ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW074

MYRRHA DAQ DEVELOPMENT

R. Modic, P. Mekuc, Cosylab, Ljubljana, Slovenia
D. Vandeplassche, P. Della Faille, SCK-CEN, Mol, Belgium

Abstract

We have implemented a generic Data Acquisition (DAQ) solution for the MYRRHA test stand at Louvain-la-Neuve (Belgium). The work was motivated by the need for better sampling performance, signal quality, arbitrary processing and storage of measurements. A full integration of the DAQ system into the global EPICS control environment was a strong requirement. An intermediate DAQ platform was put in place to satisfy the control and experiment needs. The NI PXI platform was selected to minimize integration and development effort. National Instruments (NI) LabVIEW is used to create a generic DAQ application and the CALab library, supported by DESY, is used to connect LabVIEW and EPICS. A Control System Studio GUI provides the user with the necessary control, visualization and configuration capability. The technical and organizational approach to the collaboration will be detailed in the paper, as well as the necessary customizations of CSS and CALab and experience of using NI PXI for a DAQ platform.

INTRODUCTION

The need for better sampling performance, signal quality, arbitrary processing and storage of measurements was the main motivation for this work. A full integration of the DAQ system in the global EPICS control environment was a strong requirement. An intermediate DAQ platform was put in place to satisfy the control and experiment needs.

DESIGN AND IMPLEMENTATION

National Instruments (NI) LabVIEW and the DAQmx [1] driver was used to create a generic DAQ application that runs on a PXIe [2] industrial computer with a Multifunction I/O Module. This platform was chosen because it was most suitable to comply with the requirement to acquire data with a frequency up to 2 MHz on 16 channels simultaneously and process it in real-time.

The software architecture consists of four independent modules that each executes its own function: Data acquisition, EPICS, Data Logger and Error Handler. Each of these modules has its own queue and other modules can send messages with payloads to those queues allowing the modules to interact with each other while at the same time remaining independent.

The Data Acquisition (DAQ) module (Fig. 1) is a master module and defines the workflow of the application. This main loop in the Data Acquisition module executes a state machine. States define what is considered as the positive workflow and what is not, e.g. the system is waiting for a configuration from the GUI, the system is waiting for a trigger or the system is acquiring data. Messages from other modules work as interrupt routines to the state machine execution. If a message arrives from another module, the next iteration of the main loop handles this message. Actions and state transitions are defined by the type of message and its payload. An example of a message to the DAQ module from the EPICS module is of type 'configure' with the details of the configuration data contained in the payload. This message triggers a configuration validation, storing the configuration and a state transition from the 'init' state to the 'configured' state. The DAQ module also contains an asynchronous process loop. This loop gets acquired samples from the DAQ module, processes the samples for the GUI and for storage and forwards the results to the EPICS and Data Logger modules.

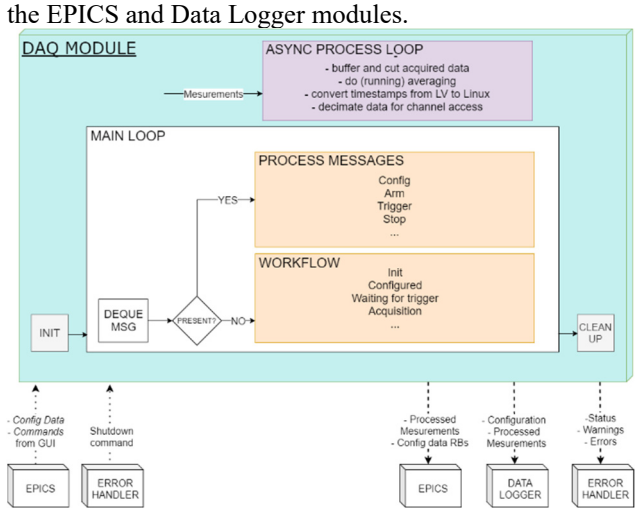

Figure 1: SW architecture of DAQ module.

The EPICS module runs SoftIOC from the CALab [3] library. This is where process variables (PVs) are stored. The EPICS module serves as a communicator with the EPICS control environment. Its main functionality is to update control and data PVs from SoftIOC to the DAQ application and vice versa. The EPICS module intercepts all user actions on the GUI and forwards it to the DAQ module.

The Data Logger module takes care of storing the acquired data and measurement configurations for post-analysis. The TFS text data format [4], defined by CERN, was chosen to the store data. The header contains configuration and static beam parameters and the body contains the acquired and processed data with the timestamp of acquisition.

The Error Handler module takes care of any events that could cause any malfunction in the operation of the application. It stores all status, warning and error events in a .log file for later inspection. The Error Handler module takes care of the proper shut-down of the system in the case of a critical error or a user shut-down action from the GUI.

To be able to integrate the DAQ application running on National Instruments hardware into the EPICS control environment, we needed a library for LabVIEW that would

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW074

be able to run SoftIOC and establish communication with it over EPICS channel access. We evaluated a number of options according to performance, ease of use, features and licensing and we chose the best two: CALab from BESSY and the National Instruments native support for EPICS. The CALab library performed better in upstream data update, which has a far more demanding requirement than downstream, and it integrated better into the LabVIEW environment. This led us to implement the DAQ application with the CALab library.

The GUI developed in Control System Studio [5] provides the user with the necessary configuration, visualization and control capability. The GUI has three tabs (Fig. 2), each with a specific functionality. Firstly, the user sets the acquisition parameters and acquisition type in the Configuration tab. In the Control tab, the user issues software triggers, stops or aborts acquisitions and shuts down the DAQ application after use. The Measurements tab (Fig. 2) displays the last measurements in real-time on an amplitude-time graph. Visible channels can be selected. There is also functionality to manipulate the graph and for quick inspection of the data.

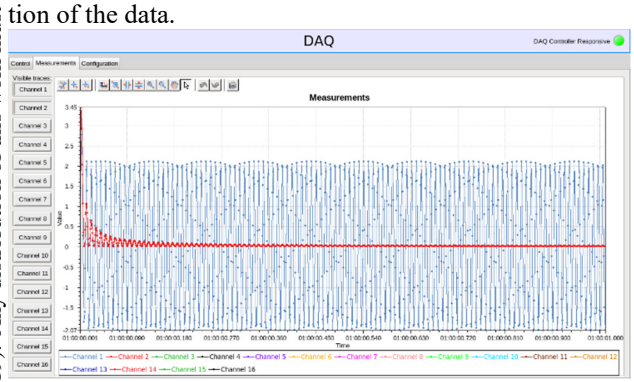

Figure 2: Measurements tab of the DAQ application GUI developed in Control System Studio.

PROJECT FORMAT

We decided to use the DAQ project as an exercise in requirement-driven engineering. An upcoming MINERVA endeavor by SCK will greatly scale the required control system efforts. Control system components developed will have to follow global control system design decisions, governed by an experienced control system architect.

We wanted to utilize the best software engineering practices early on and equip the SCK team with a scalable control system development strategy. Cosylab helped with the requirement definition and review. The requirements were then approved by SCK. The design and implementation, code review and elaborate test plan together with review process were done by Cosylab. The test plan was approved by SCK. Acceptance was done via execution of the test plan first at Cosylab (FAT). This made it possible to address any problems and comments before shipping the solution to LLN. Installation and on-site acceptance were done by the Cosylab team at LLN. The test plan was executed by SCK for final acceptance.

Cosylab managed the complete project, while SCK remained in full control through requirement and test plan approval. Using such an approach, SCK was able to make good use of the available resources, while fully controlling the deliverable to obtain the required functionality.

Any additional ideas for functionality were noted as feature requests and will be implemented in the next version of the application.

CONCLUSION

A fully functional, configurable DAQ solution was developed for MYRRHA [6] using National Instruments hardware and drivers. The DAQ application runs in LabVIEW on native hardware. For integration into the EPICS control system, the CALab was used. The user interface for configuration and data visualization was done in Control System studio.

REFERENCES

[1] NI-DAQmx Software, https://www.ni.com/dataacquisition/nidaqmx.htm

[2] PXI Systems, http://www.ni.com/sl-si/shop/pxi.html

[3] CALab, https://www.helmholtz-berlin.de/zentrum/locations/it/software/exsteuer/calab/index_en.html

[4] TFS file format, http://mad.web.cern.ch/mad/madx.old/Introduction/tfs.html

[5] Control System Studio, http://controlsystemstudio.org/records.html

[6] MYRRHA, http://sckcen.be/en/Technology_future/MYRRHA