ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

PYTHON BASED INTERFACE TO THE KARA LLRF SYSTEM

E. Blomley*, J. Gethmann, P. Schreiber, M. Schuh, W. Mexner, A. Mochihashi, A.-S. Müller
Karlsruhe Institute of Technology, Karlsruhe, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany
D. Teytelmann, Dimtel Inc., San Jose, California, USA

Abstract

The Karlsruhe Research Accelerator (KARA) at the Karlsruhe Institute of Technology (KIT) is an electron storage ring and synchrotron radiation facility. The operation at KARA can be very flexible in terms of beam energy, optics, intensity, filling structure, and operation duration. Multiple digital low-level radio-frequency (LLRF) systems are in place to control the complex dynamics of the RF cavities required to keep the electron beam stable. Each LLRF system represents a well established closed system with its own set of control logic, state machine and feedback loops. This requires additional control logic to operate all stations together. In addition, during special operation modes at KARA, extra features such as well defined beam excitation are needed. This paper presents the implementation of a Python layer created to accommodate the complex set of options as well as an easy to use interface for the operator and the general control system.

RF SETUP OF KARA

KARA makes use of two RF stations controlled by one LLRF system each. Each RF station consist of a klystron which feeds two cavities utilizing a magic T with a fixed 3dB splitting ratio. In addition, the smaller booster synchrotron is powered by one cavity with an additional LLRF system. The electron beam in KARA is accumulated at 0.5 GeV with a 1 Hz injection cycle in the booster synchrotron. The accumulation period can typically last up to one hour. After the injection is finished, the beam energy of the storage ring is slowly increased to its final operating energy. Depending on the final energy, the energy ramp lasts up to three minutes. Therefore, in our common operation scheme, the demands of the RF system differ substantially between the booster synchrotron and the storage ring. The first operates on a fast, pre-defined and synchronized voltage ramp, whereas the approach for the storage ring is to explicitly set the RF voltage level to accommodate for the increase in beam energy as the energy slowly increases. Table 1 shows a summary of the RF system characteristics.

Table 1: RF System Characteristics

Parameter	Booster	Storage Ring
f_{RF} (MHz)	500	500
Voltage Range (kV)	5 - 25	300 - 1500
Energy Range (GeV)	0.05 - 0.5	0.5 - 2.5
Ramp Duration (s)	0.8	170
LLRF Stations	1	2
Cavities	1	4

Digital LLRF System

Modern, digital LLRF systems take care of operating the RF cavities by controlling the amplitude and phase of the RF waves, while also providing access to data points to read out signals, configure the relevant parameters and operate the RF station in general. At KARA the LLRF9 [1] from Dimtel was installed in 2016 with the booster synchrotron following in 2017. Each LLRF system has up to 9 RF inputs as well as the option for additional slow inputs to, for example, monitor the cavity vacuum levels. The main internal signal processing takes place on a field-programmable gate array (FPGA) surrounded by a Linux operating system providing slow access via the EPICS control system [2], which is also the main control system of all accelerators at IBPT. The EPICS interface allows access to around 600 process variables. This includes access to a stored ramping curve, captured waveforms of the fast inputs and an integrated network analyser, among other features. The system comes with a full set of graphical user-interface (GUI) panels (see Fig. 1).

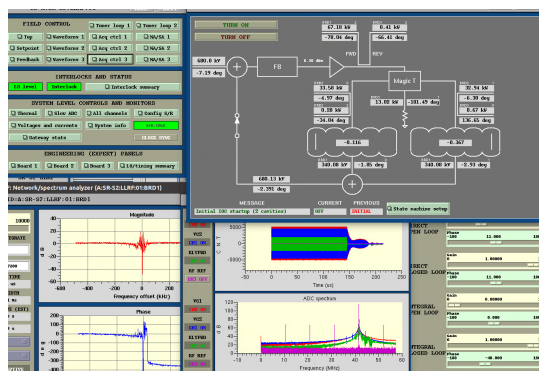

Figure 1: Native LLRF GUI. Visible is the top level control, the network analyser and the waveform acquisition tools.

USE CASES

The embedded EPICS controls and panels do an excellent job for what they were designed for, namely commissioning, configuring and monitoring the LLRF system. Still, there are several reasons as to why an additional software layer surrounding the core LLRF system might be useful.

Daily Operations

How the LLRF system is used in daily operations might be quite different from accelerator to accelerator or in this case even between the storage ring and the booster synchrotron.

* edmund.blomley@kit.edu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

While the operation modes differ, the most common ones can typically be described by a fixed routine. A surrounding framework can directly implement these routines and therefore provide the operator with a simplified method to interact with and monitor the LLRF system, without requiring the extensive knowledge and training to work with the native interface. For example, in case of the booster synchrotron, the operator is mostly interested in making sure that the system is *ON* and operational without needing to manually:

- Reset interlocks if present
- Load a pre-defined ramping curve to waveform generator
- Switch on the LLRF feedback loop
- Switch to external hardware trigger and ramping mode

Control of Multiple LLRF Stations

Each LLRF system represents its own ecosystem, taking care of monitoring and controlling various feedback loops and external hardware such as motor controllers used to tune the cavities. But if multiple LLRF systems are in place, some form of high level control is desired. Certain actions just have to be mirrored to all stations, such as resetting interlocks and getting the units into an operational state. For other areas additional data points can be introduced, such as a voltage sum across all stations, which can be monitored and set without manual interaction with the individual LLRF systems. Another typical example is adjusting the relative phasing between the booster and the storage ring as a whole, as well as, between the two stations of the storage ring.

Extending Features

An additional software layer also creates an entry point to extend the capabilities of the LLRF system. An important feature for KARA is the internal waveform generator of each LLRF unit, which allows loading arbitrary waveforms of a maximum length of 512 elements. A useful extension of this is the ability to load a user-defined curve. Since it would be cumbersome to manually define 512 elements interpolation is useful, as well as dynamic scaling before the curve is loaded to the LLRF hardware. In addition, simple periodic patterns can be generated, such as sinusoidal or step functions which are useful for various beam studies [3–5].

Working with Physical Values

Devices in an accelerator are often controlled with properties using *hardware* units, such as Ampère *A* for magnet currents or amplitudes in *dB* for cavity fields. From an accelerator physics point of view, properties in *physical* units to monitor or even control elements of the storage ring are often desired. Taking the hardware units from the LLRF system and combining this information with additional data provided by the control system such a conversion can take place. Depending on the physical quantity, there might be a need to calibrate or estimate the actual value based on some additional model input. Typical examples are the ability to control the overall voltage level by setting a desired synchrotron frequency f_s (*calibrated*) or provide an estimation of the momentum compaction factor α_c (*model input*).

DESIGN CONSIDERATIONS

A custom software layer, internally referred to as *LLRF control service*, was developed with the original installation of the digital LLRF systems to provide most of the features discussed in the previous section. The service was written as a stand-alone C++ application. Since there were no comparable applications written in C++ at that point of time, additional code was required for the basic integration and interaction with our control- and operation systems. By now this system exists as an independent artifact without any other applications making use of the same C++ interface, making it difficult to maintain, improve or extend the code base. In addition, two separate versions of the control service exist, one for booster and the other one for storage ring operation, each having a different set of features. The different feature sets made sense at the time, but the demand on especially how the booster is operated by now has changed considerably. Table 2 lists the differences.

Table 2: Feature Disparity of Prior Framework

Feature	Booster	Storage Ring
Manual control possible	no	partially
Pre-defined ramping	yes	no
Energy based ramping	no	yes
Custom waveforms	no	yes

IMPLEMENTATION

Based on the considerations above and the criticality of the LLRF control service, where even small changes might have unintended but severe side-effects, the decision was made to fully reimplement this service from scratch. A key decision was made to switch to Python, which has seen widespread usage increase across our institute, from core accelerator services to machine learning applications [6].

Control System Integration

SoftIOC is a Python module which allows for creating an EPICS IOC which runs from within the Python interpreter supporting concurrency [7]. This allows creating an application which seamlessly integrates into our control system, as well as providing the option to include modules from the Python ecosystem (see Fig. 2).

Python Integration

The LLRF SoftIOC makes use of a set of modules referred to as the *KIT Accelerator Python Tools* [8]. These modules allow for an (internally) standardized and maintained access to common resources such as network storage, electronic logbook, data archive, system logging, measurement data handling, and shared routines across our other SoftIOCs.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

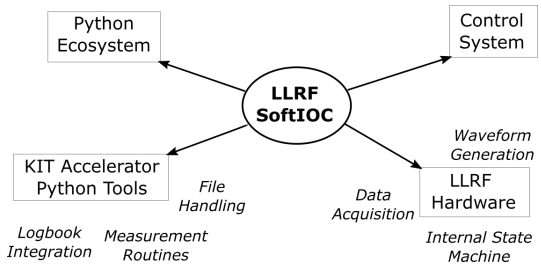

Figure 2: LLRF SoftIOC environment. SoftIOC allows seamless integration of our control system, the LLRF systems, our internal Python tools and access to the general Python ecosystem.

Access to other data points in the control system is provided using the *caproto* module [9].

Code Maintainability

To increase long term maintainability, common practices used in software engineering were applied, such as strict formatting rules and the requirement to document any public function, method and class. Tests are used to make sure that the behaviour of the code is kept consistent, also making it easier to implement potential changes in the future. Git in combination with a continuous integration workflow allows automatically running the tests as well as making sure that the code quality standards are being fulfilled before code can be added to the main code branch.

FEATURES

All features are available in both the booster and storage ring environment. In addition, each automation routine can be individually switched on or off, which allows the user to interact with the LLRF systems in varying degrees of semi- or fully manual control, if needed.

Event Logging

Certain events can automatically trigger the generation of a report for our electronic logbook [10]. For example, an interlock event during operation conditions causes the LLRF system interlock chain to be read out. Based on this information the software classifies the interlock as *internal* or *external*. In case of an internal interlock the waveforms of the RF signals are read-out and are attached as a plot to the report together with information regarding the current configuration of the LLRF system. Also, certain special routines during beam studies can create an entry to the logbook specifying the beam conditions and parameters used.

Simulator Mode

While tests of the Python code make sure that the functions behave as expected, the actual interaction with the LLRF hardware or the control system in general cannot be automatically tested. The simulator mode allows replacing either the process variables from the control system, such as the beam energy, or the process variables of the LLRF system. This is achieved by running a simulator SoftIOC which provides the necessary mock-up process variables as well as rerouting the process variables in the actual LLRF SoftIOC. In addition, the LLRF part can also be configured to use a LLRF spare unit if the actual hardware response is required.

State Machine Integration

With the updates of the LLRF system firmware, an internal state machine was added. Although this state machine cannot be used as a substitute for our typical operation modes, neither in the storage ring nor in the booster, it can take over interlock recovery, initial tuning and ramping down of the cavity fields. This allows removing a considerable amount of code and logic complexity, which was previously required for the initial start up of the LLRF system.

Measurement Routines

Thanks to having access to the Python ecosystem, including signal processing and fitting toolkits, more complex measurement or calibration routines can now directly be implemented in the LLRF SoftIOC code. One example is the calibration of the cavity voltage based on the measured synchrotron frequency f_s and subsequently a voltage scan to determine the momentum compaction factor α_c.

A more complex example is the implementation of precise phase or amplitude modulation of the cavity fields. This requires a measurement using the internal network analyser and a numerical solution based on the ratio of Bessel functions [11], whereas the user input only requires the desired modulation frequency and amplitude. To evaluate the modulation, a sequence for reading out the internal waveform is planned, which involves temporarily changing the acquisition engine from interlock detection to continuously measuring the field, taking a measurement, and switching back to interlock detection.

SUMMARY & STATUS

The dynamic range of operation modes at KARA creates a dynamic demand on our LLRF systems, from fully automated and transparent operation, to complex, semi-manual active measurements. Our approach for the LLRF SoftIOC represents a modular concept using Python, relying on robust implementations of independent modules from control system, file and logbook integration up to the inclusion of complex measurement routines with a strong focus on maintainability and extensibility. While not yet in production, first live tests in the booster are foreseen in the coming weeks.

REFERENCES

[1] LLRF9, https://dimtel.com/products/llrf9

[2] EPICS Control System, https://epics-controls.org/

[3] B. Kehrer *et al.*, "Time-Resolved Energy Spread Studies at the ANKA Storage Ring", in *Proc. IPAC2017*, Copenhagen, Denmark, May 2017, pp. 53–56. doi:10.18429/JACoW-IPAC2017-MOOCB1

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

[4] A. Mochihashi *et al.*, "Detuning Properties of RF Phase Modulation in the Electron Storage Ring KARA", pre-print: Nov. 2021. doi:10.48550/arXiv.2111.15555

[5] J. Steinmann *et al.*, "Increasing the Single-Bunch Instability Threshold by Bunch Splitting Due to RF Phase Modulation", in *Proc. IPAC'21*, Campinas, SP, Brazil, May 2021, pp. 3193–3196. doi:10.18429/JACoW-IPAC2021-WEPAB240

[6] P. Schreiber *et al.*, "Ocelot integration into KARA's control system", presented at PCaPAC'22, Prague, Czech Republic, paper THP16, this conference.

[7] pythonSoftIOC: *An EPICS IOC within the Python interpreter*, https://dls-controls.github.io/pythonSoftIOC

[8] J. Gethmann *et al.*, "Simple Python Interface to Facility-Specific Infrastructure", presented at PCaPAC'22, Prague, Czech Republic, paper THP17, this conference.

[9] caproto: *A pure-Python Channel Access protocol library*, https://github.com/caproto/caproto

[10] ELog, *Electronic Logbook package* https://elog.psi.ch/elog/

[11] D. Teytelman, "Phase modulation in storage-ring RF systems", e-print: Jul. & Sep. 2019. doi:10.48550/arXiv.1907.01381v2