madrid, Spain

HIGHLIGHTS

- The system proposed can control amplitude and phase of each cavity.
- Rapid diagnostics are refreshed in milliseconds.
- Increasing control parameters will not increase consumed time neither complexity.
- IQ demodulation can be achieved thanks to the transformed values at driver level.

ARTICLE INFO

Keywords:
EPICS
LIPAc
LLRF
Control System

ABSTRACT

The IFMIF–EVEDA (International Fusion Materials Irradiation Facility – Engineering Validation and Engineering Design Activity) linear accelerator, known as Linear IFMIF Prototype Accelerator (LIPAc), will be a 9 MeV, 125 mA continuous wave (CW) deuteron accelerator prototype to validate the technical options of the accelerator design for IFMIF. The primary mission of such facility is to test and verify materials performance when subjected to extensive neutron irradiation of the type encountered in a fusion reactor to prepare for the design, construction, licensing and safe operation of a fusion demonstration reactor (DEMO). The radio frequency (RF) power system of IFMIF–EVEDA consists of 18 RF chains working at 175 MHz with three amplification stages each. The low-level radio frequency (LLRF) controls the amplitude and phase of the signal to be synchronized with the beam and it also controls the resonance frequency of the cavities. The system is based on a commercial compact peripheral component interconnect (cPCI) field programmable gate array (FPGA) board, provided by Lyrtech and controlled by a Windows host PC. For this purpose, it is mandatory to communicate the cPCI FPGA board from EPICS Channel Access [1]. A software architecture on EPICS framework in order to control and monitor the LLRF system is presented.

1. Introduction

Paper is organized as follows: firstly the introduction. The second section, system architecture, consists of a description of the hardware and software components and a discussion of the synchrony. System operation, which describes the behavior of some parameters, some system functionalities, initialization of the IOC and the user interface, is located in the third section. The paper ends with a summary and future work exposure.

The RF system is defined as the equipment necessary to convert the high-voltage alternating current (AC) primary power to suitably conditioned RF power for input to the LIPAc accelerator cavities [2]. The quality of the RF delivered to the accelerator cavities is controlled to within ±1 degree in phase and to within ±1% in amplitude, using a low-level RF-drive modulated control system.

Each RF module local control system (RF module-LCS) is a device that will monitor and control all physical parameters within the RF chains located in the same RF module. Each RF module comprises 2 RF chains, so the 18 RF chains will be monitored and controlled by 9 RF module-LCS connected via Ethernet to the central control system (CCS) [3]. This local control system (LCS) scheme is shown in Fig. 1. The primary role of the low level radio frequency system is to monitor and control the amplitude and the phase of each cavity voltage (fast regulation) and to control the tuning of each cavity to keep its resonant frequency constant. For doing so, the LLRF will generate the RF signals (RF drives) for the amplifiers feeding the cavities, depending on their voltage and forward power. LIPAc LLRF system has to work under both the CW mode operation and the pulse mode operation (during the commissioning and tuning of the prototype accelerator) [3].

The main element of the LIPAc LLRF system is based on a high performance and commercial Lyrtech FPGA VHS-ADC board for fast control that is installed in the cPCI bus of a Windows host computer. The Virtex-4 FPGA of the VHS-ADC/DAC allows us to fulfill

* Corresponding author.
E-mail address: julio.calvo@ciemat.es (J. Calvo).

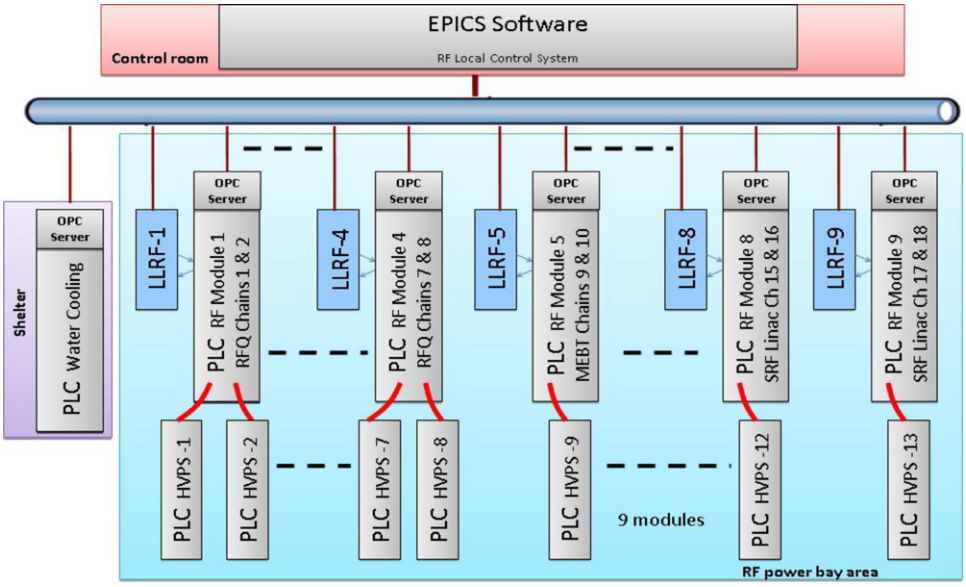

Fig. 1. LLRF local control system scheme.

our highest processing needs. If an interlock happens, FPGA card is able to make an emergency stop in less than 10 μs and it stops sending RF power to the cavities. The Virtex-4 provides us a very good logic, with one of the highest performance and density, and the memory capacity allows us to make accuracy Fast Data Logger, this FPGA board consumes only half the power needed by other FPGA families. There are two Lyrtech FPGA cards per RF module-LCS, one is called Loops and other Diagnostics. One is in charge of the amplitude, phase and tuning loops (Loops board) another one is in charge of the fast interlocks management and ancillary diagnostics (Diagnostics board). This paper presents a device support for LIPAc LLRF system, managing different applications related to control and monitor processes and data logging. Mainly, this device support is in charge of sending the parameters set by the operator to the Lyrtech board in order to modify its configuration. Communication from the device support to the board is through a bank of registers.

The application interface (API) that comes with these boards only runs under Windows operating system. Nevertheless, within the LIPAc project and for historical reasons, main control system is based on Linux OS and EPICS [4] as the main tool for the development of the control systems. Successful examples of the use of EPICS within the environment of control systems for fusion experiments can be found here [5,6]. Therefore we could consider this local system has significant differences from the model used for the main control system for the LIPAc project. Consequently, another important objective of this work is to allow a distributed control system, developing EPICS device support on the host computer and permitting the use of a different operating system thanks to properties of EPICS Channel Access.

Some examples of RF systems with EPICS can be found in [7–9]. A solution based on a digital LLRF with the same commercial board using EPICS with a JAVA IOC can be found in [10]. The work presented here has significantly more functionality than that presented in [10], such as the Fast Interlock Module, DACs and ADCs gain, system tuning, clock, VCXO programming or Fast Data Logger. In addition, there is a fundamental difference between our work and [10], namely what we present to the final user is an interface where all internal processes related to control and monitoring are transparent. In contrast, the work presented in [10], includes some tools which require programming and configuration by the final user. Thus, the work presented here is novel because no solutions in the literature have been found with the features and functionality of our software architecture for an LLRF system.

This work is based on the first paper published in [1]. The main contribution with respect to this first work is related to the description of the system architecture and the system operation, which becomes one of the fundamental cores of the control system.

2. System architecture

2.1. Hardware architecture

The LLRF system is composed by three main subsystems: a digital board with fast FPGA, the analog front end and a local timing system. It is a similar scheme as [11].

- Digital board: The digital board contains one Virtex-4 FPGA, 8 ADCs and 8 DACs with 14 bits resolution and capable to work up to 105 MHz. It is a commercial board with cPCI format provided by Lyrtech and controlled by a Windows CPU. This board acquires different kind of signals: RF control inputs (cavity voltage and forward cavity power), RF interlock inputs, digital interlocks and timing signals (gate and pulse signals). It also provides the control outputs of the LLRF: DC signals to be modulated into RF to control amplitude and phase of cavity voltage, interlock output to open a pin diode switch to stop the RF when an interlock happens and low voltage transistor-transistor logic (LVTTL) pulses to move a motor that adjusts the resonance frequency of the cavity. The cavity interlocks inputs controlled by the LLRF are: *reflected power of the cavity, vacuum pressure, arcs* and *multipacting*. Furthermore, the Machine Protection System (MPS) will be also connected to the Fast Interlock utility of the LLRF to switch off the RF Drive when required. A scheme of a LLRF module is shown in Fig. 2.
- The front end: This is in charge of up-converting the DC control outputs from the digital board into RF. For doing so, the LLRF employs a quadrature IQ modulator.
- The local timing system: It consists of a PLL board with a 100 MHz VCXO (CDC-7005-EVM from Texas Instruments). This board provides 100 MHz TTL signal to clock the digital board. This signal

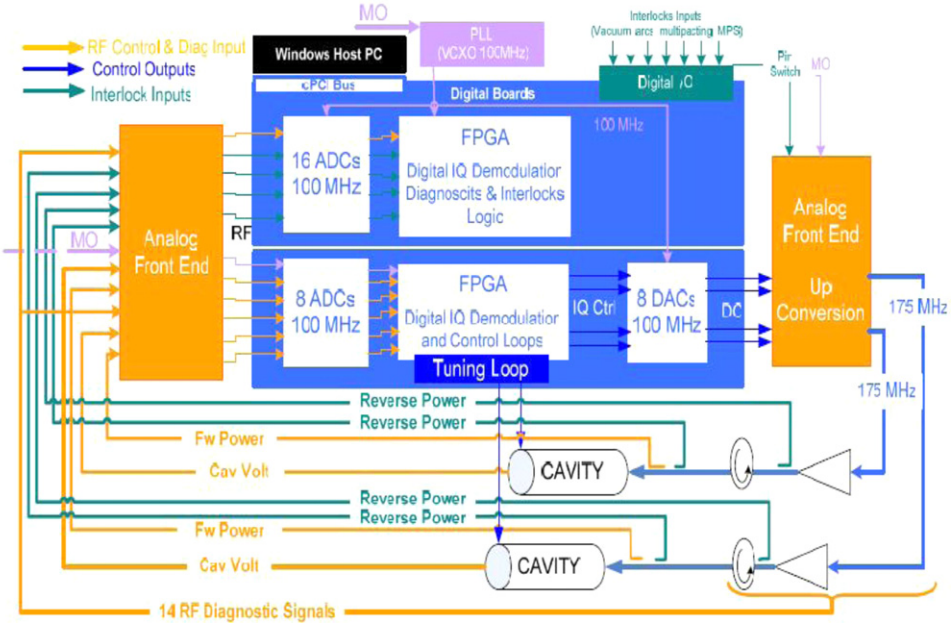

Fig. 2. LLRF system general overview [12].

will be phase locked with an external 10 MHz signal provided by the general timing system [13].

2.2. Software architecture

The presented architecture is developed using EPICS Base 3.14.11, Visual Studio 2008 C++Express Edition, asyn4-13-1, the Application Programming Interface (API) for Lyrtech boards and Control System Studio (CSS 3.0.2).

This solution consists of the following EPICS components:

- IOC (input/output controller): The IOC is any computer that supports the EPICS run-time components, including the database, and its access routines, device drivers, record types for various input and output and scanning and monitoring functionality [14]. Since the host computer system Lyrtech card is based on Windows, we chose to run the EPICS IOC on a Windows host.
- Database: The EPICS database is a basic element in an IOC. The database is a collection of records of various types. A record is an object with:
  - A unique name.
  - A behavior defined by its record type (class).
  - Controllable properties (fields).
  - Optional associated hardware I/O (device and driver support).
  - Links to other records [15].

  In our LLRF control system, the records controlling the Lyrtech boards have associated device and driver support.
- Device support: We could define it as an interface between records and hardware. A device support routine has knowledge of the record definition. It also knows how to talk to the hardware directly or how to call a driver which interfaces to the hardware. Thus, device support routines are the interface between hardware specific records in a database record and device drivers or the hardware itself [16].
- AsynDriver: It is the name of the original *asyn* package. It is written in C and provides the functions needed to write asyn servers (called asyn port drivers) and asyn clients (such as EPICS device support). It provides asynManager, which is the core of asyn, as well as a set of specific interfaces (asynInt32, asynOctet, asynFloat64, etc.). asynPortDriver is a C++class that is intended to make it much easier to write asyn port drivers. It takes care of most of the details of writing a port driver. It has a parameter library, and a set of base class methods that can be used in many cases. asynPortDriver simply calls the original asynDriver functions, which are then more or less hidden from the derived classes that are based on asynPortDriver [17].
- LAN: Local area network. This is the computer network which allows the communication between IOCs and Operator Interfaces (OPIs). EPICS provides a software component, Channel Access, which provides network transparent communication between Channel Access clients (e.g. Operator Interface, OPIs) and an arbitrary number of Channel Access servers (e.g. IOCs) [4].
- OPI: Control System Studio (CSS) is a combined effort of several parties, including DESY (Hamburg, Germany) and SNS (Oak Ridge, TN). It provides a collection of control system tools in a common environment, based on Eclipse [18].

This architecture is module based and it follows the LIPAc four layers model, as shown in Fig. 3. At the client level we have used CSS as software that allows PVs to be accessed and modified from the network, using the second layer, the Channel Access, that is the communication protocol used by EPICS to transfer information through the network. The next layer is the record support one, where the IOC and the database are located; here, we have the software which implements PVs for the use with Channel Access. This is the heart of the architecture and permits the communication between records in the database and the device. The fourth level corresponds with the equipment level, FPGA boards.

Regarding to the control flow using asynPortDriver, Fig. 4, we assume that code runs from an application thread to a port thread. Records have an associated hardware, Lyrtech boards, but they do not access to this hardware directly; rather the device support layer performs I/O operations on request calling the asyn port driver layer. In our case, most of the records are analog inputs and analog outputs. The device support provides I/O for a single record type [15]. In every record, the DTYP (Device Type) field determines which device type to use, in our case, the DTYP corresponds with the asynPortDriver [17] functions (writeInt32, writeFloat64, ...). From the record support level, the control goes to record device support which calls asynPortDriver. This device support solution

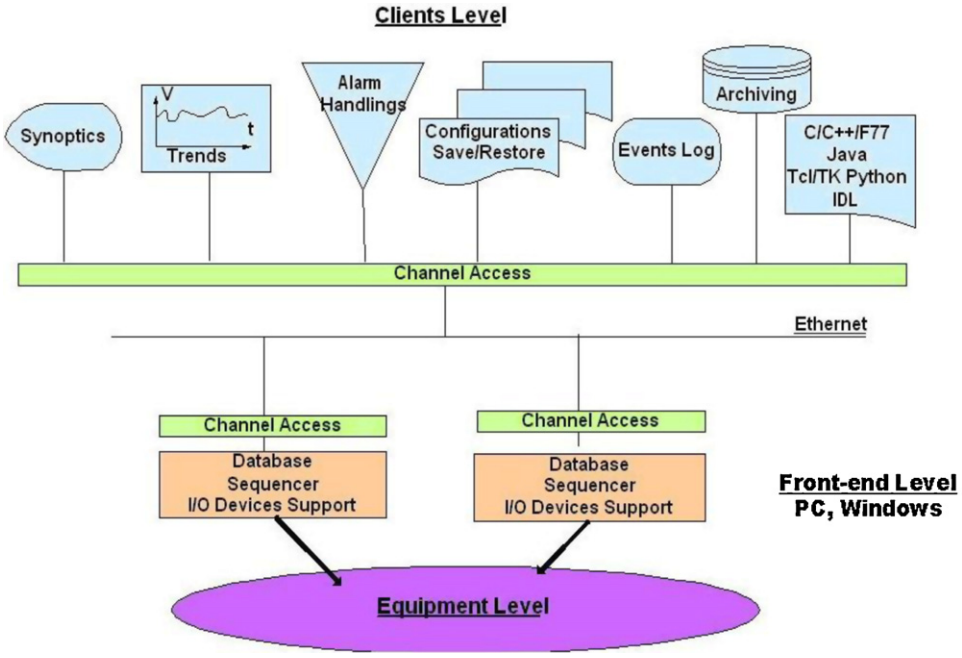

Fig. 3. LIPAc four layers structure.

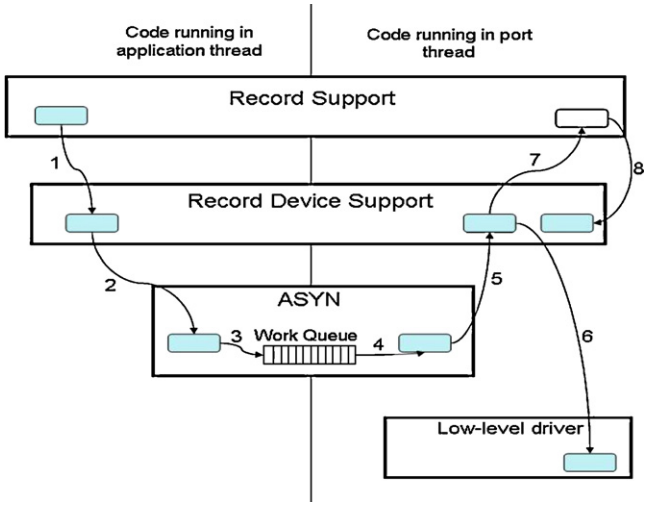

Fig. 4. Control flow in asynchronous driver [19].

increases the modularity and it makes not necessary to learn a new record type for each type of device.

One of the most interesting things about writing an asynDriver is that it is much easier to extend in the future. For example, at first it was thought necessary to use only analog input and analog output records. But later it was necessary to add support for binary output and multi output multi binary records that allow users to choose some options from a menu. This addition in any other kind of device support would have meant a new code rewriting. With an asyn driver that addition was less than 10 lines of code. Similarly, with asyn, if the users decide that they want to use a longout record rather than an analog one, we do not have to change our driver at all, changing the database file is enough. That is not true if one does not use the generic asyn device support.

Asyn provides standard facilities for debug tracing. We just add one line to our driver to use this feature. Then, when the user enables **AsynTraceIoDriver** at the EPICS shell for our driver **asynSetTraceMask(myDriver, 0, 9)** this will turn on debugging messages from the driver. Those messages have useful time stamps, and they can be routed to a file, not just to standard output (stdout).

2.2.1. Synchronous and asynchronous device support

Because we are using asynPortDriver almost all of the details of *synchronous versus asynchronous* are taken care of for us. Therefore, the choice of using a simple synchronous device is not considered. The only thing we need to do to change from synchronous to asynchronous is set the bit mask for **AsynCanblock** in the call to the asynPortDriver constructor in our driver. Our current code looks like this: **(AsynCanblock = 0, AsynMultidevice = 1, autoConnect = 1)**

That means we have created a synchronous driver as well, i.e. one that is assumed to execute quickly (e.g. under 1 ms, so it does not slow down EPICS record processing). If we find that our device is *slow* and we need to create an asynchronous driver, then we just change this to: **(AsynCanblock = 1, AsynMultidevice = 1, autoConnect = 1)**.

The difference is all handled by asynManager and the standard asyn device support.

If we do not set the **AsynCanblock** bit, then device support will directly call our driver's writeInt32, readInt32, etc. functions when the record processes from whatever thread is processing the record, i.e. EPICS periodic scan tasks, Channel Access server tasks, etc. This should only be done if our device is fast.

If we do set the **AsynCanblock** flag then asynManager will create a separate port thread for our driver. When device support wants to talk to our driver it will queue a request to communicate with it, and when the driver is not busy, device support will call the driver from that separate port thread. The driver is allowed to be slow in this case, because it will not block record processing, which is OK to be slow.

3. System operation

The traditional way to operate Lyrtech VHS-ADC/DACs boards is carried out using the manufacturer utility included in the purchase. The program communicates directly with the VHS-ADC/DACs, thus it must run on the cPCI CPU board of the cPCI chassis that contains

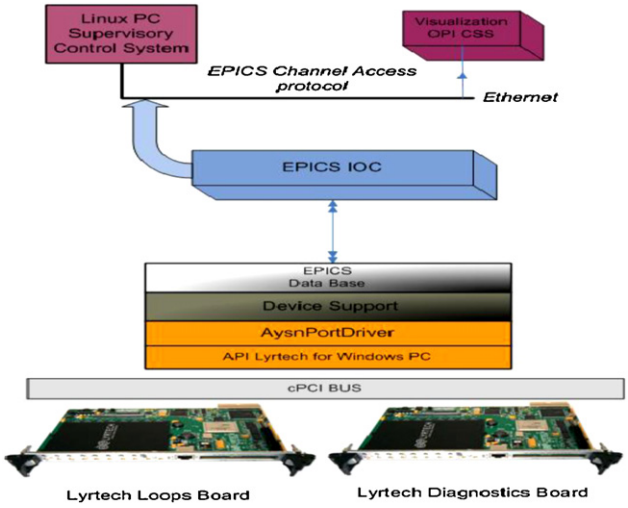

Fig. 5. LLRF control operation.

	GUI Value	Value to be sent to FPGA	Cavity Selection (bit 31)	ADDRESS (From bit 30 to bit 17)				Address selection	DATA (from bit 15 to bit 0)						
Bit Position			31	32	...	18	17	16	15	14	13	...	2	1	0
Kp A (1st address)	1	1	0	0	0	0	0	0	0	0	0	...	0	0	1
Kp A (2nd Address)		65537	0	0	0	0	0	1	0	0	0	...	0	0	1
Ki A (1st Address)	1	131073	0	0	0	0	1	0	0	0	0	...	0	0	1
Ki A (2nd Address)		196609	0	0	0	0	1	1	0	0	0	...	0	0	1
IRef A (1st Address)	100	527565	0	0	1	0	0	0	0	0	0	...	1	0	1
IRef A(2nd Address)	100	593101	0	0	1	0	0	1				...			
Kp B (1st address)	1	2147483649	1	0	0	0	0	0	0	0	0	...	0	0	1
Kp B (2nd Address)		2147549185	1	0	0	0	0	1	0	0	0	...	0	0	1
Ki B (1st Address)	1	2147614721	1	0	0	0	1	0	0	0	0	...	0	0	1
Ki B (2nd Address)		2147680257	1	0	0	0	1	1	0	0	0	...	0	0	1
IRef B (1st Address)	100	2148011213	1	0	1	0	0	0	0	0	0	...	1	0	1
IRef B (2nd Address)		2148076749	1	0	1	0	0	1				...			

Fig. 6. Example parameter values and addresses to be written in Register 0 [20].

GUI parameter	FPGA parameter	Relation beween GUI and FPGA parameter
Cavity Voltage and Phase cavity voltage	IRef & QRef	$IRef = Cavity\ Voltage \cdot \cos(Cavity\ Phase)$ $QRef = Cavity\ Voltage \cdot \sin(Cavity\ Phase)$ $Cavity\ Ref\ Amp = \sqrt{IRef^2 + QRef^2}$ $Cavity\ Ref\ Phase = \operatorname{atan}(Q/I)$ $Amplitude = \sqrt{I^2 + Q^2}$ $Phase = \operatorname{atan}(\frac{Q}{I})$

Fig. 7. Example of relation between IQ components [20].

the platforms. Through this device support, operator can control clock and trigger settings, FPGA designs, the four custom registers, and the input/output gains of all the VHS-ADC/DACs. If the program detects the proper modules within FPGA designs, operator can control data recording, as well as data playback. The device driver developed in this work is an extension and an improvement of the utility provided by the manufacturer. Moreover, this development offers the possibility to control and modify parameters that are not accessible from the traditional utility (see Fig. 5).

The general operation of the device driver is based on the use of certain functions of read/write registers in the Lyrtech card. These functions are contained within some libraries provided by the manufacturer. After two years of work, the driver code length is nearly 9000 lines and it is growing steadily thanks to the addition of new features and improvements.

3.1. Parameters to FPGA

Device server and GUI will be used to control the LLRF of two RF plants. In order to do so, the operator will set some working parameters. All these settings are gathered in different groups. These groups and the number of parameters are increasing continuously:

- Amplitude and phase loop parameters.
- General configuration parameters.
- Tuning loop.
- Manual tuning.
- Conditioning.
- Interlocks.

When the user writes a new parameter in the GUI, this parameter is sent to the FPGA through Register 0. The device server applies the right transformation to the parameter value. These registers allow to the user sending 32 bits words to the FPGA. The 16 MSBs are employed as the address of the word, while the 16 LSBs are used to send the data or value of the parameter. The MSB bit will be set to 1 when sending a parameter to cavity A and set to 0 when sending a parameter to cavity B. Both cavities will have the same number of parameters, but their values could be different. Besides, every parameter is sent twice in two different addresses in order to avoid communications problems. In case the data sent by the two addresses are not coincident, a diagnostics error bit will be asserted and stored in another register. Bits 30–17 are identical in these two addresses, and bit 16 will be set to 0 for the first address and set to 1 for the second address. Some examples of this operation are shown in Fig. 6.

After writing a new parameter, the driver reads instantly the new information from the FPGA once it has been written. To do so, the driver writes the parameter address in Register 1 and then it reads back the value of Register 1.

On the other hand, all the parameters will not be taken into account by the FPGA until the word 1 (MSB = 0) for cavity A or word 2,147,483,649 (MSB = 1) for cavity be sent to the Register 3. This is done every time a new parameter is set and after that, Register 3 is set to zero again [20].

3.1.1. Relations between GUI parameters and FPGA parameters

Some of the values displayed in the GUI are not exactly identical to the values that should be in the FPGA. The digital IQ demodulation [11] is carried out at FPGA level. For doing so, the RF inputs will be under-sampled and digitally demodulated. Since the CPU is used to send-read configuration commands to-from the FPGA board, some of the values displayed in the GUI will need to be transformed. These special parameters are cavity voltage, cavity phase and IQ components. The challenge of these transformations is to make the board understand values that the operator can easily manage and to get values introduced by the operator. Hence, the first step to control amplitude and phase components of the cavity voltage, is to transform the cavity voltage set points introduced by the operator through the GUI from polar coordinates (amplitude and phase) to rectangular coordinates (I and Q). The device support performs these operations as shown in Fig. 7.

3.2. Device support functionalities

The device driver has several functionalities and hundreds of signals. Here we show a few examples achieved in the presented development:

3.2.1. Gain

The applicable gain range of the VHS-ADC Virtex-4 ADC and DAC channels is 0–15 (i.e. 4-bit gain values). Device support allows the user modifying the values of these gains to the required setpoint.

3.2.2. Clock

All the operations carried out by the Lyrtech board should be synchronized with a clock. The source of this clock could be internal or external.

Next list shows the available clock sources of the VHS-ADC/DAC. The device support allows the user choosing any of these clock sources:

- External source.
- Fixed onboard clock (105 MHz).
- FPDP Receive Clock.
- RapidCHANNEL Receive Clock.
- FPGA Generated.
- Fixed Divided by Two.

For the LLRF internal test at Ciemat we will always use the clock option: *external source from front panel*. So before starting sending parameters, the Lyrtech board is configured to work with this clock.

3.2.3. Voltage Controlled Crystal Oscillator (VCXO)

The function of the VCXO is to synchronize Lyrtech card clock with master oscillator signal in order to properly perform the IQ demodulation of RF signals. Without the VCXO, the phase of the RF signals could not be read. Main properties of this functionality are:

- Synchronizes frequencies up to 800 MHz.
- Each output frequency is selectable by $\times 1$, /2, /4, /8, /16.
- All outputs are synchronized.

In order to check if the VCXO has been well programmed, the driver read the status of the four diagnostic signals with addresses 50–53 as shown in the next table. The way to read these bits is the same employed to read any diagnostics signals, i.e. writing the address of the corresponding signal at user defined register 2, offset 0×0074, and reading back the register value [21].

VCXO				
Power (50)	Ref (51)	Locked (52)	Cable (53)	Message to display
×	×	×	1	VCXO cable disconnected
1	0	0	0	VCXO powered
1	1	0	0	VCXO reference
1	1	1	0	VCXO locked
Any other combination of bits				Error

3.2.4. Fast data logger

Lyrtech boards, Loops and Diagnostics, have a 128 MB RAM which is storing data continuously. After an interlock happens, the LLRF will send a trigger to stop the acquisition of signals and all the data stored in the RAM will be transferred to a binary file in the Windows CPU that controls both boards.

There is a file called *CellsDataRecorder.ini* where two parameters can be set: *ChannelsSource* and *TriggerSource*. The *FirstChannelSource* should be always set to 1 and the *SecondChannelSource* should be always set to 0. The *TriggerSource* can also be set to hardware or software. By default it will be set to hardware, but this parameter can be changed through the LLRF GUI. When the user presses the button Save Acquisition, the *TriggerSource* sets to software and a software trigger is released [21].

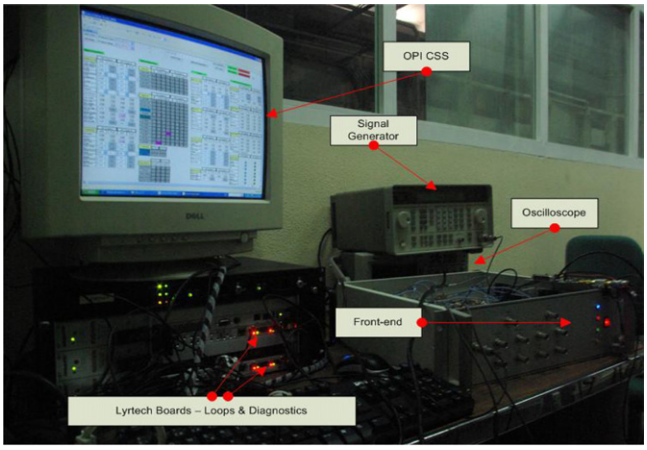

Fig. 8. LLRF system test bench at Ciemat.

3.3. Testbench

The LLRF testbench, Fig. 8, consists of a RF generator and the LLRF itself. The RF generator provides a 175 MHz, 18 dBm signal to upconvert the DC control outputs of the digital board. The RF upconverted outputs are sent to a filter and a splitter, and from there, the RF signals are sent to the ADCs of the digital board to test the acquisition, the loops and the communications of the system. The reference output of the RF generator (10 MHz) is used to synchronize the LLRF with the RF generator employing a PLL board installed in the front end of the LLRF.

To measure latency time we have used different parameters. In all measurements we have found that this time is almost negligible. A measure of cavity voltage set parameter is shown in Fig. 9. We define latency time as the time from entering a new value through the interface to read it back. Once the card has recognized the value, acquired it and shown it, we can claim to have crossed all layers of the system. Looking at the next Latency Time table of the cavity voltage set parameter, we can conclude that the system works on the scale of microseconds.

Latency time		
Cavity voltage set	Cavity voltage readback	LT
50	49.99	42 μs
100	100.01	157 μs
150	150.00	44 μs
200	199.99	38 μs
250	250.00	54 μs

3.4. EPICS IOC initialization

An EPICS IOC [14] starts by loading the binary software image and then a database definition file containing a description of all the data records and enumerated types used in the in-memory database. Our st.cmd file is short and simple, it uses macros for PVs names that follows the naming convention within IFMIF–EVEDA project. During initialization, the driver detects the boards assigning them a handle that will be the main identifier. During this initialization, a.bit file is loaded into the boards memory, which has been written properly for the special feature of the system.

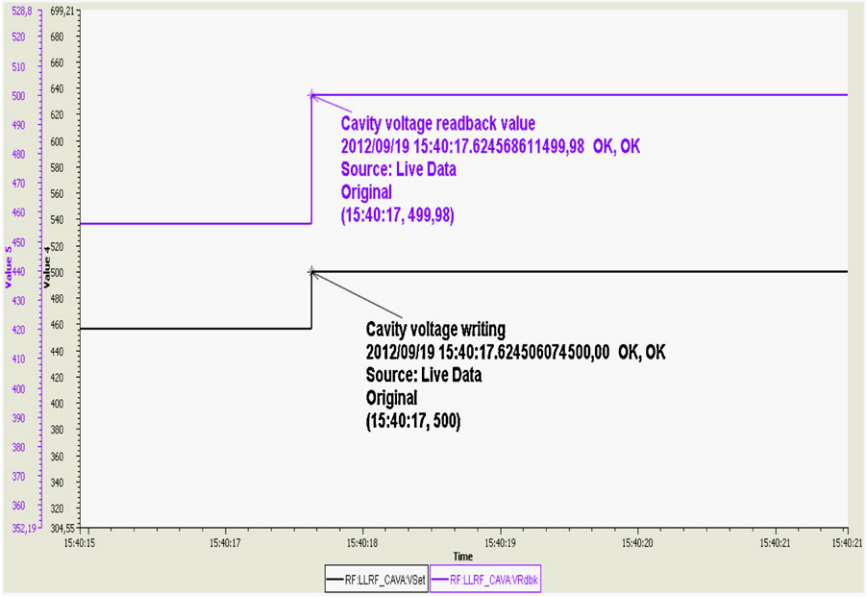

Fig. 9. Latency time measured in cavity voltage parameter.

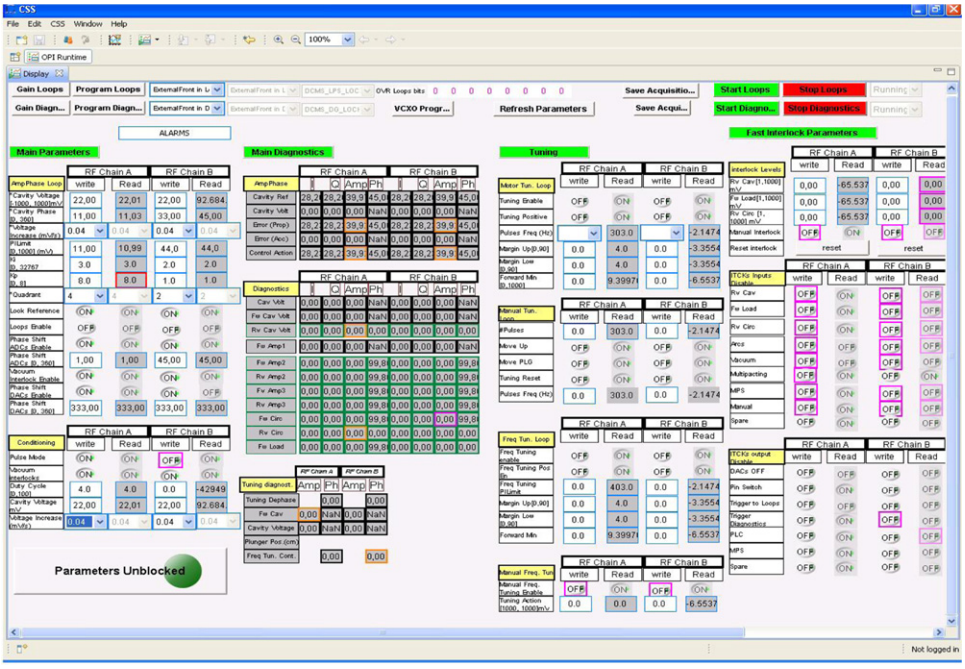

Fig. 10. OPI developed using CSS.

3.5. *User Interface*

The graphical user interface shown in Fig. 10 has been designed using CSS. To insure a consistent look and feel between the panels developed with a graphical user interfaces (GUI) development tool, basic rules have been followed. These rules are based on the previous experience of the European Union (EU) team. This aspect must be taken in consideration because GUI panels will be the most important way to interact with the different subsystems. A uniform way to represent information is very important and will avoid the *Christmas tree effect*. So to avoid unnecessary additional work when the LCSs will be integrated at the final stage in Rokkasho, these basic rules have to be adopted from the beginning of the development. The more intuitive the user interface the easier it is to use, and the easier it is to use and the more efficient to use it [22]. We expect that the operator and the machine engineer can easily tune the system, control the parameters and program Lyrtech boards through the set of designed pages. Following the EPICS philosophy, the whole logic involved has not been implemented at the client level (CSS, but rather on the IOC), thus they are available from anywhere in the control system using any client.

4. Summary and future plan

The LLRF control system based on a Lyrtech FPGA card has been solved in a new way, using the architecture explained in this paper. One of the inherent characteristics of the system is the independence of the hardware at the time of accessing the Lyrtech board

from anywhere in the distributed control system. This was one of the main challenges of the presented architecture.

The power of the class asynPortDriver has made it possible to develop a large amount of functionalities in less than two years of work. In addition to the functionalities shown, automatic conditioning of the cavities, rapid diagnostics of RF, a proprietary system of warnings and some other features have been integrated as well.

This solution will be used to control and monitor the LLRF system of two plants in the final accelerator prototype which is being built in Rokkasho, Japan. Thanks to CSS and EPICS Channel Access, process variables can be seen in any remote computer of the LIPAc Central Control System. The EPICS based characteristics of the system makes it useful because of its modularity and it can be easily upgradeable and modifiable. The choice of EPICS as a control toolset was very important to achieve this success.

The further work line moves toward adding more functionalities to the system and carrying out the testing of the system within the overall prototype accelerator. Else, adapting the warning system to the Best Ever Alarm System Toolkit (BEAST) [23] could be another choice.

Acknowledgment

This work is funded by Ministerio de Ciencia e Innovación del Gobierno de España, under projects AIC10-A-000441 and AIC-A-2011-0654.

References

[1] J. Calvo, M. Rivers, A. Ibarra, M. Patricio, IFMIF LLRF control system architecture based on EPICS, in: Proceedings of the ICALEPCS, Grenoble, France, 2011.
[2] International Energy Agency, IFMIF Comprehensive Design Report, January 2004.
[3] I. Kirpitchev, P. Méndez, M. Weber, A. Ibarra, M.A. Falagan, M. Desmons, A. Mosnier, RF power system for the IFMIF–EVEDA prototype accelerator, in: Proceedings of the EPAC, Lucerne, Switzerland, 2004.
[4] EPICS, http://www.aps.anl.gov/epics/
[5] M. Kwon, I.S. Choi, J.W. Choi, J.S. Hong, M.C. Keum, K.H. Kim, M.G. Kim, M.K. Park, S.H. Seo, S. Baek, H.G. Jhang, J.Y. Kim, The control system of KSTAR, Nuclear Instruments and Methods in Physics Research A 600 (2009) 179–181.
[6] W. Higemoto, K. Shimomura, Y. Kobayashi, S. Makimura, Y. Miyake, T. Kai, K. Sakai, J-PARC muon control system, Fusion Engineering and Design 71 (2004) 17–21.
[7] P. Corredoura, Architecture and performance of the PEP-II low-level RF system, in: 19th Annual Particle Accelerator Conference (PAC99), New York City, 1999.
[8] S. Michizono, D. Arakawa, H. Katagiri, T. Matsumoto, T. Miura, Y. Yano, S. Fukuda, Digital LLRF system for STF S1 Global, in: Proceedings of the IPAC, Kyoto, Japan, 2010.
[9] J.C. Yoon, J.W. Lee, K.M. Ha, J.H. Kim, J.M. Kim, J. Choie, EPICS based control system for the KOMAC RF system, in: Proceedings of the EPAC, Lucerne, Switzerland, 2004.
[10] I. Arredondo, M. del Campo, P. Echevarria, D. Belver, L. Muguira, N. Garmendia, H. Hassanzadegan, M. Eguiraun, Commercial FPGA based multipupose controller: implementation perspective, in: Proceedings of ICALEPCS, Grenoble, France, 2011.
[11] A. Salom, F. Pérez, Analogue and digital LLRF for ALBA synchrotron, in: Proceedings of the EPAC, Edinburgh, Scotland, 2006.
[12] Technical Description of the IFMIF–EVEDA RF Power System, IFMIF–EVEDA Accelerator System Group.
[13] A. Salom, LLRF and its Control System, Third IFMIF/EVEDA Workshop, Madrid, 2010.
[14] EPICS IOC, http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/node4.html
[15] EPICS Database, http://www.slac.stanford.edu/comp/unix/package/epics/training/documents/03_Database.pdf
[16] Device Support, http://www.aps.anl.gov/epics/docs/USPAS2010/Lectures
[17] Asyn, http://www.aps.anl.gov/epics/modules/soft/asyn
[18] CSS, http://css.desy.de/
[19] Control Flow, https://slacportal.slac.stanford.edu/sites/conf_public/epics_2012_04/presentations/asynTalk.pdf
[20] A. Salom, Device Server and GUI Specifications for IFMIF LLRF.
[21] A. Salom, Device Server and GUI Specifications for IFMIF LLRF – FIM and FDL Implementation.
[22] LIPAc EU-HT, EPICS Software Development Guidelines.
[23] K. Kasemir, X. Chen, E. Danilova, The Best Ever Alarm System Toolkit, in: Proceedings of the ICALEPCS, Kobe, Japan, 2009.